On the space–time of optimal, approximate and streaming algorithms for synopsis construction problems *

Sudipto Guha †

Abstract

Synopses construction algorithms have been found to be of interest in query optimization, approximate query answering and mining, and over the last few years several good synopsis construction algorithms have been proposed. These algorithms have mostly focused on the running time of the synopsis construction vis-a-vis the synopsis quality. However the space complexity of synopsis construction algorithms has not been investigated as thoroughly. Many of the optimum synopsis construction algorithms are expensive in space. For some of these algorithms the space required to construct the synopsis is significantly larger than the space required to store the input. These algorithms rely on the fact that they require a smaller "working space" and most of the data can be resident on disc. The large space complexity of synopsis construction algorithms is a handicap in several scenarios. In the case of streaming algorithms, space is a fundamental constraint. In case of offline optimal or approximate algorithms, a better space complexity often makes these algorithms much more attractive by allowing them to run in main memory and not use disc, or alternately allows us to scale to significantly larger problems without running out of space.

In this paper, we propose a simple and general technique that reduces space complexity of synopsis construction algorithms. As a consequence we show that the notion of "working space" proposed in these contexts is redundant. This technique can be easily applied to many existing algorithms for synopsis construction problems. We demonstrate the performance benefits of our proposal through experiments on real-life and synthetic data. We believe that our algorithm also generalizes to a broader range of dynamic programs beyond synopsis construction.

*A preliminary version of the paper appeared as "Space efficiency in synopsis construction algorithms", VLDB Conference 2005, Trondheim, [19].

†Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, PA 19104. Email: sudipto@cis.upenn.edu. Supported in part by an Alfred P. Sloan Research Fellowship and by NSF Awards CCF-0430376, CCF-0644119.

1 Introduction

Synopsis construction techniques consider representing the input in terms of the broader characteristics of the data, referred to as a synopsis or signature. These synopses or signatures, typically constructed to minimize some desired error criterion, are used subsequently in a variety of ways. These techniques are important data analysis tools and have been used in query optimization, image analysis and signal processing for a long time. Histograms were one of the earliest synopses used in the context of database query optimization [50, 41, 45]. Since the introduction of serial histograms by Ioannidis [36] this area has been a focus of a significant body of research, e.g., [35, 49, 38, 15, 27, 17, 24, 28], among many others. Matias, Vitter and Wang [44] gave one of the first proposals for using Wavelet based synopsis and over the last few years this topic has also received significant attention from different groups of researchers [5, 16, 13, 11, 46, 21]. Recently, synopses have also found applications in OLAP/DSS systems by Haas *et al.* [32], in approximate query answering by Amsaleg *et al.* [3] and Acharya *et al.* [1], and more recently in mining time series by Chakraborty *et al.* [6]. Histograms and wavelets are not the only synopses structures – quantiles and samples have been used widely as well. We will not be able to cover the entire literature and point the reader to the survey of Gibbons and Matias [14]. In this paper we will focus on histograms and wavelets mostly, but the ideas have broad applicability.

Most of the existing histogram and wavelet synopsis construction algorithms employ a dynamic programming (DP) approach and are expensive in space. The fastest algorithm for optimum histogram construction, provided by Jagadish *et al.* [38] required $\Omega(nB)$ space to construct a synopsis of size B from n numbers; algorithms proposed for Wavelet synopsis required $\Omega(n^2B)$ space. It is sometimes opined that

space used by these algorithms is not a problem with modern computers – but superlinear space becomes problematic even for moderate n. For $B = 100$ and $n = 8192$, using 4 bytes to represent a number, storing n^2B numbers amounts to 24 Gigabytes. Storing nB numbers amount to 3 Megabytes; in comparison the entire input can be stored in 32 Kilobytes. A simple fundamental question arises here: what "hidden" interaction exists between the numbers that forces us to require a table uses more space than is needed to store the input? The goal of this paper is to quantify this interaction, and to develop better algorithms in the process.

Synopses are most useful when n is not small. We would prefer a larger synopsis when n is large but those cases are exactly where extra factors of B or n hurt us the most. The space issue assumes more importance in emerging applications such as a sliding window data stream. In a fast data rate situation the total space is best allocated in the main memory. If we had larger space available then we could store larger size synopses or consider larger size windows, and increase overall accuracy.

Since space is a critical resource in many environments which require synopses, it is worthwhile being space efficient at the cost of losing a little bit in running time. However in a surprising twist, as experiments later in the paper show, we come across scenarios where the space efficient algorithm, which provably performs more work that the baseline algorithm we started with, was faster than the baseline algorithm. This is borne out in the case of the approximation algorithms, such as in [25], which were designed to improve the running time of offline algorithms and to save space in the streaming contexts. The space requirement was small to start with, and space efficient adaptation of the algorithm likely allowed the algorithm to use cache better. In hindsight, this is of course natural, and yet another demonstration of the interplay of space and time as in common in database systems.

The issue of super-linear space has long been an important one in the context of database algorithms. One of the earliest papers in this context is by Denning [10], where the notion of "working set" was introduced. The thesis of the paper, that the number of pages in main memory required by a process impacts the performance, has influenced the strategies for caching (see Smith, [51]) and buffer management (see Chou and DeWitt, [7]). In the context of algorithms, the study of memory hierarchy [2] and external memory algorithms (see [4, 53]) has been motivated by the need for algorithms that are "local", i.e., require a small footprint in memory and the I/O operations can be batched together. In terms of synopsis construction, the first part, namely the small footprint, has been known in folklore since the work of Jagadish *et al.* [38]. The idea is that the typical synopsis construction algorithms based on dynamic programming can rely on a small amount of local information, the working set, and the rest of the data structures are stored in disk. The working set approach has since been used widely. and most explicitly mentioned in the recent work of Gibbons and Garofalakis [12, 13]. However the idea of working space raises more questions than it answers – what is the best partitioning of data, the organization on disk, etc. In this paper, we show that for a number of synopsis construction problems the local structure/information is the only information we need to store. We can recompute the rest of the information without a significant increase in running time. Therefore we show that the notion of working set is redundant for a variety of these problems. The central questions we study in this paper is:

1. To construct a $O(B)$ size synopsis for $O(n)$ numbers can we design algorithms that use $O(n)$ space? Note that for offline algorithms a space complexity of $O(n)$ is optimal.

2. Are there general techniques that apply uniformly to a range of synopsis structures?

3. Can we demonstrate that the working space is the total space for many relevant applications?

4. Do such techniques allow "more", i.e., improve the running time using cleaner approaches ?

5. Are the techniques easily implementable ? Do we actually benefit from these new algorithms?

In this paper we take a fresh look at the DP approach for synopsis construction problems and identify the key aspects that lead us to space efficiency. The ideas in this paper apply to a broad spectrum of problems, but to render our discussion concrete, we limit our focus to synopsis construction problems. Although space efficient dynamic programming has been considered, in disparate contexts, such as string matching algorithms [33], those ideas do not immediately apply to synopsis construction or more general classes of dynamic programming problems. We begin by focusing on three areas which have recently seen a lot of activity.

Example 1: Histograms. The V-Optimal (VOPT) histogram was introduced by Poosala *et al.*

in [49] based on ideas proposed in [35]. Given n numbers, and a parameter B, the goal in this problem is to find the best piecewise constant representation of the data with at most B pieces, such that the sum of squares error between the data and the representation is minimized. Jagadish *et al.* [38] gave an $O(n^2B)$ time $O(nB)$ space algorithm to find the optimum B bucket histogram synopsis. Their result is based on a DP approach which extends easily to a wide variety of error measures, workloads, etc. We note that the ideas in this paper extend to the same space of measures as in [38]. As mentioned earlier, the notion of working space was shown to be effective in this context by the authors of [38], and they also proposed an $O(n^2B^2)$ algorithm which uses $O(n)$ space (as a small comment that space can be reduced using recomputations). Their algorithms implied that a "penalty" of $O(B)$ has to be paid in running time or space. We provide an algorithm which uses $O(n)$ space and $O(n^2B)$ time and improve the state-of-the-art, which shows that the working space notion is redundant for these problems. The above result is not limited to VOPT histograms; the discussion extends to other error measures, e.g., relative error, as well as range query histograms.

Example 2: Wavelets. Haar Wavelets are highly popular synopsis techniques, and in case of Euclidean error (or its square), storing the largest (normalized) coefficients of the data provides the best synopsis. This is not true for other measures such as maximum error and the problem remained open. Recently Garofalakis and Kumar [11] proposed an algorithm that uses $O(n^2B \log B)$ time and $O(n^2B)$ total space using a working space of $O(nB)$ to find the best B coefficients of the data which give the smallest maximum error. Since [11] several researchers[46, 43] observed improvements but the total space complexity remained above $O(nB)$. The same question, as in the construction of histograms, arises – do we need super-linear space? We show that we can find the best B coefficient that minimize the maximum error (also maximum relative error) in $O(n^2)$ time and $O(n)$ space, independent of B. Therefore, again we show that the "working set" is redundant in this context. For other error measures, e.g., workloads, weighted ℓ_p norms, the running time is $O(n^2 \log B)$. Our result also shows that the *error (not the coefficients)* of choosing the best B coefficients can be computed *simultaneously* for all $0 \leq B \leq n$ in time $O(n^2)$. Thus we can choose the "right" B where the marginal benefit drops by looking at the entire spectrum as B varies. This solves the "dual problem" i.e., given an error bound, find the minimum B such that the best B term synopsis will be within the error bound, also in $O(n^2)$ time and $O(n)$ space. Note, storing all the coefficients in the answer for all $0 \leq B \leq n$ may require $\Omega(n^2)$ space – in $O(n)$ space we can compute the coefficients (as well as the error) for any single B or only the error for all B.

The technique we describe here, in the context of wavelet optimization algorithms, applies only to offline algorithms since various quantities are recomputed. Karras and Mamoulis [40] and we [20, 21] provide streaming approximations for these problems. The techniques in this paper are not immediately applicable to these settings. The technique also does not apply to workload optimal Wavelet synopsis for range query problems as studied in [30]; because we are unable to capture the interaction between the coefficients in case of range query succinctly enough.

Example 3: Extended Wavelets. Extended wavelets were proposed by Deligiannakis and Rossopoulos [9] to construct wavelet representations in presence of multiple measures. They proposed a knapsack type computation for the optimum solution in $O(nMB)$ space and working set $O(nM + MB)$ for n items with M measures and size of synopsis B. In a recent work, [31] we were able to improve the space to optimum solution to $O(B^2 + BM)$. Our algorithm showed that there were a few "critical" elements in the data which can be stored in $O(BM)$ space and the optimum algorithm needed to be run only on them. But the central problem remained the same on those "critical" elements. The state-of-the-art is a fundamental $O(nB)$ space complexity of the optimum algorithm (plugging $n = B$). Using the techniques in this paper, we improve the total space complexity to $O(BM)$, which was the working space in [9], again showing that the working space notion is redundant in this case. As an added advantage, this also provides us a $O(BM)$ space streaming algorithm.

Approximation Algorithms for Histograms: So far, we have discussed optimum algorithms only. The space issue is also important in the context of approximation algorithms – the discussion on Extended Wavelets provides some intuition. If we view the optimization process as a search problem in a suitable space, most approximation algorithms seek to reduce this search space. For Extended Wavelets the reduced search space corresponding to the critical elements also contained the optimal solution. In general for approximation algorithms, the reduced search

space achieves near optimality, e.g., in approximate histograms [27] the search space to extend the DP table by one was reduced from $O(n)$ to $O(B\epsilon^{-1}\log n)$. In [23, 47] it was shown that finding the best histogram which is restricted to $O(B\epsilon^{-2}(\log n)\log\frac{1}{\epsilon})$ well chosen boundary elements, gives us a near optimum solution for the entire dataset. But the central issue of space did not disappear - it is not surprising that the space complexity of the algorithms are $O(B^2\epsilon^{-1}\log n)$ and $O(B^2\epsilon^{-2}(\log n)\log\frac{1}{\epsilon})$ (unless we pay the factor B in the running time) and we are back at the same situation where we were with the optimum algorithms. The central idea of the paper does carry over to approximate histograms. However they require significantly more technical details.

1.1 Our contributions and overview:

As mentioned earlier, our main contribution is a broad technique that improves a significant number of synopsis construction problems. More specifically

1. We present the general framework and the key ideas in Section 3.
2. We present **five** examples where our ideas improve the algorithms for optimum synopsis construction.
   (a) Histograms, discussed in Section 4. We give the first $O(n^2B)$ time $O(n)$ space optimum algorithm.
   (b) In Section 5 we show how the idea extends to approximation algorithms for histograms.
   (c) Likewise we show extensions of the idea to Range Query histograms in Section 6.
   (d) Wavelet reconstruction (non-sum of squares error), discussed in Section 7. We give the first $O(n^2)$ time $O(n)$ space algorithm; both the time and space results are the best known for this problem at present time.
   (e) Extended Wavelets, discussed in Section 8. We reduce the space requirement of the algorithm to $O(BM)$ from $O(MB + B^2)$. The problem is a variant of Knapsack, and our results demonstrates that the space efficient paradigm finds use in wider set of contexts.
3. We demonstrate that the approach is not theoretical – in the sense that there are no large constants hiding in $O()$ notations. We perform the experiments on VOPT histograms and wavelets only. Using synthetic and real life data sets which were used in previous papers we show that the space efficient algorithms give us the benefit they promise. In case of Wavelets, we show that our algorithm is superior to previous known algorithms.

Note that we do not discuss quality of synopsis across histograms and wavelets, or across error measures, which are of independent interest. Our goal in this paper is to explore the issue of space efficiency which apply to all of them, as well as to a broader spectrum of optimization problems.

2 Definitions and Preliminaries

The V-Optimal histogram construction problem is:

Problem 1 (V-Optimal Histogram: Minimize Error) *Given a set of n numbers $X = x_1, \ldots, x_n$ the problem seeks to construct a B piecewise constant representation (function) H such that $\|X - H\|_2^2 = \sum_i (x_i - H(i))^2$ is minimized. Each "piece" is a bucket and defines a subrange $[p, q]$ of the range $[1, n]$.*

We will abuse notation and will refer to the measure and the optimal histogram under the measure as VOPT. Many other error measures exist, notably workloads or weighted ℓ_p norms, maximum error $\|X-H\|_\infty = \max_i |x_i-H(i)|$ etc. Several researchers have proposed the relative error metric which uses some function of $\frac{x_i-H(i)}{\max\{|x_i|,c\}}$ to compute the error, e.g., maximum relative error $\max_i \left|\frac{x_i-H(i)}{\max\{|x_i|,c\}}\right|$. The parameter c is a constant that suppresses the contribution of very small data values. Figure 1 shows an example histogram.

Problem 2 (Histograms: Range Queries) *Given a set of n numbers $X = x_1, \ldots, x_n$ and a workload of R ranges or intervals the problem seeks to construct a B piecewise constant representation (function) H such that $\sum_{I\in R}(\sum_{i\in I} x_i - \sum_{i\in I} H(i))^2$ is minimized.*

Wavelets are multi-resolution representations of the data. There is a huge literature on this topic, including excellent textbooks [8]. Most of the database literature focus on Haar wavelets, due to their simplicity and the existence of fast algorithms for transforming the data to a wavelet representation. The inverse transformation is even simpler, it is simply the addition of at most $\log n$ numbers. Haar Wavelets

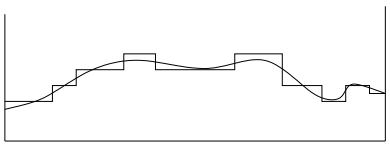


Figure 1: An example Histogram, which is a piecewise constant approximation of the distribution.

represent the data X (of length n, assumed power of 2) by the set of orthogonal vectors, ψ_i, defined below:

$$\begin{array}{rcll} \psi_1(j) = & 1 & \text{for all } j \\ \psi_{2^s+t}(j) = & \left\{ \begin{array}{l} 1 \\ -1 \end{array} \right. & \begin{array}{l} \text{if } (t-1)\frac{n}{2^s}+1 \le j \le \frac{tn}{2^s} - \frac{n}{2^{s+1}} \\ \text{if } \frac{nt}{2^s} - \frac{n}{2^{s+1}} + 1 \le j \le \frac{tn}{2^s} \end{array} \\ & & (1 \le t \le 2^s, 1 \le s \le \log n) \end{array}$$

For $n = 4$ the (non-normalized) $\{\psi_i\}$ are shown below

$$\{1,1,1,1\}, \{1,1,-1,-1\}, \{1,-1,0,0\}, \{0,0,1,-1\}$$

and they extend naturally to larger powers of 2. The inverse transform of Z is defined as $\mathcal{W}^{-1}(Z) = \sum_i Z_i \psi_i$. $\mathcal{W}^{-1}(Z)_i$ can be computed in $O(\log n)$ time without generating the full inverse. To compute $\mathcal{W}(X)$, we compute the average $\frac{x_{2i+1}+x_{2i+2}}{2}$ and the difference $\frac{x_{2i+1}-x_{2i+2}}{2}$ for each pair of consecutive elements as i ranges over $0, 2, 4, 6, \ldots$. The difference coefficients form the last $n/2$ entries of $\mathcal{W}(X)$. The process is repeated on the $n/2$ average coefficients – their difference coefficients yield the $n/4+1, \ldots, n/2$'th coefficients of $\mathcal{W}(X)$. The process stops when we compute the overall average, which is the first element of $\mathcal{W}(X)$.

Definition 1 *The* support $\textsc{Supp}(\psi_i)$ *of a vector* ψ_i *is the set* $\{j|\psi_i(j) \neq 0\}$. *These support sets are nested and are of cardinality which are powers of* 2 *(dyadic). Therefore we naturally have a complete binary "coefficient tree".*

The x_j correspond to the leaves of the coefficient tree, and the coefficients correspond to the non-leaf nodes, see Figure 2. Assigning a value c_i to a coefficient corresponds to assigning $+c_i$ to all leaves j that are *left descendants* (descendants of the left child) and $-c_j$ to all right descendants. The wavelet synopsis construction problem is defined below:

Definition 2 *The* **Normalized** *wavelet basis vectors* ψ_i^n *are defined as* $\psi_i^n = \frac{1}{\sqrt{\textsc{Supp}(\psi_i)}}\psi_i$. *It is straightforward to see that* $\psi_i^n \cdot X = \mathcal{W}(X)_i \sqrt{\textsc{Supp}(\psi_i)}$. *Further* $\psi_i^n \cdot \psi_i^n = 1$ *and* $\{\psi_i^n\}$ *defines an orthonormal (which is* ℓ_2 *length preserving) basis. We denote the normalized transform of* X *by* $\mathcal{W}_n(X)$ *(i.e.,* $\mathcal{W}_n(X)_i = X \cdot \psi_i^n$*) and the normalized inverse as* ${\mathcal{W}_n}^{-1}(Z) = \sum_i Z_i \psi_i^n$.

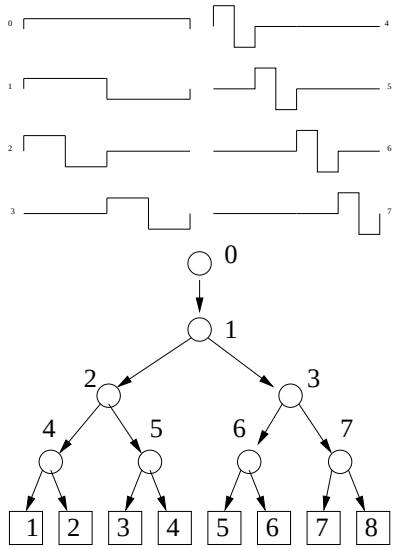


Figure 2: The basis vectors for wavelets for $n = 8$ and the coefficient tree.

Problem 3 (Wavelets: Minimize Error) *Given a set of* n *numbers* $X = x_1, \ldots, x_n$ *the problem seeks to choose at most* B *terms from the wavelet representation* $\mathcal{W}(X)$ *of* X*, say denoted by* Z*, s.t. a suitable function of* $X - \mathcal{W}^{-1}(Z)$ *denoting the error is minimized. For example,* $\|X - \mathcal{W}^{-1}(Z)\|_\infty$ *is the minimum maximum absolute error problem. Observe the problem is identical if we use* $\mathcal{W}_n()$ *and* ${\mathcal{W}_n}^{-1}()$.

We discuss maximum error since we will compare ourselves with previous work which uses the same error. The above definition extends to a broader class e.g., weighted-ℓ_p/workloads. Note that for the sum of squares error ℓ_2^2, $\|X - {\mathcal{W}_n}^{-1}(Z)\|_2^2 = \|\mathcal{W}_n(X) - Z\|_2^2$ since $\mathcal{W}_n()$ is ℓ_2 length preserving (also known as Parseval's Identity) and we arrive at a sum-of-squares problem which is minimized by choosing the largest B terms of $\mathcal{W}_n(X)$ and the rest as 0 as Z.

As mentioned earlier, we can also solve the dual problem.

Problem 4 (Wavelets: Minimizing synopsis size) *Given a target error* ϵ *find the wavelet reconstruction with smallest number of coefficients that is within the error bound for a suitable error function.*

The definition of Extended wavelets is in Section 8.

3 The general technique

The fundamental basis of our approach is the following: we want to use a divide and conquer algorithm. However simple divide and conquer (think merge-sort) does not work in these scenarios. There are two main problems in synopsis construction scenarios.

- Partition. For example, we may take the following approach: we find the best b-bucket histogram for the range $[1, \ldots, \frac{n}{2}]$ and the range $[\frac{n}{2}+1, \ldots, n]$ for all $b \leq B$ and try to merge them to find the best B bucket histogram for $[1, \ldots n]$. The idea falls flat because the optimum solution may not conform to a bucket ending at $\frac{n}{2}$. In fact, apriori, it is not clear where any of the bucket boundaries will be.

- Interaction. The above problem of bucket boundary disappears in the context of Wavelets, since the boundaries are aligned (known as dyadic intervals, since the set $\{j|V_i(j) \neq 0\}$ has cardinality a power of 2. But now a different problem arises: the wavelets are hierarchical and overlap. Thus in our simple strategy, there will be interaction between the two subproblems of half size through their parent.

The problem is that we do not know how to divide the problem into manageable subproblems, i.e., eliminate interaction and partition problems simultaneously. We will use the following strategy:

Observation 1 *We will use a dynamic program to find the interface – for a histogram this would be the boundary bucket which contains the partition; for wavelets this would be the interaction with the parent. In both cases it would matter how we divide the coefficients in the two parts. In a sense the paradigm can be viewed as Dynamic Programming meeting (being used for) Divide and Conquer. It may appear that we have achieved circularity, we have avoided a DP based solution to use another DP ! But if the interface is small size, i.e., can be specified with a few (constant) numbers, the DP for finding the interface can be smaller.*

The reader has probably guessed the game-plan by now, we will write a much smaller DP to find the interface and then recurse in each part. The parts can reuse the same space, since we will only be solving at most one subproblem at a time. The above is easier said than done. We would need to answer the following questions in order:

(A1) *Does such an interface exist?* Given a problem at hand, this will be the first step.

(A2) *Can the interface be described succinctly?* This issue also affects the simplicity and the implementability of any algorithm as well as space complexity.

(A3) *Can we compute it efficiently?* This would often involve solving for part of the original problem, e.g., if someone told us that the minimum error was 10, can we use the fact ? The answer is no, we cannot use the fact that the error is 10. But if the person could "prove" that 10 was the minimum – we can use that "proof" to find good division/interface. The fact that we also know the error is an accident. We shall see examples of this shortly, but to think ahead, a dynamic program is a recursive "proof".

The central part of the paper will be the following general theorem, the proof of the theorem is almost self evident, the difficulty is in applying it.

Theorem 1 (Single Cut) *Suppose there exists a partitioning that decomposes the problem into two subproblems of size no more than half the original, and we take $g(n, B)$ time to find the partitioning. Then the overall synopsis construction problem can be solved in time $f(n, B)$, where $f(n, B)$ is at most $g(n, B) + 2f(\frac{n}{2}, B)$, using a divide and conquer approach. In fact the running time can be further tightened to*

$$f(n, B) \leq g(n, B) + \max_{B_1 \leq B} \{f(\frac{n}{2}, B_1) + f(\frac{n}{2}, B - B_1)\}$$

The proof follows if we assume $f(n, B)$ is monotone non-decreasing in n, B. The above partitions n, since n is almost always the dominating term, the above theorem will be the most useful in our context. We can also express the above theorem in context of partitioning B.

Corollary 1 (Single Cut) *Suppose if we partition on B such that each part has at most $\lfloor \frac{B-1}{2} \rfloor$ buckets and we take $g(n, B)$ time to find the partitioning. Then the overall synopsis construction problem can be solved in time $f(n, B)$, which is at most $g(n, B) + 2f(n, \lfloor \frac{B-1}{2} \rfloor)$, using a divide and conquer*

approach. The running time can be further tightened to

$$f(n,B) \leq g(n,B) \quad + \quad \max_{n_1 \leq n-1} \{ f(n_1, \lfloor \frac{B-1}{2} \rfloor) + f(n-n_1-1, B-1-\lfloor \frac{B-1}{2} \rfloor) \}$$

Here is a more interesting version of the above theorem, which helps us in the case $g(n,B) = O(nB)$.

Theorem 2 (Double Cut) *Suppose there exists a partitioning that decomposes the synopsis construction problem into three subproblems such that both the parameters n', B' in each of the subproblems are no more than half the original, and we take $g(n,B)$ time to find the partitioning. Then the overall synopsis construction problem can be solved in time $f(n,B) \leq g(n,B) + 3f(\frac{n}{2}, \frac{B}{2})$ using a divide and conquer approach.*

Intuition: The result follows from first partitioning such that each side is smaller than $n/2$, and then partitioning the side which has larger than $B/2$ coefficients. The result follows from monotonicity of $f()$. The above approach saves one subproblem out of possible 4 if we divided n, then divided B and so on. In the case where $g(n,B) = O(nB)$ if we apply the single cut theorem we would get $f(n,B) = nB \log B$ or $f(n,B) = nB \log n$. The use of the double cut theorem will be shown in the case of Extended wavelets. For the most part of the paper, we will restrict our discussion to the Single Cut. There are however places where the Double cut provides us a better result, notably, if B is large as well. Let us now investigate the examples where the above approach helps us.

4 Example I: Histograms

The VOPT histogram is a classic problem in synopsis construction. Given a set of n numbers $X = x_1, \ldots, x_n$ the problem seeks to construct a B piecewise constant representation H (function) such that $\|X - H\|_2$ (or its square) is minimized. As mentioned earlier, since their introduction in query optimization in [36], histograms have accumulated a rich history [37], further, [38] gave a $O(n^2 B)$ time algorithm to find the optimum histogram using $O(nB)$ space. They observed that the space could be reduced to $O(n)$ at the expense of increasing the running time to $O(n^2 B^2)$.

Several different optimization criteria have been proposed for histogram construction, e.g., ℓ_1, relative error, ℓ_∞, to name a few. However most of them are based on a dynamic program similar to the VOPT case. Thus the VOPT histograms provide an excellent foil to discuss all of the measures at the same time.

4.1 The VOPT Algorithm

Jagadish *et al.* [38] show that if a "bucket" or range $[j+1, \ldots, i]$ is approximated by one value, as is the case in histograms, the best value v which minimizes the error $e(j+1,i) = \sum_{r=j+1}^{i} (x_r - v)^2$ is the mean $(\sum_{r=j+1}^{i} x_r)/(i-j)$. Based on this they gave a natural DP algorithm which is given in Figure 3.

```
Algorithm VOPT
begin
1.    Let E[i,b] be the min error b bucket histogram for [1,...,i].
2.    Initially E[i,1] ← e(1,i) for all 1 ≤ i ≤ n
3.    For b = 2 to B do
4.        For i = 2 to n do {
5.            E[i,b] = ∞
6.            For j = i − 1 downto 1 do {
7.                E[i,b] ← min{E[i,b], E[j,b−1] + e(j+1,i)}
8.                (Optional) If (E[i,b] < e(j+1,i)) break;

          /* perform book-keeping if minimum is changed */
          /* i.e., store the minimizing j in array P[i,b] */
          /* that allows the solution to be reconstructed */
              }
          }
end
```

Figure 3: The VOPT algorithm

To compute the quantities $e(j+1,i)$, we can maintain two arrays $\text{SUM}[i] = \sum_{r=1}^{i} x_i$ and $\text{SQSUM}[i] = \sum_{r=1}^{i} x_i^2$ and compute $e(j+1,i)$ in $O(1)$ time as follows (see [38]):

$$e(j+1,i) = \text{SQSUM}[i] - \text{SQSUM}[j] - \frac{(\text{SUM}[i] - \text{SUM}[j])^2}{i-j}$$

The strength of the above algorithm is its generality; it operates with virtually any definition of $e(j+1,i)$ as long as we can compute it efficiently; changing the sum in Line (7) to max for maximum error, etc. The $O(n^2 B^2)$ algorithm uses the array $E[*, b-1]$ to construct $E[*,b]$ but discards $E[*,b-1]$ immediately. It of course does not store the bookkeeping array $P[,]$ since it cannot store $O(nB)$ items. So when the algorithm computes $E[*,B]$, it uses the minimizing j, i.e., $argmin_{1 \leq j \leq n}\ E[j,b-1] + e(j+1,n)$, to recursively find the buckets for $[1,\ldots,j]$. The optional statement in Line (8) significantly improves performance since it avoids useless searching. If the precondition of Line (7) is true, then any smaller j will yield no better a solution since $e(j'+1,i) \geq e(j+1,i)$ for all $j' \leq j$. In fact in the experimental section we will see that the pruning yields significant benefits for skewed data.

4.2 Applying Our paradigm: The Algorithm SpaceOpt

As the reader knows by now, we will try to find the "interface" or the bucket that contains $\frac{n}{2}$.

Definition 3 *Given n, B, define $MB(1, i, b)$ to be the the triple (p, q, b') such that in the optimal b-bucket histogram for the subrange $[1, \ldots, i]$ that contains the point $\lfloor \frac{n}{2} \rfloor$ in the bucket $[p, q]$ and the histogram uses b' buckets to the left of p.*

Lemma 1 *If $MB(1, n, B)$ is the triple (p, q, b) then the partitioning $[1, p-1]$ and $[q+1, n]$ satisfy the three criterion $(A), (B)$,and (C) in the previous section.*

Proof: The fact that the partitioning has no interaction between the two subproblems is easy to observe. The description of the partitioning is the triple (p, q, b) and is succinct - we will prove that we can compute it by providing an $O(n^2B)$ algorithm. ∎

Assuming that such an algorithm exists (which we will see shortly), we can apply the Theorem 1 and the running time of the overall algorithm $f(n, B)$ can be bounded by $f(n, B) \leq g(n, B) + 2f(\frac{n}{2}, B)$. Since $p \leq \frac{n}{2} \leq q$ the two subproblems on $[1, \ldots, p-1]$ and $[q+1, \ldots, n]$ are of size at most $\frac{n}{2}$. If the running time of the algorithm that finds the division is $g(n, B) = an^2B$ for some constant a, then $f(n, B) \leq 2cn^2B$ solves the above equation since

$$\begin{aligned} f(n,B) &\leq an^2B + 2f(\frac{n}{2}, B) \\ &\leq an^2B + 2a(\frac{n}{2})^2B + 4f(\frac{n}{4}, B) \\ &\leq an^2B + \frac{an^2B}{2} + \frac{an^2B}{4} + \cdots \leq 2an^2B \end{aligned}$$

The above implies that the time to find the interface is at most doubled. In fact we can use the tighter formulation to set up $an^2B + \frac{an^2B}{4} + \cdots$. But in any case the algorithm is $O(n^2B)$ and we will see how the algorithm holds up in practice. Therefore summing up, If we can find the **middle bucket** in time $O(n^2B)$ and space $O(n)$ then we can solve the V-Optimal histogram problem in time $O(n^2B)$ and space $O(n)$ using our framework (the subproblems can use the same space).

4.3 Finding Middle-bucket efficiently

Consider a DP which proceeds from $i = 1$ to $i = n$ and constructs the optimum b-bucket synopsis for the interval $[1, i]$. Our goal would be to remember that bucket when we cross the $n/2$ point, that is, when

```
Algorithm Middle-Bucket(s,t,b)
begin
1.    For i = s + 1 to t {
2.        M[i] ← (s, i, 0)
3.        E[i] ← e(s, i)
4.    }
5.    For b' = 2 to b do {
6.        newE[i] = ∞
7.        For i = s + 1 to t do {
8.            For j = i − 1 downto s do {
9.                If (newE[i] > E[j] + e(j + 1, i)) {
10.                   newE[i] = E[j] + e(j + 1, i)
11.                   If (j ≥ (t−s+1)/2) then newM[i] ← M[j]
12.                   else newM[i] ← (j + 1, t, b' − 1)
13.               }
14.               (Optional) If (newM[i] < e(j + 1, i)) break;
15.           }
16.           M ← newM; E ← newE;
17.       }
18.   Now M[t] contains MB(s, t, b)
end

Algorithm SpaceOpt(1,n,B)
begin
1.    Initialize stack to empty.
2.    Push (1, n, B).
3.    While stack is non-empty {
4.        Pop the top of stack, say (s, t, b)
5.        If b > 1 then {
6.            Let (p, q, b') ← Middle − Bucket(s, t, b)
7.            Push (s, p − 1, b') and (q + 1, t, b − b' − 1)
8.        } elseif (b = 1) Output [s, t] as a bucket

              // We may have b = 0 which we need to
              // check when we push. We also compute
              // and output e(s, t), maintain sum etc.

9.    }
end
```

Figure 4: Algorithms $MB(1, n, B)$ and SpaceOpt.

$i \geq n/2$ and the last bucket is $[j, i]$ and $j < n/2$. This is quantified below:

Lemma 2 *Suppose the last bucket for the optimum b-bucket histogram for $[1, \ldots, i]$ is $[j+1, i]$. If $j < \frac{n}{2}$ then $MB(1, i, b+1) = (j, i, b)$ otherwise $MB(1, i, b+1) = MB(1, j, b)$.*

Proof: If $j \geq \frac{n}{2}$, the fact that any prefix of the buckets in an optimum histogram are optimum for their range (otherwise, the overall error would go down by choosing a different histogram) allow us to conclude that $MB(1, j, b)$ is also the bucket that contains $\frac{n}{2}$ for the best $b + 1$ bucket histogram of $[1, i]$. In the other case, $MB(1, i, b+1) = (j, i, b)$ by construction. ∎

The above lemma gives us a dynamic programming algorithm for computing $MB(1, i, b)$. The pseudocode is given in Figure 4. The overall idea is to construct $MB(1, n, B)$ from $MB(1, n, B-1)$. The array $M[i]$ keeps track of the $MB(1, n, b)$ for various b and is initialized for $b = 1$. Note that we are computing $E[i, b]$ in $newE$, but not storing it. We are computing $MB(1, i, b)$ in $newM[i]$ and we are using

it to compute $MB(1, i, b+1)$. Observe that we know the minimum error in $E[n]$ at the end of the computation – *but we do not know the buckets that give the error.* We need the divide and conquer strategy to give us that. *This brings us back to the discussion that we are computing the minimum, but only using the proof of minimality and not the value.* Therefore we are ready to conclude the following:

Theorem 3 *We can compute the V-Optimal histogram in time $O(n^2B)$ and space $O(n)$ by the SpaceOpt algorithm, i.e., repeatedly finding the middle bucket and solving the two subproblems.*

Implementation details: The above pseudocode is almost the actual code. The $O(n)$ space is due to the fact that we store $E, M, \text{SUM}, \text{SQSUM}$. We need a stack to keep track of the recursion. If we are solving to find a $b > 1$ bucket histogram for $[s, t]$, which is at the top of stack, we find the middle bucket (the 1 in the pseudocode is s and $t = n$, so $\frac{n}{2}$ is $\frac{t-s}{2}$ for this subproblem); if that is (p, q, b'), then we pop the top of stack and push a b'-bucket problem for $[s, p-1]$ and a $b-b'-1$ bucket problem for $[q+1, t]$ to the stack. If $b = 1$, we have our bucket – we simply pop and output it. Note that the size of the subproblems on the stack decrease by a factor of 2 or more. Each subproblem is represented using three numbers; two for the start and end of the subrange and the third determines the number of buckets in this subrange. Thus the overhead from the stack is at most $O(\log n)$. This implies that the space to store $E, M, \text{SUM}, \text{SQSUM}$ dominates, and the total space is $O(n)$.

4.4 Generalizations to measures other than VOPT

The basic dynamic program provided by Jagadish *et al.* [38] extend in a straightforward fashion to other error measures. This has been observed by several researchers, we quote the discussion in [28]. The properties used in the DP for VOPT can be succinctly expressed as:

(B1) The error of a single bucket is only dependent on the values in the interval defined by the bucket and nothing else. That is, $\text{SQERROR}(i, j)$ depends on the values of i, j and $x_i, x_{i+1}, \ldots, x_j$ only. Without this important property, the dynamic programming formulation would not be possible.

(B2) The overall error function, TERR, is the sum of the errors, SQERROR, in each of the B buckets.

(B3) The fact that $\text{SQERROR}(i+1, j)$ can be computed from $\text{SUM}[i]$, $\text{SQSUM}[i]$,$\text{SUM}[j]$, and $\text{SQSUM}[j]$ allowed us to compute $\text{SQERROR}(i+1, j)$ in time $O(1)$ time.

Theorem 4 (Theorem 6, [28]) *Suppose we are given a histogram construction problem where the overall error E_T is sum of the errors E_B of each of the buckets where $E_B(i, j)$ where $E_B(i, j)$ depends only on $x_i, \ldots, x_j$. If we can compute a data structure* INFO *using time $O(nT)$ and space $O(nP)$ such that any $E_B(i, j)$ can be computed in time $O(Q)$ from the records* $\text{INFO}[i]$ *and* $\text{INFO}[j]$*, then we can find the optimum histogram in time $O(nT + n^2BQ)$ time and $O(n(P + B))$ space based on dynamic programming algorithm in [38].*

Based on the above, and Theorem 3 we can immediately conclude:

Theorem 5 *Suppose we are given a histogram construction problem where the overall error E_T is sum of the errors E_B of each of the buckets where $E_B(i, j)$ where $E_B(i, j)$ depends only on $x_i, x_{i+1}, \ldots, x_j$. If we can compute a data structure* INFO *using time $O(nT)$ and space $O(nP)$ such that any $E_B(i, j)$ can be computed in time $O(Q)$ from the records* $\text{INFO}[i]$ *and* $\text{INFO}[j]$*, then we can find the optimum histogram in time $O(nT + n^2BQ)$ time and $O(nP)$ space based on the SpaceOpt algorithm.*

Applications: There are several immediate applications, we will not repeat the details which are available in [27, 31, 28]. Most notable examples are

- *Sum of Absolute Errors:* We need to efficiently compute the *median* of any interval. In this case $P = Q = T = O(\log n)$.
- *Approximation by piecewise splines:* In this case $P = T = O(d)$ and $Q = O(d^3)$ where d is the degree of the spline.
- *Relative Error:* The relative error minimizes some function of the terms $\left|\frac{x_i - H(i)}{\max\{|x_i|, c\}}\right|$. More details are available in [31].

5 Example I.(a): Approximation Algorithms for Histograms

It is natural to ask: what improvements are guaranteed by the above technique in context of approximation algorithms for histogram construction? We show

that (i) that the approximation algorithms carry over for a broad class of general measures (ii) for date stream algorithms we can show improvement for the VOPT measure only. The reason behind (ii) is that for streaming algorithms the space bounds arise from more than one context; even if we improve the space required for error evaluation the bottleneck of "how much information must be stored" cannot be overcome. We overcome the problem for VOPT due to a special Pythagorean property of the VOPT error shown in [23].

We begin by a short review of the known results for approximation algorithms. Our discussion will once again follow the V-Optimal histogram construction; we indicate generalizations/restrictions wherever appropriate. Jagadish *et al.* [38] give an approximation algorithm that runs in time $O(n^2B/\ell)$ and uses $B+\ell$ buckets while guaranteeing the quality of the $(B+\ell)$-bucket histogram constructed is no worse than the best B-bucket histogram. Since their result significantly stronger algorithms have been proposed which do not relax the number of buckets and run significantly faster. We focus on approximation schemes where given a precision parameter $\epsilon > 0$, the algorithm will return a solution which is at most $(1+\epsilon)$ times worse than the optimal solution. See [34, 52] for discussion on approximation schemes. The results can be summarized in the Table 1.

The streaming model assumes the data items x_i are presented one at a time in an increasing order of i. Thus for time series and analogous applications these algorithms are one-pass stream algorithms. The "Y/N" indicates that the algorithm applies to sliding window data streams.

The algorithm of [17] applies to a more general model of streaming, and is quite complicated. It shows that collecting a number of suitable wavelet coefficients gives us a robust histogram – whose error does not go down on adding a few extra buckets. It then uses the robust histogram to construct a histogram with B buckets using a DP similar to [38, 27], and any improvement to these algorithms translate analogously.

We have already seen the $O(n)$ space $O(n^2B)$ algorithm in the previous section. The next two results in this paper follow from our discussion in the previous section and the results of [28] and [23].

5.1 A $O(n + \frac{B}{\epsilon})$ space approximation algorithm

In [28] we showed how to find an $(1+\epsilon)$-approximation algorithm in time $O(n + B^3(\epsilon^{-2} + \log n)\log n)$ for the optimal B-bucket histogram construction problem. This is the fastest approximation algorithm till date, however the space complexity is $O(n + B^2\epsilon^{-1})$. The dominant term is $O(n)$ if with $B \gg \sqrt{n}$. But in many scenarios we may be able to store a larger number of coefficients in our synopsis for example $B \approx n^{2/3}$. The above algorithm requires superlinear space. In the extreme case we may set $B = n/100$, which achieves a factor 100 compression of the data, the above space complexity is quadratic. The above suggests that we do not have a smooth and graceful tradeoff, and a natural question arises in this context:

Question 1 *Can we achieve a fast (therefore possibly approximate) and space-efficient computation of histograms in the range where B is not too small?*

In what follows, we provide a space efficient version of the approximation algorithm and make the space linear in $\min\{n, B\epsilon^{-1}\}$. Before proceeding further, we first summarize the main ideas used in [28] and indicate the main challenges which require new ideas. Subsequently we provide the improved algorithm.

5.1.1 Review of Approximation and Challenges

The key idea behind [28] was that since all the intervals arise from the same basic sequence of n values, we can try to prune wisely and not evaluate the error at all of the $O(n^2)$ intervals. Clearly, for such an approach to succeed, there must be added structure in the relationship of the error and the intervals. This is encoded in the definition below:

Definition 4 *A function f on intervals is defined be interval monotone if for all intervals $I' \subseteq I$ $f(I') \leq f(I)$.*

And if the preconditions (B1)–(B3) to Theorem 5, as well as the following two conditions hold, then we can claim the next theorem for VOPT error.

(B4) The error $\textsc{Sqerror}(i,j)$ is an interval-monotone function. This property allows the approximation algorithm to prune the number of intervals we evaluate the function. The dynamic programming of the optimum algorithm tries all possible choices and does not require this property.

(B5) The value of the largest number R (and therefore the maximum and the minimum nonzero error) is polynomially bounded in n.

Paper	Measure	Stream	Factor	Time (for V-OPT)	Space
[38]	general	No	Opt	$O(n^2B)$	$O(nB)$
	general	Y/N	Opt	$O(n^2B^2)$	$O(n)$
[27]	general	Yes	$(1+\epsilon)$	$O(nB\tau)$	$O(B\tau)$
[26]	general	Y/N	$(1+\epsilon)$	$O(n + B\tau^2 \log n)$	$O(n + B\tau)$
[17]	VOPT,ℓ_1	Yes	$(1+\epsilon)$	$O(n + ((B \log n)/\epsilon)^{O(1)})$	$O(((B \log n)/\epsilon)^{O(1)})$
[23] ([47])	VOPT only	Yes	$(1+\epsilon)$	$O(n + B^3\epsilon^{-4} \log^2 n \log^2 \frac{1}{\epsilon})$	$O(B^2\epsilon^{-2} \log n \log \frac{1}{\epsilon})$
				$O(n + B^4\epsilon^{-4} \log^2 n \log^2 \frac{1}{\epsilon})$	$O(B\epsilon^{-2} \log n \log \frac{1}{\epsilon})$
[31]	general	Yes	$(1+\epsilon)$	$O(n + B\tau^2 \log \tau)$	$O(B\tau^2 \log n)$
[28]	general	Y/N	$(1+\epsilon)$	$O(n + B^3(\epsilon^{-2} + \log n) \log n)$	$O(n + B^2\epsilon^{-1})$
	general	Yes		$O(n + \tau M)$	$O(B\tau + M)$
Here	general	Y/N	Opt	$O(n^2B)$	$O(n)$
	general	Y/N	$(1+\epsilon)$	$O(n + B^3(\epsilon^{-2} + \log n) \log n)$	$O(n + \min\{n, \frac{B}{\epsilon}\}) = O(n)$
	VOPT only	Yes	$(1+\epsilon)$	$O(n + B^3(\epsilon^{-2} + \log n) \log^2 n)$	$O(B\epsilon^{-2} \log n)$

Table 1: Summary of results on similar problems $\tau = \min\{B\epsilon^{-1} \log n, n\}$ usually $\tau \ll n$ and $M = B(B\epsilon^{-1} \log^2 \tau + \log n \log \log n) \ll B\tau^2 \log \tau < B\tau^2 \log n$. Note that the space requirement in this paper is linear in B while preserving an $O(n)$ asymptotic running time.

Theorem 6 (Theorem 3, [28]) *In $O(n + B^3(\log n + \epsilon^{-2}) \log n)$ time and $O(n + B^2\epsilon^{-1})$ space, we can compute an $(1+\epsilon)$-approximate B-bucket VOPT histogram of n points. Furthermore, for a sliding window model we can compute a histogram of the previous n elements in time $O(B^3(\log n + \epsilon^{-2}) \log n)$.*

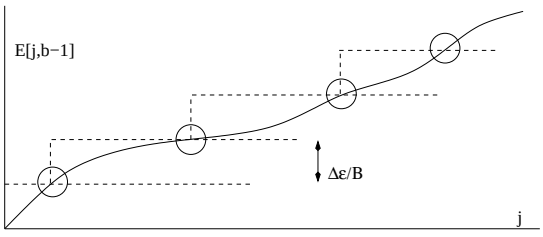


Figure 5: The approximation of $E[j, b-1]$.

For other errors, similar results can be found in Theorem 6, [28]. The central idea was to reconsider the step 4.1 in Figure 3 and observe that since $E[j, b-1]$ is an increasing function and $e(j+1, i)$ is a decreasing function (as j increases, for both), we can approximate $E[j, b-1]$ by a "staircase type" function which allows us to compare the sum only at the circled "knees" of the function, as shown in Figure 5. The savings, in both time and space, arise from the fact that the number of these knees is small. But there is a tradeoff involved, because the function $E[j, b-1]$ has to be approximated well – it was shown in [28] that if we knew that the error was between Δ and 4Δ then we can have the height of the stairs set to $\Delta\epsilon/B$ and the size of the staircase is $O(B/\epsilon)$. This precision of $\Delta\epsilon/B$ was necessary, because of the for-loop in step 4.1 of Figure 3 which essentially accumulates the approximations additively (this additive behavior requires proof, but the details of the proof are not directly relevant to the discussion here). But since the for-loop was executed B times, we accumulated an error of $\epsilon\Delta$ which gave us the $1+\epsilon$ approximation. If Δ were nonzero, we started from a provable lower bound and repeatedly doubled Δ till we got an 2-approximate solution in the range Δ and 4Δ. The $\Delta = 0$ was an easier case. However, we *still* were working with the same dynamic program and thus we needed to store the approximations for all $1 \le b \le B$ and this made the space requirement $O(n + B^2/\epsilon)$. The running time arose from three facts (i) to evaluate $E[i, b]$ (in fact this will be referred to $\textsc{Apxerr}[i, b]$ subsequently since we are computing approximations) as we needed to minimize over $O(B/\epsilon)$ values (ii) we needed to compute $\textsc{Apxerr}[i, b]$ for $O(B/\epsilon)$ values of i, which is needed for evaluating any $\textsc{Apxerr}[\cdot, b+1]$ and (iii) to find the knee points, we have to perform a binary search which introduces a $O(\log n)$ factor (note that the binary search also requires proof because we are using the approximate values rather than the exact values, once again the details of the proof will not be relevant here the interested reader should consult [28]). Therefore the running time (once we know Δ) over all the values of b, is $O(B\frac{B}{\epsilon}\frac{B}{\epsilon} \log n)$. The other term of $O(B^3 \log^2 n)$ arises from the search for Δ; here the ϵ is set to a constant but we may perform the search $\log n$ times as we raise Δ by powers of 2.

The natural direction we take is to use the alternate dynamic program in Figure 4. However several *challenges* arise:

- Consider a strategy similar to that of SpaceOpt in Figure 4 where we found the bucket which contained the $i = n/2$ point. It is conceivable that this partitions the approximate histogram

into parts which has $B-1$ and 1 buckets. Now since the running time is $O(B^3\epsilon^{-2}\log n)$ (assume the good case that we know the respective Δs) then by this process we may be invoking the routine B times (because of the split). Thus the running time would become at least $\Omega(n + B^4\epsilon^{-2}\log n)$ ignoring other terms. This tradeoff is similar to the alternate algorithm based on [38] which used $O(n)$ space but at the cost of blowing up the time by a B factor to $O(n^2B^2)$. The problem arises because n is not the dominant parameter of interest in the running time. Clearly, this is undesirable and we would have to partition the problem differently.

- The approximation algorithm introduces dependencies which are across the board; because we minimize $\textsc{Apxerr}[j, b-1] + e(j+1, i)$ the approximation of the later buckets depend on the approximation of the earlier ones. This implies that to adapt the approximation algorithm we would have to encode this dependence in the interface. Of course, we can try to come up with a different partitioning and computing scheme, but that would require a new proof of correctness (of approximation) and we will avoid that here. We reiterate that the main mission of this paper is to demonstrate that *existing* algorithms can benefit from the space efficient strategy detailed here, and to this end we would like to keep the older algorithms intact.

5.1.2 A new algorithm: The Other Middle Bucket

The discussion of challenges naturally identifies that the "interaction" between the two sub-problems must include the value of $\textsc{Apxerr}[q, \lfloor\frac{B-1}{2}\rfloor+1]$. This suggests that the second part of the problem should try to optimize a function which is the actual error plus a value of $\textsc{Apxerr}[q, \lfloor\frac{B-1}{2}\rfloor + 1]$.

Definition 5 *Given n, B, define the* other middle bucket $OMB(s,t,b,S)$ *to be the endpoint q of the $\lfloor\frac{b-1}{2}\rfloor+1$-th bucket $[p,q]$ in the optimal b-bucket histogram for $[s,\ldots,t]$ where the* Apxerr *values are shifted by S (that is we add $S>0$ to all the values). Note that this shift does not affect the optimum algorithm, but does affect the approximation. We specify the middle bucket by the $(p, q, \textsc{Apxerr}[q, \lfloor\frac{b-1}{2}\rfloor+1])$.*

The description of the partitioning is succinct, and it clearly exists – we will prove that we can compute it efficiently by providing an algorithm. Therefore the partitioning $[1, p-1]$ and $[q+1, n]$ will satisfy the three general criterion in Section 3 and we will have a space efficient approximation algorithm for VOPT histograms. The algorithm is given in Figure 6, which shows how to compute OMB. The main work is done in procedure SubSpace-Δ whose input parameters are the range $[s,t]$, the number of buckets b, the shift S and two other parameters $z, MaxEst$. $MaxEst$ determines the maximum error which we expect – this is used for pruning combinations which will not lead to small error. The parameter z is the step size of the approximation and is typically ϵ/B times the overall error for $[1,n]$. However, these parameters $z, MaxEst$ are set by Space-Δ, in increasing power of two so that the algorithm first quickly finds a 2 approximation and then the algorithm proceeds to a $1+\epsilon$ approximation. The main benefit of this is that the repeated (possibly) failed searches for the upper bound of the error is significantly fast and is independent of ϵ, and these correspond to the two terms in the running time. SubSpace-Δ uses the two queues O_Q, N_Q corresponding to computations of $\textsc{Apxerr}[*, k-1]$ and $\textsc{Apxerr}[*,k]$. The $OMB, NOMB$ are used to keep track of $OMB(s,t,b,S)$. This quantity first gets defined when $k = \lfloor\frac{b-1}{2}\rfloor + 1$ (in the procedure $CBList$). The procedure $CBList$ is an algorithm which starting from the end constructs the "knees" or the boundaries where $\textsc{Apxerr}[*,k]$ jumps by at least z. Thus the number of boundaries cannot be more that $MaxEst/z$ which will be $O(B/\epsilon)$. The implementation of $CBList$ appears to be complicated, but is in fact a simple tail recursion (which avoids duplicating binary searches). Note that O_Q, N_Q and $OMB, NOMB$ gets switched as $k \leftarrow k+1$ and that accounts for the overall $O(B/\epsilon)$ space. The procedure StackSpace-Δ which outputs the buckets using the space efficient recursion and the outermost algorithm Space-Δ are given in Figure 7. These fragments, compared to [28], show that small changes suffice in developing the space efficient approximation algorithm.

Given that an efficient algorithm to find OMB exists, the running time of the overall algorithm $f(n,B)$ can be bounded by $f(n,B) \le g(n,B) + 2f(n, \lfloor\frac{B-1}{2}\rfloor)$.

Observe that this is the best possible since the two parts may together be larger than n. This is because of the $start < b'$ described in the *CreateBestList* procedure – basically the approximation cannot guarantee a clean separation. If the running time of the algorithm that finds the division is $g(n,B) = aB^3\epsilon^{-2}\log n$ for some constant a, then $f(n,B) \le 2aB^3\epsilon^{-2}\log n$ solves the above equation

```
Procedure SubSpace-Δ(s, t, b, S, z, MaxEst)
begin
    /* Compute the OMB and approx. b-bucket histogram
    for [s, t], minimizing the error of the histogram plus S.
    MaxEst is the upper bound of the error
    z is the step size of approximation */
1.  O_Q has one element s − 1 with APXERR[s − 1, 0] = S
        /* Note: This change alone "shifts" the values. */
2.  N_Q is set empty
3.  For k = 1 to b {
4.      CBList(s,t,b,k, MaxEst, z, O_Q, N_Q, OMB, NOMB )
5.      Copy O_Q ← N_Q and OMB ← NOMB
6.      Set N_Q empty, NOMB will be overwritten
7.  }
8.  OMB(O_Q.size − 1) contains OMB(s, t, b, S)
9.  We also return APXERR[t, b].
end

Procedure CBList(s, e, b, k, C, z, O_Q, N_Q, OMB, NOMB)
begin
1.  Compute APXERR[s, k] using O_Q
2.  Suppose the index of O_Q which gave the minimum is i'
3.  if (APXERR[s, k] ≥ C)
4.      return C /* drop the interval */
5.  while ( s < e ) do {
6.      m ← (s + e + 1)/2
7.      C ← CBList(m, e, k, C, z, O_Q, N_Q, OMB, NOMB)
        /* Cchanges here */
8.      if (APXERR[start, k] ≥ C) then return C
            /* Drops interval, but queue was changed */
9.      e ← m − 1
10. }
11. if (APXERR[s, k] < C) {
12.     Insert s at the front of N_Q
13.     C ← APXERR[s, k] − z
14.     If (k = ⌊(b−1)/2⌋ + 1) then
15.         Insert (i', s, k, APXERR[s, k]) in the front of NOMB
16.     Else Insert(copy) OMB(i') in the front of NOMB.
17. }
18. return C
end
```

Figure 6: Algorithm to find $OMB(1, n, B, S)$

because

$$\begin{aligned} f(n,B) &\leq aB^3\epsilon^{-2}\log n + 2f(n, \lfloor\frac{B-1}{2}\rfloor) \\ &\leq aB^3\epsilon^{-2}\log n + 2f(n, \frac{B-1}{2}) \text{ (monotonicity)} \\ &\leq aB^3\epsilon^{-2}\log n + 2\frac{aB^3\epsilon^{-2}\log n}{8} + 4f(n, \frac{B-1}{4}) \\ &\leq aB^3\epsilon^{-2}\log n + \frac{aB^3\epsilon^{-2}\log n}{4} + \frac{aB^3\epsilon^{-2}\log n}{16} \ldots \\ &\leq \frac{5}{4}aB^3\epsilon^{-2}\log n \end{aligned}$$

Note that $g()$ corresponds to the second part of the running time, when we are performing the finer grained search as opposed to the initial part where we are only interested in finding an upper bound of the error. We conclude with the following:

Theorem 7 *We can compute an $(1+\epsilon)$-approximate B-bucket VOPT histogram in time $O(n + B^3(\log n + \epsilon^{-2})\log n)$ and space $O(n + B\epsilon^{-1})$ by repeatedly finding OMB and solving the two subproblems.*

```
Procedure StackSpace-Δ(B, MaxEst, z)
begin
1.  Initialize stack with (1, n, B, 0).
2.  error ← 0
3.  While (stack ≠ empty) {
4.      Let (s, t, b, S) ← Pop(stack)
5.      If ((t > s) and (b > 1)) {
6.          Let (p, q, b, S') ← SubSpace-Δ(s, t, b, S, z, MaxEst)
7.          Push (s, p, ⌊(b−1)/2⌋, S)
8.          Push (q, t, b − ⌊(b−1)/2⌋ − 1, S')
9.      } elseif ((b = 1) and (s ≤ t)) {
10.         Output [s, t] as a bucket, error+ = SQERROR(s, t)
11.     }
12. }
end

Algorithm Space-Δ
begin
1.  Create SUM, SQSUM, Cutoff ← 0; z ← 0
2.  E ← SubSpace-Δ(1, n, B, 0, z, Cutoff )
3.  if E = 0 return
4.  Δ ← 1/2 /* minimum error for integers */
5.  Cutoff ← 2(1 + ε)Δ; z ← Δ/2B
6.  E ← SubSpace-Δ(1, n, B, 0, z, Cutoff )
7.  while E ≥ 4Δ do {
8.      Δ ← 2 * Δ
9.      Cutoff ← 2(1 + ε)Δ; z ← Δ/2B
10.     E ← SubSpace-Δ(1, n, B, 0, z, Cutoff )
11. }
12. Δ ← E/2, Cutoff ← 2(1 + ε)Δ; z ← εΔ/2B
13. StackSpace-Δ(B, Cutoff , ρ, z)
end
```

Figure 7: Recursive Algorithm StackSpace-$\Delta(1, n, B)$ and the overall algorithm Space-Δ

5.2 Data Stream Algorithms for VOPT Measure

Data stream algorithms are algorithms that compute a function in a small number of passes over the input under the restriction that the memory is sublinear in the input size and any item not explicitly remembered cannot be accessed in the same pass. In keeping with tradition, unless we use explicitly mention the prefix multi-pass, we will use the data stream setting to denote the scenario where only a single pass is allowed.

... Constructing a synopsis in the data stream model is an attractive proposition, primarily due to the fact that the scenarios we are most interested in data compression are the scenarios where system memory is scarce. As seen in Table 1, starting from the results of [27] there has been a significant research on data stream computation of histograms. In this discussion we would limit ourselves to the time series model where the input data X is specified as $X_1, X_2, \ldots, X_n$ in order. Note that all the streaming algorithms which apply to general error measures take space quadratic in B (ignoring other terms). An interesting deviation is the case of VOPT error where the results of [23, 47] show that the VOPT histogram

approximations can be achieved via wavelets, thus shedding some light on the question regarding the comparison of these two differing synopsis techniques in the context of the VOPT error. This is an interesting "presynopsis" approach, where we first filter out most of the irrelevant information and then compress the information into our final synopsis.

However, not surprisingly, the *same* dichotomy, as in the case of the optimal VOPT algorithms remain. The main idea of [23, 47], is a different answer to the question posed in [27] namely, "can we limit the number of points at which we evaluate the dynamic program to construct the (near) optimal histogram?" The solution of [27], extended in [28], was to find these points recursively using the dynamic program itself. The solution suggested in [23] was to use the boundary points defined by the largest wavelet coefficients; however this idea was only shown to be useful in the context of VOPT error. It was proved that

Theorem 8 *[23, 48] The best B-bucket histogram (in VOPT error) where the boundaries are restricted to be the boundaries of the largest $O(\frac{B}{\epsilon^2}\log n \log\frac{1}{\epsilon})$ wavelet coefficients of the data give a $(1+\epsilon)$ approximation to the best B-bucket histogram over the entire original data.*

The suggestion in [23, 48] was to use a modified version of the optimal dynamic program of [38]. The curious reader must have asked the question that why not use the approximation algorithm of [27, 28], the answer is that then the space requirement is quadratic in B (ignoring logarithmic terms). But the downside of using the algorithm of [38] was that to guarantee $\tilde{O}(B)$ space, the running time is increased by a factor of B to B^4 (ignoring other terms, see table 1).

Therefore, we are again considering the Question 1, that can we really achieve the best of every possible world, (that is near linear, in B, space usage and not pay any excess in running time) when B is not too small. We are revisiting the same question of space efficiency as we have seen in the last few sections. The reader possibly notices that the answer, based on the previous sections, would now follow naturally. We will use the most space efficient algorithm we have – the Space-Δ algorithm. But, we will need to apply Space-Δ with a twist, since we cannot afford $O(n)$ space!

The main idea is the following two step process, but first note that the VOPT error of approximating X (defined over $[1,n]$) by H is equivalent to $\|X-H\|_2^2$ or the ℓ_2^2 difference of the two n dimensional vectors (X_i indicates the value of the i^{th} dimension).

1. We will show that we can store an approximation X' of the signal such that for *any* B bucket histograms H the error $\|X-H\| \approx \|X'-H\|$. The benefit however is that X' can be specified exactly using $O(B\epsilon^{-2}\log n\log\frac{1}{\epsilon})$ wavelet coefficients.

2. Now we will use Space-Δ on this X' and instead of storing X', generate the entries of X' as needed. The total space usage will be $O(B\epsilon^{-2}\log n\log\frac{1}{\epsilon}) + O(B\epsilon^{-1})$. Note that we may obtain boundary points *other than the boundary points of X'*; this is the point of departure from [23, 47] and gives us the savings in terms of space. Also note that Space-Δ itself is not amenable to a one pass implementation, the use of the compressed signal was necessary.

The wavelet coefficients are constructed in a streaming fashion as well (see details in [23]). For each of the endpoints of the wavelets we store $\textsc{Sum}, \textsc{Sqsum}$. We will run the Space-$\Delta$ algorithm with a twist; we will change the $\textsc{Sqerror}()$ function. Suppose we are interested in evaluating $\textsc{Sqerror}(u,v)$ then instead of using $\textsc{Sqsum}, \textsc{Sum}$ we will use $\textsc{Sqsum}', \textsc{Sum}'$ defined as follows: Since X' is a combination of several wavelet terms, it is piecewise flat. Suppose u lies between two consecutive boundary points v_1, v_2 of X'. Let that flat height between v_1, v_2 be h. Now we also know that $\textsc{Sum}'[u] = \textsc{Sum}[v_1] + h(u-v_1)$, where $\textsc{Sum}'$ refers to X'. Likewise $\textsc{Sqsum}'[u] = \textsc{Sqsum}[v_1] + h^2(u-v_1)$ and the same quantities are defined for v as well. At this point we have all the pieces of the algorithm, we run the Space-Δ algorithm. The algorithm is offline – but we are only storing X' which takes $O(B\epsilon^{-2}\log n\log\frac{1}{\epsilon})$ space only.

The overall space if therefore $O(B\epsilon^{-2}\log n\log\frac{1}{\epsilon}) + O(B\epsilon^{-1}) = O(B\epsilon^{-2}\log n\log\frac{1}{\epsilon})$. The running time is $O(n + B^3(\log n + \epsilon^{-2})\log^2 n)$, the extra $\log n$ appears from the fact that given u, we need to find v_1, v_2, h which takes $O(\log n)$ time. This immediately allows us to conclude that :

Theorem 9 *We can compute a $1+\epsilon$ approximation for the VOPT histogram in a single pass where $\ldots, x_i, \ldots$ are presented in increasing order of i, in time $O(n + B^3(\log n + \epsilon^{-2})\log^2 n)$ and space $O(B\epsilon^{-2}\log n\log\frac{1}{\epsilon})$.*

In a surprising twist, we observed in Section 9.4 that this algorithm which has a seemingly worse theoretical analysis of the running time actually performs better than the competition over the relevant range of parameters. Of course the analysis may be tightened, but we believe that the reason for this

scenario is that the space saving of a factor B has enormous impact when the overall space requirement is not very high – the secondary cache is likely to be large enough to store the entire presynopsis plus the smaller DP table. We note that this effect of caching in the small space regime is one of the appeals of streaming algorithms, even if we ignore important situations where all data cannot be stored which mandate data stream algorithms.

6 Example I.(b): Range Query Histograms

Histograms and synopsis structures are not only useful for estimation of point queries, but for most DSS/OLAP applications involve aggregate or range queries . Further a rich structure emerges in range query optimization where the ranges (and subranges) frequently are interrelated, e.g. hierarchies [39], prefixes [42], etc. Further, range queries naturally subsume point queries, i.e., where each range is a singleton element. It is quite straightforward to observe that a histogram constructed to optimize point queries is not necessarily a good histogram to estimate aggregate behavior. To address this problem, several optimal and approximate histogram algorithms have been proposed for range query workloads in [42, 18, 29, 48]. We summarize the main results below in Table 2 and note the contribution of this paper. The "uniform" refers to cases where all ranges are equally likely. The first point to observe is that the space requirements are large. In fact, some initial coding shows that the space requirement is the bottleneck in these situations. However, it is precisely when the space requirements are large space efficiency is more important. For example saving a factor 50 is useful but not critical if the space requirement is a few kilobytes. If the required space is quadratic, even for reasonable domain sizes, say $n \approx 2000$, a saving of a factor of 50 is significant and often the deciding factor that makes the program feasible.

Caveats of Table 2: The quantities in the square parenthesis indicate the restriction that the representative of a bucket is constrained to be the average. [18] contains a few approximations in the constrained case which are made redundant by [48]. [48] also gives data stream algorithms which run in time $O(N(B \log n)^{O(1)})$ and space $O((B \log n)^{O(1)})$.

Observe that there are no optimal algorithms except in simple constrained cases. *The primary issue behind this is precision.* Since the errors of the ranges are squared and added in non-trivial ways – it is not possible to guarantee that the optimal exists over rationals! The constraint forces the solution to exist over rationals. The last two algorithm [48] assume that the minimum nonzero error is bounded away from 0. This is not easy to show (no proof exists) even if the input numbers are integers since the solution can be some irrational very close to the input numbers unlike point query histograms where assuming integer input implies the minimum non-zero error is $\frac{1}{4}$. Technically, these algorithms are *asymptotic approximation*, i.e., the approximation ratio holds when the error is not too small; although this is unlikely to be a problem in most scenarios.

Sketch of Improvements: In case of the prefix queries, it was shown in [42] that a dynamic programming formulation, similar (but not same) to that of [38] gave the optimal solution. The dynamic program had a form of $E[i, b] = \min_j E[j, b-1] + C(j+1, i)$ where $C(j+1, i)$ was the sum of the errors of the ranges that ended in the range (endpoints inclusive) $j+1$ and i. The authors of [42] showed that this quantity could be computed in $O(1)$ time using some preprocessing which required $O(n)$ time and space. This naturally fits the generalization of VOPT histograms we discussed in Section 4.4 and the saving of a factor B follows immediately. The case of the complete binary tree is somewhat different, and more reminiscent of wavelets, which we will discuss shortly. The similarity arises from the fact that these synopses investigate dyadic subranges of $[1, n]$. For the hierarchy results, we could not show that a small interface exists – small interfaces can be found by increasing the running time, but there is no clear benefit. Improving their space complexity remains an interesting open question.

Similar to [42], the authors of [18] use a dynamic program of the form $E[i, b, \Lambda] = \min_j E[j, b-1, \lambda] + C'(j+1, i)$ where $C'(j+1, i)$ is a function that depends on $i, j+1$ and can be computed in $O(1)$ time. The Λ and λ are "bias" parameters and are related by an equality that depends on the values between $j+1$ and i and can be updated in $O(1)$ time as j decreases from $i-1$. The reader would immediately observe that parameter b need not be stored (but as this paper shows, can be recreated) and it is exactly what gives the space savings.

In case of uniform queries, it was shown in [48] that the optimum B bucket range query problem reduced to a problem where we were given the prefix sum of the data, namely $X'[i] = \sum_{j<i} X[i]$ and we wished to compute the best B bucket histogram of X' using VOPT error, *but* where each bucket was

Paper	Problem	# Buckets	Approx	Time	Space	**Space by this paper**
[42]	prefix	B	[Opt]	$O(n^2B)$	$O(nB)$	$\mathbf{O(n)}$
	full bin. tree	B	[Opt]	$O(n^6B^2)$	$O(n^5B)$	$\mathbf{O(n^4B\log n)}$
	hierarchy	B	[Opt]	$O(\|T\|n^6B^2)$	$O(\|T\|n^4B)$	–
[18]	uniform	B	$[1+\epsilon]$	$O(n^2BR)$	$O(nBR)$	$\mathbf{O(nR)}$
[29]	hierarchy	$2(1+\gamma)B$	[Opt]	$O(n+B^2\|T\|n^{\frac{6}{1+\gamma}})$	$O(n+B\|T\|n^{\frac{4}{1+\gamma}})$	–
[48]	uniform	$2B$	Opt	$O(n^2B)$	$O(nB)$	$\mathbf{O(n)}$
		$2B$	$(1+\epsilon)$	$O(n+B^3\epsilon^{-2}\log^2 n)$	$O(n+B^2\epsilon^{-1})$	$\mathbf{O(n+B\epsilon^{-1})}$
		B	$(1+\epsilon)$	$O(n^4B^3\epsilon^{-2})$	$O(n^3B^2\epsilon^{-1})$	–

Table 2: Summary of results on Range Query Histograms . The $|T|$ stands for the size of the hierarchy of the ranges, R stands for the largest value seen.

represented by a linear segment (as opposed to a constant in VOPT histograms). This resulting B bucket histogram was then converted to a $2B$ bucket "alternating" histogram, such that after every bucket, we introduced a new bucket of size one, but ensured that the sum of the values in X and the histogram agree up to that point. The problem therefore reduces to computing these "linear" histograms. In [28] it was shown that if the buckets are represented by small degree polynomials, we can compute the best B bucket histogram upto a factor of $(1+\epsilon)$ as well. The running time increased by a factor of $(d+1)^3$ but $d=1$ in this case.

7 Example II: Wavelets

Recall that the Wavelet construction problem is that given a set of n numbers $X = x_1, \ldots, x_n$ the problem seeks to choose at most B terms from the wavelet representation $\mathcal{W}(X)$ of X, represented by Z such that $\|X - \mathcal{W}^{-1}(Z)\|_\infty$ is minimized.

7.1 Previous algorithm(s)

We follow the description of Garofalakis and Kumar [11]. The basic idea is to define the array $E[i,b,S]$ for each node i of the coefficient tree where $0 \le b \le B$ and S is a subset of the ancestors of i. The Algorithm solves the DP given in Figure 8, we reference the algorithm as Waveopt for the rest of the paper.

Algorithm Waveopt
begin
1. We proceed bottom to top in the coefficient tree.
2. At each node i (assuming the children are i_L and i_R), set $E[i,b,S]$ as follows:

$$\min \begin{cases} \min_{b'} \max\{E[i_L, b', S\cup\{i\}], E[i_R, b-1-b', S\cup\{i\}]\} \\ \min_{b'} \max\{E[i_L, b', S], E[i_R, b-b', S]\} \end{cases}$$

end

Figure 8: The algorithm Waveopt.

It is immediate that the value of $\mathcal{W}^{-1}(Z)_j$ is fixed by the choices of all coefficients i such that j belongs to the support of i. Suppose S is a subset of the ancestors of a coefficient i. Thus a natural dynamic program emerges where we define $E[i,b,S]$ to be *the minimum contribution to the error from all j in the support of i, such that exactly b coefficients that are descendants of i are chosen along with the coefficients of S.* The algorithm is given in Figure 8. Clearly the number of entries in the array $E[]$ is Bn times 2^r where r is the maximum number of ancestors of any node. It is easy to see that $r = \log n + 1$ and thus the number of entries is n^2B. For maximum relative error, we may perform binary search for the minimization and only need $\log B$ time (see [11]) and the overall time is $O(n^2B\log B)$. The space required is $O(n^2B)$. The authors of [11] only discussed maximum error, however as pointed out by Muthukrishnan [46], the max can be suitably replaced for other measures.

7.2 Dynamic Program to Divide and Conquer

As our game-plan suggests, we will write a dynamic program to discover the "interface". In this case, the "interface" consists of two pieces of information (i) is the parent chosen (ii) how are the coefficients divided between the two subtrees. Immediately, we come across a problem; (i) is recursive! That is the parent depends on its parent and so on. We definitely do not want to consider so many cases since we want $O(n)$ space algorithms, and therefore "small interfaces".

One observation comes to our rescue: the dependence on the parent (and through it, on its parent) can be expressed in one number v as a sum or bias introduced by a combination of all of them. But we do not want to write down all possibilities, since there will be at least $O(n)$ of them for each node. The main idea we use is: We can recurse within the DP

to generate the set on the fly (without storing them). We note that although we analyze the same cases as before, the sophistication of the recurse-within-DP is higher than the previous algorithm, which is needed to prove a much better result. The overall implementation is not complicated in terms of code - the pseudocode is presented in Figure 9. The top-level call is $SpaceWave(root, 0)$. But before we proceed further, we must assure ourselves that this recursion-within-DP does not go out of hand.

```
Algorithm SpaceWave(i,v)
begin
    /* We want to allocate 0 ≤ b ≤ B coefficients to the two
    subtrees and the node i to minimize ℓ_infty error. We
    return an array of size B with the allocation and the error
    w[i] is the coefficient at node i, unless i is a leaf, in
    which case it has the data x[i] */

3.  If (i == root) { /* root has no ancestor so v = 0 */
4.      A ← SpaceWave(child, W[root]);
5.      C ← SpaceWave(child, 0);
6.      We return the array D[b] = min{A[b − 1], C[b]}
7.      D[b] stores the error as well as if it was from A,C
8.  } else {
9.      If (i == leaf) return array D[b] = |x[i] − v| /* maxerror */
10.     else {
11.         A ← SpaceWave(i_L, v + w[i]);
12.         C ← SpaceWave(i_R, v − w[i]);
13.         P ← SpaceWave(i_L, v);
14.         Q ← SpaceWave(i_R, v);
15.         Let t_1[b] be the best error and allocation of coefficients
            assuming w[i] is chosen, let
16.         t_1[b] ← min_{0≤r≤b−1} max{A[r], C[b − 1 − r]}

17.         Likewise, t_2[b] ← min_{0≤r≤b} max{P[r], Q[b − r]},
18.         for the case w[i] is not chosen

19.         return array D[b] ← min{t_1[b], t_2[b]}
20.         D[b] stores the error, split r and if it was from t_1, or t_2.
21.     }
22.}
end
```

Figure 9: The algorithm SpaceWave

Lemma 3 *Each node with ℓ ancestors is called at most $2^\ell < 2n$ times. The running time of $SpaceWave(root, 0)$ is $O(n^2 B \log B)$.*

Proof: The first part follows from induction. Each min max can be performed by binary search and there are at most B entries to compute at i, the result follows. ∎

Now $SpaceWave(root, 0)$ returns the information of the split and if the root coefficient was chosen for the best solution, based on which we can now recursively compute the coefficients.

Implementation Details: As the reader must have guessed, just like the histogram case we would maintain a stack which contains i, v, b. We pop from the stack, if $b > 0$ we call $SpaceWave(i, v)$, and look at the b'th entry. This gives us (i) if $w[i]$ is chosen, then we output, (ii) what is the split between the two children i_L, i_R, which we push into stack. If $b == 0$ we simply discard the top of stack. Observe that in the recursive calls we can adjust B in SpaceWave to the b parameter in the top of stack since we are dividing that many coefficients at most.

It is easy to see that the above stack would take $O(\log n)$ space overall since the depth of the coefficient tree is $O(\log n)$. What is interesting is that evaluating $SpaceWave(root, 0)$ we would need only at most $O(B \log n)$ space, since at most one $SpaceWave$ call will be active between children of the same parent.

Theorem 10 *We can compute the best B-term Wavelet synopsis for maximum error in time $O(n^2 B \log B)$ and space $O(n + B \log n)$.*

7.3 Further Optimization

We make the following observation in context of the algorithm SpaceWave. If a node has r children, then at most $r+1$ coefficients can be chosen in its subtree! Observe that this $r+1$ will always be a power of two.

Lemma 4 *If a node has ℓ ancestors and $r+1$ descendants (including itself) in the coefficient tree then $2^\ell(r+1) = 2n$.*

Thus we return at most arrays of size $\min\{B, r+1\}$ from $SpaceWave(i, v)$ – since more than$r+1$ coefficients cannot be chosen. A node with ℓ ancestor needs time $2^\ell \min\{B, r+1\} \log \min\{B, r+1\}$. The number of nodes with $\ell > 1$ ancestors is at most 2^ℓ. Thus the total time taken over all the nodes is

$$\sum_{\ell=0}^{\log n+1} 2^\ell 2^\ell \min\{B, r+1\} \log \min\{B, r+1\}$$
$$= \sum_{\ell=0}^{\log n+1} \min\{2^{2\ell} B, 2^\ell 2^\ell r + 1\} = \sum_{\ell=0}^{\log n+1} \min\{2^{2\ell} B, 2^\ell 2n\}$$

Now the last sum can be split into two terms

$$\sum_{\ell=0}^{\log n+1} \min\{2^{2\ell} B, 2^\ell 2n\} = \sum_{\ell=0}^{\log n - \log B} 2^{2\ell} B + \sum_{\log n - \log B}^{\log n+1} 2^\ell 2n$$

Both of the terms are geometric sums. The first is increasing by factor of 4 and its value is maximized at $\log n - \log B$, to be $\frac{n^2}{B}$. Thus the contribution from the first sum is at most $4\frac{n^2}{B}$. The second sum is also increasing, but in power of 2, the maximum is at $\log n + 1$ and we get the maximum term to be $4n^2$. Taking both the terms, the

second one dominates and we get $O(n^2)$ running time! The space required is likewise $\sum_{\ell=0}^{\log n+1} \min\{B, r+1\}$ which is $B(\log n - \log B)$ from the part where $\ell < \log n - \log B$. For the other part $r+1$ forms a geometric series in decreasing powers of 2 as ℓ increases (From Lemma 4) and adds up to $O(B)$. Observe $B(\log n - \log B) = B\log(n/B) \leq n$ (for $B \leq n$)

Theorem 11 *We can solve the maximum error wavelet reconstruction in time $O(n^2)$ and space $O(n)$.*

8 Example III: Extended Wavelets

Extended wavelets were introduced in [9]. The central idea behind Extended Wavelets is that in case of multi-dimensional data, there can be significant saving of space if we use a non-standard way of storing the information. There are several standard ways of extending 1-dimensional (Haar) wavelets to multiple dimensions. But irrespective of the number of dimensions, the format of the synopsis is a list of pairs of numbers *(coefficient index,value)*. In Extended Wavelets, we perform wavelet decomposition independently in each dimension but then we store a list of tuples consisting of the coefficient index, a bitmap indicating the dimensions for which the coefficient in that dimension is chosen, and a list of values. For example, suppose we were storing the coefficient number 15 in dimension 1 with value v, we would store a tuple $\langle 15, 1, v\rangle$. Now, if we wished to store the coefficient number 15 in dimension 3, with value u, we can store another tuple $\langle 15, 3, u\rangle$ or, leverage the stored values by storing $\langle 15, 101, v, u\rangle$ where the 101 is a bitmap which indicates that the coefficient 15 is stored in dimension 1 and 3 and the two subsequent numbers are the respective values. Since the coefficient number and the bitmap for a specific tuple is shared across the coefficients, we can store more coefficients than a simple union of unidimensional transforms. Intuitively this can be seen as a scenario where the "marginal" cost of information is smaller – or that there is a "bulk discount" in storing similar information. Formally, the Extended Wavelets are defined as:

Definition 6 *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 c_{ij}.*

Problem 5 (Extended Wavelets) *Given the NM wavelet coefficients $\{c_{ij}\}$, a storage constraint B, and a set of weights W, select the extended wavelet coefficients T to be stored in order to minimize the weighted sum $\sum_{(i,j)\notin T} W_j \cdot c_{ij}^2$ where $T = \{(i,j)|c_{ij}$ is stored $\}$. which (in optimality, but in not in approximation) is equivalent the "maximizing the benefit" where the benefit is defined as: $\sum_{(i,j)\in T} W_j \cdot c_{ij}^2$.*

Notice that there is no interaction between the benefits of storing coefficients corresponding to i and i'. The problem reduces naturally to a generalization of the **Knapsack** problem where each item (coefficient i) can be present in increasing sizes $w_{i1}, w_{i2} \ldots, w_{iM}$ (which are integers, $w_{ij} = jS + H$ for all i in our case) with increasing profits $p_{i1}, p_{i2} \ldots, p_{iM}$. We can see that defining $p_{i0} = 0$, we get $p_{ij} = p_{i(j-1)} + W_{t_j} c_{it_j}^2$ where c_{it_j} is the j^{th} largest (ignoring signs) element of $\{c_{i1}, \ldots, c_{iM}\}$; thus p_{ij} is increasing. This allows a dynamic program of $O(NMB)$ time and space, as presented in in Figure 10, where $E[i, b]$ is the maximum profit possible using b space and using indices from $[1, \ldots, i]$. As mentioned in the introduction, [24] reduced the interesting set of items from $n = N$ to $n = O(BM)$ but did not improve the DP, which is given in Figure 10. We show how to reduce the space of the algorithm *Ext-Opt*.

Algorithm Ext-Opt
begin
1. **For** $i = 1$ to n do
2. **For** $b = 0$ to B do {
3. $E[i, b] \leftarrow \max \begin{cases} E[i-1, b] \\ \max_{1\leq j\leq M}\{E[i-1, b-w_{ij}] + p_{ij} \end{cases}$
end

Figure 10: The DP in [9, 24]. In [9] $n = N$ and in [24] $n = O(BM)$.

Challenges and Intuition: The central idea is to find in $O(nBM)$ time the *middle-space* i.e., the space

allocated at $\frac{n}{2}$ as in histograms. However that gives suboptimal results since the solution of

$$g(n, B) = cnMB + g(\frac{n}{2}, b') + g(\frac{n}{2}, B - b')$$

is $g(n, B) = O(nMB \log n)$. We can do better. The solution will be a twofold approach.

1. We will use the *Double cut Theorem*, Theorem 2 where we will split the problem both in B as well as in n. However the question is how do we keep track of both these partitions?

2. To answer the above we will actually use *two* simultaneous DPs! The goal of the first, denoted as MSi below, would be to compute the total error and the partition in the number of items n. The goal of the second will be to compute the partition of the space b.

One way of viewing the solution is to observe the two nested DPs represent a factoring of a very large DP. Observe that an *arbitrary* way of solving the entire DP is not space efficient. The above factoring was critical in terms of space efficiency. The above process also gives us pseudocode which is really simple to implement. We begin by the following definition:

Definition 7 *We will define three related things:*

- *Let $E[s, i, b']$ be the maximum profit of using b' space while choosing coefficients only from the subrange $[s, \ldots, i]$.*

- *Let $MSi(s, t, i, b')$ be the tuple (b'', w) where b'' is the space allocated to $[s, \ldots, \lfloor \frac{t+s}{2} \rfloor]$ and w is the space allocated to The index $\lfloor \frac{s+t}{2} \rfloor$ in an optimal knapsack of size b' for $[s, \ldots, i]$.*

- *Let $MSb(s, i, b', b)$ be the tuple $(i', w_{i'j^*}, b'')$ such that i' is the smallest index such that the space allocated to $[s, \ldots, i']$ is $b'' \geq \lfloor \frac{b}{2} \rfloor$ in an optimum solution of $[s, \ldots, i]$ using total space b'. The space used for the index i' in this solution is $w_{i'j^*}$ which is stored to avoid recomputation.*

The next lemma is straightforward and follows from the definition of $E[s, i, b]$ as given in Figure 10 and the definition of $MSi(s, t, i, b)$.

Lemma 5 *$MSi(s, t, i, b')$ and $MSb(s, t, b', b)$ can be defined recursively as follows.*

- *Let $V_j = E[s, i-1, b' - w_{ij}] + p_{ij}$ and $j^* = \arg\max_{1 \leq j \leq M} V_j$.*

- *To define $MSi(s, t, i, b)$:*

  - *If $E[s, i-1, b'] \geq V_{j^*}$ then*

    $$MSi(s, t, i, b') = \begin{cases} (b', 0) & \text{if } i = \lfloor \frac{s+t}{2} \rfloor \\ MSi(s, t, i-1, b') & \text{Otherwise} \end{cases}$$

  - *Otherwise let*

    $$MSi(s, t, i, b') = \begin{cases} (b', w_{ij^*}) & \text{if } i = \lfloor \frac{s+t}{2} \rfloor \\ MSi(s, t, i-1, b' - w_{ij^*}) & \text{Otherwise} \end{cases}$$

- *To define $MSb(s, i, b', b)$:*

  - *If $E[s, i-1, b'] \geq V_{j^*}$ then*

    $$MSb(s, i, b', b) = \begin{cases} MSb(s, i-1, b', b) & \text{if } b' \geq \lfloor \frac{b}{2} \rfloor \\ (i, 0, b') & \text{Otherwise} \end{cases}$$

  - *Otherwise let*

    $$MSb(s, i, b', b) = \begin{cases} (i, w_{ij^*}, b') & \text{if } b' - w_{ij^*} < \lfloor \frac{b}{2} \rfloor \\ MSb(s, i-1, b' - w_{ij^*}, b) & \text{Otherwise} \end{cases}$$

Actually $MSi(s, t, i, b')$ need not be defined for $i < \lfloor \frac{t+s}{2} \rfloor$, and likewise for $MSb(s, i, b', b)$ need not be defined for $b' < \lfloor \frac{b}{2} \rfloor$; but we will do so since the algorithm becomes simpler to express. Finally we are interested in $MSi(s, t, t, b), MSb(s, t, b, b)$. Note that we need to ensure that the subproblems created have space less than $b/2$ and size less than $n/2$, we have to ensure that we do not recompute the index $\lfloor \frac{s+t}{2} \rfloor$, and the index which causes the space usage to just exceed $\lfloor b/2 \rfloor$ – this will may make the size of that problem $(n+1)/2$. This is not a very important issue, but allows us to apply the cut theorems without any further discussion. The above gives us a DP algorithm for computing $MSi(s, t, b, b)$. The algorithm actually only stores $MSi(s, i, *, b)$ and uses the same space as i changes. Thus we only need arrays of size $O(B)$. Likewise we get an algorithm for $MSb(s, t, b, b)$. The pseudo-code of the routines that find $MSi(s, t, b, b)$ and $MSb(s, t, b, b)$ are given in Figure 11. The overall algorithm (which uses a stack to maintain recursion) is given in figure 12. The overall time taken by both of them $O(nMB)$. Thus if the running time of evaluating (s, t, b) in the stack of XOpt is $f(t - s + 1, b)$ then we have the following recurrence using the following:

$$\begin{aligned} f(n, b) &\leq c \cdot nMb + 3f(\frac{n}{2}, \frac{b}{2}) \\ &\leq c \cdot nMb + \frac{3}{4}c \cdot nMb + 9f(\frac{n}{4}, \frac{b}{4}) \\ &\leq cnMb\left(1 + \frac{3}{4} + \frac{9}{16} + \cdots\right) \leq 4cnMb \end{aligned}$$

The above actually uses the fact that nb is "quadratic" in the variables n, b and that was the

```
Algorithm FindMSi(s,t,b)
begin
      /* E[b'] will store E[s,i,b'] as i keeps changing */
      /* M[b'] will store MSi(s,t,i,b') as i changes */
1.    For b' = 0 to b {
2.        E[b'] ← 0
3.        M[b'] ← 0
4.    }
5.    For i = s to t do {
6.        For b' = 0 to b do {
7.            newE[b'] ← E[b']
8.            If i ≠ ⌊(t+s)/2⌋ then newM[b'] ← M[b']
9.            else newM[b'] ← (b', 0)
10.           For j = 1 to M do {
11.               If (newE[b'] < E[b' − w_ij] + p_ij) {
12.                   newE[b'] ← E[b' − w_ij] + p_ij
13.                   If (i ≠ ⌊(t+s)/2⌋)
                              then newM[b'] ← M[b' − w_ij]
14.                   else newM[b'] ← (b', w_ij)
15.               }
16.           }
17.       }
18.       M ← newM; E ← newE;
19.   }
20.   Now M[b] contains MS(s,t,t,b)
end
Algorithm FindMSb(s,t,b)
begin
      /* E[b'] will store E[s,i,b'] as i keeps changing */
      /* M[b'] will store MSi(s,i,b',b) as i changes */
1.    For b' = 0 to b {
2.        E[b'] ← 0
3.        M[b'] ← 0
4.    }
5.    For i = s to t do {
6.        For b' = 0 to b do {
7.            newE[b'] ← E[b']
8.            If (b' ≥ ⌊b/2⌋) then newM[b'] ← M[b']
9.            else newM[b'] ← (i, 0, b')
10.           For j = 1 to M do {
11.               If (newE[b'] < E[b' − w_ij] + p_ij) {
12.                   newE[b'] ← E[b' − w_ij] + p_ij
13.                   If (b' − w_ij ≥ ⌊b/2⌋)
                              then newM[b'] ← M[b' − w_ij]
14.                   else newM[b'] ← (i, w_ij, b')
15.               }
16.           }
17.       }
18.       M ← newM; E ← newE;
19.   }
20.   Now M[b] contains MS(s,t,b,b)
end
```

Figure 11: Subroutines that evaluate $MSi(s,t,t,b)$ and $MSb(s,t,b,b)$. Note that parts of the computation is common between them, but presenting them separately is simpler to verify in terms of their respective recursive definition.

```
Algorithm XOpt(1,n,B)
begin
1.    Use the streaming algorithm [24] to identify the
      n' = B log M elements using O(BM) space.
2.    Renumber these to be 1, ..., n'.
      If an index i does not store coefficient j
      in the preprocessing step above,
      assume that coefficient to be zero.
3.    Initialize stack to (1, n', B).
4.    While stack is non-empty {
5.        Pop the top of stack, say (s,t,b)
6.        If t > s then {
7.            Let (b', w_j) ← FindMSi(s,t,b)
8.            Let (i', w'_j', b'') ← FindMSb(s,t,b)
9.            Store the best j coefficients (taking space
              w_j) corresponding to ⌊(t+s)/2⌋
10.           Store the best j' coefficients (taking space
              w'_j') corresponding to i'
11.           If (i' < (t+s)/2)
12.               Push (s, i' − 1, b'' − w'_j')
13.               Push (i' + 1, ⌊(t+s)/2⌋ − 1, b' − b'' − w_j)
14.               Push (⌊(t+s)/2⌋ + 1, t, b − b')
15.           elseif (i' > (t+s)/2)
16.               Push (s, ⌊(t+s)/2⌋ − 1, b' − w_j)
17.               Push (⌊(t+s)/2⌋ + 1, i' − 1, b'' − b' − w'_j')
18.               Push (i' + 1, t, b − b'')
19.           else /* i' = ⌊(t+s)/2⌋, and b'' = b', w_j = w'_j' */
20.               Push (s, i' − 1, b' − w_j)
21.               Push (i' + 1, t, b − b')
22.       } elseif t = s then {
23.           Let the largest j s.t. w_j ≤ b be j*
              Output the largest j* coefficients
              corresponding to s.
24.       }
25.   }
end
```

Figure 12: Algorithm XOpt

intuition alluded to in the double-cut theorem. Note that the depth of the stack is $\min\{\log n, \log B\}$ because both the parameters are decreasing by a factor.

Now we use Theorem 4.6, [24], which observes that we can restrict our attention to a set of elements which can be identified and stored by a streaming algorithm using $O(BM)$ space. The total number of elements can be bounded by $B+B/2+B/3\cdots+B/M$ elements which corresponds to $O(B \log M)$ elements. This summation is due to the fact that for $1 \leq j \leq M$ we store the best candidates corresponding to storing j indices. Each of these classes, corresponding to each j, uses at most $2B$ space.

The running time of the algorithm is $O(nM(\log M + \log B))$. The $\log B$ term can be eliminated by maintaining approximate heaps – instead of storing the top k items in a heap, we repeatedly read the next k elements of the stream and then retain the top k elements of the $2k$ elements; the other k elements are the top k elements of the previous part of the stream. This is similar to the algorithm in [23], we omit further details. Thus setting $n = \min\{n, B \log M\}$, the running time of the algorithm is $O(BM \min\{n, B \log M\})$ and the overall space requirement is $O(BM + B)$. Therefore we can conclude that:

Theorem 12 *Using algorithm XOpt we can find the best B space Extended wavelet summary of n indices with M coefficients each in $O(BM)$ space and time $O(nM \log M + BM \min\{n, B \log M\})$ in a single pass.*

9 Experimental Results

The main issue we investigated is the effect of the various additional data structures that were defined for the space efficient algorithms. We use both synthetic and real life data used in previous papers [13, 31, 27]. We did not write data to disk for the working space based algorithms – since organizing that data in the disk well is a nontrivial task and influences the running time of the working space based algorithms.

Measurables: We are interested in (i) the savings of space and (ii) the effect of that on time (the space efficient algorithms were sometimes faster). **Note** that the space component which changes across the algorithms is the size of the Dynamic Programming (DP) table. For the optimal DP algorithms, this is just allocating an array of size n or nB and the space savings is known beforehand – we checked the memory usage as the programs were running, and the total memory usage was exactly the size of the table (parts of it were virtual). So for the optimum algorithms we do not show the space usage graphs which contain relatively little information. **However** for the approximation algorithms the story is different, we plot the size of the DP table in the various implementations – this avoids discussion about representing values an `int`, `long int` or `double`. While these issues are important, the ideas in this apply to **any** encoding scheme and the relative sizes of the DP table is the robust measure. We note once again that the experiments were not trying to compare Wavelets and Histograms. They are useful synopsis techniques for data that is more suitable for each. That discussion is not the goal of this paper.

We first describe the data sets and then in Section 9.2 we present the results on optimum algorithms for V-Optimal histograms. In Section 9.3 we report the benefits of a space efficient algorithms in the context of the approximation algorithms. In Section 9.4 we consider data stream algorithms. The results show that the space efficient algorithms are the method of choice. In Section 9.5 we present the results on optimum maximum error wavelets. All experiments reported in this section were performed on a 2.4 GHz Intel Core2Duo machine with 1 GB of main memory, running Fedora 6.0. All the methods were implemented using GCC compiler of Version 4.1.1 and all implementations were single threaded.

9.1 Data Sets

We use both synthetic and real life data. The datasets are described and used in [12, 13, 31, 28], we repeat some of the description for the sake of completeness.

Synthetic Data Sets: The synthetic data sets were generated with Zipfian frequencies for various levels of skew that is controlled by the z parameter of the Zipfian. The z parameter values between 0.3 (low skew) and 1.5 (moderate skew), the distinct values between 256 ($= 2^8$) and 16384 ($= 2^{16}$), and the tuple count was set to 1,000,000. Note that the time and space complexities are not dependent on number of tuples and thus we did not vary this parameter.

A permutation step was also applied on the produced Zipfian frequencies to decide the order of frequencies over the data domain. We mostly show results with the `Normal` permutation as described in [12] where the higher frequencies are at the center of the domain. We include details on `Random`, `noPerm` and `PipeOrgan` permutations wherever appropriate. The names `Random`, `noPerm` are self-explanatory, in

case of `PipeOrgan` the high frequencies are at the two ends of the domain.

Real Life Data Sets: We also experimented with real-life data sets. We used the Dow-Jones Industrial Average (DJIA) data set available at StatLib, that contains Dow-Jones Industrial Average (DJIA) closing values from 1900 to 1993. There were a few negative values (e.g. -9), which we removed. We focused on prefixes of the dataset of size upto 16384.

9.2 Experimental study: Optimum Histograms

In this section we present the results of our ideas applied to V-Opt histogram construction algorithms. We first describe the algorithms we compare.

- **VOPT-EO:** This is a baseline implementation which uses $O(n)$ space, and does not maintain the table (and therefore cannot compute the buckets). The running time of the algorithm is a good lower bound on all $O(n)$ space algorithms.
- **VOPT-R:** It represents the $O(n^2B^2)$ time and $O(n)$ space algorithm given in [38] which computes the error as well as the buckets. This algorithm uses the VOPT-EO algorithm recursively.
- **SpaceOpt:** It represents the space efficient algorithm given in Figure 4.

Space Usage: The space usage of the $O(nB)$ space V-Opt algorithm for $n = 4096$ jumped from $2.8MB$ to $20MB$ as B was raised from 10 to 100. We used `double` and declared a nB size array for the DP (along with other structures for storing input). This is expected because the optimal DP defines a table which is 10 times larger. The space did not go up by exactly that amount because some fraction of the $2.8MB$ was used for other data structures, for example storing the data. The algorithms VOPT-EO and VOPT-R used $1.4MB$ and SpaceOpt used $1.6MB$ for entire range of settings of B. The space benefit is quite clear and as is the theme of this paper, we focus only on the algorithms which have $O(n)$ space complexity.

Ruling out VOPT-R: In Figure 13(a) we present the running times of the algorithms for $n = 4096$, for the Normal permutation and skew 1. The relative gap between VOPT-R and the other algorithms remained the same across different skew and permutation settings. This clearly demonstrates that VOPT-R is not a good alternative. *Since the algorithm VOPT-R is much slower than the others to be attractive, we do not include that algorithm for any further experiments.* The Space-Opt algorithm holds up well, and has running time no more than twice that of VOPT; but of course VOPT-EO only computes the answer and Space-Opt computes the entire histogram.

Dependence on Skew: We also investigate the dependence of skew keeping B, n a fixed constant, this is presented in Figure 13(b). The Pipe and NoPerm behaved the same as Normal permutation and we omit those results.

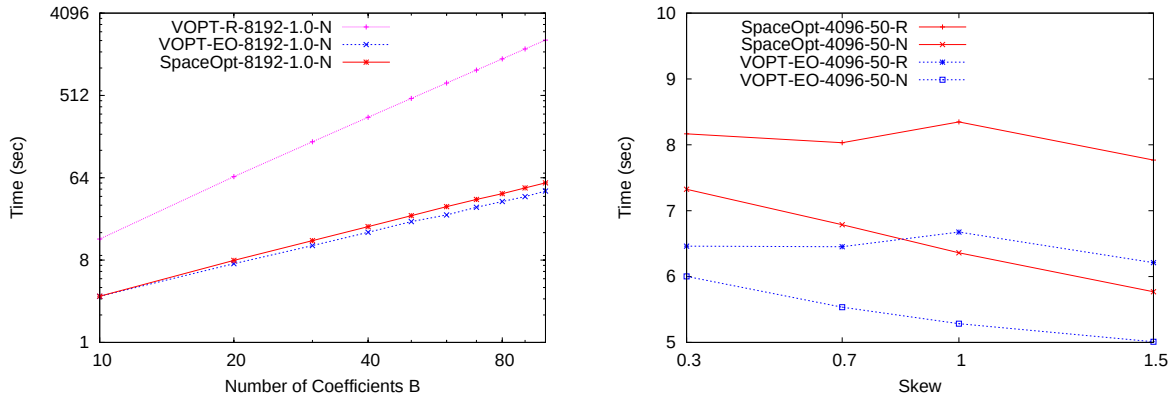
Dependence on B: We investigate the effect of B keeping n fixed. The various skew setting defined a narrow band of regions which is shown in Figure 14(a), and (b). The Random permutation defined a narrower band than the other, the permutations Pipe and NoPerm behaved the same as Normal permutation and we omit those results. Notice that the running time of SpaceOpt is about 1.3 times that of VOPT.

Dependence on n and real life dataset: Figure 15(a) shows the dependence on n, and the running time of SpaceOpt is close to that of VOPT-EO. However in the real life dataset, it became clear that the running time of SpaceOpt was the same as that VOPT on half the B parameter. This is consistent with the linear (in B) behavior of the algorithms.

Summary: Optimum Histograms The results were almost as predicted by the analysis, except that SpaceOpt performed better than twice the time of VOPT-EO as predicted, because of the pruning conditions kicking in, in the synthetic datasets. In the real life dataset, SpaceOpt performed the same as VOPT on half the B parameter, and the log-log plot shows the $O(n^2B)$ behavior of these algorithms. However as predicted, we got almost factor of B improvement in the space, because we required a significantly shorter DP table.

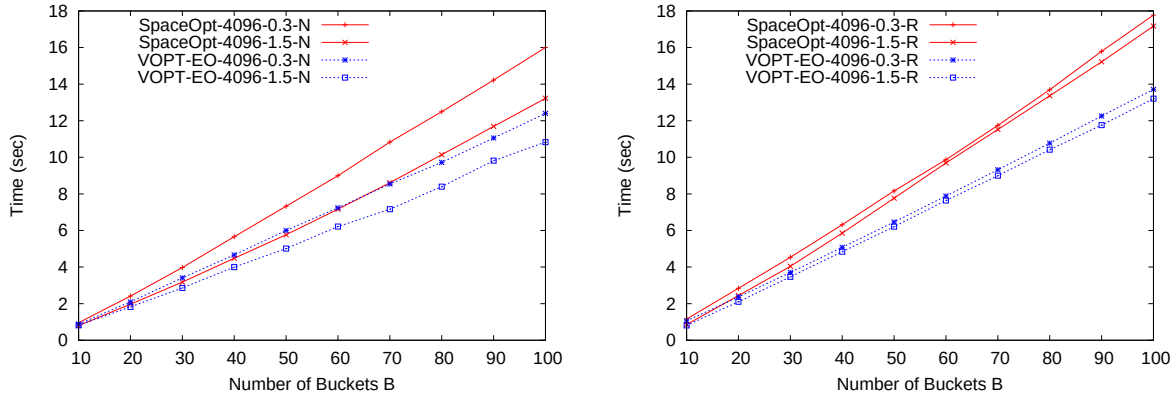
9.3 Experimental study: The Space-Time of Approximate Histograms

In this section we present the results of our ideas applied to approximate V-Opt histogram construction algorithms. We first describe the algorithms we compare.



(a) Normal Permutation, Skew=1 (b) Varying permutation and Skew

Figure 13: Running times for the algorithms and the dependence on skew



(a) Normal Permutation (b) Random Permutation

Figure 14: Running times based on B, $n = 4096$, the other permutations behaved similar to Normal.

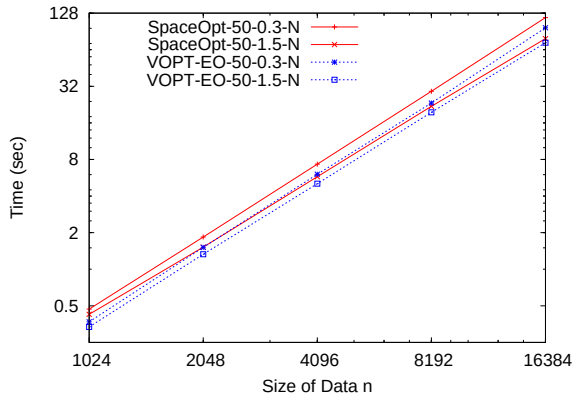
- **ApxDelta:** This is the linear time and linear space offline approximation AHIST-L-Δ introduced in [28].
- **SpaceDelta:** It represents the space efficient algorithm StackSpace-Δ given in Figure 7.

Dependence on skew: In Figure 16 we present the running times of the algorithms for $n = 16384$ and various settings of the skew in the synthetic data. This setting of n clearly requires the approximation algorithm, the quadratic algorithm will simply bee too slow. We show the effect of Normal and Random permutations (we have already seen that the other permutations behave similar to the Normal one). It is striking to see that for Random permutations that the space usage drops by a factor of $25 = B/2$ for the experiments. Although this was predicted, the theoretical gap of $\Theta(B)$ is not realized in the other data sets because the smoothness of the data enabled the pruning conditions to prune out many comparisons. Note that the running time of the space efficient algorithm is close to the other algorithm.

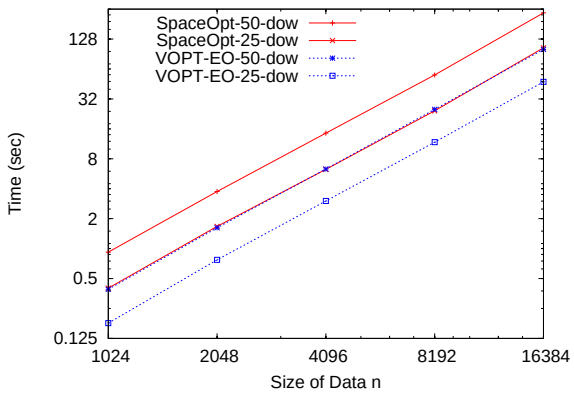
Other dependencies and Real life data sets: To save space, we show the other dependencies using the real life data set DJIA, the synthetic datasets show the same trends. In fact the running time difference between the two algorithms showed a maximum gap for the real life data set, but was below a factor of 2, and we establish that in the set of experiments shown in Fig 17.

Summary: Approximate Histograms

- The space time tradeoff of the approximation algorithms are richer than that of the optimum case. The space efficient algorithm provides a



(a) Normal Permutation



(b) Prefixes of DJIA dataset

Figure 15: Running times based on $B = 50$, skew $= 1$, as n is varied. All permutations and all skew settings behaved similarly.

significant space savings, Figures 16(b),(d) and 17(b),(d).

- Surprisingly the space efficient algorithm actually ran faster on the synthetic data! Figures 16(a),(c) demonstrate that the space efficient algorithm is faster for small ϵ and large skew – this is because the approximate DP table becomes significantly small (see Figures 17(b),(d)) and very likely these get cached. Note that this was not seen for the optimum algorithms because the table sizes were not this small. This is yet another example where a savings in space translates in savings of I/O and eventually in time.

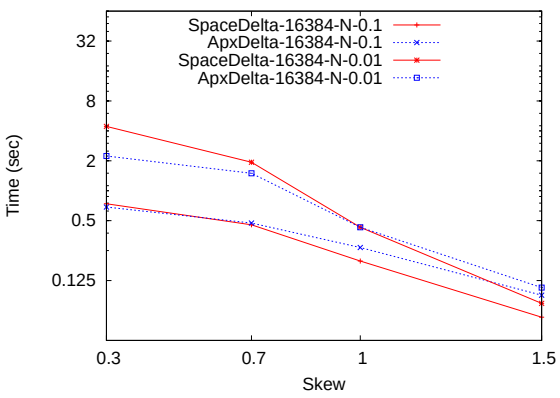
- On the real life dataset algorithm StackSpace-Δ takes no more time than that of AHIST-L-Δ on twice the input size, which is almost twice that of AHIST-L-Δ on the same size data. This is not surprising, but validates the linear running time of all the algorithms involved.

- Figure 17(b) shows that the DP table size depends mainly on B – this dependence goes away when ϵ is decreased to 0.01. This is because the B/ϵ term becomes comparable to n and the table size cannot grow beyond n. But for the larger ϵ the effect of n on the approximate DP table size is quite small.

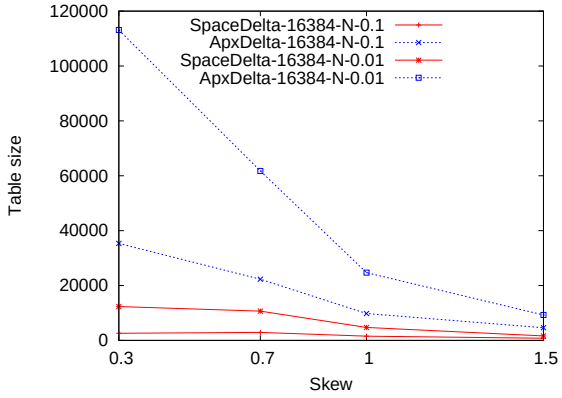
9.4 Experimental study: The space time of Approximate Histograms for Data Streams

In this section we present the results of our ideas applied to approximate V-Opt histogram construction algorithms for data streams. We first describe the algorithms we compare. Note that **all** the algorithms (i) store the largest $C = B\epsilon^{-2}(\log n)\log\frac{2}{\epsilon}$ (exact number, no other constants) wavelet coefficients and (ii) all the algorithms use a DP approach where a provably approximate DP table is stored. Based on this it is clear that the right measure to investigate is the size of the DP table. However the term $B\epsilon^{-2}(\log n)\log\frac{2}{\epsilon}$ does grow very fast, therefore we need to keep $n \geq 16384$ and $\epsilon = 0.5$ to get $B \leq 80$ as a feasible range beyond which $C \geq n/2$, which is not meaningful in a streaming setting. This may be useful for other applications such as image compression, but then we should be using the linear space offline algorithms. Although this setting of error(ϵ) appears large, we measured the accuracy of the algorithms and report it. We note that there is a play in the number of coefficients, but then there is no guarantee on what that heuristic would do on a particular dataset. However, the main ideas we are exploring here are orthogonal to the exact setting of the C, as long as it is moderate enough – even 500 or so.

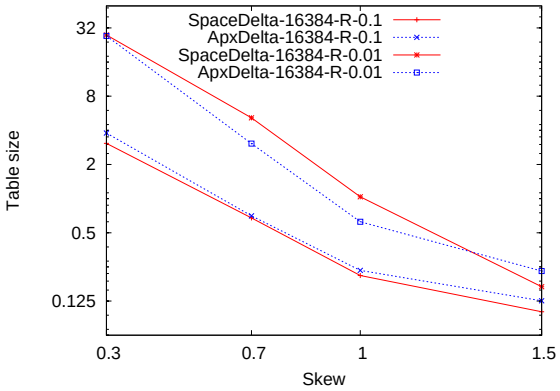
- **WaveHist:** This is the implementation of the algorithm described in [23, 47] of choosing the exact above mentioned number of coefficients and then using the optimal VOPT histogram algorithm of [38].



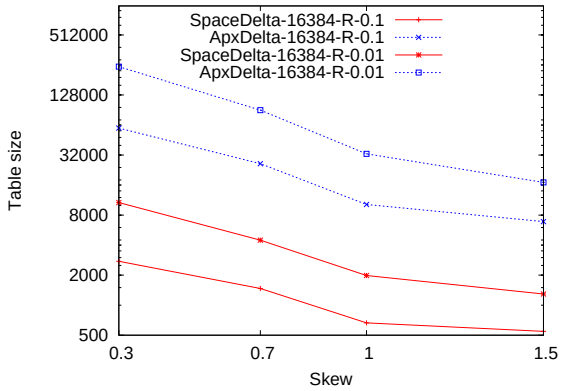
(a) Running Time, Normal Permutation



(b) Approximate DP Table, Normal Permutation



(c) Running Time, Random Permutation



(d) Approximate DP Table, Random Permutation

Figure 16: Space time tradeoff of approximation algorithms as the Skew is varied, $B = 50$. Note that for $\epsilon = 0.01$ the space use of the older approximation algorithm is high since $\frac{B^2}{\epsilon} \log n$ is large. This drops when $\epsilon = 0.1$. Note that we are not reporting the space to store the data because that is common to all algorithms, the entries in the approximate DP table contains similar information for both the algorithms and thus the number of entries is the critical part.

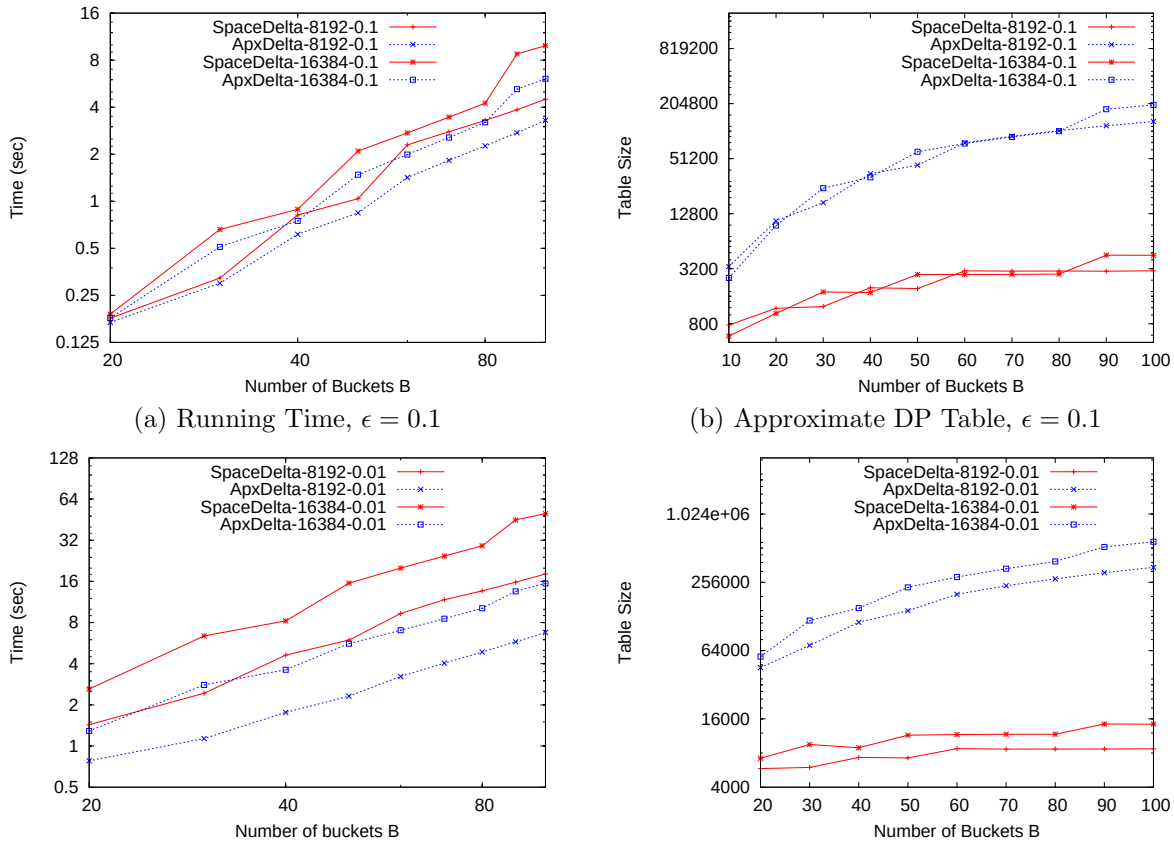
- **WaveApxHist:** This is the linear time streaming approximation AHIST-B introduced in [28]. We set the blocksize BLK to be 1024 (the results for block size 2048 were near identical). The C coefficients were first collected, and then we simulated a "virtual stream" that was fed to the algorithm AHIST-B Ṅote that this virtual stream needed to be generated BLK elements at a time and was relatively easy. This blocksize is not counted in the DP table size.

- **SpaceApxWaveHist:** It represents the space efficient algorithm streaming algorithm discussed in Section 5.2.

Synopsis Error: We begin with the following observation: even though the ϵ parameter we started with was large, the three algorithms had **identical** error on the synthetic datasets and the error on the DJIA set is plotted in Figure 18(a) – the algorithms were indistinguishable in this aspect, which was expected.

Running time: Figure 18(b) and (d) show the running times for the three algorithms. The log-log plot shows that the dependence on B is between quadratic and cubic, as expected. Note that the algorithm AHIST-B already gives almost an order of magnitude improvement in the synthetic data. The space efficient algorithm devised in this paper gives even more benefit – for the DJIA data the benefit is about an order of magnitude.

Space Efficiency: The savings in space are spectacular! It is better than the simplest strategy by three orders of magnitude and even better than the



(a) Running Time, $\epsilon = 0.1$

(b) Approximate DP Table, $\epsilon = 0.1$

(c) Running Time $\epsilon = 0.01$

(d) Approximate DP Table, $\epsilon = 0.01$.

Figure 17: Space time of approximation algorithms as B and n are varied on the DJIA dataset. Note that the space efficient algorithm for $n = 8192$ takes almost the same running time as the AHIST-L-Δ algorithm for $n = 16384$. This is the factor of 2 gap we expected. Observe that the space usage decreases by factors of $\approx B$, specially for $\epsilon = 0.01$ which forces a large table.

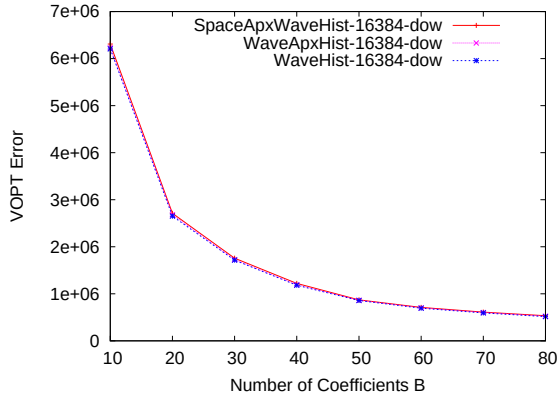
sophisticated AHIST-B – almost by a factor B, as is predicted. We note that the DP table is not the only data structure we need, but it is the most dominant term in the analysis of WaveHist and WaveApxHist.

Summary: Data Stream (Approximate) Histograms Two points immediately come to the fore from the above (i) although it has been shown before, the above reiterates that approximation is a great tool in the context of synopsis construction and already helps in improving running time and space (ii) space efficient algorithms as put forward in this paper stand to reduce the space cost of synopsis construction, and in that process improve the overall running time.

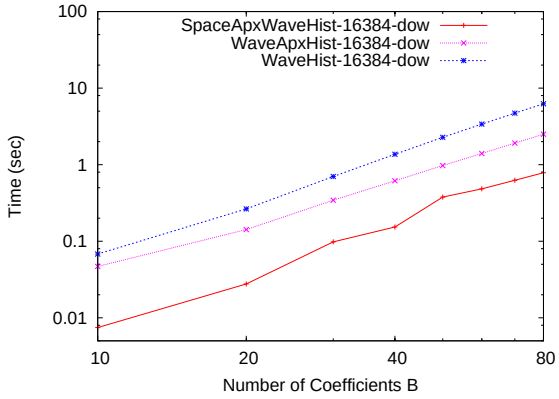
Several interesting open problems arise in this context. First, is there an analogous provable "presynopsis" for other measures? Second the best previous analysis [28] shows this to be $\tilde{O}(B^2/\epsilon)$. The algorithm we present here has space complexity $\tilde{O}(B/\epsilon^2)$. The former is better at low B small ϵ scenario and the latter is beneficial in a large B large ϵ scenario. This of course begs the question that is a $\tilde{O}(B/\epsilon)$ space $(1+\epsilon)$ algorithm possible for the VOPT error – this remains an interesting open question in this area.

9.5 Experimental study: Wavelets

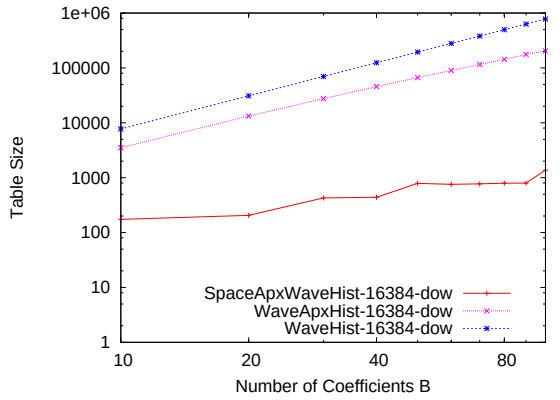
For the Wavelet synopsis construction problem we show only the results for the maximum relative error measure. We could not use the $O(n^2B)$ space algorithm described in [11] since we ran out of memory quickly. As mentioned earlier, choosing parameters of a working space based algorithm is always unclear and biases comparisons; we eschewed the approach. We therefore restrict ourselves to comparing the $O(B \log n)$ space *WaveOpt()* algorithm and



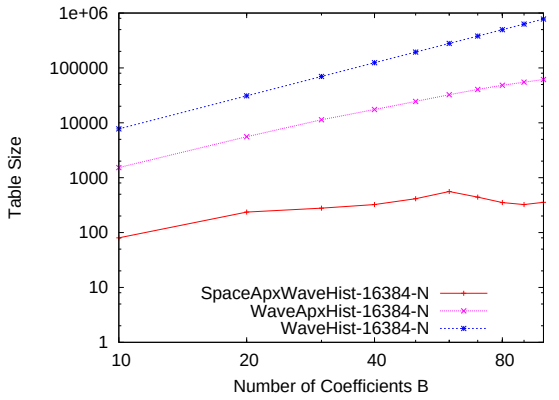
(a) DJIA, $n = 16384$, VOPT Error



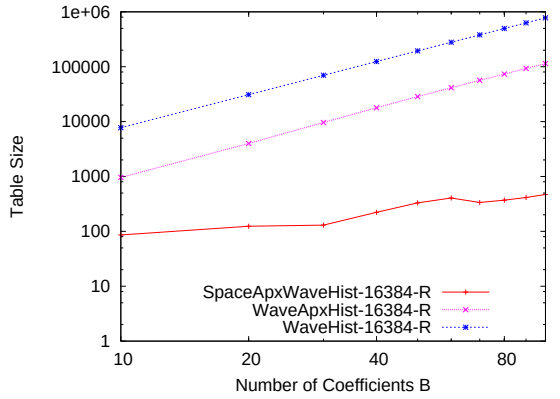
(b) Running time



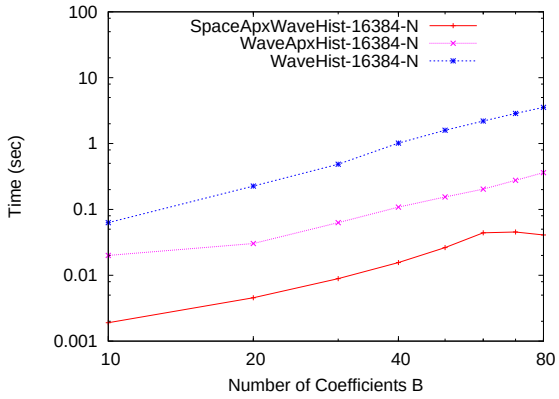
(c) DP table size, DJIA



(d) DP table size, Normal Permutation

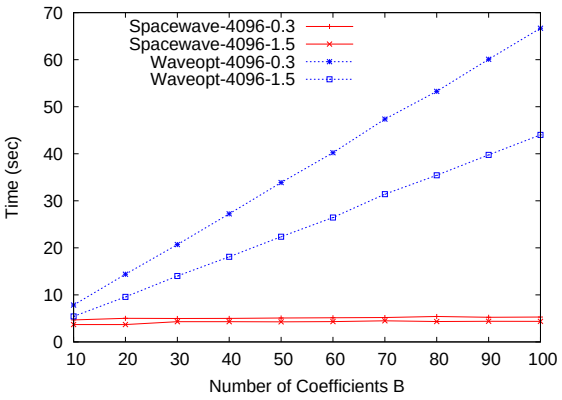


(c) DP table size, Random Permutation

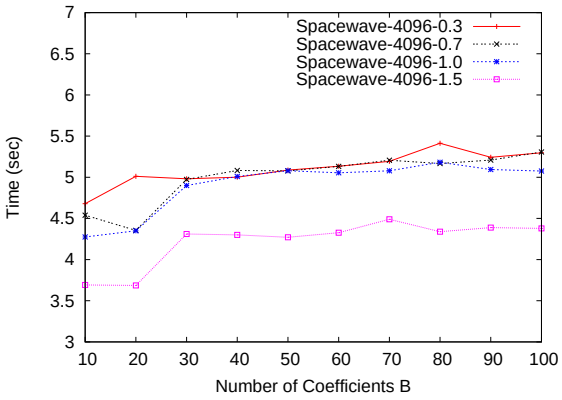


(d) Running time, Normal Permutation

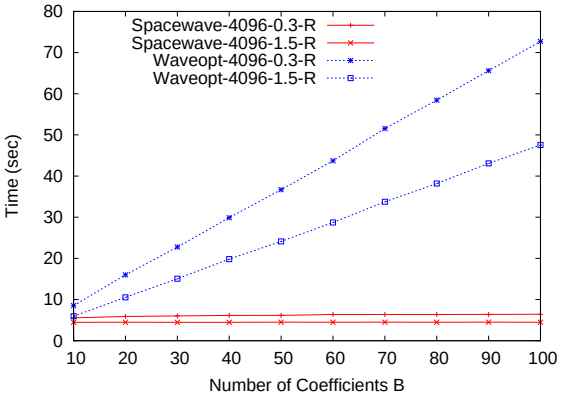
Figure 18: Results on Data Stream algorithms, $n = 16384$ for all of the above. The synthetic data has $Skew = 1$



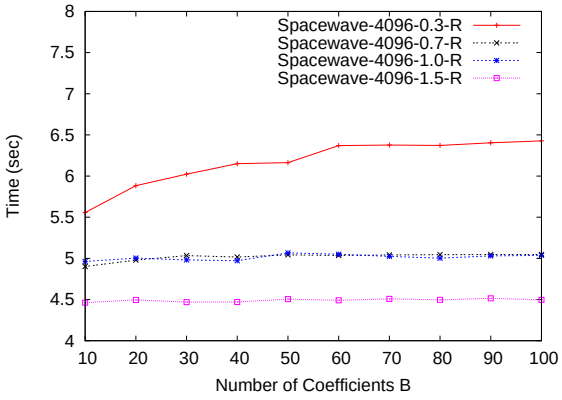
(a) Normal Permutation, Both algorithms



(b) Normal Permutation, SpaceWave only

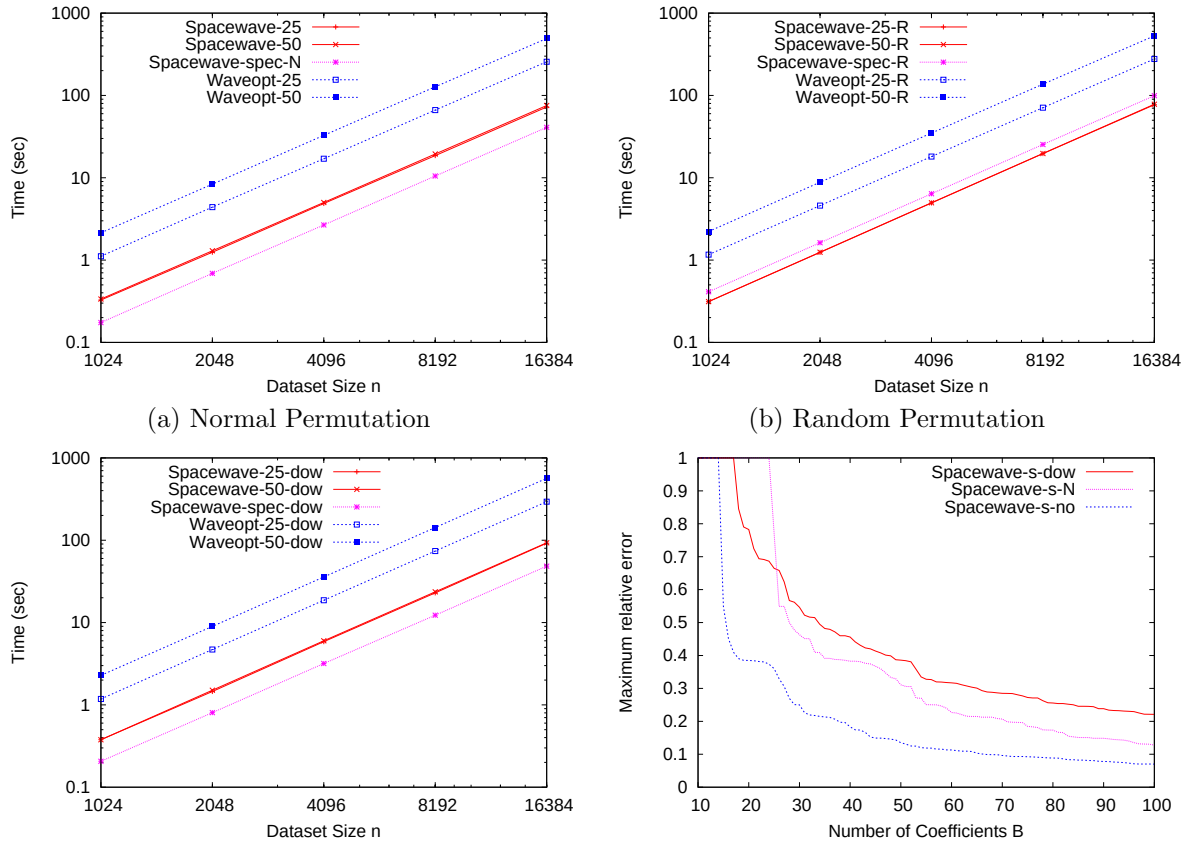


(c) Random Permutation, Both algorithms



(d) Random Permutation, SpaceWave only

Figure 19: Running times as skew varies, Normal Permutation, $n = 1024$



(a) Normal Permutation (b) Random Permutation

(c) Dow data (prefixes) (d) Spectrum

Figure 20: Running times as function of n

the optimized algorithm *SpaceWave()*. As discussed earlier, the algorithm *WaveOpt()* serves as a lower bound/baseline for any algorithm that computes the same table as in [11]. We reiterate that the algorithms compute the same quantity and we only compare the running times.

- **WaveOpt:** This algorithm is described in Section 7.2. Since the algorithm is only a baseline, as mentioned earlier, we did not compute the coefficients/answer but the final error only.
- **SpaceWave:** It represents the space efficient algorithm with the optimization mentioned in Section 7.3. This algorithm computes the error as well as the coefficients, and the final reconstruction based on those coefficients.

Dependence on Skew We present the comparison of running times for various settings of the skew parameter in Figure 19. The performance of the algorithms for the Random permutation was different compared to the other permutations which were similar in nature.

Dependence on n and real life data sets: We present the comparison of running times if B is fixed and we vary n Figure 20. Interestingly Figure 20(d) shows that the DJIA dataset has a behavior similar to the noPerm data for small number of coefficients and then switches to a behavior similar to Normal permutation. This is expected, because the overall trend in DJIA is Zipfian, which is what is realized in the small number of coefficients. For larger number of coefficients, the deviations from the Zipfian determine the error and has a stronger effect.

Summary: Wavelets

- The Wavelet algorithms were more insensitive to skew, Figures 19, compared to Histograms, Figure 13. Once n, B were fixed, the algorithm has little variation (except the binary search) that depends on the data. There are no obvious pruning strategies which we believe remains an open problem in this area, i.e., can the wavelet algorithms be made more data driven? Note that some progress has been achieved in [22] in

the context of unrestricted synopses.

- However the Random permutation behaved differently compared to the other permutations. First the binary search affected the random permutation significantly (at low skew). Second, although usually, it took less less time to compute the spectrum than to evaluate the coefficients for a fixed B since there were no recursion/recomputation involved (see Figures 20(a) and (c)) – the Random permutation was an exception to that rule. This is because the spectrum computation computes the optimum allocation of all n possible coefficient allocation values. In case of the Random permutation, in absence of any smoothness, this process took significantly longer than the recomputation required for small number of coefficients.

- All the algorithms are (approximately) quadratic in n as can be seen from the slopes of the lines in the log-log plot in Figure 20(a),(b) and (c).

- The running time of WaveOpt is $O(n^2 B \log B)$. Figures 19(a) and (c) verify that the running time is almost linear in B.

- SpaceWave is the clear winner and its running time is almost independent of B as the analysis suggested.

10 Summary

In this paper we took a fresh look at DP techniques for synopsis construction problems. We provided a novel framework of divide and conquer and DP algorithms to reduce the space bounds for Wavelet and histogram synopsis construction problems to $O(n)$. We provided the best wavelet reconstruction algorithm which is independent of B in space and running time. We also provided the first $O(n)$ space $O(n^2 B)$ time algorithm for V-opt histograms and its generalizations. We indicated (briefly) how the ideas affect other synopsis construction problems as well.

Acknowledgments

We thank Rajeev Motwani, Sampath Kannan, Nick Koudas, Kamesh Munagala, S. Muthukrishnan, Kyuseok Shim and Torten Suel for discussions that were extremely useful in improving the presentation. We would also like to thank Boulos Harb, Chulyun Kim, Hyoungmin Park, Jungchul Woo in helping us with various projects on synopsis construction which has led to sharpened intuition on these problems. We would like to thank Kyuseok Shim doubly, for providing datasets etc., without which this work would never reached fruition.

References

[1] Acharya, S., Gibbons, P., Poosala, V., Ramaswamy, S.: The Aqua Approximate Query Answering System. Proc. of ACM SIGMOD (1999)

[2] Aggarwal, A., Alpern, B., Chandra, A., Snir, M.: A model for hierarchical memory. Proceedings of the Symposium on Theory of Computing (STOC) pp. 305–314 (1987)

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

[4] Arge, L.: External memory data structures. Proc. of ESA pp. 1–29 (2001)

[5] Chakrabarti, K., Garofalakis, M.N., Rastogi, R., Shim, K.: Approximate query processing using wavelets. In: Proceedings of the International Conference on Very Large Databases (VLDB) (2000)

[6] Chakrabarti, K., Keogh, E.J., Mehrotra, S., Pazzani, M.J.: Locally adaptive dimensionality reduction for indexing large time series databases. ACM TODS **27**(2), 188–228 (2002)

[7] Chou, H.T., DeWitt, D.J.: An evaluation of buffer management strategies for relational database systems. pp. 127–141 (1985)

[8] Daubechies, I.: Ten Lectures on Wavelets. SIAM (1992)

[9] Deligiannakis, A., Roussopoulos, N.: Extended wavelets for multiple measures. In: SIGMOD Conference (2003)

[10] Denning, P.J.: The working set model for program behaviour. CACM **11**(5), 323–333 (1968)

[11] Garofalakis, M., Kumar, A.: Deterministic wavelet thresholding for maximum error metric. Proc. of PODS (2004)

[12] Garofalakis, M.N., Gibbons, P.B.: Wavelet synopses with error guarantees. In: Proc. of ACM SIGMOD (2002)

[13] Garofalakis, M.N., Gibbons, P.B.: Probabilistic wavelet synopses. ACM TODS **29**, 43–90 (2004)

[14] Gibbons, P., Matias, Y.: Synopsis data structures for massive data sets. Proceedings of the ACM-SIAM Symposium on Discrete Algorithms pp. S909–S910 (1999)

[15] Gibbons, P., Matias, Y., Poosala, V.: Fast Incremental Maintenance of Approximate Histograms. Proc. of VLDB, Athens Greece pp. 466–475 (1997)

[16] Gilbert, A., Kotadis, Y., Muthukrishnan, S., Strauss, M.: Surfing Wavelets on Streams: One Pass Summaries for Approximate Aggregate Queries. Proceedings of VLDB pp. 79–88 (2001)

[17] Gilbert, A.C., Guha, S., Indyk, P., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Fast, small-space algorithms for approximate histogram maintenance. In: Proc. of ACM STOC (2002)

[18] Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Optimal and approximate computation of summary statistics for range aggregates. In: Proc. of ACM PODS (2001)

[19] Guha, S.: Space efficiency in synopsis construction algorithms. Proceedings of the International Conference on Very Large Databases (VLDB) (2005)

[20] Guha, S., Harb, B.: Wavelet synopsis for data streams: Minimizing non-euclidean error. Proc. of SIGKDD Conference (2005)

[21] Guha, S., Harb, B.: Approximation algorithms for wavelet transform coding of data streams. Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) (2006)

[22] Guha, S., Harb, B.: Approximation algorithms for wavelet transform coding of data streams. Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) (2006)

[23] Guha, S., Indyk, P., Muthukrishnan, S., Strauss, M.: Histogramming data streams with fast per-item processing. In: Proc. of ICALP (2002)

[24] Guha, S., Kim, C., Shim, K.: XWAVE: Optimal and approximate extended wavelets for streaming data. Proceedings of the International Conference on Very Large Databases (VLDB) (2004)

[25] Guha, S., Kim, C., Shim, K.: XWAVE: Optimal and approximate extended wavelets for streaming data. Extended version of [24] (2005)

[26] Guha, S., Koudas, N.: Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. In: Proc. of ICDE (2002)

[27] Guha, S., Koudas, N., Shim, K.: Data Streams and Histograms. In: Proc. of STOC (2001)

[28] Guha, S., Koudas, N., Shim, K.: Approximation algorithms for histogram construction problems. ACM TODS **31**(1), 396–438 (2006)

[29] Guha, S., Koudas, N., Srivastava, D.: Fast algorithms for hierarchical range histogram construction. In: Proc. of ACM PODS (2002)

[30] Guha, S., Park, H., Shim, K.: Wavelet synopsis for hierarchical range queries with workloads. Manuscript. Email for copy. (2005)

[31] Guha, S., Shim, K., Woo, J.: REHIST: Relative error histogram construction algorithms. Proceedings of the International Conference on Very Large Databases (VLDB) (2004)

[32] Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. In: SIGMOD Conference (1997)

[33] Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. CACM **18**(6), 341–343 (1975)

[34] Hochbaum, D. (ed.): Approximation Algorithms for NP Hard Problems. Brooks/Cole Pub. Co (1996)

[35] Ioannidis, Y., Poosala, V.: Balancing Histogram Optimality and Practicality for Query Result Size Estimation. Proc. of ACM SIGMOD (1995)

[36] Ioannidis, Y.E.: Universality of serial histograms. In: Proc. of the VLDB Conference (1993)

[37] Ioannidis, Y.E.: The history of histograms (abridged). Proc. of VLDB Conference pp. 19–30 (2003)

[38] Jagadish, H.V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K.C., Suel, T.: Optimal Histograms with Quality Guarantees. In: Proc. of the VLDB Conference (1998)

[39] Jagadish, H.V., Lakshmanan, V.S., Srivastava, D.: What can hierarchies do for data warehouse? Proceedings of the International Conference on Very Large Databases (VLDB) (1999)

[40] Karras, P., Mamoulis, N.: One pass wavelet synopis for maximum error metrics. Proceedings of the International Conference on Very Large Databases (VLDB) (2005)

[41] Kooi, R.P.: The Optimization of Queries in Relational Databases. PhD Thesis, Case Western Reserve University (1980)

[42] Koudas, N., Muthukrishnan, S., Srivastava, D.: Optimal histograms for hierarchical range queries. In: Proc. of ACM PODS (2000)

[43] Matias, Y., Urieli, D.: Personal communication (2004)

[44] Matias, Y., Vitter, J.S., Wang, M.: Wavelet-Based Histograms for Selectivity Estimation. Proc. of ACM SIGMOD (1998)

[45] Muralikrishna, M., DeWitt, D.J.: Equi-depth histograms for estimating selectivity factors for multidimensional queries. Proc. of ACM SIGMOD, Chicago, IL pp. 28–36 (1998)

[46] Muthukrishnan, S.: Workload optimal wavelet synopsis. DIMACS TR (2004)

[47] Muthukrishnan, S., Strauss, M.: Approximate histogram and wavelet summaries of streaming data. DIMACS TR 52 (2003)

[48] Muthukrishnan, S., Strauss, M.: Rangesum histograms. Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) (2003)

[49] Poosala, V., Ioannidis, Y., Haas, P., Shekita, E.: Improved Histograms for Selectivity Estimation of Range Predicates. Proc. of ACM SIGMOD, Montreal Canada pp. 294–305 (1996)

[50] Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Proc. of the ACM SIGMOD, pp. 23–34 (1979)

[51] Smith, A.J.: Cache memories. ACM Comput. Surv. **14**(3), 473–530 (1982)

[52] Vazirani, V.: Approximation Algorithms. Springer Verlag (2001)

[53] Vitter, J.S.: External memory algorithms and data structures: dealing with massive data. ACM Comput. Surv. **33**(2), 209–271 (2001)