An Adaptive View Element Framework for Multi-dimensional Data Management

John R. Smith and Chung-Sheng Li
IBM T.J. Watson Research Center
Data Management
30 Saw Mill River Road
Hawthorne, NY 10532
{jrsmith, csli}@watson.ibm.com

Abstract

We present an adaptive wavelet view element framework for managing different types of multi-dimensional data in storage and retrieval applications. We consider the problems of multi-dimensional data compression, multi-resolution subregion access, selective materialization, progressive retrieval and similarity searching. The framework uses wavelets to partition the multi-dimensional data into view elements that form the building blocks for synthesizing views of the data. The view elements are organized and managed using different view element graphs. The graphs are used to guide cost-based view element selection algorithms for optimizing compression, access, retrieval and search performance.

We present the adaptive wavelet view element framework and describe its application in managing multi-dimensional data such as 1-D time series data, 2-D images, video sequences, and multi-dimensional data cubes. We present experimental results that demonstrate that the adaptive wavelet view element framework improves performance of compressing, accessing, and retrieving multi-dimensional data compared to non-adaptive methods.

Keywords – Multimedia database systems, data management, OLAP, data cubes, content-based search, digital libraries, and information retrieval.

1 Introduction

Enabling the efficient storage, access, query and retrieval of large volumes of multi-dimensional data is one of the important emerging problems in databases. Many multi-dimensional database systems are beginning to be deployed on-line, such as those that serve time series data, large images, video sequences and views of data cubes. In many of these applications, the data items have great size and require significant storage space and transmission bandwidth. Furthermore, the large volume of data greatly complicates the handling of the multi-dimensional data by the database systems. As a result, specialized solutions are needed for compressing, storing, accessing, retrieving and searching the multi-dimensional data.

CIKM '99 11/99 Kansas City, MO, USA

Some of the problems are addressed by partitioning the multi-dimensional data and adaptively compressing, storing, retrieving and querying the partitions. However, so far, no attempt has been made to develop a unified framework across different types of multi-dimensional data and all of these functions. We previously explored applications of storage and retrieval of large images [1], progressive retrieval of video sequences [2], similarity query of multi-dimensional vectors [3] and on-line analytic processing (OLAP) of multi-dimensional data cubes [4]. In this paper, we develop a common wavelet view element framework that integrates the adaptation strategies across all of the facilities of the database for better managing multi-dimensional data.

1.1 Related work

One of the key elements of multi-dimensional data management is the partitioning of the data. By breaking the data into smaller more manageable units, the data can be more easily handled by the storage, retrieval and query subsystems. Once partitioned, the units can be differentially compressed, stored, accessed, retrieved and searched.

Adaptive partitioning methods have been previously investigated for compression and storage. In particular, adaptive wavelet partitionings [5, 6] and spatial quad-trees [7] have been shown to be effective in compressing images [8], video [2] and time-series data [9, 10]. Similarly, wavelet partitionings have been used to speed-up access to sub-regions of large images [11]. For example, allocating wavelet partitions of images to different storage facilities allows higher image view access parallelism and data throughput [12]. Wavelet partitioning also enables progressive retrieval in which data is incrementally retrieved from the server, cached, and re-used by the client in synthesizing views locally. In progressive retrieval, methods of dividing the view synthesis computation between server and client have been shown to speed-up the retrieval of views of large images [1, 13].

In database applications, similar problems are found in OLAP [14, 15, 16]. In OLAP, the queries perform range-aggregations over the cells of multi-dimensional data cubes. Previously, we applied wavelet partitioning to data cubes to address problems of selective materialization of views [4]. Vitter, et al. also used wavelets to compress and approximate the data cubes [17].

For similarity searching of multi-dimensional data, dimensionality reduction is a form of partitioning that speeds-up querying and allows indexing. Previously, we used wavelets to partition and compress multi-dimensional histogram data for content-based image retrieval [3, 18]. In general, dimen-

sionality reduction and multi-dimensional indexing can be integrated to index high-dimensional data (for example, see Fastmap [19], RCSVD [20], SVDD [21], and QBIC [22]).

1.2 Overview

In order to provide a uniform approach for managing multi-dimensional data, we develop an adaptive wavelet view element framework. The framework performs different structured iterative partitionings of the data managed by view element graphs. Depending on the application and data type, the partitioning is performed along multiple dimensions in *space* and *frequency*. Initially, the view element partitionings generate *over-complete* and *redundant* representations of the multi-dimensional data.

The framework uses different view element selection algorithms that select among candidate sub-sets of the view elements in order to optimize compression, storage, query and retrieval. In general, we are concerned in the selection processes with properties of *completeness* and *non-redundancy* of the view element sets. We briefly discuss the significance of *completeness* and *non-redundancy* and describe how the wavelet view element framework provides advantages for compression, access, storage, retrieval and searching.

1. **Compression** – the wavelet view element approach partitions the multi-dimensional data in space and frequency. Typically, much of the energy is compacted into a small number of view elements. By assigning a compression cost, i.e., entropy [5], energy, variance, rate-distortion [6], to each view element, we select the *complete* and *non-redundant* set of view elements that yields the lowest total compression cost, or highest compression performance [8].

2. **View access** – given that different views are likely to be accessed with different frequency, the mult-dimensional data can be adaptively partitioned and stored in a form that minimizes the average access cost. To optimize the storage for efficient view access, we assign each view element an access frequency (actual or estimated) and processing cost of supporting the query population. We then select the *complete* and *non-redundant* set of view elements that yields the lowest total support cost.

3. **Selective materialization** – selective materialization is similar to view access pattern adaptation except that we relax the *non-redundancy* constraint of the view element sets. View element sets are selected to minimize a total cost for supporting the population of queries without exceeding a storage budget. The selected set of view elements is stored in the database and used to generate the views at query time.

4. **Progressive retrieval** – in progressive retrieval, the client caches view elements retrieved from the server. For each client request, we determine the needed processing at the server and client and which view elements need to be transmitted to the client. The choices are made by assigning a support cost to each view element and selecting the least cost *complete* set that intersects with the requested view. This determines the necessary retrieval and processing operations that produce the requested view in each stage of progressive retrieval.

5. **Similarity search** – For M-D vectors and time-series data, the selection of the view elements allows a flexible trade-off in query precision and query response time. Given a multi-dimensional query, we approximate the data using an *incomplete* set of view elements. In order to compare the query data to the target data, we select the set of view elements that minimizes the work for matching and guarantees a given precision.

1.3 Outline

In this paper, we describe the adaptive wavelet view element framework for managing multi-dimensional data. In section 2, we describe different multi-dimensional storage and retrieval applications. In section 3, we propose different view element graphs for time series data, images, video sequences and M-D data cubes. In section 4, we present in detail the cost-based view element selection algorithms. In section 5, we describe application of the view element selection algorithms for optimizing multi-dimensional data compression, access, selective materialization, progressive retrieval and similarity searching, Finally, in Section 6, we evaluate the adaptive wavelet view element framework in compressing, storing and retrieving views of large 2-D images.

2 Multi-dimensional data management

Database systems for multi-dimensional data need to provide efficient storage and retrieval. The typical application environment consists of facilities for compressing, storing, accessing, analyzing, retrieving and querying the data, as illustrated in Figure 1. The primary function of the storage sub-system is to compress and store the non-structured multi-dimensional data in the database.

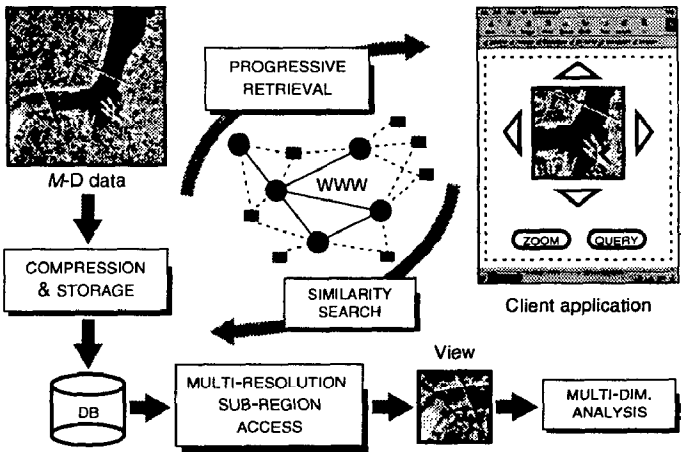

Figure 1: Typical functions of multi-dimensional databases include multi-dimensional data compression, efficient storage, multi-resolution sub-region view access, progressive view retrieval and similarity searching.

As described earlier, there are two dimensions for partitioning multi-dimensional data: space (includes time) and frequency (includes spatial- and temporal-frequency). Wavelets partition the data in frequency into logarithmically spaced subbands [23]. The low-frequency subband serves as a coarse approximation of the data. On the other hand, spatial grids and spatial quad-trees partition the data only in space.

For each of the different data types shown in Table 1, we develop a view element graph structure for partitioning the data in space and/or frequency. For 1-D time series data, vectors, and images, we partition the data in both space and frequency using a space and frequency graph [8]. For the video sequences, we partition the sequence first in time, then each temporal unit is partitioned in both spatial- and temporal-frequency. For M-D data cubes, we partition the data in frequency along each dimension.

Dimensions	Data type	View element structure
1-D	time series data	Space and frequency graph
2-D	large images	Space and frequency graph
3-D	video sequences	Spatio-temporal video graph
M-D	data cubes	View element graph

Table 1: Overview of view element structures for managing different types of multi-dimensional data.

3 View element graphs

In general, the view element graphs organize the hierarchies of transitions between view elements. The transitions correspond to the operations that partition (parent to child) or synthesize (children to parent) the data. Each view element is generated by applying the sequence of partitionings that follow from the root node of the view element graph.

In general, we distinguish between three types of view elements in a view element graph – views, intermediate view elements and residual view elements. Typically, the views and intermediate view elements are of most interest to users. For example, the views and intermediate view elements of the 2-D images correspond to image sub-regions depicted at different resolutions. For M-D data cubes, the intermediate view elements correspond to range-aggregations of the data. The residual view elements are used in combination with the views and intermediate view elements to synthesize the more detailed views.

3.1 Space and frequency graph

The space and frequency graph is constructed by iteratively partitioning the data in space and frequency [1]. For 1-D data, the data is partitioned spatially (S) via binary segmentation and in frequency (F) via a two-band Haar filter bank. For 2-D data, the data is partitioned spatially (S) via quad-tree segmentation and in frequency (F) via a four-band QMF filter bank. In order to guarantee commutativity in the space and frequency partitionings, the filter banks need to have a partitionable form. The Haar filter bank inherently has this property, and in general, any QMF filter-bank can be converted to a partitionable form [8].

The space and frequency decompositions are integrated to iteratively partition the data into the view elements. Figure 2(a) illustrates the process for partitioning the data using the space and frequency graph. Each element $v_{i,j,k,l}$ in the library corresponds to a space and frequency localized projection of the data, where i and j indicate the spatial and frequency resolution of the projection, and k and l indicate the space and frequency location of the projection.

3.1.1 Analysis

In order to partition the data, the space and frequency partitionings are iterated as follows:

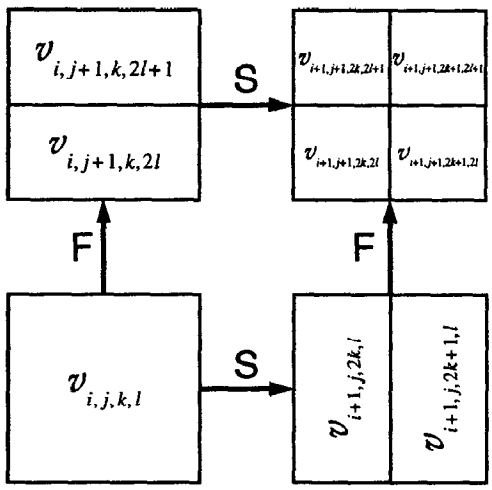

Figure 2: Commutavitity of the space and frequency partitioning operations in the space and frequency graph.

- Spatial partitioning S – $\mathbf{S}_0$ and $\mathbf{S}_1$ segment each view element $v_{i,j,k,l}$ to generate projections $v_{i+1,j,2k,l}$ and $v_{i+1,j,2k+1,l}$ as follows:

$$\begin{aligned} v_{i+1,j,2k,l} &= \mathbf{S}_0 v_{i,j,k,l} \text{ and} \\ v_{i+1,j,2k+1,l} &= \mathbf{S}_1 v_{i,j,k,l}. \end{aligned} \quad (1)$$

- Frequency partitioning F – the frequency partitioning operators $\mathbf{H}_0$ and $\mathbf{H}_1$ segment each view element $v_{i,j,k,l}$ into two frequency subbands to generate the projections $v_{i,j+1,k,2l}$ and $v_{i,j+1,k,2l+1}$ as follows:

$$\begin{aligned} v_{i,j+1,k,2l} &= \mathbf{H}_0 v_{i,j,k,l}, \text{ and} \\ v_{i,j+1,k,2l+1} &= \mathbf{H}_1 v_{i,j,k,l}. \end{aligned} \quad (2)$$

3.1.2 Synthesis

Given that each space (S) and frequency (F) partitioning is *non-redundant* and *complete* , a parent view element is synthesized from the children view elements. The synthesis of parent view elements in the space and frequency graph follows from:

$$\begin{aligned} v_{i,j,k,l} &= v_{i+1,j,2k,l} + v_{i+1,j,2k+1,l}, \text{ and} \\ v_{i,j,k,l} &= \mathbf{G}_0 v_{i,j+1,k,2l} + \mathbf{G}_1 v_{i,j+1,k,2l+1}, \end{aligned} \quad (3)$$

where the view element is synthesized from its frequency children using $\mathbf{G}_0$ and $\mathbf{G}_1$, where $\mathbf{G}_i = \mathbf{H}_i^T$ since the QMF filter banks, including Haar, satisfy the perfect reconstruction condition, $\mathbf{H}_0\mathbf{G}_0 + \mathbf{H}_1\mathbf{G}_1 = \mathbf{I}$. The perfect reconstruction property allows that the data is reconstructed from any *complete* set of elements.

3.2 Haar projection library (1-D)

For the 1-D data, the space and frequency partitionings are generated as follows using a Haar filter bank, as follows [3]:

- Spatial partitioning S – the spatial partitioning operators $\mathbf{S}_0$ and $\mathbf{S}_1$ partition v into two halves as follows, let $x_0 = \mathbf{S}_0 v$ and $x_1 = \mathbf{S}_1 v$, then

$$x_0[n] = \begin{cases} v[n] & n < N/2 \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

$$x_1[n] = \begin{cases} 0 & n < N/2 \\ v[n] & \text{otherwise.} \end{cases} \quad (5)$$

- Frequency partitioning F – the frequency partitioning operations correspond to the low-pass $\mathbf{H}_0$ and high-pass $\mathbf{H}_1$ signal transformations in the two-band Haar filter bank, respectively, and split v into two frequency subbands as follows, let $y_0 = \mathbf{H}_0 v$ and $y_1 = \mathbf{H}_1 v$, then

$$y_0[n] = \frac{\sqrt{2}}{2}(x[2n] + x[2n+1]) \quad (6)$$

$$y_1[n] = \frac{\sqrt{2}}{2}(x[2n] - x[2n+1]). \quad (7)$$

The Haar projection library is shown in Figure 3 for a time-series of eight points. The Haar projection library provides an extremely large number of different ways for representing the data. For example, for a time-series with 128 points, the library has 448 view elements that provide $O(10^{16})$ unique *complete* representations and $O(10^{33})$ unique *incomplete* projections of the data [3].

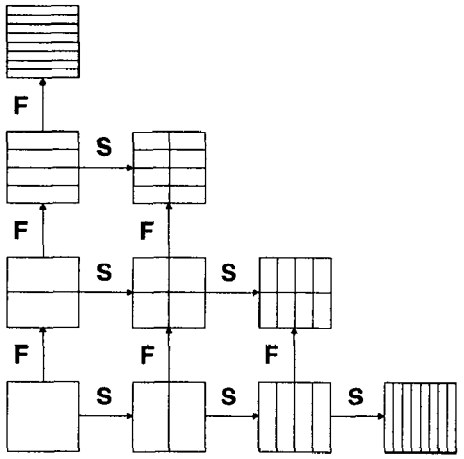

Figure 3: The Haar projection library partitions the 1-D and time-series data into wavelet view elements in space and frequency.

3.3 Space and frequency graph (2-D)

For 2-D data, the space and frequency partitioning operations are integrated symmetrically in a graph structured cascade as shown in Figure 4 [1, 8]. The partitioning combines spatial-quad-trees (S) and 2-D wavelet packet trees (F) to form a directed acyclic graph.

3.4 Video spatio-temporal graph (3-D)

For video sequences, we use a video graph to partition the sequences into the wavelet view elements [2]. The video sequences are first partitioned temporally into fixed size units (i.e., groups of 64 frames). Then, each unit is partitioned using the spatio-temporal video graph, as shown in Figure 5. The video graph is constructed by integrating spatial and temporal filter bank building blocks. Overall, the view elements generated by the video graph correspond to subbands with different locations and sizes in spatial- and temporal-frequency.

3.5 View element graph (M-D)

For M-D data cubes, the view element graph partitions the data along each dimension by taking the sum (P_1^i) and difference (R_1^i) of adjacent cells along each dimension i [4]. The view element graph manages the view elements and provides a data structure for evaluating the *completeness*, *non-redundancy* and benefits of different view element sets. The view element graph organizes the view elements according to the forward- and reverse-dependencies among the view elements. An example 2-D view element graph for a 2-D data cube is depicted in Figure 6.

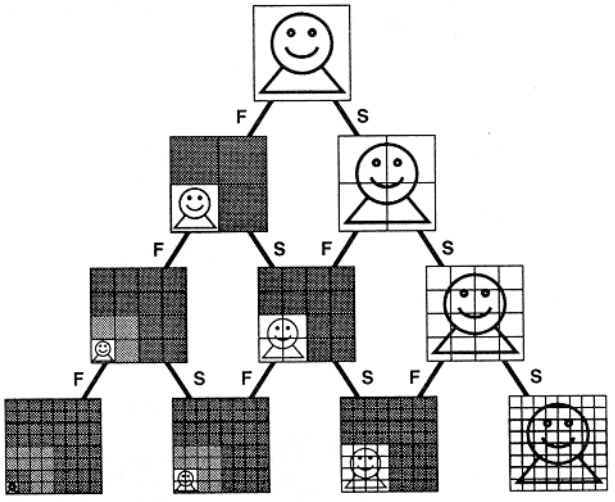

Figure 4: The space and frequency graph partitions 2-D lattice data such as large images into wavelet view elements in space and frequency.

4 Selection algorithms

Given the different view element graphs for 1-D time series data, 2-D images, video sequences and M-D data cubes, we develop a number of cost-based methods for selecting sets of view elements that optimize the performance of compressing, accessing, selective materialization, progressive retrieval and similarity searching. These methods utilize several cost-based algorithms for the selection of sets of view elements under different conditions of *completeness* and *non-redundancy*. For simplicity, we describe the algorithms in a form suitable for the space and frequency graph partitioning of 1-D time-series data(Fig 3). However, application to the other data types and view element graphs follows directly.

We consider first an algorithm for selecting a *complete* and *non-redundant* set of view elements based on any additive cost assigned to each view element.

Algorithm 1 (View element basis selection) *Given the assignment of an additive cost $C_{i,j,k,l}$ to each view element $v_{i,j,k,l}$, the complete and non-redundant set of view elements of least total cost is found as follows:*

1. Start from the root view element $v_{0,0,0,0}$ in the graph, and recursively at each child view element $v_{i,j,k,l}$, compute the cost $C^*_{i,j,k,l}$ of the selected least-cost path given by:

$$C^*_{i,j,k,l} = \min(\; C_{i,j,k,l}, \; C^*_{i+1,j,2k,l} C^*_{i+1,j,2k+1,l}, \; C^*_{i,j+1,k,2l} + C^*_{i,j+1,k,2l+1}), \quad (8)$$

where $C_{i,j,k,l}$ is the cost of view element $v_{i,j,k,l}$,

$$C^*_{i+1,j,2k,l} + C^*_{i+1,j,2k+1,l}$$

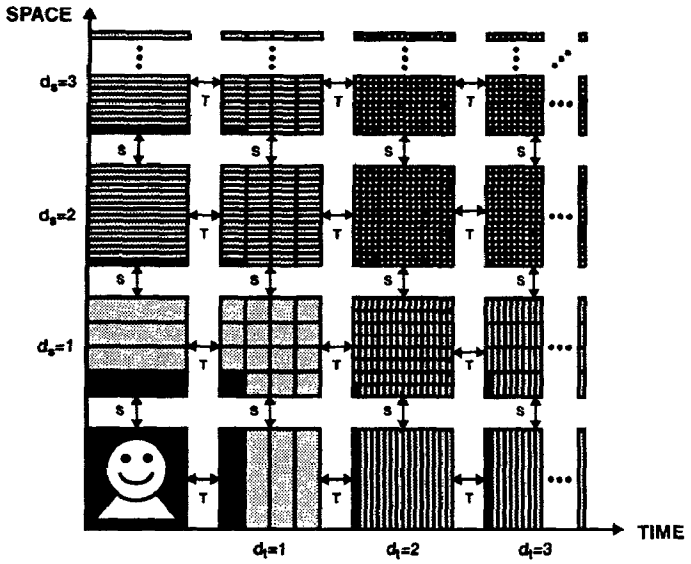

Figure 5: Video spatio-temporal graph partitions the video sequences into view elements in spatial- and temporal-frequency.

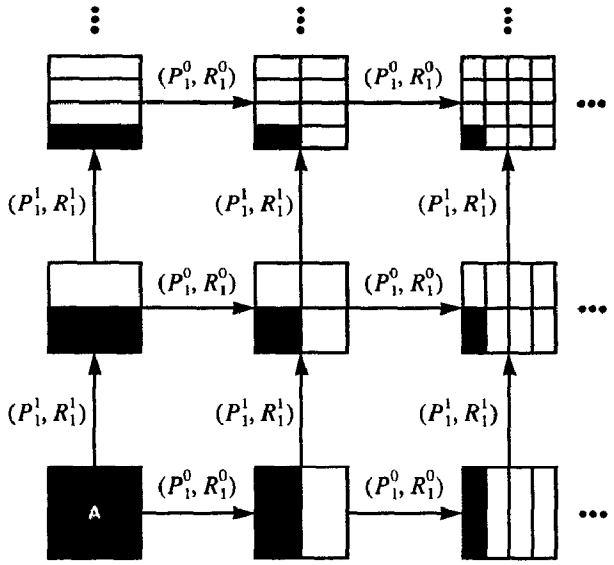

Figure 6: The M-D view element graph partitions M-D data cubes into wavelet view elements in frequency along each dimension. The view element graph organizes the view elements into a two-way (analysis, synthesis) dependency graph.

is the optimal total cost of the S child path, and

$$C^*_{i,j+1,k,2l} + C^*_{i,j+1,k,2l+1}$$

is the optimal total cost of the F child path.

2. Mark by $L_{i,j,k,l}$ the choice with lowest cost.
3. Start again from the root view element $v_{0,0,0,0}$ and follow the paths according to the marked choices $L_{i,j,k,l}$.
4. Traverse again the view element graph from the root node. The set of terminal view elements encountered in the traversal form the *complete* and *non-redundant* set of lowest cost.

The view element basis selection algorithm is suited for selecting representations of the multi-dimensional data that do not expand the amount of data and reconstruct the data without information loss. Basis selection is well suited for compression. However, in some applications, such as similarity searching, it is desirable to select an *incomplete* set of view elements. The following algorithm utilizes a greedy approach for selection of an *incomplete* and *non-redundant* set of view elements.

Algorithm 2 (Incomplete view element set selection) *Given the assignment of an additive cost $C_{i,j,k,l}$ to each view element $v_{i,j,k,l}$, an incomplete and non-redundant set of view elements of low total cost and set size K is found as follows:*

1. Initialize all view elements $v_{i,j,k,l}$ to be unblocked and not selected.
2. For $n = 0$ to $K-1$ do
   (a) Find the view element $v^n_{i,j,k,l}$ that has lowest cost $C^n_{i,j,k,l}$, is not blocked and is not selected. Mark $v^n_{i,j,k,l}$ selected.
   (b) Find the remaining view elements that intersect with $v^n_{i,j,k,l}$ in the space-frequency plane. Mark those view elements as blocked.
   (c) Let $n \leftarrow n+1$.
3. Read off the K selected view elements.

Alg. 2 is not optimal since it uses a greedy approach. However, Alg. 2 can also be used to select view elements for a given storage capacity rather than based on the set size limit K. We next consider the case of the selection of a *complete* and *redundant* set of view elements.

Algorithm 3 (Redundant view element set selection) *Given the assignment of an additive cost $C_{i,j,k,l}$ to each view element $v_{i,j,k,l}$, the complete and redundant set of elements of that has low cost and does not exceed a given storage capacity is found follows:*

1. Use Alg. 1 to select the *complete* and *non-redundant* set of view elements with least total cost.
2. Mark all of the selected view elements as blocked.
3. Use Alg. 2 to select additional view elements without exceeding the storage capacity (greedy addition).

5 View element management

We next describe how to use the view element selection algorithms for compression, access, selective materialization, progressive retrieval and similarity searching.

5.1 Adaptive compression

For both the lossless and lossy compression of multi-dimensional data, we use Alg. 1 for selecting the basis that best compacts the data. For example, in the case of large images, Alg. 1 adapts the partitioning of the image in space and spatial-frequency. We assign each of the view elements a coding cost, where in the lossless coding case, we base the cost on

the actual data size of each losslessly encoded view element. In the lossy case, we compress each view elements using different compression factors to generate an operational rate-distortion curve, from which we compute the compression cost. Then, Alg. 1 is used to select the *complete* and *non-redundant* set of view elements that has the lowest total cost. The system deletes the remaining view elements, and compresses and stores the selected ones in the database [8].

5.2 View access

For optimizing view access, we define a cost function that provides a basis for improving view access performance as follows: we derive the cost of processing each view element from the volume of the intersection of the view element with the requested view (as in [4]). Then, the total cost of generating a view is given by the sum of the view element processing costs. This cost function allows us to measure the view access performance of the system. Given a population of queries, and given a set of stored view elements, we compute the total cost of processing the queries. Next, importantly, given a population of queries, we determine the optimal set of view elements that need to be stored in the database to give the lowest total processing cost [1].

For a given access pattern we determine the optimal set of stored view elements, or equivalently, the space and frequency partitioning of the image as follows:

Algorithm 4 (Access pattern adaptation) *Given the access frequency $p_{i,j,k,l}$ of each view element $v_{i,j,k,l}$, the complete and non-redundant set of view elements with lowest total access cost for the population of queries is found as follows:*

1. For each view element $v_{i,j,k,l}$, compute the processing cost $C_{(i,j,k,l)\rightarrow(i',j',k',l')}$ for supporting each other view $v_{i',j',k',l'}$ (this cost is set to zero in absence of dependency).
2. Let $C_{i,j,k,l} = \sum_{i',j',k',l'} p_{i,j,k,l} C_{(i,j,k,l)\rightarrow(i',j',k',l')}$ give the total cost of each view element $v_{i,j,k,l}$.
3. Use Alg. 1 to determine the *complete* and *non-redundant* set of view elements with the lowest total processing cost.

5.3 Selective materialization

We extend the access pattern adaptation method to the selective materialization of view elements. In this case, we have storage space that exceeds the volume of the multi-dimensional data. We use this additional storage space to further reduce the processing cost for the different views of the data [4].

Algorithm 5 (Selective materialization) *Given the access frequency $p_{i,j,k,l}$ of each view element $v_{i,j,k,l}$, the complete and redundant set of view elements that has the least total access cost for a population of queries is found as follows:*

1. For each view element $v_{i,j,k,l}$, compute the processing cost $C_{(i,j,k,l)\rightarrow(i',j',k',l')}$ for supporting each other view $v_{i',j',k',l'}$.
2. Let $C_{i,j,k,l} = \sum_{k} p_{i,j,k,l} C_{(i,j,k,l)\rightarrow(i',j',k',l')}$ give the total cost of each view element $v_{i,j,k,l}$.
3. Use Alg. 3 to determine the *complete* and *redundant* set of view elements with the lowest total processing cost.

5.4 Progressive retrieval partitioning

In progressive retrieval, we consider the access cost of each view using view elements at the client and server [1]. We determine the optimal division of work between client and server for each client request of view $v_{i,j,k,l}$ as follows:

Algorithm 6 (Progressive retrieval algorithm) *Given the access cost $C^s_{i,j,k,l}$ of each view element $v_{i,j,k,l}$ at the server, and transmission cost $C^t_{i,j,k,l}$ for transmitting $v_{i,j,k,l}$ to the client, the set of view elements to retrieve in each step of progressive retrieval is found as follows:*

1. Assign an access cost $C^c_{i,j,k,l} = 0$ to each of view element $v_{i,j,k,l}$ in the client cache, otherwise $C^c_{i,j,k,l} = \infty$.
2. Use Alg. 1 to select a *complete* and *non-redundant* set of view elements from the server and client, considering the server $C^s_{i,j,k,l}$ and client $C^c_{i,j,k,l}$ access cost of each view element.
3. Retrieve and process the view elements accordingly to synthesize the view $v_{i,j,k,l}$.

5.5 Similarity search

The similarity search of the multi-dimensional data can be computed with some precision loss using an *incomplete* set of view elements. We devise a query computation procedure that guarantees a query precision bound of $1-\epsilon$, where ϵ can be determined by the system or user. The query and target vectors are first partitioned into wavelet view elements. Then at each successive stages of matching, the residual energy of the query and target vectors is compared to the threshold $1-\epsilon$ to terminate the similarity computation. We make the worst case assumption that the residual energy is retained in the same view elements for the query and target vectors. This alignment maximizes the residual energy intersection and provides a bound for the error in the query computation [3].

6 Evaluation

In order to evaluate the adaptive wavelet view element selection methods, we performed experiments for compression, access, progressive retrieval of large images.

6.1 Compression evaluation

In order to evaluate compression performance, we compare the adaptive wavelet view element method using Alg. 1 to compression algorithms based on JPEG, wavelets and spatial segmentation. Figure 7 shows the resulting rate-distortion compression results for two images. For JPEG, we compressed each image several times using different JPEG quality factors. We obtained the rate from the compressed file size. We measured the distortion from the fidelity (PSNR) of the decompressed image. For wavelets, segments, blobs and the space and frequency graph, we obtained the rate-distortion results by partitioning the image, quantizing the partitions, and measuring the entropy (rate) and fidelity (PSNR).

Figure 7 shows that, in practise, the space and frequency graph performs measurably better. The rate-distortion plots in Figure 7(a) correspond to the compression of the 512×512 *Barbara* image [23]. For a given rate, space and frequency graph gives 2.3 to 3.1 dB higher fidelity than wavelets and 3.7 to 4.9 dB higher fidelity than JPEG. Compression based

on spatial segmentation and blobs performs substantially worse than the space and frequency graph.

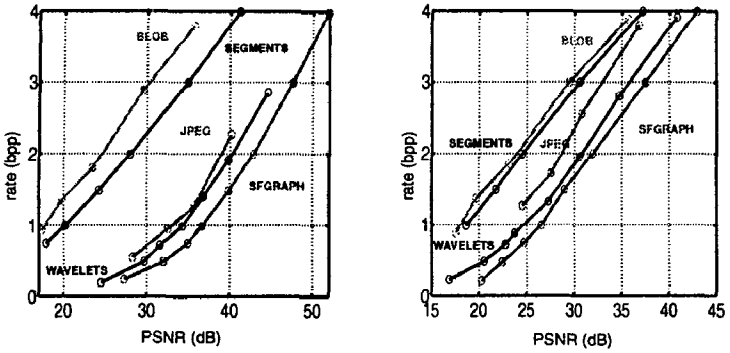

Figure 7: Lossy compression evaluation using adaptive wavelet view element framework: (a) results for 512 × 512 "Barbara" image, and (b) results for 5962 × 5962 satellite image.

6.2 Access adaptation evaluation

In order to evaluate the wavelet view element access adaptation strategy using Alg. 4, we simulated different access modes by randomly accessing views. We repeated the experiments for the space and frequency graph of different depths for large 2-D images. Figure 8 shows the significant reduction in view access cost by adapting the selection and storage of the view elements to the access patterns. We compared the view access performance to that of other image partitioning and storage schemes based on segments, blobs and wavelets [1].

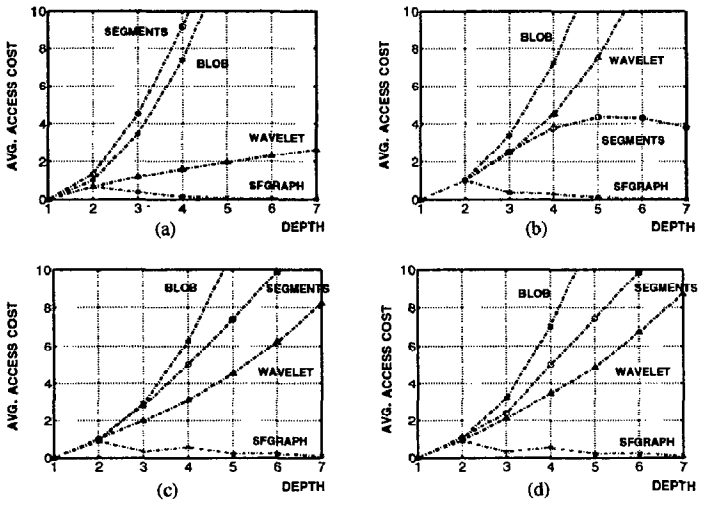

Figure 8: Evaluation of average view access costs using the adaptive wavelet view element framework ("sfgraph") for different access patterns (a) fixed-grid drill-down, (b) arbitrary spatial drill-down, (c) equal probability access, and (d) arbitrary multi-resolution access.

1. **Fixed-spatial grid drill-down.** The view element method adapts to this access pattern by storing a tiled-wavelet set of view elements. Figure 8(a) shows a 22 - 231× reduction in access costs over the other methods. The fixed wavelet transform performs worse but is somewhat more suited for this type of drill-down browsing than segments and blobs.

2. **Arbitrary spatial drill-down.** The view element method adapts to this access pattern also by storing a tiled-wavelet set of view elements. Figure 8(b) shows a 14.7 - 109× reduction in access costs over the other methods. Interestingly, segments are somewhat better for this type of drill-down than fixed wavelets. However, the space and frequency graph performs significantly better than both methods.

3. **Equal probability view access.** When all views are equally likely to be accessed, the adaptive wavelet view element framework, as shown in Figure 8(c) gives a 9.3 - 13.9× reduction in access costs over the other methods. Fixed wavelets, segments and blobs are not well suited for supporting this type of access.

4. **Arbitrary multi-resolution access.** When the users drill-down into the images, but vary the spatial size and location of the zoom, the adaptive wavelet view element framework, as shown in Figure 8(d) gives s 9.8 - 29.1× reduction in access costs over the other methods.

6.3 Progressive retrieval evaluation

In order to evaluate the retrieval adaptation strategy using Alg. 6, we simulated the random zooming and panning of large images by a remote client. We varied the relative processing power of the server and client, and varied the transmission bandwidth. Figure 9 shows the comparison of the adaptive strategy ("adapt") where both client and server participate in synthesizing views to strategies where only the client ("client") or server ("server") synthesize the views. In each user click shown in Figure 9, the user randomly zooms-in, zooms-out or pans up, down, left or right.

The results in Figure 9 show that the adaptive strategy minimizes the cumulative latency in progressively retrieving the views over the network. For example, Figure 9(a) shows the result for a thin client with 0.1× the processing power of the server. Performing all of the processing at the client is not optimal. On the other hand, performing all of the work at the server does not take full advantage of the view elements in the client cache. The optimal strategy adaptively partitions the space and frequency graph view synthesis between server and client [1].

7 Summary

We developed a common adaptive wavelet view element framework for managing different types of multi-dimensional data. We considered the problems of optimizing storage, compression, access, progressive retrieval and similarity searching of 1-D time series data, 2-D images, video sequences, and multi-dimensional data cubes. We presented experimental results on large 2-D images that demonstrate the significant performance improvements of the view element framework in applications that require efficient storage and compression, multi-resolution sub-region view access and progressive retrieval of the multi-dimensional data.

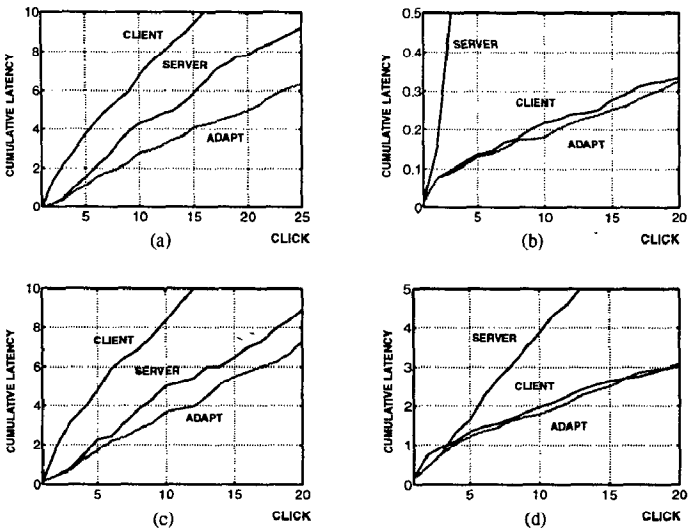

Figure 9: Progressive retrieval with adaptive partitioning of view element synthesis between server and client: (a) thin client, (b) powerful client, (c) thin client and low bandwidth, (d) normal client and low bandwidth.

References

[1] J. R. Smith, V. Castelli, and C.-S. Li. Adaptive storage and retrieval of large compressed images. In *IS&T/SPIE Symposium on Electronic Imaging: Science and Technology - Storage & Retrieval for Image and Video Databases VII*, San Jose, CA, January 1999.

[2] J. R. Smith. VideoZoom spatio-temporal video browser. *IEEE Trans. Multimedia*, 1(2):157 – 171, 1999.

[3] J. R. Smith. Query vector projection access method. In *IS&T/SPIE Symposium on Electronic Imaging: Science and Technology - Storage & Retrieval for Image and Video Databases VII*, San Jose, CA, January 1999.

[4] J. R. Smith, V. Castelli, A. Jhingran, and C.-S. Li. Dynamic assembly of views in data cubes. In *Proc. ACM Principles of Database Systems (PODS)*, pages 274–283, June 1998.

[5] R. R. Coifman and M. V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Trans. Inform. Theory*, 38(2), March 1992.

[6] K. Ramchandran and M. Vetterli. Best wavelet packet bases in a rate-distortion sense. *IEEE Trans. Image Processing*, June 1993.

[7] E. Shusterman and M. Feder. Image compression via improved quadtree decomposition algorithms. *IEEE Trans. Image Processing*, 3(2):207 – 215, 1994.

[8] J. R. Smith and S.-F. Chang. Joint adaptive space and frequency graph basis selection. In *IEEE Proc. Int. Conf. Image Processing (ICIP)*, Santa Barbara, CA, October 1997.

[9] C. Herley, J. Kovačević, K. Ramchandran, and M. Vetterli. Tilings of the time-frequency plane: Constructions of arbitrary orthogonal bases and fast tiling algorithms. *IEEE Trans. Signal Processing*, December 1993.

[10] S. Mallat and Z. Zhang. Matching pursuit with time-frequency dictionaries. *IEEE Trans. Signal Processing*, December 1993.

[11] A. S. Poulikidas, A. Srinivasan, O. Egecioglu, O. Ibarra, and T. Yang. A compact storage scheme for fast wavelet-based subregion retrieval. In *Proc. Computing and Combinatorics Conference (COCOON '97)*, 1997.

[12] S. Prabhakar, S. Agrawal, A. El Abbadi, A. Singh, and T. R. Smith. Browsing and placement of multiresolution images on secondary storage. Technical Report TRCS96-22, UCSB, 1996.

[13] D. Andresen, T. Yang, D. Watson, and A. Poulakidas. Dynamic processor scheduling with client resources for fast multi-resolution WWW image browsing. In *Proc. Intern. Parallel Processing Symposium (IPPS)*, 1997.

[14] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proc. of the 12th Int. Conf. on Data Engineering*, pages 152–159, 1996.

[15] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *ACM Proc. Int. Conf. Manag. Data (SIGMOD)*, May 1997.

[16] R. Agrawal, A. Gupta, and S. Sarawagi. Modeing multidimensional databases. In *13th Int'l Conf. on Data Engineering*, April 1997.

[17] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In *Proc. ACM Intern. Conf. on Information and Knowledge Management (CIKM)*, Washington, DC, November 1998.

[18] J. R. Smith and S.-F. Chang. VisualSEEk: a fully automated content-based image query system. In *Proc. ACM Intern. Conf. Multimedia (ACMMM)*, pages 87 – 98, Boston, MA, November 1996.

[19] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets. In *ACM Proc. Int. Conf. Manag. Data (SIGMOD)*, pages 163 – 174, 1995.

[20] A. Thomasian, V. Castelli, and C.-S. Li. Clustering and singular value decomposition for approximate indexing in high dimensional spaces. In *Proc. ACM Intern. Conf. on Information and Knowledge Management (CIKM)*, November 1998.

[21] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting Ad Hoc queries in large datasets of time sequences. In *ACM Proc. Int. Conf. Manag. Data (SIGMOD)*, May 1997.

[22] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23 – 32, September 1995.

[23] M. Vetterli and J. Kovačević. *Wavelets and Subband Coding*. Prentice-Hall, Inc, Englewood Cliffs, NJ, 1995.