SHIFT-SPLIT: I/O Efficient Maintenance of Wavelet-Transformed Multidimensional Data*

Mehrdad Jahangiri
University of Southern California
Los Angeles, CA 90089-0781
jahangir@usc.edu

Dimitris Sacharidis†
National Technical University of Athens
Athens, GR 15773
dsachar@dblab.ntua.gr

Cyrus Shahabi
University of Southern California
Los Angeles, CA 90089-0781
shahabi@usc.edu

ABSTRACT

The Discrete Wavelet Transform is a proven tool for a wide range of database applications. However, despite broad acceptance, some of its properties have not been fully explored and thus not exploited, particularly for two common forms of multidimensional decomposition. We introduce two novel operations for wavelet transformed data, termed SHIFT and SPLIT, based on the properties of wavelet trees, which work directly in the wavelet domain. We demonstrate their significance and usefulness by analytically proving six important results in four common data maintenance scenarios, i.e., transformation of massive datasets, appending data, approximation of data streams and partial data reconstruction, leading to significant I/O cost reduction in all cases. Furthermore, we show how these operations can be further improved in combination with the optimal coefficient-to-disk-block allocation strategy. Our exhaustive set of empirical experiments with real-world datasets verifies our claims.

1. INTRODUCTION

The Discrete Wavelet Transform is a well established tool, used extensively in signal processing applications for many years since its introduction. Recently, it has proven useful for a number of database applications as well. The wavelet transformation has been used to provide approximate, progressive or even fast exact answers to OLAP range-aggregate queries [2, 3, 7, 9, 12, 13, 15], with its performance rivaling traditional histogram and sampling techniques. In the domain of time-series analysis and mining, wavelets are used to automate feature extraction and expedite pattern discovery and outlier detection [1, 8]. The wavelet transformation is also used to provide compact synopses of data streams [4, 5] in support of approximate query processing.

However, despite its broad acceptance, the wavelet transformation has not been explored to its full potential for data intensive applications. Namely, the compact support and the multi-scale properties of the wavelets, as illustrated by the wavelet tree of decomposition, lead to some overlooked but interesting properties. With the exception of [2], where traditional relational algebra operations are re-defined to work directly in the wavelet domain, most applications resort to reconstruction of many data values to support even the simplest operations in the original domain.

We introduce two novel operations for wavelet decomposed data, named **SHIFT** and **SPLIT**, that stem from the multiresolution properties of wavelets to provide general purpose functionality. They are designed to work directly in the wavelet domain and can be utilized in a wide range of data intensive applications, resulting in significant improvements in every case.

Furthermore, queries on wavelet-transformed data exhibit a particular access pattern. There is a strong dependency among wavelet coefficients, enforced by the multi-scale nesting property, so that we always know which coefficients must be retrieved alongside any coefficient to reconstruct a data point or a range. This observation leads to constructing multidimensional tiles containing wavelet coefficients that are related with each other under a particular access pattern. These tiles are then stored directly into the secondary storage, as their size is adjusted to fit a disk block. By using this tiling approach we can minimize the number of disk I/Os needed to perform any operation in the wavelet domain, including the important reconstruction operation which results in significant query cost reductions. We designed the **SHIFT** and **SPLIT** operations to work with multidimensional tiles, as these operations benefit significantly from their existence.

1.1 Data Maintenance Scenarios

To demonstrate the usefulness of our **SHIFT** and **SPLIT** operations we look into four common data maintenance scenarios, and examine these operations in each context. The scenarios are diverse enough to cover most of the areas where wavelets are used, but not exhaustive, as we conjecture that the applications that can benefit from the **SHIFT** and **SPLIT** operations are plenty. The scenarios examined

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and IIS-0238560 (PECASE), unrestricted cash gifts from Microsoft, an on-going collaboration under NASA's GENESIS-II REASON project and partly funded by the Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (CiSoft); CiSoft is a joint University of Southern California - ChevronTexaco initiative.

†This work was done while the author was a graduate student of Infolab at the University of Southern California.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA
Copyright 2005 ACM 1-59593-060-4/05/06 ...$5.00.

here share the fact that the problem they deal with has a straightforward solution when dealing with untransformed data. Therefore, one is compelled to first reconstruct the original data from the transformed data. However, we are interested in working entirely in the wavelet domain, and as we see, this becomes a complicating, but fruitful, factor. Beside the **SHIFT** and **SPLIT** operations, a major contribution of this paper includes the six analytically proven improvement results in these four applications by utilizing **SHIFT** and **SPLIT**:

- *Transformation of Massive Multidimensional Datasets*: In the simplest scenario, we are faced with the transformation of a multidimensional dataset into the wavelet domain in an I/O efficient manner, where available memory is limited. Our approach is transformation by chunks, small enough to fit in memory. Each transformed chunk is then shifted, to relocate its coefficients, and split, to update some of the already calculated coefficients. We show that our new transformation technique significantly outperforms the current state of the art methods [12, 13] for transforming large multidimensional datasets.

- *Appending to Wavelet Decomposed Transforms*: To illustrate, suppose we have already accumulated and transformed data for 10 years of measurements to expedite query processing. Now, what should we do in the case that new data for one more year arrive? Should we throw the old transformed data and do the transformation from scratch? Certainly, we cannot perform updates; there is nothing to update since the new data involve a part of the data we have not transformed yet. This scenario, seen in the untransformed domain, involves a number of inserts. However, in the transformed domain, each of these inserts require some dimensions to grow, and therefore not only do coefficients have to be updated, but new coefficients need to be created as well. In [2], the authors define the relational algebra operations in the wavelet domain, but do not propose a solution for insert operations. Generally, expanding the transformed data to larger dimension sizes has a high asymptotic cost, even when using our **SHIFT** and **SPLIT** operations; however, since these operations are faster than computing coefficients, our approach results in faster execution times.

- *Data Stream Approximation*: Gilbert et al. [5] demonstrated that a best K-term wavelet approximation of a single dimensional data stream of domain size N in the time series model is possible using space of $O(K + \log N)$, by keeping the $O(\log N)$ coefficients that can change, with per-item cost of $O(\log N)$. We show that the **SHIFT-SPLIT** operations can further reduce the per-item cost to $O\left(\frac{1}{B}\log\frac{N}{B}\right)$ at the expense of additional storage of B coefficients.

  Furthermore, we investigate the case of multidimensional data streams, decomposed under two different forms of wavelet transformation. We conclude that we can maintain a K-term approximation, under certain restrictions. To the best of our knowledge, this is the first work dealing with wavelet approximation of multidimensional data streams, as previous works [8, 1, 4, 5] focused on the single dimensional case.

- *Partial Reconstruction from Wavelet Transforms*: Consider the scenario in which we wish to extract a region of the original data from its wavelet transform. We are faced with the following dilemma: either decompose the entire data and then extract the desired region, which is reasonable if the region extend over large part of the data; or reconstruct point by point the desired region, which is preferable for small regions.

  Chakrabarti et al. [2] propose a solution to deal with relational algebra selection operations in the wavelet domain. Their approach examines the wavelet coefficients to calculate their contribution to the selected range. Our **SHIFT-SPLIT** approach generalizes this notion and therefore can be applied to other forms of wavelet decomposition.

1.2 Outline

We begin our discussion, in Section 2, with presenting an overview of discrete wavelet transform and the notion of wavelet trees. In Section 3, we introduce the disk block allocation strategy which leads to the efficient tiling of wavelet coefficients and then we extend this strategy to the multidimensional case. The **SHIFT** and **SPLIT** operations are presented in full details in Section 4, with their most important applications appearing in Section 5. We present our experimental studies in Section 6 and we conclude our discussion in Section 7.

2. PRELIMINARIES

In this section, we define the preliminary concepts that we use throughout this paper. For a more detailed treatment of wavelet basics please refer to [11].

2.1 Discrete Wavelet Transform

The Discrete Wavelet Transformation (DWT) provides a multi-scale decomposition of data by creating "rough" and "smooth" views of the data at different resolutions. In the case of Haar wavelets that we use throughout this paper, the "smooth" view consists of averages or average coefficients, whereas the "rough" view consists of differences or detail coefficients. At each resolution, termed level of decomposition or scale, the averages and details are constructed by pairwise averaging and differencing of the averages of the previous level.

Let us consider a vector of 4 values $\{3, 5, 7, 5\}$ and let us apply DWT. We start by first taking the pairwise averages: $\{4, 6\}$ and the pairwise differences $\{-1, 1\}$. For any two consecutive and non-overlapping pair of data values a, b we get their average: $\frac{a+b}{2}$ and their difference divided by 2: $\frac{a-b}{2}$. The result is 2 vectors each of half size containing a smoother version of the data, the averages, and a rougher version, the differences; these coefficients form the first level of decomposition. We continue by constructing the average and difference from the smooth version of the data: $\{4, 6\}$. The new average is $\{5\}$ and the difference is $\{-1\}$, forming the second and last level of decomposition. Notice that 5 is the average of the entire vector as it is produced by iteratively averaging pairwise averages. Similarly, -1 represents the difference between the average of the first half of the vector and the average of the second half. The final average and the differences produced at all levels of decomposition form the Haar transform: $\{5, -1, -1, 1\}$. Notice, that at

each level of decomposition the averages and differences can be used to reconstruct the averages of the previous level.

We denote by $u_{j,k}$ and $w_{j,k}$ the k-th average (a.k.a. scaling coefficient) and the k-th detail coefficient (a.k.a. wavelet coefficient), respectively, for the j-th level of decomposition. The averages at level j are decomposed into averages and details of level $j+1$. If we denote the set of scaling coefficients at the j-th level by U_j and the set of wavelet coefficients at the j-th level by W_j, we can formally write the previous statement as $U_j = U_{j+1} \oplus W_{j+1}$, where the direct-sum $\oplus$ notation refers to the decomposition process. The original data are the scaling coefficients of the 0-th level. See also Appendix A.

For example, the 3 level decomposition, shown in Figure 1, is formally written as:

$$\begin{aligned} U_0 &= U_1 \oplus W_1 \\ &= U_2 \oplus W_2 \oplus W_1 \\ &= U_3 \oplus W_3 \oplus W_2 \oplus W_1 \end{aligned}$$

Figure 1 also shows that for each level of decomposition j, there are 2^{n-j} wavelet coefficients $w_{j,k}$ and 2^{n-j} scaling coefficients, for $0 \le k \le 2^{n-j}-1$. The transformed vector $\widehat{\mathbf{a}}$ consists of the average, $u_{n,0}$, as its first element, followed by the details $w_{j,k}$ sorted decreasing by level j and increasing by position k: $u_{n,0}$, $w_{n,0}$, $w_{n-1,0}$, $w_{n-1,1}$, $w_{n-2,0}$, ..., $w_{1,0}$, ..., $w_{1,2^{n-1}-1}$. We denote that vector $\widehat{\mathbf{a}}$ is the Discrete Wavelet Transform of $\mathbf{a}$ by $\widehat{\mathbf{a}} = \texttt{DWT}(\mathbf{a})$. Note that the untransformed vector $\mathbf{a}$ contains the scaling coefficients at the 0-th level of decomposition: $\mathbf{a}[k] = u_{0,k}$, for $0 \le k \le N$.

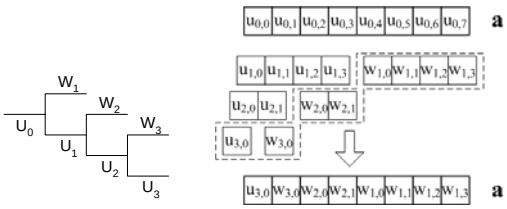


Figure 1: Haar Wavelet Decomposition

We now define the term "support interval" that we will use frequently in this paper.

Definition 1. The *support interval* of a (wavelet or scaling) coefficient is the part of the original data from which this coefficient is computed.

Figure 2 shows the support intervals of Haar wavelets for a vector of size 8.

Definition 2. A (wavelet or scaling) coefficient *covers* another (wavelet or scaling) coefficient if the support interval of the latter is (completely) contained in the support interval of the former.

For example, the first coefficient in the second level of decomposition $w_{2,0}$ covers the first and second coefficients of the first level of decomposition, $w_{1,0}$ and $w_{1,1}$; see Figure 2.

Definition 3. An interval I is a *dyadic interval* if $I = \left[k2^j, (k+1)2^j - 1\right]$, for $0 \le j \le n$ and $0 \le k \le 2^{n-j}-1$.

Haar wavelet coefficients $w_{j,k}$ and Haar scaling coefficients $u_{j,k}$ have the property that their support intervals are dyadic intervals.

Property 1. The support interval of a Haar wavelet coefficient $w_{j,k}$ (or scaling coefficient $u_{j,k}$) is the $I_{j,k}$ dyadic interval $\left[k2^j, (k+1)2^j - 1\right]$.

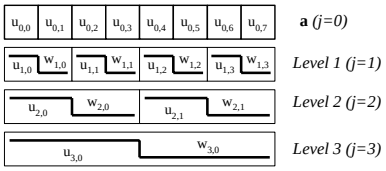


Figure 2: Support Intervals of Haar Wavelets

Multidimensional Wavelet Transformation There are two ways to perform a multidimensional wavelet decomposition, the *standard* and the *non-standard*. In short, the standard form is performed by applying a series of one-dimensional decompositions along each dimension whereas the non-standard form does not decompose each dimension separately. In the non-standard form, after each level of decomposition only the averages corresponding to the same level are further decomposed. We refer the reader to Appendix B for further discussion.

2.2 Wavelet Tree

In this section, we review the notion of wavelet tree. Our purpose is two-fold. This tree exploits the relationships between coefficients, and thus identifies the access patterns which lead to the block allocation strategy described in Section 3. Furthermore, the **SHIFT** and **SPLIT** operations that we define in Section 4 are easily understood in the context of wavelet trees, as they essentially are operations on trees.

The multiresolution property of the Haar wavelets induces a tree construct capturing and illustrating this property. A wavelet coefficient w is the parent of another coefficient w', when w is the coefficient with the smallest support that covers w'. For Haar wavelets, which is our case, this tree is a binary tree where each node $w_{j,k}$ has exactly two children, $w_{j-1,2k}$ and $w_{j-1,2k+1}$. The scaling coefficient $u_{n,0}$ is the root of the tree having only one child $w_{n,0}$. This tree structure has been given several names in the wavelet bibliography, such as error tree [13, 12], dependency graph [10], etc. Figure 3 shows this tree for a vector of size 8; scaling coefficients are shown with squares, whereas wavelet coefficients are shown in circles. Figure 3 also shows the original data as children of the leaf nodes of the tree, drawn with dotted line.

The beautiful property of this tree is that it portrays the way Haar wavelets partition the time-frequency plane; see Figure 3. As j decreases we gain accuracy in the time domain, but simultaneously, we lose accuracy in the frequency domain and vice versa.

The following lemma is a consequence of the time-frequency trade-off. A single point in time domain depends on those wavelet coefficients in the path to the root. As a result, a data value can be reconstructed in time proportional to the tree height and thus in time logarithmic to the vector size.

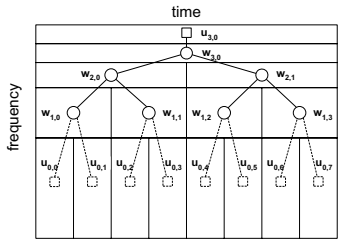


Figure 3: Haar Wavelet Tree

Lemma 1. Let $\widehat{\mathbf{a}}$ be the wavelet transform of vector $\mathbf{a}$ of size $N = 2^n$, $\widehat{\mathbf{a}} = \texttt{DWT}(\mathbf{a})$. Any value of $\mathbf{a}$ can be reconstructed using exactly $n + 1 = \log N + 1$ coefficients from $\widehat{\mathbf{a}}$.

PROOF. Let $\mathbf{a}[i]$ be the $(i + 1)$-th value of $\mathbf{a}$. At each level of decomposition j, there is exactly one wavelet coefficient $w_{j,\lfloor \frac{i}{2^j} \rfloor}$ that covers $\mathbf{a}[i]$, because of the fact that Haar wavelets of the same level have non-overlapping support. This together with the parent-child relationship existent in the wavelet tree results in the covering wavelet coefficients $w_{j,\lfloor \frac{i}{2^j} \rfloor}, \forall j \in [1, n]$ and the scaling coefficient $u_{n,0}$ belonging to a $(n + 1)$-long path in the tree. □

Therefore, as Lemma 1 suggests a point query can be answered using $O(\log N)$ coefficients. However, the wavelet transformation is mainly used for its ability to answer range-sum queries also using $O(\log N)$ coefficients, as the following lemma suggests.

Lemma 2. Let $\widehat{\mathbf{a}}$ be the wavelet transform of vector $\mathbf{a}$ of size $N = 2^n$, $\widehat{\mathbf{a}} = \texttt{DWT}(\mathbf{a})$. A range-sum query $\sum_{i=l}^{r-1} \mathbf{a}[i]$ can be answered using not more than $2n + 1 = 2 \log N + 1$ coefficients from $\widehat{\mathbf{a}}$.

PROOF. This lemma holds because of the fact that Haar wavelets have a 0-th vanishing moment. For more details refer to [9]. □

3. DISK BLOCK ALLOCATION OF WAVELET COEFFICIENTS

The purpose of this section is to assign wavelet coefficients to disk blocks in such a way that the number of blocks required for answering queries is minimized. We have already seen that the wavelet tree captures the dependency among coefficients. In particular, if a coefficient is required to be retrieved then all coefficients on the path to the root must also be retrieved. This property creates an access pattern of wavelets that must be exploited by the disk block allocation strategy.

Intuitively, a disk block should contain coefficients with overlapping support intervals, so that the utilization of the in-block coefficients is high. However, we must take under consideration the fact that the disk block allocation strategy should not allow redundancy, in that a wavelet coefficient should belong to one block only. Under this restriction, in order to be fair across all coefficients, we partition the wavelet tree into binary subtree tiles and store each tile on a disk block. Assuming that the disk block size B is a power of 2, $B = 2^b$, we achieve logarithmic utilization of the blocks. At least b coefficients inside the block, lying in a path, are to be utilized any time this disk block is needed. Logarithmic utilization may seem low at first, but it is the best we can hope for under our restrictions, as proven in [10].

One final issue is that the size of the binary subtree tiles is $2^b - 1$, whereas the block size is 2^b. We are wasting space of 1 coefficient in our block allocation strategy. Therefore, we choose to store the scaling coefficient corresponding to the root of the subtree, along with the wavelet coefficients of the tile. The extra scaling coefficients that we store are useful for query answering, as they can dramatically reduce query costs. An example of the disk block allocation strategy for a wavelet tree of 32 coefficients is shown in Figure 4.

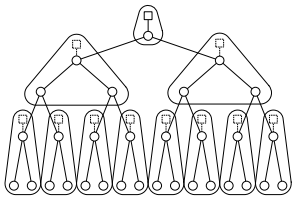


Figure 4: Disk Block Allocation Strategy

We continue our discussion by generalizing the disk block allocation schema for both standard and non-standard multidimensional wavelet transformation in Section 3.2; but first, we need to extend the wavelet tree notion to the multidimensional case.

3.1 Multidimensional Wavelet Trees

As mentioned before, there are two forms of multidimensional wavelet decomposition, the standard and non-standard. The non-standard form of decomposition involves fewer operation and thus is faster to compute but does not compress as efficiently as the standard form. Particularly, range aggregate queries can be highly compressed using the standard form as shown in [9]. In the database literature both transformation forms have been used: standard by [13, 12, 9, 7] and non-standard by [15, 2]. However, we are not aware of any study on the extension of the wavelet tree concept for either form of the multidimensional transformations. More details on the multidimensional transformation forms can be found in Appendix B.

In the standard multidimensional transformation each dimension is decomposed independently. Therefore, there cannot be a single tree capturing the levels of decomposition. In case of 2-d, considering a 1-d wavelet tree for each of the decomposed dimensions, two 1-d wavelet trees are required. Every coefficient in a transformed 2-d array has two indices, one for each dimension. Each of these indices identifies a position in the 1-d tree, which as we have seen corresponds to a decomposition level and to a translation inside that level. Figure 5 shows a coefficient in an 8×8 2-d array and the corresponding indices on the two wavelet trees.

The two 1-d trees can be used to determine which coefficients need to be retrieved for reconstructing data values on the 2-d array. Subsequently, they provide information about the access pattern of 2-d wavelets. A single data value on the untransformed (original) 2-d array corresponds to a path in each of the 1-d wavelet trees, or better, a set of 1-d indices, as mentioned before. The cross product among all indices

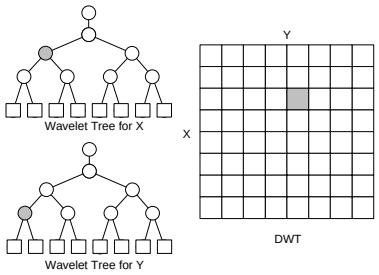


Figure 5: Standard Form Wavelet Trees

across these sets, construct the 2-d indices whose coefficients must be retrieved. For a $N \times N$ array, where $N = 2^n$, each of the paths contains $(n+1)$ 1-d indices, therefore there are $(n+1)^2$ 2-d indices. Figure 6 shows the two paths on the 1-d wavelet trees, as well as the required coefficient resulting from the cross product between 1-d indices.

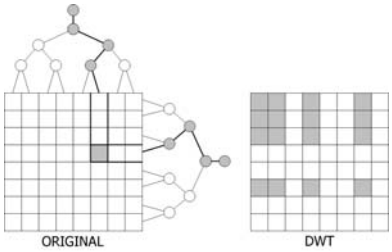


Figure 6: Standard Form Data Point Reconstruction

In contrast to the standard multidimensional transformation, a single wavelet tree can capture the levels of decomposition and dependency among coefficients for the non-standard transformation. The support intervals of the wavelet coefficients form a quad-tree, as each support interval is further decomposed in quadrants at the next level of decomposition. At the j-th level of d-dimensional decomposition we have $(2^d)^j$ nodes, each containing $2^d - 1$ coefficients with support interval hypercubes with edge length 2^j.

In the 2-d case, the support intervals of the coefficients are squares with side length of power 2. There are 3 coefficients for each support interval, one for each of the wavelet subspaces: W^d, W^v and W^h; thus, each quad tree node contains its 3 corresponding coefficients. Figure 7 shows the wavelet tree for an 8×8 array and zooms in on a multidimensional tile, described in Section 3.2. The support interval of the children nodes, which are the four quadrants of the support interval of the parent node, are shown in dark grey. To reconstruct a point in the original 2-d array, one has to traverse the quad tree bottom up and use all 3 coefficients in each node.

3.2 Disk Block Allocation of Multidimensional Wavelets

As in the single dimensional case, our main concern is to pack coefficients in disk blocks so that we achieve the highest possible block utilization on query time and thus decreasing retrieval cost. The solution is to assign as many coefficients with the same support to the same disk block as possible.

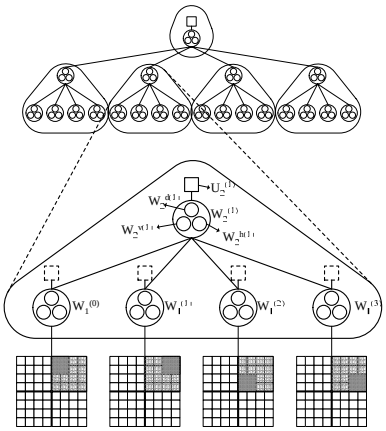


Figure 7: Non-Standard Form Wavelet Tree

This results in different disk block allocation strategies for the two multidimensional forms of decomposition. We assume d-dimensional dataset, where each dimension has size $N = 2^n$. Furthermore, disk block size is B^d, where $B = 2^b$.

In the standard multidimensional decomposition, each dimension can be treated independently. Therefore, for each dimension we construct tiles of size B containing the B coefficients of a subtree, similar to the single dimensional case. The cross product of these d sets of single dimensional bases construct B^d multidimensional bases. The coefficients corresponding to these B^d bases are stored in the same block and form a multidimensional tile.

In the non-standard multidimensional decomposition, tiles are subtrees of the quad tree. The branching factor of a d-dimensional quad tree is $D = 2^d$ and each node contains $D - 1$ coefficients. Therefore, a tile of height b contains $\frac{D^b - 1}{D - 1}$ nodes or equivalently $D^b - 1$ coefficients. By also storing the scaling corresponding to the root node we create tiles of $D^b = (2^d)^b = (2^b)^d = B^d$ coefficients which fit in a disk block of size B^d. Figure 7 shows the tiling of a 8×8 array, for disk blocks of size 16.

4. SHIFT-SPLIT

In this Section we describe our general purpose operations, **SHIFT** and **SPLIT**, on wavelet transformed vectors. Later, in Section 5, we discuss the applications that can benefit from our operations.

There is a relationship among the coefficients in the transform of a vector, **a** and in the transform of a dyadic region **b** of the vector. This relationship is captured by *shifting*, re-indexing, the wavelet coefficients (details) of **b** and by *splitting*, calculating contributions from the scaling coefficient (average).

The **SHIFT-SPLIT** operations are better understood in the context of wavelet trees. Let **a** be a vector of size $N = 2^n$ and let **b** be the $(k+1)$-th dyadic range of vector **a** with size $M = 2^m$. The wavelet coefficients of $\widehat{\mathbf{a}}$ are denoted by $w^a_{j,l}$, whereas the wavelet coefficients of $\widehat{\mathbf{b}}$ are denoted by $w^b_{j,l}$; similarly for scaling coefficients, $u^a_{j,l}$ and $u^b_{j,l}$. Also, let T_a and T_b denote the wavelet trees of $\widehat{\mathbf{a}}$ and $\widehat{\mathbf{b}}$, respectively. Figure 8 illustrates the above.

The support of the wavelet coefficient $w^a_{m,k}$ is the dyadic

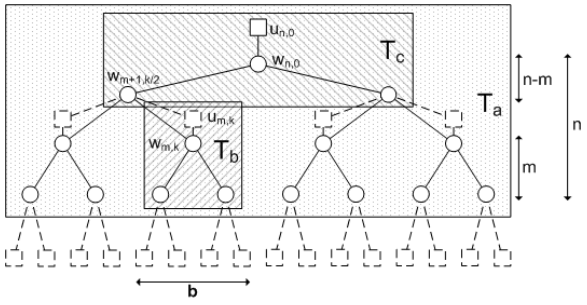


Figure 8: Shift-Split Operations

range that **b** represents. Therefore, $w^a_{m,k}$ covers $w^b_{m,0}$ and vice versa, since their support is the same range of **a**; see T_b in Figure 8. Furthermore, all children of $w^a_{m,k}$ in T_a have common support with the corresponding children of $w^b_{m,0}$ in T_b. Specifically, at the j-th level of decomposition, the i-th coefficient $w^b_{j,i}$ of T_b has the same support with the $\big(k2^{m-j}+i\big)$-th coefficient $w^a_{j,k2^{m-j}+i}$.

Definition of SHIFT. Let **a** be a vector of size $N = 2^n$ and let **b** be the $(k+1)$-th dyadic range of vector **a** with size $M = 2^m$. Also, let $f : \mathbb{Z} \to \mathbb{Z}$, $f(i) = \big(\frac{k}{2^{m-j}} + i\big)$, be a function that translates the indices i of $\widehat{\mathbf{b}}$ to indices $f(i)$ of $\widehat{\mathbf{a}}$. The **SHIFT** operation on the transformed vector $\widehat{\mathbf{b}}$ is defined as the re-indexing of the wavelet coefficients by function f. □

The wavelet coefficients of $\widehat{\mathbf{a}}$ that cover the interval represented by **b** contain a portion of the energy of the average of vector **b**. To be exact, the value of the wavelet coefficients $w^a_{j,\lfloor \frac{k}{2^{j-m}} \rfloor}$ for $j \in [m+1, n]$, as well as the average $u^a_{n,0}$ depend on the value of the average $u^b_{m,0}$; these coefficients lie in the path from $w^a_{m,k}$ to the root and are contained in T_c of Figure 8. Essentially the value of the average $u^b_{m,0}$ is *split* across these $n-m+1$ coefficients, contributing either positively, or negatively.

Definition of SPLIT. Let **a** be a vector of size $N = 2^n$ and let **b** be the $(k+1)$-th dyadic range of vector **a** with size $M = 2^m$. Also, let $g : [m+1, n] \to \mathbb{R}$,

$$g(j) = \begin{cases} \frac{1}{\sqrt{2}^{j-m}} u^b_{m,k}, & \text{if } k \mod 2^{j-m} \text{ even} \\ -\frac{1}{\sqrt{2}^{j-m}} u^b_{m,k}, & \text{if } k \mod 2^{j-m} \text{ odd} \end{cases}$$

be the function that calculates the contribution of $u^b_{m,k}$ per level j. The **SPLIT** operation on the transformed vector $\widehat{\mathbf{b}}$ calculates the contribution of $u^b_{m,k}$ to the $n-m$ wavelet coefficients: $\delta w^a_{j,\lfloor \frac{k}{2^{j-m}} \rfloor} = g(j)$ for $j \in [m+1, n]$ and to the average: $\delta u^a_{n,0} = \frac{1}{\sqrt{2}^{n-m}} u^b_{m,k}$. □

To demonstrate the use of the **SHIFT-SPLIT** operations, let us look at two examples.

Example 1. Assume we are to transform a very large vector **a** of size $N = 2^n$ into the wavelet domain, where only the subregion $[k2^m, (k+1)2^m - 1]$ of the vector contains non-zero values. Let **b** be that non-zero subregion of size $M = 2^m$. Because of the fact that **b** forms a dyadic interval, we can apply the **SHIFT-SPLIT** operations to construct $\widehat{\mathbf{a}}$ as follows. First, we obtain the wavelet transform $\widehat{\mathbf{b}}$ in time $O(M)$. Next, we apply the **SHIFT** operation to place the wavelet coefficients of $\widehat{\mathbf{b}}$ in their corresponding position in $\widehat{\mathbf{a}}$. Finally, we apply the **SPLIT** operation on the average of **b** to obtain $n-m+1$ contributions and construct the remaining $n-m+1$ coefficients. We have completed the wavelet transformation of **a** in time $O(M+n-m) = O(M+\log \frac{N}{M})$, instead of $O(N)$. □

Example 2. Assume we have already transformed vector **a** of size $N = 2^n$ into the wavelet domain. There are updates, stored in vector **b**, coming for a subregion $[k2^m, (k+1)2^m-1]$ of **a**. The goal is to update the wavelet transform of **a** as efficiently as possible. Each of $|\mathbf{b}| = \mathbf{M} = \mathbf{2^m}$ updates requires $n+1$ values to be updated, leading to a total cost of $O(M \log N)$. However, we can use the **SHIFT-SPLIT** operations to batch updates and reduce cost, as follows. First, we obtain the wavelet transform $\widehat{\mathbf{b}}$ in time $O(M)$. Next, we apply the **SHIFT** operation to calculate the indices of the wavelet coefficients of $\widehat{\mathbf{a}}$ which need to be updated by the wavelet coefficients of $\widehat{\mathbf{b}}$. Finally, we apply the **SPLIT** operation on the average of **b** to obtain $n-m+1$ contributions and update the corresponding coefficients in $\widehat{\mathbf{a}}$. The total update cost using **SHIFT-SPLIT** has been reduced to $O(M + \log \frac{N}{M})$. □

4.1 Multidimensional Shift-Split

The **SHIFT-SPLIT** operations in the multidimensional decomposition exploit the relationship between the wavelet coefficients of the entire dateset and those in a multidimensional dyadic range. A multidimensional dyadic range is formed by the cross product of single dimensional dyadic intervals. For the non-standard decomposition we will only consider *cubic* multidimensional dyadic ranges resulting from dyadic intervals of equal length for all dimensions; arbitrary multidimensional dyadic ranges can always be seen as a collection of cubic intervals.

To perform the **SHIFT-SPLIT** operations for the standard multidimensional decomposition, one has to perform the operations for each dimension separately. Any coefficient in the d-dimensional dyadic interval can only be shifted or split in each dimension, and thus can sustain d operations in total. Consider as an example a d-dimensional dataset, where each dimension has size $N = 2^n$, and a cubic dyadic range of edge $M = 2^m$. The **SHIFT** operation affects $(M-1)^d$ coefficients and the **SPLIT** operation calculates $(M+n-m)^d - (M-1)^d$ contributions.

With the non-standard multidimensional transformation, all the wavelet coefficients in the cubic dyadic range must be shifted similar to the standard transformation. However, only the scaling coefficient has to be split and the contributions for the coefficients inside nodes on the path to the root have to be calculated. Therefore, in the non-standard transformation, the **SHIFT** operation affects $M^d - 1$ coefficients and the **SPLIT** operation calculates $(2^d-1)(n-m)+1$ contributions.

4.2 Shift-Split of Tiles

In this section we assume that the coefficients are stored using the optimal block allocation strategy described in Section 3. We will calculate the number of tiles affected by the operations for the single dimensional and extend to the two multidimensional wavelet transformations.

We start with the single dimensional case of a vector of size $N = 2^n$ and its $k+1$-th dyadic interval of size $M = 2^m$,

	SHIFT	SPLIT
Standard	$O\left(\lceil\frac{M}{B}\rceil^d\right)$	$O\left(\left(\lceil\frac{M}{B}\rceil - \lceil\log_B\frac{N}{M}\rceil\right)^d - \lceil\frac{M}{B}\rceil^d\right)$
Non-Standard	$O\left(\lceil\frac{M}{B}\rceil^d\right)$	$O\left((2^d-1)\lceil\log_B\frac{N}{M}\rceil\right)$

Table 1: Shift-Split of Tiles

when the disk block size is $B = 2^b$. The coefficients affected by the **SHIFT** operation belong to a subtree of the wavelet tree, and that subtree contains exactly $\lceil\frac{M}{B}\rceil$ tiles. On the other hand, the **SPLIT** operation calculates $\log\frac{N}{M}$ contributions. Because these contributions lie on a single path to the root inside every tile, there are $\log B$ coefficients affected per tile. This results in exactly $\lceil\log_B\frac{N}{M}\rceil$ tiles containing the contributions of the **SPLIT** operation. To summarize for the single dimensional case, the **SHIFT** operation affects B times less tiles than coefficients, whereas the **SPLIT** operation affects $\log B$ times less tiles than coefficients.

Extending to d-dimensional tiles of size $B^d = (2^b)^d$ and applying the observation for the single dimensional case, we derive the number of d-dimensional tiles affected by the operations in each multidimensional form. The results are summarized in Table 1. For the remainder of this paper we will drop the ceiling operations to increase readability.

5. SHIFT-SPLIT APPLICATIONS

In this section, we describe some of the applications where the **SHIFT-SPLIT** operations prove useful and draw comparisons to the existing alternatives.

5.1 Transformation of Massive Multidimensional Datasets

One of the most important application of the **SHIFT-SPLIT** operations is I/O efficient transformation of massive multidimensional datasets. In the following, we assume that the dataset is d-dimensional with each dimension having a domain of size $N = 2^n$, so that the hypercube has N^d cells. The available memory for performing the transformation is M^d, where $M = 2^m$, measured in units of coefficients. Therefore, at any point in time, there can only be $M^d \ll N^d$ coefficients in main memory. Given these restrictions we need to construct an efficient, in terms of required I/O operations, algorithm for decomposing the dataset. We begin by assuming that one I/O operation involves a single data value, or coefficient. Later, we measure I/O operations in units of disk blocks, as we consider the optimal disk block allocation strategy described in Section 3.2.

The intuition behind our approach is simple. We assume that the data are either organized and stored in multidimensional chunks of equal size and shape, or that the chunk-organization process has been performed, similar to [2, 12]. We transform each chunk and use the **SHIFT** operation to relocate the coefficients and the **SPLIT** operation to update the stored coefficients. The chunks are hypercubes of size M^d so that they fit in main memory. Figure 9 shows a one dimensional example, for $N = 16$ and $M = 4$, where the current chunk is C. The transformation of C results in the wavelet coefficients inside the box needing to be shifted. The scaling coefficient of C must be split to calculate the contributions to the coefficients shown in grey. With black are shown the coefficients that have a finalized value; that is, coefficients that will not be affected by C or by chunks coming after C. With white are shown the coefficients that do not cover any of the chunks seen so far.

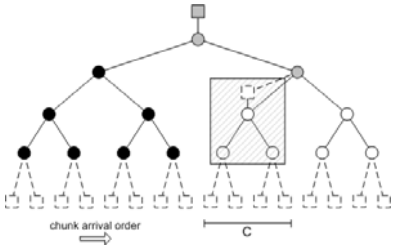


Figure 9: Transformation by Chunks

RESULT 1. *The I/O complexity for transforming a d-dimensional dataset with each dimension having domain size $N = 2^n$ into the standard form of decomposition using memory of M^d coefficients is $O\left(\left(\frac{N}{B} + \frac{N}{M}\log_B\frac{N}{M}\right)^d\right)$ disk blocks of size B^d.*

PROOF. As mentioned in Section 4.1, the **SHIFT** operation, for the standard decomposition, affects $(M-1)^d$ coefficients, whereas the **SPLIT** operation affects $(M+n-m)^d - (M-1)^d$ coefficients. Consequently, each chunk requires $O\left((M+n-m)^d\right) = O\left((M+\log\frac{N}{M})^d\right)$ I/O operations. Summing for all $\left(\frac{N}{M}\right)^d$ chunks, we derive the I/O complexity for the standard multidimensional wavelet transformation measured in coefficients: $O\left(\left(N + \frac{N}{M}\log\frac{N}{M}\right)^d\right)$. Now, let us consider disk blocks of size B^d, for $B = 2^b$. In this case, the I/O cost per chunk in units of disk blocks is: $O\left(\left(\frac{M}{B} + \log_B\frac{N}{M}\right)^d\right)$. Summing for all chunks we derive the I/O complexity, measured in terms of disk blocks, for the standard multidimensional wavelet transformation: $O\left(\left(\frac{N}{B} + \frac{N}{M}\log_B\frac{N}{M}\right)^d\right)$ □

Vitter et al. [12, 13] use the standard form to decompose multidimensional datasets, without taking under consideration, however, our optimal block allocation strategy. They transform a dense d-dimensional dataset in $O(N_z^d \log_M N)$disk I/O operations; in the case of sparse data with N_z non-zero values the I/O complexity is $O(N_z^d \log_M N)$. We can modify our **SHIFT-SPLIT** approach to accommodate for sparseness similar to the latter case, where only N_z non-zero values exist; the modified I/O complexity is $O\left(\left(N_z + \frac{N_z}{M}\log\frac{N}{M}\right)^d\right)$. However, for comparison purposes we omit the effect of sparseness in the original data. The I/O complexities are summarized in Table 2.

RESULT 2. *The I/O complexity for transforming a d-dimensional dataset with each dimension having domain size $N = 2^n$ into the non-standard form of decomposition using memory of $M^d + (2^d-1)\log\frac{N}{M}$ coefficients is $O\left(\left(\frac{N}{B}\right)^d\right)$ disk blocks of size B^d.*

PROOF. In the case of the non-standard multidimensional wavelet transformation, the **SHIFT** operation affects M^d-1 coefficients, whereas the **SPLIT** operation affects $(D-1)(n-m)+1$coefficients, where $D=2^d$. The per chunk I/O cost is $O\left(M^d+(D-1)\log\frac{N}{M}\right)$. Summing for all $\left(\frac{N}{M}\right)^d$chunks, we derive the I/O complexity, measured in terms of coefficients, for the non-standard multidimensional wavelet transformation: $O\left(N^d+(D-1)\left(\frac{N}{M}\right)^d\log\frac{N}{M}\right)$. When tiling is used, the I/O cost per chunk in units of disk blocks becomes: $O\left(\left(\frac{M}{B}\right)^d+(D-1)\log_B\frac{N}{M}\right)$. Summing for all chunks we derive the I/O complexity, measured in terms of disk blocks, for the non-standard multidimensional wavelet transformation: $O\left(\left(\frac{N}{B}\right)^d+(D-1)\left(\frac{N}{M}\right)^d\log_B\frac{N}{M}\right)$. However, if we enforce a particular access pattern on the chunks, namely a z-ordering, and allow some extra amount of memory $(2^d-1)\log\frac{N}{M}$ to store those coefficients that are affected by the splitting of the scaling coefficient of the chunks, we can reduce the cost to the optimal $O(N^d)$, as seen in Table 2. A similar approach has been suggested in [2], where a recursive procedure is used to ensure values come in the particular access pattern. □

5.2 Appending to Wavelet Decomposed Transforms

In this section we investigate the problem of appending new data to existing transformed data. Appending is fundamentally different from updating in that it results in the increase of the domain of one or more dimensions. As a result, the wavelet decomposed dimensions also grow, new levels of transformation are introduced and therefore the transform itself changes. We would like to perform appending directly in the wavelet domain, preserving as much of the transformed data as possible and avoiding reconstruction of the original data. The **SHIFT-SPLIT** operations helps us achieve this goal. To make complexity analysis easier, we omit the effect of the optimal disk block allocation strategy, or equivalently assume disk block size of 1 coefficient. Also, we use the standard form of decomposition, as analysis for the non-standard form is similar.

As a motivation, consider the scenario where a massive multidimensional dataset containing measurements over 10 years is decomposed into the wavelet domain to expedite query processing. A new set of data for the following year has become available, which results in appending to the time domain and possibly on other measure dimensions. Let us assume that the 10-year decomposed d-dimensional dataset has size of N^d, and that the available memory is M^d, for $N=2^n$ and $M=2^m$.

Our **SHIFT-SPLIT** approach to the problem is the following, repeating for each M^d data values that we gather in memory. We start by performing the d-dimensional DWT on the gathered data. Next, if required, we make the necessary space on the original transformed data (expand) to accommodate for the new data to be appended. The final step is to shift and split the gathered data to update the expanded data. The second step is the most important in the appending application. Let us assume that we must expand on one of the dimensions to accommodate for the coefficients held in memory. The expansion means that the wavelet tree for that dimension has to increase its height by 1, and thus double its domain range. This expansion process is carried out by shifting and splitting the decomposed data in this dimension. Figure 10 shows expansion in one dimension, where T_{old} becomes T_{new} and $|T_{new}|=2|T_{old}|$. The expansion step creates the necessary space for the current chunk of M^d coefficients in memory, as well as for some of the next chunks. Therefore, this step, although costly, is rather rare.

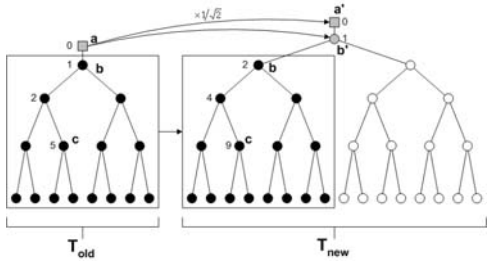


Figure 10: Wavelet Tree Expanding

The I/O cost of expanding transformed data in one dimension is $O(N^d)$ as all coefficients have to be shifted to construct the new data cube of size $2N^d$. Note, that although the asymptotic cost is high, the required **SHIFT-SPLIT** operations are very fast, which leads to fast execution times for expanding the domain of one dimension. This phenomenon is amplified by the use of tiling and is demonstrated in Section 6.2. Moreover, this operation, unlike reconstruction, does not require memory to process. The I/O cost of applying the **SHIFT-SPLIT** operations on the memory chunk of size M^d is $O\left(\left(M+\log\frac{N}{M}\right)^d\right)$.

5.3 Data Stream Approximation

In this section, we revisit the appending problem, this time in the context of stream query processing: we wish to maintain a wavelet approximation of a multidimensional datastream in the time-series model, when dimension sizes are unbounded and new data are coming. The focus here is to construct a space and time efficient algorithm for maintaining the best K-term synopsis. We show that we cannot, in general, maintain a K-term synopsis for multidimensional datasets decomposed using the standard form under bounded space. However, if certain conditions are met we can maintain a K-term synopsis effectively.

Let us start with the simple one dimensional case. As shown in [5], we can maintain the best K-term approximation of a data of length $N=2^n$ by using space $K+\log N+1$. We always store the K highest coefficients encountered so far, plus those coefficients whose value can change by subsequent data arrivals. These coefficients, termed wavelet crest in [8], lie on the path from the current value to the root of the wavelet tree and therefore, they are exactly $\log N+1$. Equivalently, if we consider a range containing just the data values under consideration, the **SPLIT** operation results in contributions lying in the wavelet crest. Therefore, at any time we have to keep the coefficients that can be affected by the **SPLIT** operation in memory.

RESULT 3. *A K-term wavelet synopsis of a data stream of size N in the time series model can be maintained using memory of $O(K+B+\log\frac{N}{B})$ coefficients with $O\left(\frac{1}{B}\log\frac{N}{B}\right)$ per-item computational cost.*

PROOF. If we keep in memory a buffer of size $B=2^b$ we can reduce per-item processing time at the expense of

Transformation Method	I/O cost *(in coefficients)*	I/O cost *(in blocks)*
Vitter et al. (Standard)	$O\left(N^d \log_M N\right)$	
Shift-Split (Standard)	$O\left((N + \frac{N}{M}\log\frac{N}{M})^d\right)$	$O\left((\frac{N}{B} + \frac{N}{M}\log_B\frac{N}{M})^d\right)$
Shift-Split (Non-Standard)	$O\left(N^d\right)$	$O\left((\frac{N}{B})^d\right)$

Table 2: I/O Complexities

extra space. We collect B coefficients in the buffer, transform them and apply the **SHIFT** operation to obtain the $B-1$ relocated wavelet coefficients. Next, we compare these coefficients with the K highest, to obtain the new set of K highest coefficients. Finally, we have to update the coefficients that can change by using the contributions derived from the **SPLIT** operation. The number of contributions for a buffer of size B is $\log\frac{N}{B}$ and thus the space required for the coefficients on the crest is $\log\frac{N}{B}$. The total computational cost for the buffer, which includes the cost for transformation and the cost for updating the coefficients on the crest, is $O\left(B + \log\frac{N}{B}\right)$. As a result, the per-item computational cost is $O\left(\frac{1}{B}\log\frac{N}{B}\right)$ reduced from $O(\log N)$, at the expense of extra space of B. □

The key for being able to maintain a wavelet approximation in the one dimensional case is the fact that only a single path to the root of the wavelet tree has to be maintained at any time. Let us turn our attention to the multidimensional case. We assume that the data needs to be appended in only one dimension (usually the time dimension), which is the case for multidimensional data streams of the time series model. To separate the continuously increasing dimension, we let T denote its current size, whereas the other dimensions have a constant size of N. Therefore, the d-dimensional data stream has a size of $N^{d-1}T$. The amount of space, besides the K terms, required to maintain a K-term approximation depends on the number of coefficient that can be affected by a **SPLIT** operation. We calculate the number of these coefficients for each of the multidimensional forms, assuming that we have extra storage to buffer M^d coefficients, where $M = 2^m$. Note that when $M = 1$, we achieve maximum efficiency in terms of space, in the expense of increasing the per-item processing time.

RESULT 4. *A K-term standard wavelet synopsis of a d-dimensional data stream growing in the T dimension can be maintained using memory of $O\left(K + M^d + N^{d-1}\log\frac{T}{M}\right)$ coefficients.*

PROOF. In the standard form, there are $d-1$ wavelet trees of size N and a single wavelet tree of size T. Since, the stream expands on the dimension of size T, we only have to keep a path to the root for the wavelet tree corresponding to that dimension. However, a new data value can arrive in any position on the other trees, which means that we have to keep all the paths to the root for the $d-1$ trees. To recap, we need to keep all N 1-d basis functions from the $d-1$ trees of size N and only $\log\frac{T}{M}$ 1-d basis functions for the tree corresponding to the dimension which increases. The cross product between these sets of 1-d basis functions results in $N^{d-1}\log\frac{T}{M}$ d-dimensional basis functions and thus that many coefficients have to be maintained, besides the K highest coefficients and the extra storage space of M^d coefficients used for buffering. Therefore, the required space of $O\left(K + M^d + N^{d-1}\log\frac{T}{M}\right)$ coefficients is prohibitive, except in the case where the constant dimensions have very small domain size, so that N^{d-1} is small. □

RESULT 5. *A K-term non-standard wavelet synopsis of a d-dimensional data stream growing in the T dimension can be maintained using memory of $O\left(K + M^d + (2^d-1)\log\frac{N}{M} + \log\frac{T}{N}\right)$ coefficients.*

PROOF. Since the dimension with size T is constantly expanding, we have to deal with non equal dimension sizes, similar to [2]. Such a data stream can be seen as a $\frac{T}{N}$ hypercubes of size N^d, where each of these hypercubes can be decomposed with the non-standard form. Each of these $\frac{T}{N}$ hypercubes results in a wavelet tree capturing the non-standard decomposition, where there exists a single average as the root of each of these trees. We apply the single dimensional transformation on the $\frac{T}{N}$ data constructed by these averages. The final result consists of $\frac{T}{N}$ non-standard multidimensional trees and a single one dimensional tree which has as leaf nodes the averages of the non-standard trees. We assume the z-ordered access pattern, described in Section 5.1, and we allow for extra buffering space of M^d coefficients. Under these restrictions, the coefficients we have to retain lie in a path to the root in the last tree of the hypercubes, and in the path to the root in the single dimensional wavelet tree. Therefore we need to keep $(2^d-1)\log\frac{N}{M}$ coefficients from the non-standard tree and $\log\frac{T}{N}$ coefficients from the 1-d tree, resulting in a total space cost of $O\left(K + M^d + (2^d-1)\log\frac{N}{M} + \log\frac{T}{N}\right)$. □

5.4 Partial Reconstruction from Wavelet Transforms

In this section, we discuss the problem of reconstructing a set of values specified by a range on a multidimensional dataset. The problem is equivalent to translating the selection operation of relational algebra to the wavelet domain. Chakrabarti et al. [2] have provided a solution for the non-standard form, in which they identify the coefficients who cover the range and calculate their contribution. Here, we present a similar approach, based on the inverse of **SHIFT**-**SPLIT** operations, which generalizes to both forms of decomposition. The inverse of **SHIFT** is essentially the inverse index translation, whereas the inverse of **SPLIT** is Lemma 1, which shows how to reconstruct a value from contributions on a path to the root. Therefore, the cost of the inverses of these operations is the same.

We focus our discussion here to multidimensional ranges that are dyadic ranges; an arbitrary selection range can be seen as a number of such dyadic ranges. Therefore, our problem degenerates to the reconstruction of a d-dimensional dyadic range of size M^d, given the transformation of the entire data of size N^d. The scaling coefficients of the dyadic range are calculated using the inverse **SHIFT**, whereas the

rest of the coefficients are simply calculated from the coefficients in the original dataset by re-indexing, using the inverse **SPLIT**.

RESULT 6. *The time complexity for reconstructing a d-dimensional dyadic range of size M^d from a wavelet transformed signal of size N^d is $O\left(\left(M+\log\frac{N}{M}\right)^d\right)$ for the standard form and $O\left(M^d+(D-1)\log\frac{N}{M}\right)$ for the non-standard.*

PROOF. It follows from the complexity of the **SHIFT-SPLIT** operations. □

6. EXPERIMENTS

In this section, we study the performance of the **SHIFT-SPLIT** operations in three real world scenarios. First, we use these operations to transform a large dataset into the wavelet domain. Next, we show how **SHIFT-SPLIT** operations are employed for the maintenance of transformed data in an appending scenario. Finally, we show the significant improvement in the update cost for maintaining a wavelet synopsis in a data stream application by employing additional memory as buffer. We would like to emphasize that the experiments are accurate implementations of the operations on real disks with real disk blocks (see [6] for further information).

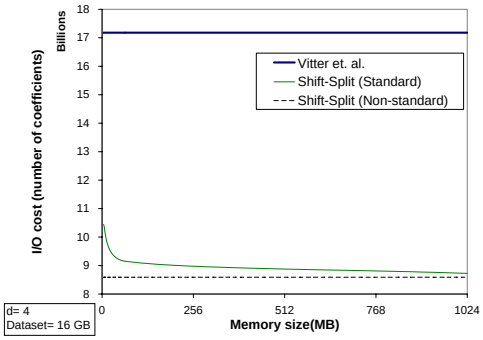


Figure 11: Effect of Larger Memory

6.1 Transformation of Massive Multidimensional Datasets

In this set of experiments, we transform a large dataset, **TEMPERATURE**, into the wavelet domain using limited available memory. The **TEMPERATURE** dataset is a real-world dataset provided to us by JPL that measures the temperatures at points all over the globe at different altitudes for 18 months, sampled twice every day. We construct a 4-dimensional cube with latitude, longitude, altitude and time as dimension attributes, and temperature as the measure attribute, with the total size of the cube being 16GB.

Figure 11 shows that larger memory considerably reduces transformation cost of **SHIFT-SPLIT** in the Standard form but it does not noticeably affect **SHIFT-SPLIT** in the Non-Standard form. The reason behind this is that the cost of the **SPLIT** operation is considerably different for the two forms of multidimensional wavelet transformation. Increasing memory size causes a significant decrease in **SPLIT** cost and consequently a major decrease of the Standard form transformation as there are many coefficients affected by the contributions of the **SPLIT** operation. However, **SPLIT** cost is almost negligible in the Non-Standard form (see Table 1). Finally, this figure also states that our **SHIFT-SPLIT** approach outperforms the Vitter et al. [12] algorithm for any memory size.

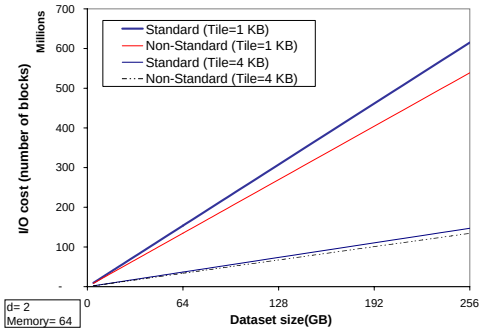


Figure 12: Effect of Larger Tiles

As we have shown in Section 3, not only Tiling is the optimal wavelet coefficient blocking for query processing, but it is also a **SHIFT-SPLIT** friendly schema which introduces significant cost improvements in the transformation process. Figure 12 demonstrates this fact by using different tile sizes and thus illustrates the scalability of the SHIFT-SPLIT algorithms.

6.2 Appending to Wavelet-Transformed Data

We examine our proposed appending technique on the **PRECIPITATION** [14] dataset, where we incrementally receive new sets of data every month. **PRECIPITATION** is a real-life dataset that measures the daily precipitation for the Pacific NorthWest for 45 years. We built a 3-dimensional cube with latitude, longitude and time as dimensional attributes, and precipitation as the measure attribute for every day. The sizes of these dimensions are 8, 8 and 32 respectively for each month. Figure 13 demonstrates the **SHIFT-SPLIT** I/O cost as new sets of data are appended. The sudden jumps in the figure correspond to the expansion process, where all coefficients must be shifted to accommodate for new data values. One can observe that this expansion process is not such a dominating factor as described in Section 5.2, especially for larger disk block sizes.

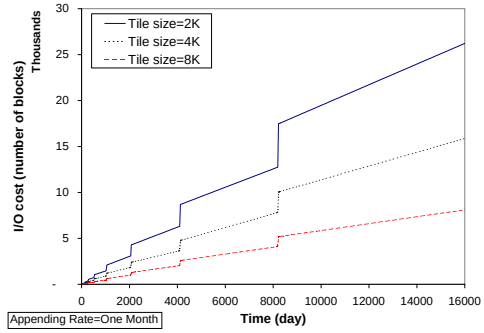


Figure 13: SHIFT-SPLIT in Appending

6.3 Data Stream Approximation

In this scenario we only need to preserve the synopsis of the **PRECIPITATION** dataset, limited to a memory footprint of 40KB. Figure 14 demonstrates the computational cost versus the extra storage trade-off described in Section 5.3. As the figure suggests, the update cost can be improved by 88% by employing additional memory buffer of only 6% of the total synopsis size.

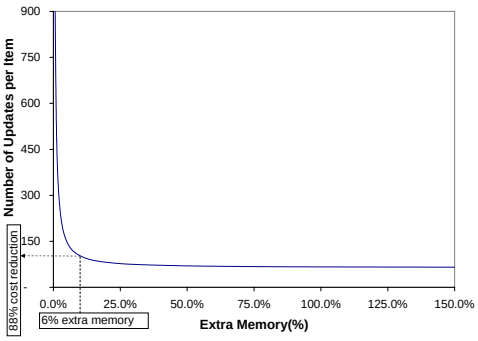


Figure 14: SHIFT-SPLIT in Multidimensional Streaming

7. CONCLUSIONS

We have introduced two general purpose operations, termed **SHIFT** and **SPLIT**, that work directly in the wavelet domain and can also be applied in combination with the optimal disk block allocation strategy. We analyze their costs for both the single dimensional case and the two forms of multidimensional transformation.

There is a significant number of applications that can benefit from these operations. We have revisited some data maintenance scenarios, such as transforming massive multidimensional datasets and reconstructing large ranges from wavelet decomposed data, and utilized the **SHIFT-SPLIT** operations to draw comparisons with the current state of the art techniques. Furthermore, we have provided solutions to some previously un-explored maintenance scenarios, namely, appending data to an existing transformation and approximation of multidimensional data streams. We demonstrated the effectiveness of the proposed techniques both analytically and experimentally, and we conjecture that the introduced operations can prove useful in a plethora of other applications, as the **SHIFT-SPLIT** operations stem from the general properties and behavior of wavelets.

8. REFERENCES

[1] A. Bulut and A. K. Singh. SWAT: Hierarchical stream summarization in large networks. In *Proceedings of ICDE*, pages 303–314, 2003.

[2] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *Proceedings of VLDB*, pages 111–122, 2000.

[3] A. Deligiannakis and N. Roussopoulos. Extended wavelets for multiple measures. In *Proceedings of ACM SIGMOD*, pages 229–240, 2003.

[4] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *Proceedings of ACM SIGMOD*, 2002.

[5] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of VLDB*, pages 79–88, 2001.

[6] M. Jahangiri and C. Shahabi. ProDA: A Suite of WebServices for Progressive Data Analysis. In *Proceedings of ACM SIGMOD (demonstration)*, 2005.

[7] D. Lemire. Wavelet-based relative prefix sum methods for range sum queries in data cubes. In *Proceedings of CASCON*. IBM, October 2002.

[8] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Awsom: Adaptive, hands-off stream mining. In *Proceedings of VLDB*, 2003.

[9] R. Schmidt and C. Shahabi. Propolyne: A fast wavelet-based technique for progressive evaluation of polynomial range-sum queries. In *Proceedings of EDBT*, 2002.

[10] C. Shahabi and R. Schmidt. Wavelet disk placement for efficient querying of large multidimensional data sets. In *Department of Computer Science Technical Reports*. University Of Southern California, 2004.

[11] E. J. Stollnitz, T. D. Derose, and D. H. Salesin. *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann Publishers Inc., 1996.

[12] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of ACM SIGMOD*, pages 193–204, 1999.

[13] J. S. Vitter, M. Wang, and B. R. Iyer. Data cube approximation and histograms via wavelets. In *Proceedings of CIKM*, pages 96–104, 1998.

[14] M. Widmann and C.Bretherton. 50 km resolution daily precipitation for the pacific northwest, 1949-94.

[15] Y.-L. Wu, D. Agrawal, and A. E. Abbadi. Using wavelet decomposition to support progressive and approximate range-sum queries over data cubes. In *Proceedings of CIKM*, pages 414–421, 2000.

APPENDIX

A. WAVELET TRANSFORM

Let us define wavelet transformation more precisely. Given a lowpass scaling function ϕ and a highpass wavelet function ψ, we can define a family of functions by scaling and translating:

$$\phi_{j,k}(t) \equiv 2^{-j/2}\phi(2^{-j}t-k), \quad \psi_{j,k}(t) \equiv 2^{-j/2}\psi(2^{-j}t-k),$$

where j indexes scale and k indexes position in time. The scales and translations of these functions form an orthogonal basis for L_2. Therefore any function $f \in L_2$ can be represented in the wavelet domain:

$$f = \langle f, \phi_{0,0}\rangle\phi_{0,0} + \sum_j \sum_k \langle f, \psi_{j,k}\rangle\psi_{j,k}$$

The wavelet decomposition of a vector $\mathbf{a}$ of size 2^n consists of the scaling coefficient (average) $u_{n,0} = \langle f, \phi_{n,0}\rangle$ and $2^n - 1$ wavelet coefficients (details) across all levels of decomposition; for each level of decomposition j, there are 2^{n-j} wavelet coefficients $w_{j,k} = \langle f, \psi_{j,k}\rangle$ (See Figure 1). We,

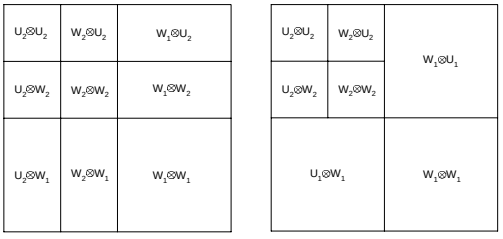
also, represent all scaling and wavelet coefficients of level j with vectors of U_j and W_j where U_j and W_j are spanned by $\phi_{j,k}$ and $\psi_{j,k}$ respectively. Therefore, we can demonstrate wavelet decomposition by a series of vector decompositions as following:

$$U_0 = U_3 \oplus W_3 \oplus W_2 \oplus W_1$$

B. MULTIDIMENSIONAL WAVELET TRANSFORMATION

To perform the wavelet decomposition of multidimensional datasets we need multidimensional wavelets and scaling functions. For illustration purposes, we focus our discussion on 2-dimensional transformations. The extension to higher dimensionality is straightforward. In general, the multidimensional wavelets and scaling spaces are constructed from tensor products of single dimensional wavelets W_j and scaling vectors U_k, where j and k index scale, or equivalently, levels of decomposition.

The tensor product of the vectors U_j and U_k results in a 2-dimensional subspace $U_{j,k} = U_j \otimes U_k$. Similarly for W_j and W_k we shape the subspace $W^d_{j,k} = W_j \otimes W_k$. Tensor products among scaling and wavelet vectors result in 2 more sets of subspaces: $W^h_{j,k} = U_j \otimes W_k$ and $W^v_{j,k} = W_j \otimes U_k$. These 4 sets of 2-d subspaces, $U_{j,k}$, $W^d_{j,k}$, $W^h_{j,k}$, $W^v_{j,k}$ can decompose 2-d space. However there are 2 ways to perform multidimensional wavelet decomposition, the *standard* and the *non-standard*.



a. Standard Form b. Non-Standard Form

Figure 15: 2-level decomposition

B.1 Standard Form

To decompose a 2-d array of size N^2 using the standard form, we first completely decompose one dimension and then the other, with the order not being important. This means that we first transform each of the N rows of the array to construct a new array and then take each of the N columns of the new array and again perform 1-d DWT on them. The final array is the 2-d standard transform of the original array.

In terms of subspaces, each 1-d untransformed vector is initially expressed using the 0-th level vector U_0. Therefore, each cell in the untransformed array is expressed by the 2-d space $U_{0,0} = U_0 \otimes U_0$

Decomposing both vectors U_0 to the first level of decomposition $U_0 = W_1 \oplus U_1$ and distributing the tensor product, we get:

$$\begin{aligned} U_{0,0} &= U_0 \otimes U_0 \\ &= (W_1 \oplus U_1) \otimes (W_1 \oplus U_1) \\ &= (W_1 \otimes W_1) \oplus (W_1 \otimes U_1) \oplus (U_1 \otimes W_1) \oplus (U_1 \otimes U_1) \\ &= W^d_{1,1} \oplus W^v_{1,1} \oplus W^h_{1,1} \oplus U_{1,1} \end{aligned}$$

Decomposing both vectors U_0 to the second level of decomposition $U_0 = W_1 \oplus W_2 \oplus U_2$ and distributing the tensor product, we get:

$$\begin{aligned} U_{0,0} &= U_0 \otimes U_0 \\ &= (W_1 \oplus W_2 \oplus U_2) \otimes (W_1 \oplus W_2 \oplus U_2) \\ &= (W_1 \otimes W_1) \oplus (W_1 \otimes W_2) \oplus (W_1 \otimes U_2) \\ &\oplus (W_2 \otimes W_1) \oplus (W_2 \otimes W_2) \oplus (W_2 \otimes U_2) \\ &\oplus (U_2 \otimes W_1) \oplus (U_2 \otimes W_2) \oplus (U_2 \otimes U_2) \\ &= W^d_{1,1} \oplus W^d_{1,2} \oplus W^v_{1,2} \oplus W^d_{2,1} \oplus W^d_{2,2} \oplus W^v_{2,2} \\ &\oplus W^h_{2,1} \oplus W^h_{2,2} \oplus U_{2,2} \end{aligned}$$

Figure 15a shows how the 2-d array is partitioned to 9 subspaces for 2 level decomposition.

B.2 Non-Standard Form

The non-standard form differs in that the decomposition is not happening on each dimension separately. Rather, after each level of decomposition only the coefficients corresponding to the $U_{j,j}$ subspace are further decomposed. The first level of decomposition results from decomposing each dimension to the first level, exactly as in the standard form:

$$\begin{aligned} U_{0,0} &= U_0 \otimes U_0 \\ &= (W_1 \oplus U_1) \otimes (W_1 \oplus U_1) \\ &= (W_1 \otimes W_1) \oplus (W_1 \otimes U_1) \oplus (U_1 \otimes W_1) \oplus (U_1 \otimes U_1) \\ &= W^d_{1,1} \oplus W^v_{1,1} \oplus W^h_{1,1} \oplus U_{1,1} \end{aligned}$$

By decomposing the scaling vector U_1 into the next level of decomposition $U_1 = W_2 \oplus U_2$ we decompose only $U_{1,1}$ subspace.

$$\begin{aligned} U_{0,0} &= W^d_{1,1} \oplus W^v_{1,1} \oplus W^h_{1,1} \oplus U_{1,1} \\ &= W^d_{1,1} \oplus W^v_{1,1} \oplus W^h_{1,1} \oplus (U_1 \otimes U_1) \\ &= W^d_{1,1} \oplus W^v_{1,1} \oplus W^h_{1,1} \oplus ((W_2 \oplus U_2) \otimes (W_2 \oplus U_2)) \\ &= W^d_{1,1} \oplus W^v_{1,1} \oplus W^h_{1,1} \oplus \\ &(W_2 \otimes W_2) \oplus (W_2 \otimes U_2) \oplus (U_2 \otimes W_2) \oplus (U_2 \otimes U_2) \\ &= W^d_{1,1} \oplus W^v_{1,1} \oplus W^h_{1,1} \oplus W^d_{2,2} \oplus W^v_{2,2} \oplus W^h_{2,2} \oplus U_{2,2} \end{aligned}$$

Figure 15b shows how the 2-d array is partitioned to 7 subspaces for 2 level decomposition.