

Dynamic Assembly of Views in Data Cubes

John R. Smith

Vittorio Castelli

Anant Jhingran

Chung-Sheng Li

IBM Thomas J. Watson Research Center
30 Saw Mill River Road, Hawthorne, NY 10532
{jrsmith,vittorio,anant,csli}@watson.ibm.com

Abstract

In this paper, we present a method for dynamically assembling views in multi-dimensional data cubes in order to more efficiently support data analysis and querying involving aggregations. The proposed method decomposes the data cubes into an indexed hierarchy of view elements. The view elements differ from traditional data cube cells in that they correspond to partial and residual aggregations of the data cube. The view elements provide highly granular building blocks for synthesizing the aggregated and range-aggregated views of the data cubes.

We propose a strategy for selecting and materializing the view elements based on the frequency of view access. This allows the dynamic adaptation of the view element sets to patterns of retrieval. We present a fast and optimal algorithm for selecting non-expansive view element sets that minimize the processing costs for generating a population of aggregated views. We also present a greedy algorithm for selecting redundant view element sets in order to further reduce processing costs. We demonstrate that the view element approaches perform better in terms of lower processing and storage costs than methods based on materializing views.

1 Introduction

On-line Analytical Processing (OLAP) is a recent form of decision-support system that analyzes data across many dimensions [4]. In the OLAP applications, the data cube provides a conceptual representation of the multi-dimensional data that is generated by mapping the functional attributes of the data to the dimensions of the cube. As a result, the operations in OLAP that aggregate and summarize the data correspond to operations over the cells of the data cube.

Two approaches are emerging for handling data cubes in OLAP applications. Relational OLAP (ROLAP) uses standard relational databases for storing, generating and querying the data cubes. In order to support queries involving aggregations, Gray et al. [6] proposed the cube operator, which computes the aggregations over all combinations of dimensions of the data cube. Multidimensional OLAP (MOLAP)

systems, on the other hand, store the data cubes explicitly in the form of multi-dimensional arrays. They require specialized software for generating, manipulating and querying the data cubes. However, this approach allows for techniques in signal processing, such as the view element method, to be incorporated to add new functionality.

There are many issues involved with MOLAP data cubes. For one, the data cubes can be large for high-dimensional data. Furthermore, the nature of the data in databases is often such that it results in sparse and inefficient data cubes [10]. More importantly, in order for the OLAP applications to be interactive, the operations performed on the data cubes need to be fast [4]. Much of the work on data cubes is directed towards the efficiency in accessing aggregated views. The difficult problem often addressed concerns identifying the best set of views to materialize [7, 8]. One common theme in the treatment of data cubes is that redundant data needs to be generated in order to gain efficiency. This results largely from the lack of a framework that allows the aggregated data to be re-used for synthesizing parent views in the data cube.

1.1 Related Work

There has been recent work on developing methods for generating, querying, and maintaining data cubes [1, 2, 6, 9]. Agrawal, et al., developed several algorithms for computing collections of related aggregations in data cubes [2]. Ross and Srivastava proposed an algorithm for the fast computation of data cubes over sparse relations [10]. Zhao et al., recently investigated the problem of generating data cubes that are stored explicitly using sparse multi-dimensional arrays [13].

In the case of high-dimensional data sets, the storage overhead of the views can become significant. As a result, it is difficult to materialize and store all of the aggregated views in the data cube. Harinarayan, et al., [8] and Gupta, et al., [7] solve this problem by organizing the views into a view dependency hierarchy. They observe that since some views may be computed from other views, it is possible to trade-off the storage requirements for work at query-time. A greedy, polynomial-time algorithm for selecting the best set of views to answer a given population of queries was provided in [8]. Gupta, et al., address the problem of selecting, jointly, the best set of views and indices given space constraints. [7]

We point out some limitations of the view dependency approaches. First, the dependencies between views is always one-way: from parent to child. Second, redundant information is retained by storing views in addition to the

data cube. The view dependency hierarchy illustrates this redundancy, since all views are dependent on the original data cube [8]. We develop a new framework that addresses these limitations by decomposing the data cube into view elements.

1.2 Contribution

In this paper, we present a method for representing the data cube in terms of *aggregated*, *intermediate* and *residual* view elements. The view elements provide higher granularity in decomposing the data cube than the view dependency hierarchy. For each aggregation operator, we design a pair of partial decomposition operators that satisfy the properties of *perfect reconstruction*, *non-expansiveness*, *distributivity* and *separability*. The partial decomposition operators are cascaded to compute the aggregations in the data cube. The perfect reconstruction property allows parent views to be synthesized from the children view elements. The non-expansiveness allows the generation of complete sets of view elements that do not increase the volume of the data cube.

We develop a view element graph to manage the view elements. The view element graph assists in evaluating the relative benefits of view element sets for allowing fast retrieval of a specified population of queries. In this paper we make the following contributions:

1. We develop a framework for decomposing data cubes into view elements using partial and residual aggregation operators.
2. We develop partial and residual aggregation operators for the SUM function that are complete, non-expansive, distributive and separable.
3. We describe a view element graph data structure for managing (generating, synthesizing, evaluating, and selecting) view elements in MOLAP applications.
4. We develop a fast algorithm for selecting the optimal non-redundant view element set for a given population of queries that minimizes the processing costs.
5. We develop a greedy algorithm for adding redundant view elements in order to minimize the processing and storage costs.

1.3 Outline

The paper is organized as follows. In Section 2, we present the data cube model for representing views in OLAP applications. In Section 3, we develop the framework for decomposing data cubes into view elements using partial and residual aggregation operators. In Section 4 we describe how to manage the view elements using a view element graph. Then, in Section 5, we present two algorithms for selecting the view element sets according to populations of queries. In section 6, we examine how to compute range-aggregation queries using the view element graph. Finally, in Section 7, we demonstrate the reduction in the processing and storage costs of the view element method compared to other methods of representing views in data cubes.

2 Preliminary

We start out by defining some preliminary information about the data cubes. We consider that the data set is initially stored in a relational table R that has d *functional*

attributes and at least one *measure attribute*. We have an aggregation operator, denoted by S , which is to be applied to the measure attribute.

Then, we denote by A , the d -dimensional data cube generated from relation R by mapping the m -th functional attribute of R to dimension i_m in A . Given a domain of size n_m for the m -th functional attribute, the data cube A has a volume of $Vol(A) = \prod_{m=0}^{d-1} n_m$. We assume for convenience in the following that $\forall_m n_m = 2^{k_m}$ for integer k_m . Each cell in A contains an aggregation of the measure attribute of all records in R that map to that cell. In the following, we denote by S^m the aggregation of a dimension i_m of A .

3 View elements

We present a new framework for decomposing the data cube into view elements. The decomposition is achieved using partial and residual aggregation operators. These operators provide the building blocks for computing the SUM aggregation over various combinations of dimensions in the data cube.

3.1 Partial sum

We define the first partial sum operator pair (P_1^m, R_1^m) for a dimension i_m of A as follows:

$$P_1^m(A) = \sum_{l=0}^1 A[i_0, \dots, 2i_m + l, \dots, i_{d-1}] \quad (1)$$

$$R_1^m(A) = \sum_{l=0}^1 (-1)^l A[i_0, \dots, 2i_m + l, \dots, i_{d-1}]. \quad (2)$$

We recognize (P_1^m, R_1^m) as the multi-dimensional extension of the discrete-time Haar filter bank [12, pages 98 – 103]. We denote by $P_1^m(A)$, the first partial aggregation and by $R_1^m(A)$, the first partial residual. We can see from Eq 1 that $P_1^m(A)$ takes the sums of neighboring pairs of points along each dimension of A , and subsamples by two. R_1^m simply takes the differences and subsamples by two. We define the following important properties of (P_1^m, R_1^m) :

Property 1 Perfect reconstruction – *The pair (P_1^m, R_1^m) satisfies the perfect reconstruction property since A can be perfectly reconstructed from $P_1^m(A)$ and $R_1^m(A)$.*

The perfect reconstruction of A is achieved from:

$$A[i_0, \dots, 2i_m, \dots, i_{d-1}] = \frac{1}{2}(P_1^m(A) + R_1^m(A)) \quad (3)$$

$$A[i_0, \dots, 2i_m + 1, \dots, i_{d-1}] = \frac{1}{2}(P_1^m(A) - R_1^m(A)). \quad (4)$$

It is well know that the Haar filters satisfy the perfect reconstruction property. Many other filter pairs, which we do not investigate here, satisfy this property. We justify the choice of the Haar filters because of their short length (two taps) and low complexity. Furthermore, they enable the intermediate view elements to be used in range-aggregation queries, which we explore later in Section 6.

In order to generalize the operators, we first define $P_0^m(A) = R_0^m(A) = A$. Then, we have the following recursive expression for computing the partial and residual aggregations,

where for convenience we denote $P_{k-1}^m(A)$ and $R_{k-1}^m(A)$ by P_{k-1}^m and R_{k-1}^m , respectively,

$$P_k^m(P_{k-1}^m) = \sum_{l=0}^1 P_{k-1}^m[i_0, \dots, 2i_m + l, \dots, i_{d-1}] \quad (5)$$

$$R_k^m(P_{k-1}^m) = \sum_{l=0}^1 (-1)^l P_{k-1}^m[i_0, \dots, 2i_m + l, \dots, i_{d-1}]. \quad (6)$$

The distributivity of P_k^m and R_k^m enable the first partial aggregation operators to be cascaded along a dimension of A to generate the k -th partial aggregation.

Property 2 Distributivity – $P_k^m(A)$ can be computed by applying P_1^m recursively to $P_0(A)$.

The k -th partial aggregations are derived recursively as follows:

$$\begin{aligned} P_k^m(A) &= P_1^m(P_{k-1}^m(A)) \\ R_k^m(A) &= R_1^m(P_{k-1}^m(A)). \end{aligned} \quad (7)$$

This gives P_1^m and R_1^m the telescopic properties, i.e.,

$$P_k^m(A) = \underbrace{P_1^m(P_1^m(\dots(P_1^m(A))))}_k, \quad (8)$$

in which $P_k^m(A)$ is computed progressively. The distributivity allows us to generalize the property of perfect reconstruction in Property 1 to the k -th partial aggregation as follows:

$$P_{k-1}^m[i_0, \dots, 2i_m, \dots, i_{d-1}] = \frac{1}{2}(P_k^m(A) + R_k^m(A)) \quad (9)$$

$$P_{k-1}^m[i_0, \dots, 2i_m + 1, \dots, i_{d-1}] = \frac{1}{2}(P_k^m(A) - R_k^m(A)) \quad (10)$$

Not only does the pair (P_1^m, R_1^m) satisfy the perfect reconstruction property in Property 1, it is also non-expansive. This property is important in generating decompositions of the data cube that do not expand the volume of the data cube.

Property 3 Non-expansiveness – The pair (P_1^m, R_1^m) is non-expansive since it does not increase the volume of data.

We demonstrate the non-expansion property as follows: consider that we have a volume of A of

$$Vol(A) = \prod_{m=0}^{d-1} n_m. \quad (11)$$

Then, from Eq 1, we have a volume of $Vol(P_1^m(A))$ as follows:

$$Vol(P_1^m(A)) = n_0 n_1 \dots \frac{n_m}{2} \dots n_{d-1}, \quad (12)$$

and the same for $R_1^m(A)$. It follows that

$$Vol(P_1^m(A)) + Vol(R_1^m(A)) = Vol(A). \quad (13)$$

Finally, we have that P_1^m and R_1^m are separable since they operate independently on each dimension i_m of A .

Property 4 Separability – P_1^m and R_1^m are separable since they operate solely on dimension i_m of the input.

The separability and distributivity of P_1^m and R_1^m enable the input to be aggregated along multiple dimensions by cascading the aggregation operators in any order as follows:

$$P_{1,1}^{m,n}(A) = P_1^m(P_1^n(A)) = P_1^n(P_1^m(A)). \quad (14)$$

We now examine how to use the partial aggregation operators to form the building blocks for generating the aggregated views in the data cube.

3.2 Total aggregation

The total aggregation along a dimension i_m is computed by cascading P_1^m in succession $\log_2 n_m$ times, where n_m is the size of dimension i_m . For example, the total sum $S^m(A)$ of A along dimension i_m is given by

$$\begin{aligned} S^m(A) &= P_{\log_2 n_m}^m(A) \\ &= \underbrace{P_1^m(P_1^m(\dots(P_1^m(A))))}_{\log_2 n_m}. \end{aligned} \quad (15)$$

Using the Property 4, this is generalized to compute the total sum of A as follows:

$$\begin{aligned} S(A) &= \underbrace{S^0(S^1(\dots(S^{d-1}(A))))}_d \\ &= P_{\log_2 n_0}^0(P_{\log_2 n_1}^1(\dots(P_{\log_2 n_{d-1}}^{d-1}(A)))). \end{aligned} \quad (16)$$

We now relate the notion of views in the data cube A to the partial, residual and total aggregations that are computed from A . We define an aggregated view as follows:

Definition 1 Aggregated view – An aggregated view of data cube A is generated by totally aggregating A along any number of dimensions.

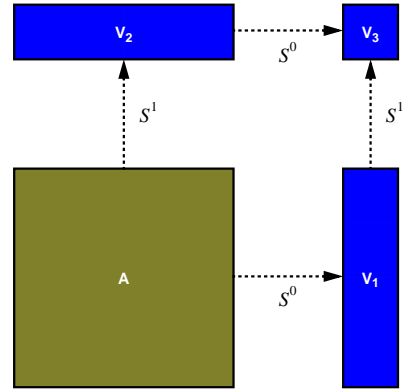


Figure 1: Example view hierarchy. The aggregated views (V_1 , V_2 , and V_3) are generated by totally aggregating A along dimensions i_0 and i_1 .

Figure 1 illustrates an example 2-D view hierarchy. The aggregated views of A are computed from $V_1 = S^0(A)$, $V_2 = S^1(A)$, and $V_3 = S(A) = S^0(S^1(A)) = S^1(S^0(A))$. Given that we can compute the total aggregations by cascading the first partial aggregators, we have the foundation for building a hierarchy of view elements. In general, we refer to the aggregated views, and all partial and residual aggregations as *view elements*.

Definition 2 View element – A view element of data cube A is generated by the total, partial, or residual aggregation of A along any number of dimensions.

We further distinguish two special cases of view elements, the *residual view elements* and *intermediate view elements*.

Definition 3 Residual view element – A residual view element of data cube A is generated by the residual aggregation of any view element of A along any number of dimensions.

If, in generating a view element, a residual aggregation has been used at any point, then that view element is defined to be a residual view element. The residual view elements are almost never of direct interest to the OLAP users. In the signal-processing, the residual views correspond to the high-frequency subbands, and are often important in signal analysis. We name the remaining view elements, some of which are aggregated views, intermediate view elements. The intermediate view elements that are not aggregated views are often of interest in OLAP applications that involve range-aggregation queries [9].

Definition 4 Intermediate view element – An intermediate view element is any view element that is not a residual view element.

The intermediate view elements are generated by applying the partial aggregation operator along some of the dimensions. Figure 2 illustrates an example 2-D view element hierarchy. The partial and residual aggregations of A are computed by applying P_1^m and R_1^m on each of the dimensions i_m of A . The view elements V_4 , V_6 , and V_{11} are intermediate view elements and V_5 , V_7 , V_8 , V_9 , and V_{10} are residual view elements.

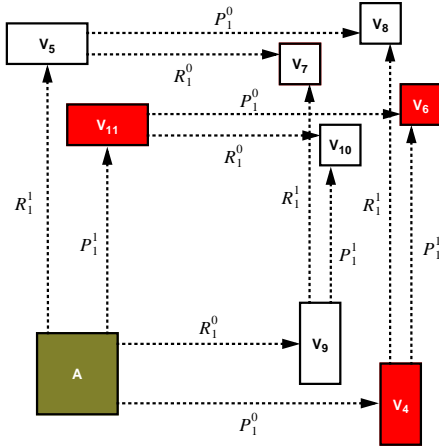


Figure 2: Example view element hierarchy with residual and intermediate view elements.

Figure 3 illustrates the view element hierarchy along one dimension of the data cube. The view element hierarchy is a super-structure of the view hierarchy in Figure 1. In the view hierarchy, the aggregated view V_1 is generated by applying S^0 to A . In the view element hierarchy, V_1 is generated by applying P_1^0 to V_4 . V_4 is generated by applying P_1^0 to A . The view element hierarchy in Figure 3 also illustrates the two-way dependencies among the view elements. For example, V_4 is capable of being synthesized from V_1 and

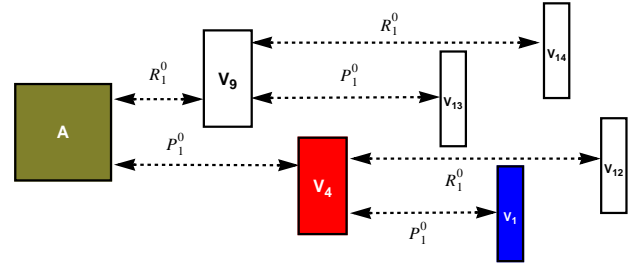


Figure 3: Example view element hierarchy with aggregated views and intermediate and residual view elements.

V_{12} due to the perfect reconstruction property of the first partial aggregation pair (P_1^m, R_1^m) .

Definition 5 View element set – A view element set is a unique set of aggregated views, partial-, or residual-view elements.

We investigate how the data cube is represented using view element sets. We first define two properties of view element sets: *completeness* and *non-redundancy*. These properties are important in representing the views in the data cubes using the view element sets.

Definition 6 Completeness – A view element set is complete with respect to another view element V if the members of the view element set can be used to perfectly reconstruct V .

By this definition, the partial aggregations of A generated by P_1^m and R_1^m are complete with respect to A , since, by Property 1, A can be reconstructed from $P_1^m(A)$ and $R_1^m(A)$. In general, we can test the completeness of a set of view elements using the recursive and perfect reconstruction properties of the first partial aggregation pair. We present a method that uses the view element graph in Section 4.2.

Definition 7 Non-redundancy – A view element set is non-redundant only if there does not exist a partial, residual, or total aggregation of two of the view elements that generates like view elements.

The non-redundancy ensures that there is no forward or reverse dependencies among the view elements in the view element set. We now have the following definition for the view element sets that form a basis for the data cube.

Definition 8 View element basis – A view element set that is complete with respect to data cube A is a basis of A .

In general, the definition of a basis applies to the view elements also, that is, a set of view elements may form the basis for representing another view element. Of particular interest are the view element sets that form a non-redundant bases of the data cube.

Definition 9 Non-redundant view element basis – A set of view elements that is non-redundant and complete (a basis) with respect to data cube A is a non-redundant basis of A .

The notion of a basis is important for selecting view elements that are sufficient for representing the data cube. A non-redundant basis is able to represent the data cube without expanding the volume of data. This is important if storage space is minimal. In many cases, a minimum of redundancy can be allowed to provide gain in processing performance, as we demonstrate in Section 7.2.2.

4 View element graph

We now develop the view element graph. The view element graph manages the view elements and provides a data structure for evaluating the completeness, non-redundancy and benefits of various 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 is depicted in Figure 4.

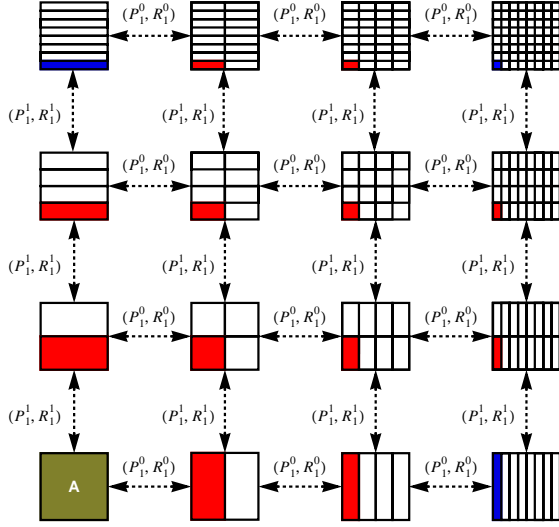


Figure 4: The view element graph organizes the view element into a two-way dependency graph (dark = aggregated views, light = intermediate view elements, white = residual view elements).

4.1 View element graph size

We examine the size and complexity of the view element graph. Consider a d -dim data cube A , where each dimension i_m has size n_m . The view element graph has a depth $D_m = \log_2 n_m$ on each dimension i_m of the data cube A . The depth D_m corresponds to the number partial aggregations P_1^m needed in cascade to totally aggregate dimension i_m of A .

Size The entire view element graph has N_{ve} view elements, where

$$N_{ve} = \prod_{m=0}^{d-1} (2n_m - 1). \quad (17)$$

Of these view elements, N_{av} are aggregated views, which are of direct interest in OLAP applications, where

$$N_{av} = 2^d. \quad (18)$$

N_{iv} of the view elements are intermediate view elements, which can be used for range-aggregated queries, where

$$N_{iv} = \prod_{m=0}^{d-1} (\log_2 n_m + 1). \quad (19)$$

The remaining N_{rv} view elements are residual view elements

$$N_{rv} = N_{ve} - N_{iv}. \quad (20)$$

Table 1 summarizes some example view element graph sizes for various values of d (dimensionality) and n (domain size). In general, we are not interested in generating all of the view elements in the view element graph. However, we enumerate them in order to indicate the complexity involved with the view element method.

	$d = 2$ $n = 256$	$d = 3$ $n = 32$	$d = 4$ $n = 16$	$d = 5$ $n = 8$	$d = 8$ $n = 4$
N_{av}	4	8	16	32	256
N_{iv}	81	216	625	1024	6,561
N_{rv}	261,040	249,831	922,896	758,351	5,758,240
N_{ve}	261,121	250,047	923,521	759,375	5,764,801

Table 1: Number of view elements of each type in the view element graphs of various sizes (d = number of dimensions and n = domain size of each dimension).

Complexity The view element graph groups the view elements into blocks according to the stage in the cascade of partial and residual aggregation, as depicted in Figure 4. In general, each block of view elements is generated by partially and residually aggregating a set of parent view elements. We can see from Eq 1 and Eq 2 that there is a total of $Vol(A)$ additions/subtractions involved in generating each block of view elements.

The view element graph has $N_b = \prod_{m=0}^{d-1} (\log_2 n_m + 1)$ blocks, counting the original data cube. The generation of all of the view elements in the view element graph requires $\mathcal{O}((N_b - 1)Vol(A))$ additions/subtractions. The entire view element graph would require $N_b Vol(A)$ in storage space if stored explicitly.

4.2 Completeness and non-redundancy

We can determine completeness and redundancy of a view element set by evaluating its coverage of the d -dim frequency plane. We assign each view element V_m a position X_m and size W_m in the frequency plane. The position and size are determined by the cascade of partial and residual aggregation operators used to generate the view element.

We define the position of the root view element A by $X(A) = [0, 0, \dots, 0]$ and size by $W(A) = [1, 1, \dots, 1]$, since A completely covers the frequency plane. The partial aggregation P_1^m along dimension i_m of A generates the child view element with position and size as follows:

$$\begin{aligned} X(P_1^m(A)) &= [0, 0, \dots, 0, \dots, 0], \\ W(P_1^m(A)) &= [0, 0, \dots, 1/2, \dots, 0]. \end{aligned} \quad (21)$$

The partial aggregation along dimension i_m reduces the width of the view element in dimension i_m of the frequency plane by two. The partial residual aggregation R_1^m along dimension i_m of A generates a child view element with position and size

$$\begin{aligned} X(R_1^m(A)) &= [0, 0, \dots, 1/2, \dots, 0], \\ W(R_1^m(A)) &= [0, 0, \dots, 1/2, \dots, 0]. \end{aligned} \quad (22)$$

We can see that $P_1^m(A)$ and $R_1^m(A)$ correspond to the low-frequency and high-frequency subbands of A , respectively,

along dimension i_m . In general, we obtain the relative position and size of the view elements as follows: consider the input view element V with position $X(V) = [x_0, x_1, \dots, x_{d-1}]$ and size $W(V) = [w_0, w_1, \dots, w_{d-1}]$. Then, the position and size of the first partial and residual aggregations of V are given by:

$$\begin{aligned} X(P_1^m(V)) &= [x_0, x_1, \dots, x_m, \dots, x_{d-1}], \\ W(P_1^m(V)) &= [w_0, w_1, \dots, w_m/2, \dots, w_{d-1}], \\ X(R_1^m(V)) &= [x_0, x_1, \dots, x_m + w_m/2, \dots, x_{d-1}], \\ W(R_1^m(V)) &= [w_0, w_1, \dots, w_m/2, \dots, w_{d-1}]. \end{aligned} \quad (23)$$

We determine whether the view elements overlap in the frequency plane by evaluating the intersection of the multidimensional rectangles in the frequency-plane defined by their position and size. We define the intersection $V_A \cap V_B$ of view elements V_A and V_B , which gives the volume of their overlap as follows:

$$V_A \cap V_B = \begin{cases} 0 & \exists_m x_m^A \geq x_m^B + w_m^B \text{ or} \\ & \exists_m x_m^B > x_m^A + w_m^A \\ I(V_A, V_B) & \text{otherwise} \end{cases} \quad (24)$$

where

$$I(V_A, V_B) = \prod_{m=0}^{d-1} (\min(x_m^A + w_m^A, x_m^B + w_m^B) - \max(x_m^A, x_m^B)) \quad (25)$$

We can now determine non-redundancy and completeness as follows:

- **Non-redundancy** – a view element set is non-redundant if and only if the view elements in the set do not overlap in the frequency plane: $\forall_{A \neq B} V_A \cap V_B = 0$.
- **Completeness** – a view element set is complete if and only if the view elements in the set completely cover the frequency plane.

We can evaluate the completeness of a view element set using the view element graph and the following recursive procedure:

Procedure 1 View element set completeness – a view element set is complete with respect to A if and only if:

1. A is in the view element set or,
2. the view element set is complete with respect to the partial- and residual-aggregation children on at least one dimension i_m of A .

Any view element set that completely and non-redundantly covers the frequency plane is a non-redundant view element basis of the data cube.

4.3 View element bases

There are many view element bases that have corollaries in signal processing. Since the view element graph is a form of multi-dimensional filter bank, we characterize some of the view element bases in signal processing terms.

Wavelet basis The wavelet basis ([12]) is a non-redundant basis generated by decomposing the intermediate view elements on all dimensions, jointly, starting from the root data cube. The view element wavelet basis set is depicted in Figure 5(a). The final total aggregation and all of the generated residual view elements form the wavelet basis. The wavelet basis has a volume of $Vol = n^d$.

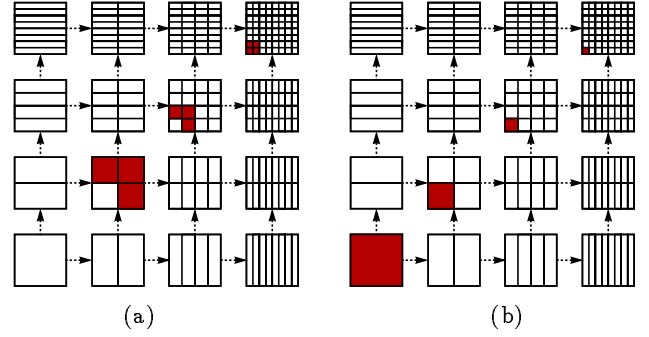


Figure 5: Example view element sets: (a) wavelet basis, and (b) redundant Gaussian pyramid.

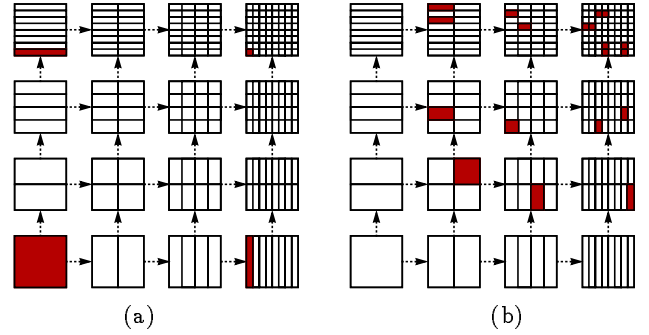


Figure 6: Example view element sets: (a) redundant view hierarchy, and (b) non-redundant wavelet packet basis.

Gaussian pyramid The Gaussian pyramid ([3]) is generated by partially aggregating the intermediate view elements on all dimensions, jointly, starting from the root data cube. The total aggregation view element and all of the generated intermediate view elements form the Gaussian pyramid. The view element Gaussian pyramid set is depicted in Figure 5(b). The Gaussian pyramid, which is a redundant basis, has a volume of $Vol = \sum_{m=0}^{d-1} \frac{1}{d^m}$.

View hierarchy The view hierarchy ([8]), is generated by totally aggregating the data cube along all combinations of its dimensions. The view element set that corresponds to the view hierarchy is depicted in Figure 6(a). The view hierarchy forms a redundant basis and has a volume of $Vol = (n+1)^d$.

Wavelet packets In general, the approach of representing the data cubes adaptively by non-redundant sets of view elements is related to the wavelet packet method in signal processing [5]. A wavelet packet basis corresponds to any non-redundant basis (any complete and non-redundant set of view elements). An example of a view element wavelet packet basis set is depicted in Figure 6(b).

Wavelet packets have great capacity for compressing potentially sparse data cubes. Although we do not explore it here, by selecting the bases that best isolate the non-zero data from the zero areas of the data cube, the view element wavelet packet basis can represent the data cube in a compact form. Since the wavelet packet bases are non-redundant and complete, they have a volume of $Vol = n^d$.

5 View element selection

In an OLAP application, we would like to select a set of view elements for representing the data cube. We assume that the database administrator anticipates the relative frequency in which various views of the data cube are accessed. Alternatively, the frequencies of access can be observed online, allowing the system to dynamically reconfigure. We define two algorithms for selecting a view element set for representing the data cube according to the frequencies of view access.

In the first algorithm, we consider that the selected view element set needs to form a non-redundant and complete basis of the data cube. Given this constraint, the algorithm selects the complete and non-redundant set of view elements that minimizes the processing cost for generating the population of queries. The second algorithm relaxes the non-redundancy constraint and focuses on selecting a set of view elements that minimizes processing cost for a target storage cost.

We point out that since the view dependency hierarchy is embedded within the view element graph, the use of view elements does not preclude the selection of a redundant set of views. In fact, when such configurations represent the best alternatives, they will be selected. In this way, the view element approach represents an extension of the methods that pre-materialize a set of views.

5.1 View element basis selection

We first describe a fast procedure for extracting a non-redundant basis from the view element graph [11]. Then, in Algorithm 1, we present an algorithm for selecting the non-redundant view element basis that minimizes the processing costs for accessing views. In general, we are guaranteed to select a non-redundant view element basis by following the following procedure:

Procedure 2 Basis extraction – a non-redundant basis is extracted from the view element graph as follows: start from the root view element (A):

1. Choose one of the dimensions i_{m^*} , or choose nothing.
2. If choose nothing then terminate. Otherwise, repeat steps 1 & 2 for the partial and residual aggregation children on the chosen dimension i_{m^*} .
3. Follow the chosen paths from the root view element, and mark all of the encountered terminal view elements.
4. The marked elements form a non-redundant view element basis.

The extraction of a non-redundant basis requires a single $d + 1$ -way choice for each view element. Considering that there are N_{ve} view elements (Eq 17), the procedure requires only $\mathcal{O}((d + 1)N_{ve})$ comparisons in order to extract a non-redundant basis. We now apply this procedure to the extraction of the least-cost basis.

5.2 Minimum processing cost

The fast procedure for basis extraction can be used to select the view element basis that has the minimum processing cost for generating a population of queries. Let $\{Z_k\}$ define a population of K views, or, in general, view elements. Let

f_k denote the relative frequency of access of Z_k such that $\sum_{k=0}^{K-1} f_k = 1$. For each view element V_n in the view element graph, and each Z_k , we compute the processing cost $C_{n,k}$ for V_n to support Z_k .

In general, the views Z_k are not direct descendents of all view elements. However, in order for a view element V_a to construct an intersecting view element V_b , it must, in general, be aggregated and synthesized with other view elements. We can express this cost as the sum of the costs for V_a and V_b to generate their largest common descendent, that is:

$$C_{a,b} = D_{a,l} + D_{b,l}, \quad (26)$$

where l is chosen such that $Vol(V_l) = Vol(I(V_a, V_b))$, and $D_{a,l}$ and $D_{b,l}$ give the costs to aggregate view elements V_a and V_b , respectively, to generate V_l . We base these costs on the number of additions/subtractions required in the partial aggregation cascades (see Section 4.1).

We can compute $C_{a,b}$ directly from the volumes of the involved view elements if they intersect. Let $I(V_a, V_b)$ give the intersection of V_a and V_b from Eq 25. Then,

$$C_{a,b} = \begin{cases} F_{a,l} + F_{b,l} & V_a \cap V_b \\ 0 & \text{otherwise.} \end{cases} \quad (27)$$

where, in general, $F_{a,l}$ is computed from

$$F_{a,l} = \sum_{j=\log_2(I(V_a, V_b))}^{\log_2(Vol(V_a))-1} 2^j. \quad (28)$$

Then, we select the non-redundant basis from the view element graph that minimizes the total processing cost using the following recursive algorithm:

Algorithm 1 Minimum cost non-redundant basis selection – A complete and non-redundant basis of minimum cost is selected from the view element graph as follows:

1. Assign a support cost $C_n(V_n)$ to each view element V_n in the view element graph, where

$$C_n(V_n) = \sum_{k=0}^{K-1} f_k C_{n,k}. \quad (29)$$

2. Let $T_m(V_n)$ give the minimum cost of V_n 's children on dimension i_m :

$$T_m(V_n) = \mathcal{D}(P_1^m(V_n)) + \mathcal{D}(R_1^m(V_n)), \quad (30)$$

where $\mathcal{D}(V_n)$ is defined for each view element V_n as follows:

3. Let $m^* = \operatorname{argmin}_m T_m(V_n)$ select the minimum cost dimension i_{m^*}
4. If $C_n(V_n) \leq T_{m^*}(V_n)$ then terminate, otherwise choose dimension i_{m^*} . Compute $\mathcal{D}(V_n)$ as follows:

$$\mathcal{D}(V_n) = \begin{cases} C_n(V_n) & C_n(V_n) \leq T_{m^*}(V_n) \\ T_{m^*}(V_n) & \text{otherwise.} \end{cases} \quad (31)$$

5. Use Procedure 2 to extract the selected basis.

Algorithm 1 selects the optimal non-redundant view element basis using the fast Procedure 2 for the basis extraction. An experiment that demonstrates that Algorithm 1 selects representations of the data cube that minimize the processing cost is given later in Section 7.2.1.

5.3 Minimum processing and storage cost

In some cases, it is desirable to select a redundant basis if the processing costs can be reduced significantly. Unfortunately, the fast algorithm does not apply in this case. In [8], a sub-optimal greedy algorithm was proposed for selecting redundant views. We follow a similar approach for selecting redundant view elements.

The greedy algorithm selects the redundant view element basis that minimizes the processing cost for a target storage cost. The algorithm utilizes the following procedure for computing the total processing cost of a redundant view element set in supporting a population of queries:

Procedure 3 Total processing cost – *the total processing cost for a redundant view element set to support a population of queries is computed as follows:*

1. Aggregation costs:

- (a) For each selected view element V_s , let $F_{s,l}$ give the aggregation cost to generate each dependent view element V_l (Eq 28).
- (b) Let $\mathcal{F}_l = \min_s F_{s,l}$ give the minimum cost to compute view element V_l by aggregating some view element.

2. Synthesis costs:

- (a) For each non-selected view element V_n , let \mathcal{R}_n give the minimum cost to compute V_n by synthesis:

$$\mathcal{R}_n = \text{Vol}(V_n) \min_m (T_p^m + T_r^m), \quad (32)$$

where T_p^m and T_r^m are the minimum costs of the partial and residual children, respectively, on dimension i_m , and for each child $j \in \{p, r\}$ on dimension i_m , $T_j^m = \min(\mathcal{F}_j^m, \mathcal{R}_j^m)$.

3. Let T_j give the least cost option (aggregation or synthesis) of generating each view element V_j :

$$T_j = \min(\mathcal{F}_j, \mathcal{R}_j). \quad (33)$$

4. Then, the total cost to generate a population of views $\{Z_k\}$ is given by

$$T = \sum_k f_k T_k, \quad (34)$$

where f_k is the frequency of access of view V_k .

The following greedy algorithm adds view elements (or views) in successive stages. At each stage, the view element that results in the largest reduction of the total processing cost T is added.

Algorithm 2 Minimum cost redundant basis selection – *The complete, and possibly redundant basis that minimizes processing cost for a target storage cost S_T is selected from the view element graph as follows:*

1. Select the minimum cost non-redundant view element basis $\{V_s\}$ using Algorithm 1.
2. For each non-selected view element (or view) V_n , if $\text{Vol}(\{V_s\}) + \text{Vol}(V_n) \leq S_T$ then
 - (a) Select the view element.

- (b) Compute the new total processing cost T^n using Procedure 3.
- (c) De-select V_n .

3. Let $n^* = \text{argmin}_n T^n$ choose the view element that results in the largest reduction of the processing cost.
4. Add V_{n^*} to the selected view element set, and repeat steps 2, 3 & 4 while $\text{Vol}(\{V_s\}) < S_T$.

An experiment that demonstrates the performance of Algorithm 2 in selecting the redundant representations of the data cube that have the minimum processing cost for a target storage cost is given later in Section 7.2.2.

6 Range-aggregation

Many queries of interest in OLAP involve the aggregation over contiguous attribute ranges. For example, one such view computes the total sales of a particular product to a particular customer between a range of dates. In general, a range refers to an embedded sub-cube within the data cube. The range is defined by its position $X = [x_n]$ and size $W = [w_n]$ as follows:

$$G(A) = A[x_0 : w_0, x_1 : w_1, \dots, x_{d-1} : w_{d-1}]. \quad (35)$$

The range-aggregation computes an aggregation S over the range $G(A)$ of the data cube A , by

$$S(G(A)) = \sum_{i_0=l_0}^{l_0+w_0-1} \sum_{i_1=l_1}^{l_1+w_1-1} \dots \sum_{i_{d-1}=l_{d-1}}^{l_{d-1}+w_{d-1}-1} A[i_0, i_1, \dots, i_{d-1}]. \quad (36)$$

The range-aggregations are related to the intermediate views due to the commutativity between range extraction G and partial aggregation P_1 under conditions on the range values. Consider the partial aggregation P_1^m applied after range extraction G^m along dimension i_m of A

$$P_1^m(G^m(A)) = \sum_{l=0}^1 A[i_0, \dots, x_m + 2i_m + l, \dots, i_{d-1}]. \quad (37)$$

We define G_2^m by sub-sampling G^m by two such that $X_2 = X/2$ and $W_2 = W/2$. If P_1^m and G_2^m are cascaded, then we have

$$\begin{aligned} G_2^m(P_1^m(A)) &= G_2^m\left(\sum_{l=0}^1 A[i_0, \dots, 2i_m + l, \dots, i_{d-1}]\right) \\ &= \sum_{l=0}^1 A[i_0, \dots, x_m + 2i_m + l, \dots, i_{d-1}], \end{aligned} \quad (38)$$

which gives the desired commutativity as follows:

$$P_1^m(G^m(A)) = G_2^m(P_1^m(A)). \quad (39)$$

As long as the range in G falls along powers of two in position and size, then the range can be extracted by G_2 from the first partially aggregated intermediate view. In general, we can extract the range-aggregation from the k -th partial aggregation intermediate view if the range falls along powers of 2^k as follows:

$$P_k^m(G^m(A)) = G_{2^k}^m(P_k^m(A)). \quad (40)$$

Otherwise, the range-aggregations can be generated by aggregating the extracted ranges from the data cube as usual.

7 Evaluation

We examine the benefits of the view element graph first through a pedagogical example, and then through experimental analysis.

7.1 Pedagogical view element example

We construct a simple example to illustrate the advantages of the view element method. In this example, we consider that two views, V_1 and V_7 , are equally likely to be retrieved. The remaining views are not. That is, $f_1 = f_7 = 0.5$, and $\forall_{n \neq 1,7} f_n = 0$. The processing costs for each of the transitions in the view element graph are indicated in Figure 7 by (i, j) , where i is the aggregation cost and j is the synthesis cost. Table 2 gives the storage and processing costs of various sets of view elements.

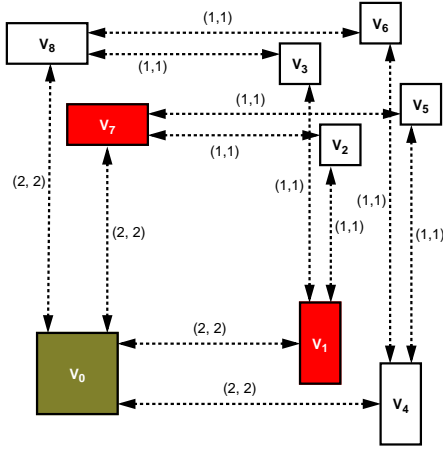


Figure 7: Example view element graph in which two views are accessed with equal frequency: $f_1 = f_7 = 0.5$.

View element set	Basis	Redundant	Processing Cost	Storage Cost
$\{V_3, V_6, V_7\}$	Yes	No	3	4
$\{V_1, V_5, V_6\}$	Yes	No	3	4
$\{V_0\}$	Yes	No	4	4
$\{V_1, V_4\}$	Yes	No	4	4
$\{V_7, V_8\}$	Yes	No	4	4
$\{V_2, V_3, V_5, V_6\}$	Yes	No	4	4
$\{V_0, V_1, V_7\}$	Yes	Yes	0	8
$\{V_1, V_7\}$	No	Yes	0	4
$\{V_3, V_7\}$	No	No	3	3
$\{V_2, V_3, V_5\}$	No	No	4	3

Table 2: The processing and storage costs of various view element sets.

The first six view element sets listed in Table 2 are non-redundant bases for data cube V_0 . Two of these view element sets, $\{V_3, V_6, V_7\}$, and $\{V_1, V_5, V_6\}$, have the minimum total processing cost = 3. For example, the processing cost of $\{V_1, V_5, V_6\}$ is computed from $(V_1 \xrightarrow{0} V_1) + (V_5 \xrightarrow{1} V_7)$, $(V_1 \xrightarrow{1} V_2)$, $(V_2 \xrightarrow{1} V_7)$. The remaining non-redundant and complete view element sets are sub-optimal.

On the other hand, consider that only views can be materialized. By materializing that root data cube V_0 , the populations of views are generated with a higher processing cost = 4. Furthermore, the redundant set of views $\{V_0, V_1, V_7\}$

has zero processing cost but increases the storage cost to 8. The incomplete set of views $\{V_1, V_7\}$ has zero processing cost but is not capable of constructing all views in the data cube. In summary, this example illustrates that without using view elements, the processing cost is reduced only by increasing the storage cost, or by sacrificing the completeness of the set of views.

7.2 Experiments

We evaluate the performance of the view element method in two experiments. In both experiments, we have a d -dim data cube with a size of n on each dimension (square). We assign a random probability of access to each of the aggregated views in the data cube. We denote the k -th aggregated view by Z_k , and the probability of access by f_k .

7.2.1 Experiment 1 – non-redundant bases

In the first experiment, we compare three strategies of selecting non-redundant view element bases: ([D]) store data cube A , ([W]) store wavelet view element basis, and ([V]) select the best non-redundant view element basis using Algorithm 1.

The experiment uses a 4-dimensional data cube with a domain size of 16 on each dimension. The view element graph for the 4-D data cube consists of 923,521 view elements, of which 16 are aggregated views. We conducted 100 trials in which the view access frequencies were chosen at random as described above.

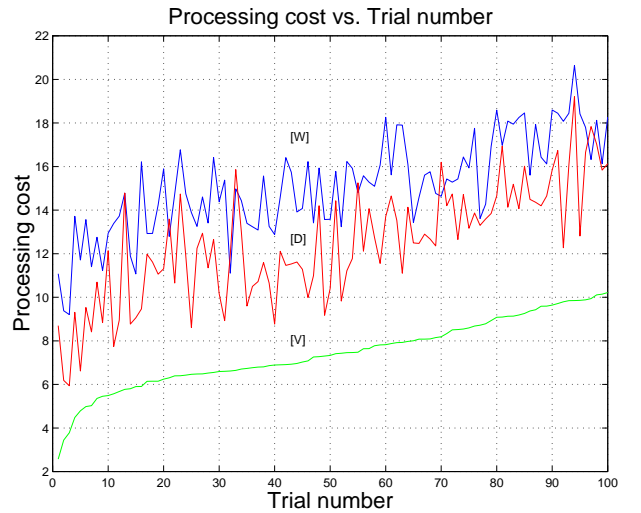


Figure 8: The processing costs to support randomly selected distributions of aggregated view queries: [D] data cube only, [W] wavelet basis, [V] best view element basis.

The resulting processing costs are given in Figure 8. We can see that Algorithm 1 (plot [V]) selects view element sets that greatly reduce the processing cost for supporting the queries. On average, the processing costs are 53.8% of the cost of the data cube (plot [D]). The wavelet basis (plot ([W])) performs worse than both methods.

In general, the view element method is guaranteed have a lower processing costs than these methods since the view element graph is a superset of the data cube, the wavelet basis, and other non-redundant, complete representations of the data cube.

7.2.2 Experiment 2 – storage and processing costs

In the second experiment we investigate two approaches for materializing redundant sets views and view elements. In the first approach ([D]), we start by materializing the data cube, then add views in a greedy fashion. In the second approach ([V]), we first select the minimum cost non-redundant view element basis using Algorithm 1, then add view elements in a greedy fashion using Algorithm 2.

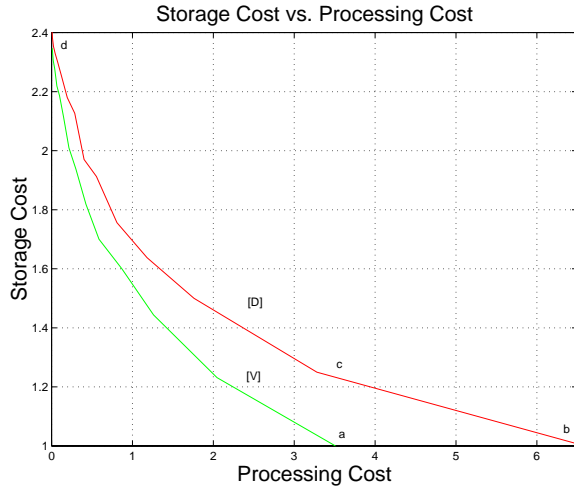


Figure 9: Average storage cost vs. processing cost over ten trials: [D] greedy view method, [V] greedy view element method.

The average results for the storage and processing costs over ten trials are given in Figure 9. The experiment used a $d = 4$ -dim data cube with a domain size of $n = 4$ on each dimension. The storage costs were computed from relative volume of the selected view elements (or views) compared to the volume of the data cube. We point out that the maximal storage cost, which is given by materializing all views, is given by $(n + 1)^d / n^d = 2.44$.

Figure 9 demonstrates the reduction in processing and storage costs achieved by the view element method (plot [V]). Initially, the view element method selects the best non-redundant view element basis (point a). In order to provide an equivalent processing cost, the view selection method (plot [D]) requires an increase in storage cost by approximately 1.25 (point c). Furthermore, as the storage budget is increased, the view element method consistently represents the data cube in a form that has lower processing cost than the best set of redundant views.

We can guarantee that the view element method selects a representation that has lower processing cost for any target storage cost as follows: at each stage in Algorithm 2, add the best view, and remove the obsolete view elements. The initial non-redundant view element method solution (point a) is never worse than that provided by the data cube (point b). Furthermore, both methods converge to the same zero processing cost solution (point d). By greedily adding views in both methods, the view element sets have a lower processing cost for any target storage cost.

8 Conclusion

We presented a new method for representing data cubes using aggregated, intermediate and residual view elements.

We described a view element graph data structure for managing the generation, selection and evaluation of the view elements. We presented and evaluated the performance of a fast and optimal algorithm for selecting non-expansive and complete sets of view elements for representing the data cubes. We also presented a greedy algorithm for selecting redundant view elements in order to further reduce processing costs. We demonstrated that the greedy view element approach performs better than the greedy approaches of adding views to the data cube in terms of storage and processing costs.

References

- [1] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *13th Int'l Conf. on Data Engineering*, April 1997.
- [2] S. Agrawal, R. Agrawal, P. M. Deshpande, A. Gupta, J. E. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. Conf. on Very Large Databases (VLDB)*, pages 506–521, 1997.
- [3] P. J. Burt and E. Adelson. The Laplacian Pyramid as a compact image code. *IEEE Trans. Commun.*, COM-31(4), April 1983.
- [4] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (On-line Analytical Processing) to user-analysts: An IT Mandate. Technical report, E. F. Codd & Associates, 1993.
- [5] R. R. Coifman and M. V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Trans. Inform. Theory*, 38(2), March 1992.
- [6] 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.
- [7] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Proc. of the 13th Int'l Conf. on Data Engineering*, 1997.
- [8] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing Data Cubes efficiently. In *Proc. of the 1996 ACM-SIGMOD Conf.*, 1996.
- [9] 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.
- [10] K. A. Ross and D. Srivastava. Fast computation of sparse datacubes. In *Proc. Conf. on Very Large Databases (VLDB)*, 1997.
- [11] 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.
- [12] M. Vetterli and J. Kovačević. *Wavelets and Subband Coding*. Prentice-Hall, Inc, Englewood Cliffs, NJ, 1995.
- [13] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *ACM Proc. Int. Conf. Manag. Data (SIGMOD)*, pages 159 – 170, 1997.