XWAVE: Optimal and Approximate Extended Wavelets for Streaming Data

Sudipto Guha
University of Pennsylvania
sudipto@cis.upenn.edu

Chulyun Kim
Seoul National University
cykim@kdd.snu.ac.kr

Kyuseok Shim
Seoul National University
shim@ee.snu.ac.kr

Abstract

Wavelet synopses have been found to be of interest in query optimization and approximate query answering. Recently, extended wavelets were proposed by Deligiannakis and Roussopoulos for data sets containing multiple measures. Extended wavelets optimize the storage utilization by attempting to store the same wavelet coefficient across different measures. This reduces the bookkeeping overhead and more coefficients can be stored. An optimal algorithm for minimizing the error in representation and an approximation algorithm for the complementary problem was provided.

However, both their algorithms take linear space. Synopsis structures are often used in environments where space is at a premium and the data arrives as a continuous stream which is too expensive to store. In this paper, we give algorithms for extended wavelets which are space sensitive, i.e., use space which is dependent on the size of the synopsis (and at most on the logarithm of the total data) and operates in a streaming fashion. We present better optimal algorithms based on dynamic programming and a near optimal approximate greedy algorithm. We also demonstrate the performance benefits of our algorithms compared to previous ones through experiments on real-life and synthetic data sets.

1 Introduction

Approximate query processing has recently emerged as a viable solution for dealing with the huge amounts of data, the high query complexities, and the increasingly stringent response-time requirements that characterize decision support systems (DSS) applications.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

Due to the exploratory nature of many DSS applications, in several scenarios such as ad-hoc mining or dealing with remote data [8, 1] approximate answers obtained from synopses suffice. In DSS applications, databases with multiple measures are common. For example, market basket database may include information on the revenue, the quantity of being sold and the profits. One natural and widely used tool for synopses with multiple measures is approximate Wavelet representation.

Traditionally, wavelet approximation methods in this multi-measure scenario used either decomposition on individual dimensions, or treated the data as a vector and applied a multidimensional decomposition. As pointed out by Deligiannakis and Roussopoulos in [3], these methods may result in suboptimal solutions. It is not hard to see that the former may store the same coordinate for more than one measure – which stores the coordinate of the coefficient multiple times and wastes space. The latter on the other hand, may be forced to store very small number of coordinate values of which only a few coefficients might help reduce significantly the error and not be effective as well. To remedy this, extended wavelets were proposed in [3]. This problem seeks to optimize the storage utilization by attempting to store the same wavelet coefficient across different measures, thereby eliminating the bookkeeping overhead for one (or possibly, more) of them. They gave an optimal algorithm for the sum of squared error between the representation of the data achieved by the synopsis and the input. They also gave a faster 2-approximation algorithm for the problem of maximizing the sum of weighted squares of the representation, termed as "benefit". The benefit and error add up to weighted sum of squares of the input coefficients and is therefore fixed, thus the problems can be thought of as"complimentary". They also demonstrated that extended wavelets achieve better estimation quality compared to multidimensional wavelets in several cases.

However, there are two problems with the proposed solutions. (i) Both their algorithms require linear space. Synopsis structures are frequently used in environments where space is at a premium and the data arrives as a continuous stream which is too expensive to store. Thus, linear space algorithms are not desirable in such scenarios. (ii) An approximation algorithm for maximizing the benefit does not give a good approximation algorithm for minimizing error, e.g., suppose the optimum solution had benefit 99 and error 1. Suppose a 2-approximation of the benefit achieved a benefit of 50 (which is more than $\frac{99}{2}$) — but the error of this solution is 50 as well, 50 times the optimum error.

1.1 Our contributions

We make the following contributions:

- To address the problem of linear space, we present optimal algorithms for extended wavelets which are space sensitive, i.e., use space which is dependent on the size of the synopsis (and at most on the logarithm of the total data size) and operates in a streaming fashion.
- For the problem of no guarantee on error by the previous approximation algorithm, we give an algorithm that has error *less than or equal* to the error of the optimum solution (therefore at least as much benefit), but relaxes the space bound to store a few extra coefficients (at most as many as the number of different measures).
- We also demonstrate how to adapt all the above algorithms to the context of streaming data (which connects to the linear space requirement of previous work), i.e., given multidimensional points as a stream we construct the coefficients on the fly as well as maintain the synopsis. This is particularly of use in modeling time series data.
- Through experiments on real-life and synthetic data sets, we demonstrate that our proposed algorithms have significant performance benefits while requiring much less space.

We would like to mention that if the space bound is indeed *strict*, we can give a different $(1 + \epsilon)$-approximation algorithm for the optimum error preserving the space bound. In this process, we show a *non-trivial connection* between extended wavelets and histograms similar to the V-Optimal objective. The complexity of the algorithm is asymptotically the same as the approximation algorithm presented here. However, due to space limitations, this connection cannot be described in this paper. It can be found in [7]. Furthermore, the algorithm that arises from the connection to histograms is somewhat theoretical and significantly complicated to implement, which we relegate to future work.

1.2 Organization

The paper is organized as follows. In the next section, we present related work. In Section 3, we introduce preliminary definitions and the problem of constructing extended wavelet on databases with multiple measures. In Section 4, we introduce improved optimal algorithms. We then present the approximation algorithm in Section 5. Section 6 discusses how to adapt the extended wavelet to stream. In Section 7, we present experimental results. Finally, we make concluding remarks in Section 8. Due to the lack of space, we are unable to present any proofs of the lemmas and theorems. They can be found in [7].

2 Related Work

Several approximation techniques using small summary have been developed for selectivity estimation and approximate query answering. These techniques include histograms [14, 15, 9, 13], wavelets [10, 2] and sampling [4, 20].

Wavelet-based approaches provide a mathematical tool for the hierarchical decomposition of functions, with a long history of successful applications in image processing [12, 16]. In [2, 10, 16], they demonstrated that wavelets can be accurate even in high-dimensional datasets. Recent studies have also demonstrated the applicability of wavelets in selectivity estimation [10], answering range-sum aggregates queries over data cubes [19, 18], approximate query processing [2] and data streams [11, 5].

3 Preliminaries

Wavelets, particularly Haar wavelets, provide useful tools for multi-resolution summarization. In context of databases they have been found to be of interest in query optimization, approximate query answering, and similarity estimation. We review the definition of wavelets before discussing our problem.

3.1 Wavelets

We consider signals indexed on $\{1, \ldots, N\}$, where N is a power of 2. Given a sequence of N numbers $\mathbf{X} = x_1, \ldots, x_N$, which can thought of belonging to the Euclidean space $\Re^N$, we can represent the sequence as a linear combination $\sum_{i=1}^{N} x_i \mathbf{u}_i$ where $\mathbf{u}_i$ is the N-dimensional vector where the i-th coordinate is set to 1 and all other coordinates are 0.

Definition 3.1 The function that equals 1 on set S and zero elsewhere is denoted by $\mathbf{\Gamma}(S)$. A (Haar) wavelet is a function $\mathbf{\Psi}$ on $[1, N]$ of one of the following forms:

- $\frac{1}{\sqrt{N}}\mathbf{\Gamma}([1, N])$

- $\dfrac{\mathbf{\Gamma}([i+1, i+2^j]) - \mathbf{\Gamma}([i+2^j+1, i+2\cdot 2^j])}{2^{(j+1)/2}}$
  where $i = 2k2^j$ for some integer k and $j \geq 0$.

The first type of wavelets is a vector with all coordinates equal to $1/\sqrt{N}$. Example wavelets of the second type are

$$\begin{array}{l}(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0, 0, 0, \ldots, 0, 0),\\ (0, 0, \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, 0, \ldots, 0, 0), \ldots\\ (\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}, 0, 0, 0, 0, \ldots,) \ldots\end{array}$$

There are N wavelets altogether, and they form an orthonormal basis. Thus any N-dimensional vector can be decomposed uniquely as a linear combination of the wavelet vectors (Basis property). Further if $\mathbf{\Psi}, \mathbf{\Psi}'$ are two wavelets then $\langle \mathbf{\Psi}, \mathbf{\Psi}' \rangle$ is 1 if $\mathbf{\Psi} = \mathbf{\Psi}'$ and 0 otherwise (orthonormality).

Every signal can be reconstructed exactly from all its wavelet coefficients (its full *wavelet transform*, an orthonormal linear transformation), as $\mathbf{X} = \sum_i w_i \mathbf{\Psi}_i$ where $w_i = \langle \mathbf{X}, \mathbf{\Psi}_i \rangle$ is defined as the *i-th coefficient*. We will term i as the *index* of $\mathbf{\Psi}_i$. Observe that the wavelet $\frac{1}{\sqrt{N}}\mathbf{\Gamma}([1, N])$ will have a coefficient which is $\sqrt{N}$ times the average of all the values.

Both the transformations (to the wavelet representation and from wavelets to the original representation) can be performed in linear time. There are several ways of computing them, for more details consult [16, 10, 5].

3.2 Extended Wavelets

Given N points*, each having M "measures" we have several choices in choosing a wavelet representation. In [16], two choices were outlined – "individual", i.e., to treat each of the dimensions independently or "combined", i.e., to compute the wavelet transform of the columns (dimensions) and then perform a wavelet transform along the rows. In [3], a more flexible strategy termed as "extended wavelets" were proposed, which outperformed the earlier two strategies.

Definition 3.2 An extended wavelet coefficients of N points with M measures is a triplet $< Bit, i, V >$ consisting of:

- A bitmap Bit consisting of M bits. $Bit(j)$ indicates whether the coefficient corresponding to the j-th measure has been stored.
- The i indicates the coefficient number. The space to store Bit and i is denoted by H.
- The stored list of coefficient values V, where the r-th item in the list corresponds to the i-th coefficient of measure j if $Bit(j) = 1$ and $\sum_{j'=1}^{j} Bit(j') = r$. Each of the stored coefficients are assumed to take space S. We denote the i-th coefficient of measure j by w_{ij}.

*Without loss of generality, N is a power of 2.

Extended wavelet provides a flexible storage method and bridges the gap between the two extreme approaches of individual or multi-dimensional decompositions.

Since the most common objective used as an error measure is minimizing L_2 norm of approximation, a natural extension for datasets with multiple measures is how to minimize the weighted sum of the squared error for all measures. If the error (the difference between the original data and the wavelet reconstruction) for i-th data item in j-th measure is denoted by e_{ij} then the optimization problem is the following:

Problem 1 *Given a set of NM wavelet coefficients $\{w_{ij}\}$ points in D-dimensional dataset with M measures, a storage constraint B, and a set of weights W, select the extended wavelet coefficients to be stored in order to minimize the weighted sum* $\sum_{i=1}^{N}\sum_{j=1}^{M} W_j \cdot e_{ij}^2$

Following [16], it is shown in [3] that the above is equivalent to the following:

Problem 2 *Given the NM wavelet coefficients $\{w_{ij}\}$, a storage constraint B, and a set of weights W, select the extended wavelet coefficients to be stored in order to minimize the weighted sum*

$$\sum_{i,j:w_{ij}\ \textit{is}\ \textbf{not}\ \textit{stored}} W_j \cdot w_{ij}^2$$

Observe that Problem 2 is equivalent the "maximizing the benefit" where the benefit is defined as:

$$\sum_{i,j:w_{ij}\ \text{is stored}} W_j \cdot w_{ij}^2$$

But a good approximation for one may not mean a good approximation for the other. A more useful problem in the DSS scenario is:

Problem 3 *Given a N data points in D-dimensions with M measures, a storage constraint B, and a set of weights W, compute and select the extended wavelet coefficients to be stored in order to minimize the error*

$$\sum_{i=1}^{n}\sum_{j=1}^{M} W_j \cdot e_{ij}^2 = \sum_{i,j:w_{ij}\ \textit{is not stored}} W_j w_{ij}^2$$

in a single pass over the data.

3.3 DynL2: An Optimal Algorithm for Problem 2

An optimal dynamic programming algorithm, *DynL2*, in Figure 1 was proposed in [3] to solve Problem 2.

```
Procedure DynL2(InCoeffs, B, W)
begin
1.  for u :=1 to N*M do
2.      for v :=0 to S+H-1 do {
3.          /* Nothing can be stored! */
4.          OPT [u,v].ben := FORCE[u,v].ben := 0
5.          OPT[u,v].choice := FORCE[u,v].choice := 1
6.      }
7.  for v :=S+H to B do {
8.      OPT[1,v].ben := FORCE[1,v].ben := W[1]*$w^2$[1,1]
9.      OPT[1,v].choice := FORCE[1,v].choice := 3
10. }
11. for u :=2 to N*M do {
12.     x := 1 + (u-1) div M (Coefficient index)
13.     y := 1 + (u-1) mod M (Measure index)
14.     for j :=S+H to B do {
15.         a := OPT[u-1,v].ben
16.         b := OPT[u-1,v-S-H].ben+W[y]*$w^2$[x,y]
17.         c := FORCE[u-1,v-S].ben+W[y]*$w^2$[x,y]
18.         d := FORCE[u-1,v].ben
19.         if y > 1 {
20.             OPT[u,v].ben := max(a, b, c)
21.             FORCE[u,v].ben := max(d, b, c)
22.         }
23.         else {
24.             OPT[u,v].ben := max(a, b)
25.             FORCE[u,v].ben := b
26.         }
27.         Set OPT[u,v].choice and FORCE[i,j].choice
                to the value 2, 3 or 4, appropriately.
28.             /* If ben = a or ben = d, set choice to 2 */
29.             /* If ben = b, set choice to 3 */
30.             /* If ben = c, set choice to 4 */
31.     }
32. }
33. Reconstruct optimum solution by doing a reverse
        traversal starting from the entry OPT[NM,B], and
        moving based on the choice field of the current entry.
34. return OPT[NM,B].ben (Maximum benefit)
end
```

Figure 1: The DynL2 Algorithm in [3]

Suppose the coefficients are ordered in the canonical order, i.e., w_{ij} is in position $u = i * M + j$ in an array. Let $\text{OPT}[u, b]$ the optimal solution with at most b units of space where no coefficient which occurs later than u in the order is being used. We also let $\text{FORCE}[u, b]$ be defined as the same as $\text{OPT}[u, b]$ except that the solution of the former should store some $w_{ij'}$ with $j' \leq j$ where $u = i * M + j$, i.e., at least for one of the measures the i-th coefficient is stored.

The intuition behind the algorithm is that from $u-1$ to u the algorithm tries to figure out the extra space it would have to use and the benefit derived. Thus four choices, denoted by the variables a, b, c, d in the pseudocode arise depending on w_{ij} being added or not to $\text{OPT}[u, b]$ or $\text{FORCE}[u, b]$.

The algorithm stores OPT and FORCE, and thus takes $O(NMB)$ space. Since each entry of both OPT and FORCE is evaluated in $O(1)$ time, the time complexity of *DynL2* is $O(NMB)$.

In [3], a 2-approximation algorithm with $O(NM^2 \log(NM))$ time and $O(NM)$ space was also proposed for Problem 2.

4 Improved Optimum Algorithms

4.1 OptWaveI: A Simple Algorithm

We first present the definitions that will be used to describe our improved optimum algorithms.

Definition 4.1 Let $\text{NEWOPT}[i, b]$ denote the minimum error (maximum benefit) of using at most b space and no coefficient of index larger than i (irrespective of measure). To aid the presentation, we also define:

- Let $\text{ALL}[i] = \sum_{j=1}^{M} W_j w_{ij}^2$.
- Let $\text{BOTTOM}[i, j]$ be the sum of the j *smallest* items of the set of numbers $W_1 w_{i1}^2, W_2 w_{i2}^2, \ldots, W_M w_{iM}^2$.
- Let $\text{TOP}[i, j]$ be the sum of the j *largest* items in the set of numbers $W_1 w_{i1}^2, \ldots, W_M w_{iM}^2$. Naturally,

$$\text{ALL}[i] = \text{TOP}[i, j] + \text{BOTTOM}[i, M - j]$$

Lemma 4.2 *If we are storing the i-th coefficient for a subset $C \subseteq \{1, \ldots, M\}$ of the measures, the best solution is to store the $|C|$ coefficients which have the largest contribution, i.e., $W_j w_{ij}^2$. Thus $\text{TOP}[i, p]$ gives the best benefit over all subsets C with $|C| = p$. The minimum error of choosing to store $|C|$ coefficients is therefore $\text{BOTTOM}[i, p]$.*

Intuition: The above lemma allows us to decouple the choices of subsets C' and C for the coefficients with indexes $i - 1$ and i respectively. Thus, the choice reduces to how much space we allocate to all coefficients of index $i - 1$ versus all coefficients of index i.

As a result, we have the naive optimum algorithm *OptWaveI* shown in Figure 2. This algorithm, as we will see later in the experiments, already performs much better than the *DynL2*. *Observe that, without any further improvements (which we will make later), the space required is $O(NM + NB)$.* In the $O(NB)$ space, we need to maintain the choice of p for each $\text{NEWOPT}[i, b]$. Given p and i, the subset of the measures for which we store the coefficient is automatically the coefficients being in the sorted order of the contribution $W_j w_{ij}^2$. We can prove the following:

Theorem 4.3 *The algorithm OptWaveI evaluates $\text{NEWOPT}[i, b]$ correctly for all i and b.*

The running time of the algorithm is $O(NMB + NM \log M)$. A sorting with $O(M \log M)$ time allows us to compute all of $\text{BOTTOM}[i, M - p]$.

```
Procedure OptWaveI()
begin
1.  SUMALL:= 0
2.  for b :=1 to B do
3.    NEWOPT[0, b] := 0
4.  for i :=1 to N do {
5.    for j :=1 to M do
6.       Compute BOTTOM[i,j]
7.    SUMALL:= SUMALL+ BOTTOM[i,M]
8.    for b :=1 to B do {
9.       if b < H + S (cannot store anything)
10.         NEWOPT[$i, b$] := SUMALL
11.      else {
12.         NEWOPT[$i, b$] := NEWOPT[$i$-1, $b$] + BOTTOM[$i, M$]
13.         for p :=1 to M do
14.            if $b$-$H$-$S * p \geq 0$
15.               NEWOPT[$i, b$] := min(NEWOPT[$i, b$],
16.                  NEWOPT[$i$-1, $b$-$H$-$S * p$]+
17.                     BOTTOM[$i, M$-$p$])
18.      }
19.   }
20. }
end
```

Figure 2: The OptWaveI

The rest of the algorithm take $O(NMB)$ time.

Improved Space Requirement: Note that, to evaluate NEWOPT$[i, j]$ for $1 \leq j \leq B$, we *only* need the array of NEWOPT$[i-1, j]$ for $1 \leq j \leq B$. Since for each of NEWOPT$[i, j]$ we need $O(B)$ space to store the information regarding chosen coefficients so far, the space required is $O(B^2)$ only. If we try to *copy* the solutions, the running time will increase due to $O(B)$ time for a single copy operation. We instead use a pointer to speed up coping operation of chosen coefficients so far. Since multiple entries of NEWOPT$[i, b]$ table may be extended from the same NEWOPT$[i', b']$ with $i' < i$, the selected coefficients for the latter may be shared. Thus, when we deallocate space for NEWOPT$[i', b']$, we have to make sure not to delete its selected coefficients if they are still pointed by other NEWOPT$[i, b]$. To handle this, a counter of how many different NEWOPT$[i, b]$ are using NEWOPT$[i', b']$ is maintained in each NEWOPT$[i', b']$. We delete all information stored for NEWOPT$[i', b']$ if it is not used by any NEWOPT$[i, b]$ for $i > i'$. In this process, we will decrease the counters of earlier coefficients selected and may need to delete them. Notice that the amortized cost of all the delete is $O(NB)$ since the total number of deletions can be at most the total number of NEWOPT$[i, b]$ being computed which is $O(NB)$. Thus, we arrive at a $O(NMB)$ time and $O(B^2)$ space algorithm.

Notice that $B \ll NM$, since otherwise there is no benefit in space by storing a synopsis, and thus we improve over *DynL2* in terms of the space bounds.

4.2 OptWaveII: A Better Optimum Algorithm

Definition 4.4 Define $L = \lfloor \frac{B}{S+\frac{H}{M}} \rfloor$. Observe that the optimum algorithm can store at most L coefficients since each coefficient w_{ij} takes up *at least* $S + \frac{H}{M}$ space on the average. The extra space H has to be shared by the coefficients – and the best case scenario is when all coefficients corresponding to an index i have been chosen.

Definition 4.5 Suppose we ordered the coordinates of i for $1 \leq i \leq N$ in a non-increasing order of TOP$[i, p]$ (the maximum benefit of storing p coefficients) into $i_1^p, i_2^p, \ldots, i_n^p$. That is, TOP$[i_j^p, p] \geq$ TOP$[i_{j'}^p, p]$ if $1 \leq j \leq j' \leq N$. We let BEST$[p] = \{i_j^p | j \leq L\}$. Notice that BEST$[p]$ need not be disjoint as p varies.

Ideally, we would like to say that the subsets of coefficients with index i stored by optimum must belong to $\bigcup_{p=1}^{M}$ BEST$[p]$ for some p, but it may be that there are several solutions with equal error. But in that case one of them will always select coordinates in $\bigcup_{p=1}^{M}$ BEST$[p]$ only and this is captured in the following theorem:

Theorem 4.6 *There exists an optimum solution which only stores the coefficients of the coordinates from* $\bigcup_{p=1}^{M}$ BEST$[p]$.

Intuition: The motivation behind the above theorem is to introduce a *filtering step* where we try to recognize the more useful coordinates. The idea is that given two subsets of coefficients C', all of which correspond to index i' and C which corresponds to i, if we have $|C| = |C'|$, it is better to choose the subset which has the greater contribution to the benefit. Now, the proof of the theorem is more involved, since the optimum may store coefficients with indices i and i' but $|C| \neq |C'|$.

Using the above theorem, we develop a significantly better optimal algorithm *OptWaveII*. The *OptWaveII* first invokes the *OptWavePreProcess* shown in Figure 3 to compute $\bigcup_{p=1}^{M}$ BEST$[p]$. It then runs the same algorithm as *OptWaveI* in Figure 2 except that the for-loop in line (4) is replaced as below:

for each i $\in \bigcup_{p'=1}^{M}$ BEST$[p']$ **do** {

Since we consider only $i \in \bigcup_{p'=1}^{M}$ BEST$[p']$, we add

NEWOPT$[n, B]$:= NEWOPT$[n, B]$+TOTSUM−SUMALL

where $TOTSUM = \sum_{i=1}^{N}$ BOTTOM$[i, M]$ at end of *OptWaveII* to compute correct value of NEWOPT$[n, B]$. $TOTSUM$ is computed by *OptWavePreProcess*.

We maintain M min-heaps of size L for BEST$[p]$. The min-heaps are implemented using M arrays of

```
Procedure OptWavePreProcess()
begin
1.  for i :=1 to N do {
2.      Sort the coefficients of $i$-th coordinate
3.      for p :=1 to M do {
4.          Compute Top[$i,p$] using Top[$i,p-1$]
5.          if sizeof(Best[$p$]) $< L$ {
6.              Insert $i$ with key Top[$i,p$] to Best[$p$]
7.              count[i] := count[i]+1;
8.          }
9.          else {
10.             if findmin(Best[$p$]).key $<$ Top[$i,p$] {
11.                 Let $m$ be the minimum element in Best[$p$]
12.                 count[m] := count[m]-1
13.                 if count[m] $= 0$
14.                     Eliminate $m$ from hash table.
15.                 Insert $i$ with key Top[$i,p$] to Best[$p$]
16.                 count[i] := count[i]+1;
17.             }
18.         }
19.     }
20.     If count[i] $> 0$
21.         Store $w_{i1},\dots,w_{iM}$ in the hash table with count[i].
22. }
end
```

Figure 3: The OptWavePreProcess()

size L. The key of element i (for heap p) is $\textsc{Top}[i,p]$. the operations $findmin()$ costs $O(1)$ time and insert/delete costs $O(\log L)$ time. We also maintain a hash table which for the key i will store $w_{i1},\dots,w_{iM}$ and $count_i$, the number of heaps i is currently present in.

We first compute $\bigcup_{p=1}^{M}\textsc{Best}[p]$ using *OptWavePreProcess* as in Figure 3. For each i, we compute $\textsc{Top}[i,p]$ for all p in $O(MlogM)$ time, and the cost of at most M insertions is $O(M\log L)$. Thus in this phase we take $O(NM(\log M+\log L))$ time.

For the second phase, the outer for loop will be executed at most ML times, which is an upper bound on $\bigcup_{p'=1}^{M}\textsc{Best}[p']$. The inner loops take time MB.

Thus the time for the second phase is M^2LB, and in total we take $O(NM(\log M+\log L)+\frac{M^2B^2}{S+\frac{H}{M}})$[†]. The total space bound is $O(MB+B^2)$ where the two terms correspond to the two different phases.

5 An Approximation Algorithm

A natural question arises in this context – if the $O(NMB)$ worst case algorithm can be speeded up. In [3], the authors proposed a greedy 2-approximation of the "benefit" which runs in time $O(NM^2\log(NM))$. Recall that the benefit of choosing a subset C of coefficients in $\{1,\dots,M\}$ that is i-th coordinate is defined as $\sum_{j\in C}W_j w_{ij}^2$. The benefit quantifies the error "saved" by choosing the coefficients in C.

Our approximation at the surface is similar to the 2-approximation algorithm in that it uses benefit to space ratios. However our algorithm will take a different road. Our algorithm will *try to reduce the space subject to maintaining the same quality of solution as the optimum.*

It may appear that we are comparing apples and oranges, but these algorithms are known as *pseudo approximation algorithms* – where we find an approximate solution, but relax some constraints (in this case, space bound) slightly (by $MS+H$). But because in this case we are able to prove that the quality of our solution is at least as good as the optimum (restricted to B), the result can be viewed as approximating the space while keeping the quality fixed. Note, that we assume no knowledge of the quality of the solution beforehand.

Technically our algorithm in this section differs from the 2-approximation in [3] in the following aspects:

1. We use our previous idea of not considering all coefficients.
2. We are more cautious in selecting the subsets for which we compute benefit.
3. Unlike previous work, *our definition of the benefit-space ratio will not be uniform over all subsets.*
4. Our algorithm operates in $O(B)$ space.
5. Unlike previous algorithm, we do not consider inserting coefficients corresponding to the same index i more than once (i.e., if they are ejected).

Before describing the algorithm, we make a few important observations.

Lemma 5.1 *Let $S,H>0$. Given a sequence $X=\{x_1,x_2,\dots,x_M\}$ of non-negative numbers in non-increasing order, let $\hat{p}$ be the value of p that maximizes $\max_{p=1,\dots,M}\frac{\sum_{j=1}^{p}x_j}{S*p+H}$. Among every subset $Y\subseteq X=\{x_1,x_2,\dots,x_M\}$, the subset $Y=\{x_1,x_2,\dots,x_{\hat{p}}\}$ maximizes $\frac{\sum_{x_j\in Y}x_j}{S|Y|+H}$.*

Definition 5.2 Let us define $\textsc{Ratio}[i].wt$ as follows:

$$\textsc{Ratio}[i].wt=\max_{p=1,\dots,M}\textsc{Top}[i,p]/(S*p+H)$$

Let us also define $\textsc{Ratio}[i].p$ as the smallest value p for which the maximum ratio is obtained.

Lemma 5.3 *Suppose $w_{ij_1},w_{ij_2},\dots,w_{ij_M}$ are the coefficients with index i such that $W_{j_u}w_{ij_u}^2\ge W_{j_v}w_{ij_v}^2$ whenever $u\le v$. For all $u>\textsc{Ratio}[i].p$, we have $\textsc{Ratio}[i].wt\ge W_{j_u}w_{ij_u}^2/S$. Furthermore, for all $1\le u\le\textsc{Ratio}[i].p$, we have $W_{j_u}w_{ij_u}^2/S\ge\textsc{Ratio}[i].wt$.*

[†]We can assume $ML\le N$ otherwise we can use a bound of N on the set, and thus the total time is at most $O(NM(\log M+\log L)+NMB)$.

The algorithm: The algorithm *ApproxWave* is given in Figure 4. We will maintain a min-heap of size *at most* $B + M * S + H$. The heap will shrink and grow — but the total space required to store all the coefficients associated with the heap will not exceed $O(B)$. The min-heap will be implemented using an array as before. The elements in the heap will be tuples $(i, p, flag)$ where $flag = 1$ will indicate all the coefficients $w_{ij_1}, \ldots, w_{ij_p}$ are (which define $\textsc{Top}[i, p]$) are chosen. $flag = 0$ would mean only the coefficient w_{ij_p} is chosen. We can implement $findmin()$ in $O(1)$ time and insert/delete in $O(\log B)$ time. We will maintain a hash table to store the wavelet coefficients for every tuple $(i, p, flag)$ in min-heap. It is straightforward to observe that the time complexity of the algorithm is $O(NM(\log M + \log B))$ since the size of the heap is bound by B.

Due to the lack of space we cannot prove any of the claims made, but we indicate an outline of the proof. In a very high level, we will show that if we consider the elements in our solution which are not in the optimum, they must have a better benefit to space ratio than the elements in the optimum solution which we have not included. We show the separation by bounding the former from below, and the latter from above, by the key value of the minimum element in the heap.

Observation 5.4 *If the key value of the minimum element in the heap is λ, then any item with key value at least λ must be present in the heap.*

Lemma 5.5 *In the above algorithm, once we reach the condition of $Free < 0$, we have $-(M * S + H) \leq Free < 0$ in all subsequent steps.*

The above lemma can be proved by induction and the proof is omitted due to lack of space. But the lemma implies that we exceed the space bound by *at most* M coefficients. Thus we can bound the size of the heap to be $O(B)$.

Lemma 5.6 *Let* $\textsc{Ones} = \{i | (i, p, 1) \text{ in heap}\}$ *and* $\textsc{Zeros} = \{i | (i, p, 0) \text{ in heap}\}$*, then the condition of* $\textsc{Ones} \supseteq \textsc{Zeros}$ *always holds. That is, if $(i, u, 0)$ is present in the heap at any point, so must be* $(i, \textsc{Ratio}[i].p, 1)$.

The next set of lemmas will be used as the critical part of the proof for the main theorem of this section that states that the benefit of our solution is *no less* than that of the optimal solution.

Lemma 5.7 *Further let $|\textsc{Zeros}|$ be the number of elements of the form $(i, p, 0)$ in the heap, then*

$$B - Free = \sum_{i \in \textsc{Ones}} (S * \textsc{Ratio}[i].p + H) + S * |\textsc{Zeros}|$$

```
Procedure ApproxWave()
begin
1.  $Free := B$
2.  for i :=1 to N do {
3.    Compute $\textsc{Ratio}[i]$
4.    if ($Free \geq 0$) {
5.      Insert $(i, \textsc{Ratio}[i].p, 1)$ in heap
              with $key = \textsc{Ratio}[i].wt$
6.      $Free \leftarrow Free - S \cdot \textsc{Ratio}[i].p - H$
7.    } else if ($findmin().key < \textsc{Ratio}[i].wt$) {
8.      while ( ($findmin().key < \textsc{Ratio}[i].wt$)
                  and ($Free < 0$) ) {
9.        Suppose the min element was $(i', m_p, flag)$.
10.       if ($flag = 1$) {
11.         if ($Free + S(m_p - \textsc{Ratio}[i].p) \geq 0$) {
12.           /* leave while loop */
13.           break
14.         }
15.         $Free \leftarrow Free + S \cdot m_p + H$
16.       }
17.       else
18.         $Free \leftarrow Free + S$
19.       Remove the min element in the heap.
20.     }
21.     Insert $(i, \textsc{Ratio}[i].p, 1)$ in heap
              with $key = \textsc{Ratio}[i].wt$
22.     $Free \leftarrow Free - S \cdot \textsc{Ratio}[i].p - H$
23.   }
24.   for $u := \textsc{Ratio}[i].p + 1$ to $M$ {
25.     if ($Free \geq 0$) {
26.       Insert $(i, u, 0)$ with $key = \frac{W_{j_u} w_{ij_u}^2}{S}$
27.       $Free \leftarrow Free - S$
28.     } else if ($findmin().key \geq \frac{W_{j_u} w_{ij_u}^2}{S}$) break
29.     else {
30.       Suppose the min element was $(i', m_p, flag)$
31.       if $flag = 0$ {
32.         Remove the min element in the heap.
33.         Set $Free \leftarrow Free + S$
34.       } else if ($Free + S(m_p - 1) + H < 0$) {
35.         Remove the min element in the heap.
36.         Set $Free \leftarrow Free + S \cdot m_p + H$
37.       }
38.       Insert $(i, u, 0)$ with $key = \frac{W_{j_u} w_{ij_u}^2}{S}$
39.       $Free \leftarrow Free - S$
40.     }
41.   }
42. }
43. Include all the coefficients stored in the heap.
end
```

Figure 4: The ApproxWave

Corollary 5.8 *The space taken by the solution is at most $B+M*S+H$, i.e., we exceed the budgeted space at most M extra coefficients.*

Before proving the quality of our solution we will have to define some notation. Suppose that λ^* is the ratio of the minimum element after all coefficients have been processed. Thus for any item in the heap and not considered by our algorithm, the key could have been at most λ^*. Suppose the optimum solution uses a set of coefficients corresponding to $\mathcal{S}_{opt} \subseteq [1,\ldots,n]$ and we choose a set of coefficients corresponding to $\mathcal{S}_{sol} \subseteq [1,\ldots,n]$. For each $i \in \mathcal{S}_{opt}$ suppose that the optimal chooses $o(i)$ coefficients (it will choose the best ones) and for $i \in \mathcal{S}_{sol}$ suppose that we chose $p(i)$ coefficients.

Lemma 5.9 *Recall that by $w_{ij_1}, w_{ij_2}, \ldots, w_{ij_M}$ we refer to the coefficients of i sorted in non-increasing order of $W_j w_{ij}^2$. The following four conditions hold.*

(P1): *If $i \in \mathcal{S}_{opt}$ and $i \notin \mathcal{S}_{sol}$ then*

$$\sum_{u=1}^{o(i)} W_{j_u} w_{ij_u}^2 \leq \lambda^*(So(i)+H)$$

(P2): *If $i \notin \mathcal{S}_{opt}$ and $i \in \mathcal{S}_{sol}$ then*

$$\sum_{u=1}^{p(i)} W_{j_u} w_{ij_u}^2 \geq \lambda^*(Sp(i)+H)$$

(P3): *If i is in both $\mathcal{S}_{opt}, \mathcal{S}_{sol}$ and $p(i) < o(i)$ then for all $p(i) < u \leq o(i)$ we have $W_{j_u} w_{ij_u}^2 \leq \lambda^* S$.*

(P4): *If i is in both $\mathcal{S}_{opt}, \mathcal{S}_{sol}$ and $p(i) > o(i)$ then for all $o(i) < u \leq p(i)$ we have $W_{j_u} w_{ij_u}^2 \geq \lambda^* S$.*

Observe that the above lemma shows that for all the coefficients not stored by our solution the benefit is at most λ^* times the space taken by those coefficients[‡]. Likewise, the lemma says that for the coefficients we stored and the optimal solution did not store, the benefit is at least λ^* times the space[§].

Theorem 5.10 *The benefit of our solution is no less than the benefit of the optimum solution (which takes at most B space).*

6 Adapting small space extended wavelet algorithms to streams

Formally, a data stream computation is a space bounded algorithm, where the space is sub-linear in the input. Any input items are accessed sequentially and any item not explicitly stored cannot be accessed again in the same pass. A comprehensive discussion of streaming is beyond the current scope. For our current problem, we need a small space algorithm that that makes one-pass over the data and generates the extended wavelet synopsis. In this scenario we see the data items $x_{i1}, x_{i2}, \ldots, x_{iM}$, and then we proceed to see the data items $x_{(i+1)1}, x_{(i+1)2}, \ldots, x_{(i+1)M}$ and likewise to $i+2$, etc.

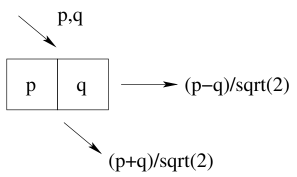


Figure 5: The algorithm A_ℓ

We will draw upon [6] for computing wavelet decompositions. Suppose we have a simple algorithm illustrated in Figure 5 which takes two numbers p, q and outputs the two wavelet coefficients $(p-q)/\sqrt{2}$ and $(p+q)/\sqrt{2}$. Now suppose we made this algorithm, say A_1, repeatedly pick pairs of items from the stream and output the two associated coefficients, but with a difference. Suppose the coefficients corresponding to the difference, e.g. $(p-q)/\sqrt{2}$ are output directly and the sums are handed to a different algorithm A_2 which does exactly the same thing.

As an example consider a single dimension with 8 data values: $8, 7, 6, 5, 4, 3, 2, 1$. A_2 will receive $(8+7)/\sqrt{2}$,$(6+5)/\sqrt{2}$, $(4+3)/\sqrt{2}$ and $(2+1)/\sqrt{2}$. From the first two numbers, A_2 will output $\left(\frac{8+7}{\sqrt{2}} - \frac{6+5}{\sqrt{2}}\right)/\sqrt{2}$ immediately and pass the corresponding sum to A_3. The number which is being output is actually $\frac{8+7-6-5}{2}$ and is the coefficient of the wavelet vector whose support is the first half of the input! Thus A_2 will output the coefficients corresponding to wavelet vectors with support 4. A_3 will receive $\frac{8+7+6+5}{2}$, $\frac{4+3+2+1}{2}$ and output the wavelet coefficient of the wavelet vector with support 8. A_3 will pass the coefficient corresponding to the sum of all the coefficients to A_4. On seeing the end of input A_4 will output this single number.

Observe that each A_ℓ will need only space $O(1)$ since it stores the first value it receives; on seeing a second value it outputs the difference (divided by $\sqrt{2}$), sends the other value to $A_{\ell+1}$ and forgets both the numbers. A_ℓ keeps repeating this process for every pair of answers it sees. When the end-of-input signal is seen it passes the signal to $A_{\ell+1}$.

If one $A_{\ell'}$ receives only one input and then the end of input of the original data, it just output the stored input. *Notice that the division by $\sqrt{2}$ happens at each stage recursively* and the last output is the sum of all the numbers divided by $\sqrt{N}$, or the average of all numbers multiplied by $\sqrt{N}$. Several researchers compute the coefficients differently, by computing the

[‡]Including the space required by the bitmaps if we did not store any coefficient for i, whereas optimum stored at least one coefficient.

[§]Once again including the space required by bitmaps.

recursive averages and multiplying by *normalization constants*. The processes are equivalent, and the process described here adapts to streams readily. Due to lack of space, we relegate further details to [7]. See also [16, 10, 5] for further details on wavelets. Summarizing all the above, we get:

Fact 6.1 *Using $O(\log n)$ space we can compute the wavelet decomposition in a single pass. The order in which the coefficients will be output will correspond to a post-order traversal of the complete binary tree defined on the input indices $\{i\}$. Note that n need not be known in advance for the process.*¶

An immediate corollary would be

Corollary 6.2 *Any one-pass algorithm for Problem 1 can be used to get a one-pass algorithm for Problem 3 with an additional space of $O(M \log N)$*

7 Experimental result

We conducted experiments on real-life as well as synthetic data sets. We implemented algorithms DynL2 and GreedyL2 proposed in [3] and compared them against the algorithms proposed in this paper. All experiments reported in this section were performed on Pentium-4 2.8 GHz machine with 512 MB of main memory, running Linux operating system. All the methods are implemented using GCC compiler of Version 2.95.3

Experimental results confirm that our optimal algorithms, optWaveI and optWaveII, are faster than DynL2. The ApproxWave is at least as accurate as GreedyL2 but much faster.

7.1 Synthetic Data Sets

For our synthetic data sets, we implemented a data generator suggested in [3, 2, 17]. The input parameters to the data generator along with their description and default values are as illustrated in Table 1. The generator begins by randomly selecting *n_regions* rectangular regions in a N-dimensional array. The volume of any dense region is randomly chosen between V_{min} and V_{max}. Sum_i is the summation of the values for all the cells contained in *n_regions* dense regions for measure *i*. Through the use of Zipf function with parameter Z, Sum_i is partitioned across the *n_regions* rectangular regions. Within each region, the values are distributed by using one of the four distributions in Table 2 with skew parameter between z_{max_i} and z_{min_i}. Note that we use the notion of the *Altered-X*‖ distribution to help create pairs of measures with

Parameter	Description	Default Value
N	Number of dimensions	2
M	Number of measures	30
$Card_i$	Cardinality of dimension i	512
n_regions	Number of dense regions	10
V_{min} V_{max}	Minimum and maximum volume of regions	4900 4900
Z	Skew across regions	0.5
z_{min_i}, z_{max_i}	Minimum and maximum skew within region i	1, 1
Sum_i	Sum of values for measure i	1,000,000
spCount	Fraction of populated cells in sparse areas	0.05
$spSum_i$	Sum of values of populated cells in sparse area i	0.05

Table 1: Data Generator Input Parameters

Distribution	Description
Center	Cells with smaller L1-distance from center have larger values
Reverse	Cells with smaller L1-distance from center have smaller values
Middle	Consider a hyper-rectangle centered at the region's center, and having for each dimension, half the length of the corresponding region length. Cells with smaller L1-distance from this hyper-rectangle have larger values
Altered-X	This measure follows the same distribution as X distribution, but its values are randomly altered by up to 50%

Table 2: Data Generator Value Distributions

similar, but not identical , data distribution. The generator also populates nonzero cells outside the dense regions. The fraction of such cells is defined by spCount parameter and the total sum of the values of these cells is denoted by the $spSum_i$ parameter.

In each experiment, the parameters of the data generator were set to the default values, unless specified otherwise.

7.2 Algorithms

We conducted a comprehensive performance evaluation of the various schemes. Specifically, we show the performance figures of the following schemes:

- **DynL2:** This is our implementation of the dynamic programming algorithm DynL2 in [3]. It is an improved version of [3] using less space.
- **GreedyL2:** It is the greedy approximate algorithm in [3] which uses a heap instead of an AVL-

¶As an aside, each of the A_ℓ form a transducer and the above describes a wavelet coefficient computation using transducers.

‖X can be either one of the Center, Middle or Reverse distributions.

tree.

- **OptWaveI and OptWaveII:** It represents our optimal algorithms presented in Section 4.1 and Section 4.2 respectively.

- **ApproxWave:** It represents approximate algorithm described in Section 5.

7.3 Synthetic Data Sets

7.3.1 Behavior of All Algorithms

Figure 6 shows the results of optimum and approximate algorithms with varying the number of measures M from 2 to 6. We also varied the space constraint B from 1K to 10K bytes. The initial two measures are the ones with distributions Center and Middle, and the measures that are later added are: Reverse, Altered-Center, Altered-Reverse, Altered-Middle. We fixed $Card_i$ to 1024. Other parameters were set to default values. The execution times are shown using a log scale.

Both OptWaveI and OptWaveII are much faster than traditional optimum algorithm DynL2. The execution time for OptWaveII is again at least two orders of magnitude faster than DynL2. Our ApproxWave is also much faster than traditional GreedyL2. Furthermore, when the value of B is small (e.g. B = 1K bytes), the optimum algorithm OptWaveII is actually even faster than the approximation algorithm GreedyL2. As we increase B, GreedyL2 starts to win optWaveII with larger M values and finally becomes better regardless of the value of M (e.g. B = 4K bytes). In Figure 6, we present two graphs only for B = 1K and B = 4K bytes.

7.3.2 Behavior of Approximate Algorithms

To see the behavior of approximate algorithms, we tested synthetic data sets with both GreedyL2 and ApproxWave.

Storage Space: In Figure 7-(a) and Figure 7-(b), we present the execution time and average weighted sum squared error for approximation algorithms as the storage space is varied from 1K to 1M bytes. We set the number of measures M to 30 and the skew parameter of the data distributions within each region was set to 1.0. The errors of both GreedyL2 and ApproxWave are very close, but the error of ApproxWave is slightly better than that of GreedyL2. Furthermore, the speed of ApproxWave is much faster than that of GreedyL2.

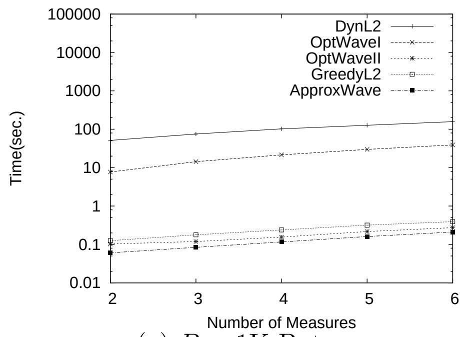
Skew within Region: We modified the zipfian parameter controlling the skew of the measure's data distributions within each region from 0.5 to 4 and we set B to 100K bytes. The Figure 7-(c) and Figure 7-(d) presents the obtained results for the average weighted sum of squared error and execution time respectively. When the skew is small, the values of the measures are very similar and most of *detail wavelet coefficients* become zero. Thus, the error becomes very small. As the skew increases more, the measure values become different and the detail wavelet coefficients start to have non-zero values. It results in larger errors. However, as the skew parameter increases even larger (after 1.5) values, the large coefficients are restricted to a smaller area for each distribution. This produces the reduction of the weighted sum of squared error of the results, as the number of significantly influencing coefficients becomes smaller. On the other hand, the probability that coefficients from multiple measures become important simultaneously becomes smaller. This explains the shape of the average weighted sum of squared error graph in Figure 7-(d). Even though the graphs of the errors of both algorithms are close, the error of ApproxWave is slightly better that that of GreedyL2. However, regardless of skew changes, the execution times do not show any big change. It is expected since the time complexities of both algorithms are not dependent on skew in data distribution. ApproxWave is again much faster than that of GreedyL2.

Number of Measures: We present the execution time with varying the number of measures M from 5 to 50 in Figure 8. The distributions of measures are cyclically repeated in order of Center, Middle, Reverse, Altered-Center, Altered-Reverse and Altered-Middle. We set B to 100K bytes. The experiments validate our earlier analysis that the time complexities of *GreedyL2* and *ApproxWave* are $O(NM^2 \log(NM))$ and $O(NM(\log M + \log B))$ respectively. Furthermore, since the space requirement of GreedyL2 is much larger than that of ApproxWave, as the number of measures increases, GreedyL2 starts to suffer from some memory problem and thus its running time degrades significantly. The execution time of GreedyL2 is slower than that of ApproxWave up to in an order of magnitude.

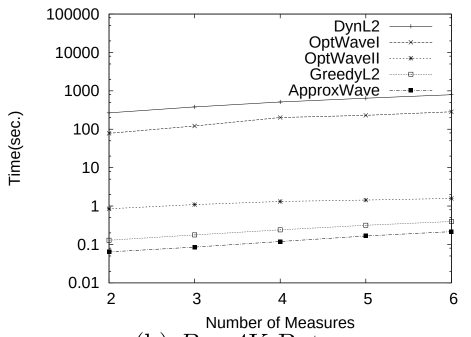
7.4 Real-life Data Set

For our real-life data experiments, we used the Pacific Northwest weather measurement data from the state of Washington ** The coordinates of the dataset are day and time. We selected one year dataset and used the solar irradiance, wind speed, wind peak, air temperature, dewpoint temperature and relative humidity for the station in the university of Washington. We set a weight value of 3 to the first two measure, a weight value of 2 to the next two measures and a weight value of 1 to the remaining measures. The number of tuples in the dataset is 525600. We computed the average weighted sum of squared absolute errors.

**It is available from http://www-k12.atmos.washington.edu/k12/grayskies/nw_weather.html.

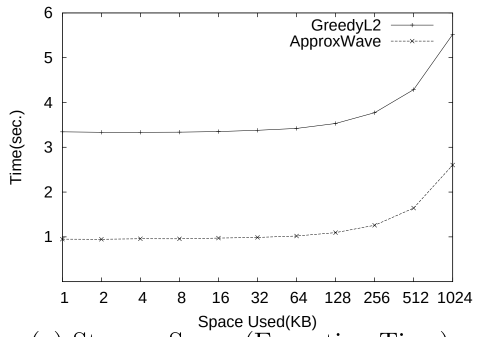


(a) $B = 1$K Bytes

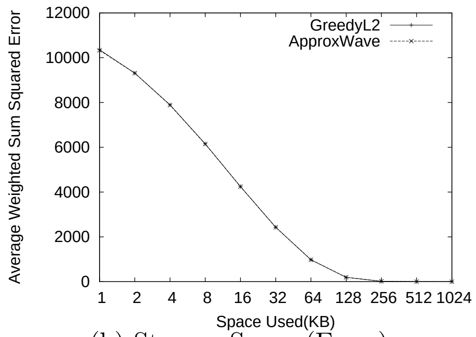


(b) $B = 4$K Bytes

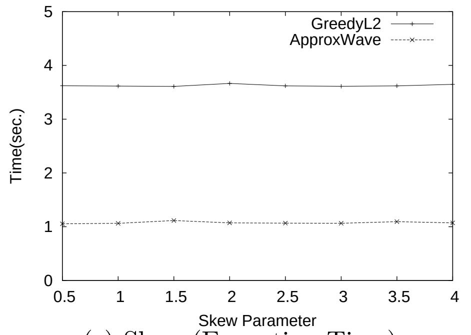
Figure 6: Execution Time with Varying M



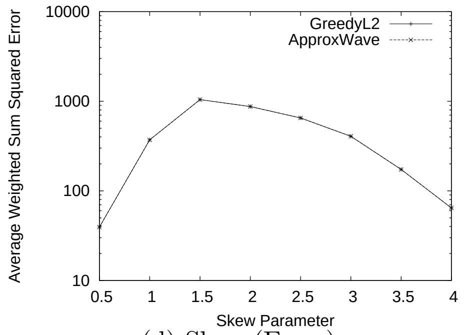
(a) Storage Space (Execution Time)



(b) Storage Space (Error)



(c) Skew (Execution Time)



(d) Skew (Error)

Figure 7: Approximate Algorithms

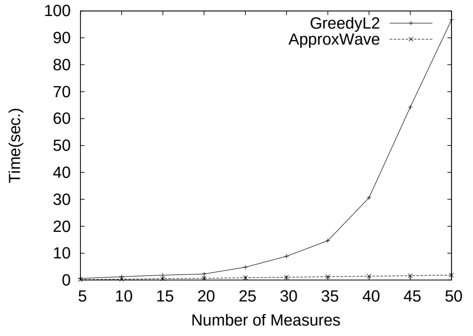


Figure 8: Execution Time with Varying M by Approximate Algorithms

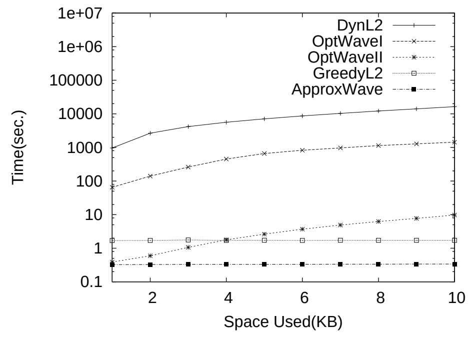


Figure 9: Execution Time for Real-life Dataset

Figure 9 presents the result for execution time, as the storage bound B was varied from 1K to 10K bytes. The execution times of the three optimum algorithms grow linearly in increasing B as the time complexities of the algorithms illustrate. The value of B affects the execution times of the approximation algorithms very little as expected. Our results show that the execution time for OptWaveII is at least two orders of magnitude better than DynL2. The execution time for ApproxWave is again much faster than GreedyL2 too. We believe that our results clearly demonstrate the significant performance gains by both OptWave and ApproxWave algorithms compared to traditional DynL2 and GreedyL2 algorithms.

8 Summary

Degligiannakis and Roussopoulos [3], pointed out that traditional wavelet approximation methods for multiple dimensions may result in suboptimal space utilization in constructing synopses, and proposed extended wavelets. They gave an optimal algorithm for the sum of squared error and a faster 2-approximation for the problem of maximizing the weighted sum of squares of the representation, termed as "benefit". This approximation algorithm however does not guarantee the error. However, both their algorithms required linear space.

To address the problem of linear space, we presented optimal algorithms for extended wavelets which are space sensitive. We gave an algorithm that has error *less than or equal* to the error of the optimum solution (therefore at least as much benefit), but relaxes the space bound to store a few extra coefficients (at most as many as the number of different measures). We also demonstrate how to adapt all the above algorithms to the context of streaming data. Through experiments on real and synthetic data sets, we demonstrated that our proposed algorithms have significant performance benefits while requiring much less space.

Acknowledgments

The authors wish to thank the referees for many useful comments which have helped in improving the presentation of the manuscript. The work was supported by the Ministry of Information and Communication in Korea through the University Information Technology Research Center (ITRC) Support Program.

References

[1] L. Amsaleg, P. Bonnet, M. J. Franklin, A. Tomasic, and T. Urhan. Improving responsiveness for wide-area data access. *IEEE Data Eng.*, 20(3):3–11, 1997.

[2] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB Conference*, 2000.

[3] A. Deligiannakis and N. Roussopoulos. Extended wavelets for multiple measures. In *SIGMOD Conference*, 2003.

[4] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD Conference*, 1998.

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

[6] S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Histogramming data streams with fast per-item processing. In *ICALP*, 2002.

[7] S. Guha, C. Kim, and K. Shim. XWAVE: Optimal and approximate extended wavelets for streaming data. Techical Report, Seoul National University, Seoul, Korea, July 2004.

[8] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD Conference*, 1997.

[9] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *Proceedings of 25th International Conference on Very Large Data Bases*, Edinburgh, Scotland, September 1999.

[10] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. *SIGMOD Conference*, 1998.

[11] Y. Mattias, J. S. Vitter, and M. Wang. Dynamic Maintenance of Wavelet-Based Histograms. *VLDB Conference*, 2000.

[12] A. Natsev, R. Rastogi, and K. Shim. WALRUS: A Similarity Retrieval Algorithm for Image Databases. In *SIGMOD Conference*, 1999.

[13] V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. In *SSDBM*, 1999.

[14] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB Conference*, 1997.

[15] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD Conference*, 1996.

[16] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics - Theory and Applications.* Morgan Kaufmann, San Francisco, CA, 1996.

[17] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates on sparse data u sing wavelets. *Proceedings of SIGMOD*, pages 193–204, June 1999.

[18] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD Conference*, 1999.

[19] J. S. Vitter, M. Wang, and B. R. Iyer. Data cube approximation and histograms via wavelets. In *CIKM*, 1998.

[20] Y.-L. Wu, D. Agrawal, and A. E. Abbadi. Using the golden rule of sampling for query estimation. In *SIGMOD Conference*, 2001.