DIMACS Technical Report 2005-19
May 2005

Workload-Optimal Histograms on Streams

by

S. Muthukrishnan[1,2] M. Strauss[3,4] X. Zheng[5]

[1]Permanent Member
[2]Supported by NSF ITR 0220280 and NSF 0354690. Rutgers University. muthu@cs.rutgers.edu.
[3]Permanent Member
[4]Supported by NSF DMS 0354600. Dept. of Mathematics, University of Michigan, 2074 East Hall, 530 Church St., Ann Arbor, Michigan, 48109-1403. Fax: +1-734-763-0937. martinjs@umich.edu
[5]Supported by NSF DMS 0354600. University of Michigan. xuanzh@eecs.umich.edu

ABSTRACT

Histograms are used in many ways in conventional databases and in data stream processing for summarizing massive data distributions. Previous work on constructing histograms on data streams with provable guarantees have not taken into account the workload characteristics of databases which show some parts of the distributions to be more frequently used than the others; on the other hand, previous work for constructing histograms that do make use of the workload characteristics—and have demonstrated the significant advantage of exploiting workload information—have not come with provable guarantees on the accuracy of the histograms or the time and space bounds needed to obtain reasonable accuracy.

We study the algorithmic complexity of constructing workload-optimal histograms on data streams, and expose the effect of the workload on the complexity of histogram construction problems.

Consider the case when the workload is explicitly stored and the input data is streamed in the *time series* model—the input is a vector of N components, and we are presented with the components, one at a time, in order. We present an algorithm that uses polylogarithmic space, processes each new item in constant time, and, in polylogarithmic post-processing time, determines a $(1+\epsilon)$-approximation to the optimal histogram. This matches the space complexity, up to polylogarithmic factors, of the histogram construction on the stream when workload is uniform [10]. To get this result, we need to identify and exploit the notion of *linearly robust* histograms. These is the first known algorithmic result with provable guarantees for workload-optimal histogram construction on streams.

Now consider the case when workload is not stored fully, but is compressed. As we show, trivial lossy compression can be used (one can drop low-order bits of individual weights) but algorithmic results such as the above are not possible if any *nontrivial lossy* compression is used. However, we show that our algorithmic results can be extended efficiently to the case when the workload is compressed *without loss* by using, for example, a universal compression scheme of Lempel-Ziv. This requires supporting a *symbol-range-count* data structure on compressed data which may be of independent interest. Also, the direction of exploring data stream problems when one of the inputs (workload) is compressed losslessly, is novel.

1 Introduction

A histogram is a piecewise-constant approximation of an observed data distribution. A histogram is used as a small-space, approximate synopsis of the underlying data distribution, which is often too large to be stored precisely. Histograms have found many applications in database management systems, perhaps most commonly for query selectivity estimation in query optimizers [15], but have also found applications in approximate query answering [2], load balancing in parallel join execution [25], mining time-series data [18], partition-based temporal join execution, query profiling for user feedback, etc. Ioannidis has a nice overview of the history of histograms, their applications, and their use in commercial DBMSs [14]. Also, Poosala's thesis provides a systematic treatment of different types of histograms [25]. Formally:

Definition 1 *A B-bucket* histogram $\mathbf{H}$ *of length N is a partition of $[0, N)$ into intervals $[b_0, b_1) \cup [b_1, b_2) \cup \cdots \cup [b_{B-1}, b_B)$, where $b_0 = 0$ and $b_B = N$, together with a collection of B heights h_j, for $0 \leq j < B$, one for each bucket. A point query $\mathbf{A}[i]$ to $\mathbf{H}$ returns the estimate h_j where $b_j \leq i < b_{j+1}$.*

In building a B-bucket histogram, we want to choose $B - 1$ boundaries b_j and B heights h_j, dependent on $\mathbf{A}$. A number of different criteria are known [25] for choosing b_j's and h_j's; a popular and effective one is the *V-Opt histogram* [16], where b_j's and h_j's are chosen to minimize the total square error, taken *uniformly over the set of all point queries*, or, equivalently, $\|\mathbf{A} - \mathbf{H}\|^2 = \sum_i (\mathbf{A}[i] - h_{j(i)})^2$. (Once we have chosen the boundaries, the best bucket height on an interval I is the average of $\mathbf{A}$ over I.)

In [17], the authors presented an $O(N^2 B)$ time algorithm for determining the optimal histogram $\mathbf{H}_{\mathrm{opt}}$ that minimizes the total square error. This algorithm has two drawbacks.

- It is expensive—quadratic in N. In order to overcome this drawback, focus has been on $(1+\epsilon)$-*approximations*, that is, algorithms to find a histogram $\mathbf{H}$ such that $\|\mathbf{A}-\mathbf{H}\|^2 \leq (1 + \epsilon)\|\mathbf{A} - \mathbf{H}_{\mathrm{opt}}\|^2$.
- It needs $\mathbf{A}$ to be stored explicitly which is prohibitive in space for large distributions where histograms are used as synopses.

In order to overcome the second drawback, the focus has been on the *data stream model* of computation where (a) the algorithm reads the signal left to right in one pass as $\mathbf{A}[1], \mathbf{A}[2], \ldots$, using space *polylogarithmic* in the input length N; this is the so-called *time-series model* [22], or, (b) $\mathbf{A}$ is specified as a series of *updates* and the algorithm has to track the changes to $\mathbf{A}$ in space and time per update *polylogarithmic* in input N; this is the so-called *cash register model* if only additions are allowed, or more generally, the *dynamic maintenance model* if both additions and deletions are allowed [22]. Notice that with the polylogarithmic space requirement, the input can only be represented *lossily* since accurate representation of the signal or the workload will need at least linear space in the worst case.

Besides the parameters B, N, and ϵ, the algorithms' costs depend on the numerical precision involved; we let M be a parameter such that $\log(M)$ is roughly the number of bits of precision used (see below for a formal definition). A series of $(1+\epsilon)$-approximation algorithms have been proposed that work in time in $O(N+\mathrm{poly}(B, \log N, \log M, 1/\epsilon))$ in the time series model using $\mathrm{poly}(B, \log N, \log M, 1/\epsilon)$ space enroute [12, 11, 10]. In the dynamic maintenance model, the authors in [8] present an algorithm that uses time per update, space, and post-processing time time $\mathrm{poly}(B, \log N, \log M, 1/\epsilon)$. This solves the approximate V-Opt histogram computation problem from a theoretical point of view, modulo getting the constants involved to be as small as possible.

It has, however, long been an issue that the V-Opt histogram as defined above is limited in its applications because it does not take into account the *workload* of queries for which the histogram is optimized. In particular, when some of the point queries are more frequent than the others, the histogram needs to be better at approximating answers to the frequent queries rather than the infrequent ones. In other words, the metric to minimize is not the sum of squared errors *uniformly* over all point queries, but that obtained by weighting the error on each point query by the workload of how frequently each point query is posed. Formally:

Definition 2 *Given an input signal* $\mathbf{A}[0 \cdots N-1]$ *and* workload $w[0 \cdots N-1]$, $0 \le w_i \le 1$, $\sum_i w(i) = 1$, *the* workload-optimal *B-bucket histogram* $\mathbf{H}_{\mathrm{opt}}$ *is the choice of b_j's and h_j's that minimize*

$$\|\mathbf{A} - \mathbf{H}\|_w^2 = \sum_i w_i(\mathbf{A}[i] - h_{j(i)})^2.$$

The problem of finding $\mathbf{H}_{\mathrm{opt}}$ is interesting on stored or streamed signals as well as stored or streamed workloads.

The database community has proposed methods not merely to synopsize data distributions but also to take the workload into account. Query feedback from the execution engine of a DBMS was used in [5] to modify the synopsis. Histogram boundaries are refined adaptively in [19, 1, 26] based on a dynamically evolving workload that is continuously updated based on feedback from the query engine; they differ in how they approximate values within buckets, how they weight the workload etc. Still, these methods do not give any provable results on approximating $\mathbf{H}_{\mathrm{opt}}$. There has been some work on *other* synopsis that are workload-aware. For example, [7] proposed sampling methods that adapt to recent workload. IBM's LEO optimizer [27] uses workload information for variety of synopsis. In [24], a $O(N^2 B/\log B)$ time algorithm is presented for determining the optimal choice of B Haar wavelet synopsis; this has recently been improved to $O(N^2)$ time [9]. The Haar basis is modified in [20] with the knowledge of the workload and algorithms for obtaining B-term synopsis are designed for this new basis; while this algorithm works in linear time, it does not provide near-optimal B-term Haar wavelet synopsis. For special workloads, [24] presented a near-linear algorithm for finding the optimal B-term Haar wavelet basis. All of these results for Haar and related basis [24, 9, 20] work only when *both* the signal and workload are available in a stored form without any loss of information, and not streamed with polylogarithmic space. However, when both the signal and workload are stored explicitly without loss of information, the dynamic programming from [17] immediately gives an $O(N^2 B)$ time algorithm for finding the optimal $\mathbf{H}_{\text{opt}}$, so the challenge in [24, 9] arises from working with the Haar wavelet basis and does not reflect on the difficulty in constructing $\mathbf{H}_{\text{opt}}$. To summarize, the *significant open problem with finding $\mathbf{H}_{\text{opt}}$ is when either the signal or the workload is streamed or both are streamed, with space polylogarithmic in N.*

In this paper, we address the problem of computing $\mathbf{H}_{\text{opt}}$ on data streams. Our primary question is, do the powerful theoretical results known for uniform histogram construction on data streams [8, 10, 12] hold for the workload-aware case as well? Is there a difference in streaming the signal versus the streaming the workload in polylogarithmic space? What is the information-content of the workload and how does it affect the complexity of histogram construction?

Our contributions are as follows. Suppose the data items are integers, and the weights are positive integers between the minimum weight, $w_{\min}$, and the maximum weight, $w_{\max}$. Let $M = \max\{||A||^2, \frac{w_{\max}}{w_{\min}}\}$ be a bound on the range of data and weights.

- *Workload w is stored without loss of information.*

  We present an $O(N+\text{poly}(B, \log N, \log M, 1/\epsilon))$ time algorithm to compute a B-bucket histogram $\mathbf{H}$ with $\|\mathbf{A}-\mathbf{H}\|_w^2 \leq (1+\epsilon)\|\mathbf{A}-\mathbf{H}_{\text{opt}}\|_w^2$ where $\mathbf{H}_{\text{opt}}$ is the workload-optimal B-bucket histogram, with respect to arbitrary w. This is the first near-linear[1] time algorithm for approximating $\mathbf{H}_{\text{opt}}$ under non-uniform workloads. The above algorithm can be run in the time series model taking only $O(1)$ time per new item and using $\text{poly}(B, \log N, \log M, 1/\epsilon)$ space and time to construct the $(1+\epsilon)$-approximate histogram. Under the more general dynamic maintenance model, the above algorithm can be modified using previously known techniques so that the time per update, total space used, and postprocessing time are all $\text{poly}(B, \log N, \log M, 1/\epsilon)$. This is the first known set of algorithms that use sublinear—polynomial in B, $1/\epsilon$ and polylogarithmic in N, M—space for dealing with data stream signals and yet yields $(1+\epsilon)$ approximate $\mathbf{H}_{\text{opt}}$ histograms for any w. It matches the previously known bounds for the special case when the workload is uniform [8].

- *Workload w is compressed.* We consider two cases.

  - Given a workload vector w, let w' denote the vector of weights in w, rounded to a power of $(1+\epsilon)$; it is easy to see that w' can be substituted for w giving up no more than a $(1+\epsilon)$-approximation factor in histogram estimation. This is a trivial lossy compression of w.

    A simple argument shows that, in general, if the *rounded* weight vector w' of worst-case space $|w'| = \Theta(\log(\log(M)/\epsilon)N)$ bits is compressed to $|w'| - \omega(\log(M)/\epsilon)$ bits, then no algorithmic result of the type is possible (even if the entire signal is stored without loss). That is, no significant lossy compression of w is possible beyond discarding low-order bits.

[1] Note that, for moderate values of the parameters other than N, the run time is dominated by $O(N)$. In this paper, we use the term "near-linear" for this type of cost.

– We focus on the case when w' is compressed without loss. We show that if w' is losslessly compressible to a structure $C(w')$ of size $|C(w')|$ by the Ziv-Lempel method [30], then the preprocessing time and space can be reduced from linear in N to linear in $|C(w')|$. We present a $O(|C(w')|)$-spaced data structure to answer "symbol-range-count" queries which enable all our workload-optimal histogram construction algorithms above to be implemented in space and time $|C(w')|\text{poly}(B, \log N, \log M, 1/\epsilon)$. This is an advantage for highly compressible workload w's, where $|C(w')| \ll o(|w'|)$. We also present an alternative algorithm with different tradeoffs among resources.

  This work integrates aspects of streaming algorithms (where input is compressed lossily) with algorithms that work with losslessly compressed input. This is a novel direction suggested in [24].

Section 2 has preliminaries. Section 3 has our results for case when w is uncompressed, and is one of the main results here. The algorithm working in a *lockstep model* is presented in Section 4. The lower bound on space when the workload is streamed is in Section 5 and is fairly simple. In Section 6, we describe methods for managing a compressed workload w. In this extended abstract, formal proofs are omitted.

2 Preliminaries

Definition 3 *Inner Product with Weight: For any two signals* $\mathbf{A}$ *and* $\mathbf{B}$ *with length* N *respectively and the same length* N *weight vector* w*, define* $\langle \mathbf{A}, \mathbf{B} \rangle_w = \sum_{i=1}^{N} \mathbf{A}_i \mathbf{B}_i w_i$ *and* $\|\mathbf{A}\|_w = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle_w}$ *where* w_i *is a non-negative weight at index* i*. We continue to write* $\langle \mathbf{A}, \mathbf{B} \rangle$ *and* $\|\mathbf{A}\|$ *for the dot product and norm under uniform workload where all* w_i*'s are equal.*

Definition 4 *Robust Representation [8, 10]. Fix a signal* $\mathbf{A}$*. A representation* $\mathbf{H}_{\mathrm{r}}$ *is called a* (B, ϵ)*-robust approximation to* $\mathbf{A}$ *if (i)* $\|\mathbf{H}_{\mathrm{r}} - \mathbf{A}\| \leq \epsilon \|\mathbf{H}_{\mathrm{opt}} - \mathbf{A}\|$*, or, (ii) for any representation* $\mathbf{H}$ *on the boundaries of* $\mathbf{H}_{\mathrm{r}}$ *and any other* $B-1$ *boundaries, with optimal parameters, we have* $(1-\epsilon)\|\mathbf{A} - \mathbf{H}_{\mathrm{r}}\|^2 \leq \|\mathbf{A} - \mathbf{H}\|^2$*.*

Here we call this property *bucket robustness*, to distinguish it from *linear robustness*, defined later.

3 Uncompressed Workload

We first give an algorithm for time series data stream model that takes time $O(1)$ per item, space and post-processing time $\text{poly}(B, \log N, \log M, 1/\epsilon)$, where the signal and workload have length N and bound M, and a $(1+\epsilon)$-approximate B-bucket histogram is desired. Our algorithm's overall structure follows closely the algorithm in [10] for the uniform workload, so we first sketch that algorithm and selected parts of its analysis, then describe in detail the changes needed for non-uniform workloads.

3.1 A Previous Uniform-Workload Algorithm

The algorithm in [10] proceeds as follows.

1. (Selection of large wavelet terms.) Read in a length-N stream $\mathbf{A}$ of time-series data and output a list L of the $B' \leq \text{poly}(B, \log N, \log M, 1/\epsilon)$ terms with largest coefficients.[2]

2. (Construction of bucket-robust representation.) Select the largest $B'' = \text{poly}(B, \log N, \log M, 1/\epsilon)$ terms from L, greedily, using a particular 2-part stopping rule (described in detail below). Call the result $\mathbf{H}_\mathrm{r}$, a *bucket-robust* histogram of $O(B'')$ buckets.

3. (Construction of output.) Find a best B-bucket histogram $\mathbf{H}$ to $\mathbf{H}_\mathrm{r}$, and output $\mathbf{H}$ as a $(1+\epsilon)$-approximate histogram to $\mathbf{A}$.

Note that the first step is performed on the stream, but the last two steps are full-space, polynomial-time post-processing algorithms on small input, that is, input of polylogarithmic size. In [10], the authors showed that the computational cost of each step meets the claim.

We now consider in more detail the relevant parts of the [10] algorithm. In Step 2, we need an additional parameter, $\epsilon_\mathrm{r} = \Theta(\epsilon)$. We take terms from L, from biggest to smallest, $4B\log(N)$ at a time, and add them to $\mathbf{H}_\mathrm{r}$, which is initially the zero histogram. Let $\mathbf{H}'_\mathrm{r}$ denote the *next* value of $\mathbf{H}_\mathrm{r}$, *i.e.*, $\mathbf{H}_\mathrm{r}$ plus the next $4B\log(N)$ terms to be taken. We stop when either of the following conditions is met:

- (No Progress.) $(1-\epsilon_\mathrm{r})\,\|\mathbf{A}-\mathbf{H}_\mathrm{r}\|_2^2 \geq \|\mathbf{A}-\mathbf{H}'_\mathrm{r}\|_2^2$.

- (Many Terms.) We have accumulated T terms, for some $T = O(\epsilon_\mathrm{r}^2 \log(1/\epsilon_\mathrm{r}) B \log(N))$.

Using a case analysis, the output $\mathbf{H}$ is shown to be correct whichever stopping rule is used. The conditions in bucket-robustness (Definition 4) correspond to the Many Terms and No Progress stopping rules, respectively.

In Step 3, dynamic programming similar to [17] is used. In particular, the dynamic programming algorithm accesses $\mathbf{H}_\mathrm{r}$ only by making the following query. Given interval $[\ell, r)$, what is the best height h of a 1-bucket histogram $a\chi_{[\ell,r)}$ and what is the resulting error $\sum_{\ell\leq i<r}(\mathbf{H}_\mathrm{r}[i]-a)^2$ on that interval? This query must be answered in time to meet the post-processing bound.

3.2 Our Algorithm for Non-Uniform Workloads

Our algorithm generalizes the algorithm in [10] to non-uniform workloads. We first give a high-level description of our algorithm. To state the algorithm clearly, it is convenient to give some abstract definitions along the way. Analysis will be given later.

1. (Weight-class splitting.) We regard each weight w_i as rounded to w'_i, a power of $(1+\epsilon)$. There is a small number $p = \log_{1+\epsilon}(M)$ of these classes. We split the incoming time series into p new time series, according to the associated rounded weight.

[2] We do not need the definition of wavelets in this paper.

2. (Conversion to wavelets, selection of large terms, and construction of bucket-robust representation.) For each substream, we create a bucket-robust representation, as in [10]. For each substream, record which of the two stopping conditions was used.

3. (Recombination of substreams.) Conceptually, recombine the bucket-robust representations with stopping rule No Progress, getting $\mathbf{H}'_{\mathrm{r}}$, and recombine the bucket-robust representations with stopping rule Many Terms, getting $\mathbf{H}''_{\mathrm{r}}$. Conceptually, combine $\mathbf{H}'_{\mathrm{r}}$ and $\mathbf{H}''_{\mathrm{r}}$, getting $\mathbf{H}_{\mathrm{r}}$. The algorithm does nothing; it represents $\mathbf{H}'_{\mathrm{r}}$ and $\mathbf{H}''_{\mathrm{r}}$ as the appropriate collection of bucket-robust histograms.

4. (Construction of output.) Find a best B-bucket histogram $\mathbf{H}$ to $\mathbf{H}_{\mathrm{r}}$, and output $\mathbf{H}$ as a near-best histogram to $\mathbf{A}$.

In what follows, we will give details of each step where it significantly differs from [10]. The technical aspects are manifold, but the crux is $\mathbf{H}_{\mathrm{r}}$ is *not* bucket-robust and as a result [10] fails. We now proceed formally.

Step 1. Rounding Weights. We consider signals of length N, with weights $w_1, \cdots, w_N$, and such that $\|\mathbf{A}\|^2 \le M$. We will assume that data items are integers and that weights are positive integers in the range $w_{\min} = 1$ to some $w_{\max} \le M$ (so we abandon the normalization $\sum_i w_i = 1$.) Define $p = \log_{1+\epsilon} M + 1$, and p different rounded weights $w^1, w^2, \cdots, w^p$, where $w^i = (1+\epsilon)^{i-1}$. Round all the original weights $w_1, \cdots, w_N$ *down* to rounded weights $w'_1, \cdots, w'_N$ respectively, *i.e.*, $w'_i = w^j$ where $w^j \le w_i < w^{j+1}$. We use $w' = (w'_1 \cdots w'_N)$ to represent the length-N rounded weight vector.

Lemma 1 *Fix a signal* $\mathbf{A}$ *of dimension* N*. Then* $\|\mathbf{A} - \mathbf{H}'_{\mathrm{opt}}\|_w^2 \le (1+\epsilon)\|\mathbf{A} - \mathbf{H}_{\mathrm{opt}}\|_w^2$*, where* $\mathbf{H}_{\mathrm{opt}}$ *is the optimal* B *bucket representation to* $\mathbf{A}$ *under weight* w*, and* $\mathbf{H}'_{\mathrm{opt}}$ *is the optimal* B *bucket representation to* $\mathbf{A}$ *under weight* w'*.*

Proof. We have $\|\mathbf{A} - \mathbf{H}'_{\mathrm{opt}}\|_w^2 = \sum_{i=1}^N [\mathbf{A}_i - \mathbf{H}'_{\mathrm{opt}}(i)]^2 w_i \le (1+\epsilon)\sum_{i=1}^N [\mathbf{A}_i - \mathbf{H}'_{\mathrm{opt}}(i)]^2 w'_i$ from the relationship between w_i and w'_i. Further, by optimality of $\mathbf{H}_{\mathrm{opt}}$, $\|\mathbf{A} - \mathbf{H}'_{\mathrm{opt}}\|_w^2 \le (1+\epsilon)\sum_{i=1}^N [\mathbf{A}_i - \mathbf{H}_{\mathrm{opt}}(i)]^2 w'_i \le (1+\epsilon)\sum_{i=1}^N [\mathbf{A}_i - \mathbf{H}_{\mathrm{opt}}(i)]^2 w_i$. Hence, $\|\mathbf{A} - \mathbf{H}'_{\mathrm{opt}}\|_w^2 \le (1+\epsilon)\|\mathbf{A} - \mathbf{H}_{\mathrm{opt}}\|_w^2$. ∎

Step 3. Here is the technical crux. Unfortunately, the old definition of bucket-robustness fails in our framework of separate substreams based on weight classes. We give a brief illustration why. Suppose there are just two weight classes, that partition $[0, N)$ precisely into the even and odd indices. Suppose $\mathbf{H}'$ and $\mathbf{H}''$ are two bucket-robust histograms, defined on the even and odd indices, respectively, and each has a small number of buckets. Suppose that all heights represented in $\mathbf{H}'$ and $\mathbf{H}''$ are distinct. Suppose that $\mathbf{H}'$ and $\mathbf{H}''$ have about equal shares of the error. Finally, suppose that the reason for bucket-robustness is *only* that they are not much improved by refinement by B more buckets; that is, suppose $\|\mathbf{A} - \mathbf{H}'\|^2$ and $\|\mathbf{A} - \mathbf{H}''\|^2$ are each approximately $\frac{1}{2}\|\mathbf{A} - \mathbf{H}_{\mathrm{opt}}\|^2 > 0$, which can happen if $\mathbf{A} - \mathbf{H}_{\mathrm{opt}}$ is noisy. We claim that all of these assumptions are consistent.[3] Now, let $\mathbf{H}$ be the

[3]In any case, we cannot provably rule out the existence of such $\mathbf{H}'$ and $\mathbf{H}''$.

combination of $\mathbf{H}'$ and $\mathbf{H}''$; we will show that $\mathbf{H}$ is not bucket-robust. Note that, because of the even/odd partitition induced by the given weights, $\mathbf{H}$ has N buckets, each of size 1. Then one of the bucket-robust condition says that if we further refine $\mathbf{H}$ and then *optimize the heights*, we do not get much improvement, multiplicatively. Clearly, by optimizing the heights, we can get error zero! The other condition says that $\|\mathbf{A}-\mathbf{H}\|^2 \approx \epsilon^2 \|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|^2$. But $\|\mathbf{A}-\mathbf{H}\|^2 = \|\mathbf{A}-\mathbf{H}'\|^2 + \|\mathbf{A}-\mathbf{H}''\|^2 \approx \|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|^2 \gg \epsilon^2 \|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|^2$. It follows that neither condition of bucket-robustness is satisfied.

Instead, we introduce the following new notion.

Definition 5 *Fix a signal $\mathbf{A}$ and rounded weight vector w'. Given parameters B and ϵ, let $\mathbf{H}_{\mathrm{opt}}$ denote an optimal B-bucket histogram for $\mathbf{A}$ under w'. A representation $\mathbf{H}_{\mathrm{r}}$ is called a (B,ϵ)-linearly-robust (or just (B,ϵ)-robust henceforth) approximation to $\mathbf{A}$ under weight w', if, for any B-bucket histogram H_B and any scalars a and b, either $\|\mathbf{A}-\mathbf{H}_{\mathrm{r}}\|^2 \leq \epsilon^2 \|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|^2$ or $(1-\epsilon)\|\mathbf{A}-\mathbf{H}_{\mathrm{r}}\|_w^2 \leq \|\mathbf{A}-(a\mathbf{H}_{\mathrm{r}}+b\mathbf{H}_B)\|_w^2$.*

The first condition is similar to the corresponding condition in bucket-robustness: the error $\|\mathbf{A}-\mathbf{H}_{\mathrm{r}}\|^2$ is already tiny compared with $\|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|^2$. The second condition is implied by bucket-robustness. Hence, linear-robustness is a weaker notion than bucket-robustness.

Returning to our overall algorithm, recall that $\mathbf{H}'_{\mathrm{r}}$ and $\mathbf{H}''_{\mathrm{r}}$ are the recombinations of bucket-robust histograms for substreams where the stopping rule is No Progress or Many Terms, respectively.

Lemma 2 *Each of $\mathbf{H}'_{\mathrm{r}}$ and $\mathbf{H}''_{\mathrm{r}}$ is linearly robust. Further, the combination $\mathbf{H}_{\mathrm{r}}$ of $\mathbf{H}'_{\mathrm{r}}$ and $\mathbf{H}''_{\mathrm{r}}$ is linearly robust.*

Proof. Consider $\mathbf{H}'_{\mathrm{r}}$. Since, on each substream, bucket-robustness implies linear-robustness, the multiplicative improvement of $a\mathbf{H}'_{\mathrm{r}} + b\mathbf{H}_B$ over $\mathbf{H}'_{\mathrm{r}}$ on each substream is bounded by $(1-\epsilon)$. Since the substreams are disjoint, by orthogonality, their square error sums to the square error of $\mathbf{H}'_{\mathrm{r}}$ and their improvements sum to the improvement for $\mathbf{H}'_{\mathrm{r}}$. It follows by summing over substreams that the overall error improvement is bounded by $(1-\epsilon)$. Now consider $\mathbf{H}''_{\mathrm{r}}$. In each substream, the (unsquared) error is within $\epsilon\|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|$. Summing over $p = O(\log(M)/\epsilon)$ substreams and using the triangle inequality, it follows that the overall unsquared error is at most $p\epsilon\|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|$. By substituting ϵ/p for ϵ, we have error at most $\epsilon\|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|$ on the set of positions where $\mathbf{H}''$ is defined. Finally, to see that the combination $\mathbf{H}_{\mathrm{r}}$ of $\mathbf{H}'_{\mathrm{r}}$ and $\mathbf{H}''_{\mathrm{r}}$ is linearly robust, let E' be the (unsquared) error in $\mathbf{H}'_{\mathrm{r}}$ and E'' be the (unsquared) error in $\mathbf{H}''_{\mathrm{r}}$ on their respective substreams. we consider whether $\epsilon(E')^2$ or $(E'')^2$ is larger. If $\epsilon(E')^2 \geq (E'')^2$, then little relative progress is made on $(E')^2$ by the linear combination $a\mathbf{H}'_{\mathrm{r}} + b\mathbf{H}_B$. The progress made on $(E'')^2$ is at most $\epsilon(E')^2$, which is also small compared to $(E')^2$. It follows that $\mathbf{H}_{\mathrm{r}}$ is linearly robust. On the other hand, suppose $\epsilon(E')^2 \leq (E'')^2$. By choosing appropriate robustness parameters, we may assume that $(E'')^2 \leq (\epsilon^3/2)\|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|$, so that $(E')^2 \leq (\epsilon^2/2)\|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|$. It follows that the total error $(E')^2 + (E'')^2$ is at most $\epsilon^2\|\mathbf{A}-\mathbf{H}_{\mathrm{opt}}\|$. We conclude that $\mathbf{H}_{\mathrm{r}}$ is linearly robust. ∎

Step 4. This involves two aspects, first a proof that the output $\mathbf{H}$ is correct, that is, a best B-bucket histogram $\mathbf{H}$ to $\mathbf{H}_{\mathrm{r}}$ is a $(1+\epsilon)$-approximation of the optimal histogram to $\mathbf{A}$, and second, how to find $\mathbf{H}$.

For the uniform workload, the fact that $\mathbf{H}_{\mathrm{r}}$ is bucket-robust was shown in [10] to imply that the output $\mathbf{H}$ is correct. This was done by showing that bucket-robustness implies linear-robustness and that linear-robustness implies correctness, though the concept of linear-robustness was not isolated as important in [10]. To extend the proof in [8, 10] from the uniform workload to workload w', we observe that the proof under the uniform workload in [10] uses only the triangle inequality and an approximate Pythagorean theorem, which hold also under weight w'. Therefore we conclude that $\mathbf{H}$ is a correct (approximate) output. For the reader familiar with [10], we note that the approximate weighted Pythagorean Theorem says that if $\langle \mathbf{A}-\mathbf{C}, \mathbf{B}-\mathbf{C}\rangle_{w'} \leq \epsilon \|\mathbf{A}-\mathbf{C}\|_{w'} \|\mathbf{B}-\mathbf{C}\|_{w'}$ (a near-right angle at C), then $\|\mathbf{A}-\mathbf{B}\|_{w'}^2 = (1\pm\epsilon)\left(\|\mathbf{A}-\mathbf{C}\|_{w'}^2 + \|\mathbf{B}-\mathbf{C}\|_{w'}^2\right)$.

Second, we perform dynamic programming. As remarked earlier, it suffices to find the best 1-bucket histogram representation to $\mathbf{H}_{\mathrm{r}}$ on a given range to perform the dynamic programming. In order to do this, we need to be able to answer the following type of query, which we call a *symbol-range-count* query. Given a weight class label, i, and a position, j, how many times does the weight class i occur in $[j', j]$? We use the following:

Lemma 3 *Let histogram $\mathbf{H}_{\mathrm{r}}$ equal the conceptual combination of p histograms $\mathbf{H}_{\mathrm{r}}^i$ of length N and at most B' buckets with respect to a partition of $[0, N)$ corresponding to p weight classes in the rounded weight vector w'. In time (and space) $O(N)$, one can build a data structure that, on query $[j', j)$, gives the best one-bucket approximation to $\mathbf{H}_{\mathrm{r}}$ on $[j', j)$ and the associated error, in query time $O(pB'\log(N))$.*

Proof. (Sketch.) We may assume that $j' = 0$, by combining appropriately the queries $[0, j)$ and $[0, j')$. For each weight class, it is straightforward to build a linear-sized data structure, in linear time, that answers this query in time $O(\log(N))$ or better. Since the weight classes partition $[0, N)$, by combining the data structures for the various weight classes, we can construct, for the overall collection of weights, a data structure of size $O(N)$, in time $O(N)$, as desired.

Next, let $w_{\mathrm{sum}} = \sum_{k=0}^{j} w'_k$. Define a random variable X by letting $X = \mathbf{H}_{\mathrm{r}}[k]$ with probability $\frac{w'_k}{w_{\mathrm{sum}}}$, for $k \in [0, j)$ at random. Then $E[X] = h$ is the height of the optimal one-bucket representation to $\mathbf{H}_{\mathrm{r}}$ between 0 and j. Therefore, $h = \frac{\sum_{k=0}^{j} w'_k \mathbf{H}_{\mathrm{r}}[k]}{\sum_{k=0}^{j}(w'_k)} = \frac{\sum_{k=0}^{j} w'_k \mathbf{H}_{\mathrm{r}}[k]}{w_{\mathrm{sum}}}$. Note that all the data in one bucket of a robust representation in each stream i have the same rounded weight and the same height in $\mathbf{H}_{\mathrm{r}}$. So we will break $[0, j)$ into at most B' buckets from the corresponding $\mathbf{H}^i$. For each such bucket $[\ell, r)$ and each weight class, we need to make a symbol-range-count query for ℓ and another query for r, and subtract; we can then compute the optimal height. Computing the error, $\sum_{k=0}^{j} w'_k(\mathbf{H}_{\mathrm{r}}[k]-h)^2$, is similar. ∎

We can now summarize:

Theorem 1 *There is an algorithm that, given parameters B, N, M, ϵ and weight vector w of length N and bound M, preprocesses w in time and space $O(N)$, reads data $\mathbf{A}$ with $\|\mathbf{A}\|_w^2 \leq M$ in time series, then outputs a B-bucket histogram $\mathbf{H}$ with $\|\mathbf{A}-\mathbf{H}\|_w^2 \leq (1+$*

$O(\epsilon))\|\mathbf{A} - \mathbf{H}_{\text{opt}}\|_w^2$, where $\mathbf{H}_{\text{opt}}$ is the best possible B-bucket histogram representation to $\mathbf{A}$ under weight w. The algorithm uses space $O(poly(B, \log N, \log M, 1/\epsilon))$ in addition to the space associated with w independent of the input. The algorithm uses time $O(N)$ to read the stream of data and post-processing time $O(poly(B, \log N, \log M, 1/\epsilon))$ to build $\mathbf{H}$.

This meets the complexity for the uniform workload case, and settles the problem modulo improving the polynomials. Using prior work on the general *dynamic maintenance* model for uniform workload case [8] we get the following corollary (which also meets the best-known complexity for the uniform workload case [8]):

Corollary 1 *For parameters N, M, B, ϵ, there is a randomized data structure for an array $\mathbf{A}$ that preprocesses a workload w in time and space $O(N)$, requires additional space $O(poly(B, \log N, \log M, 1/\epsilon))$ and supports the following operations in time $O(poly(B, \log N, \log M, 1/\epsilon))$:* update *(Add v to $\mathbf{A}[i]$, where v may be positive or negative);* build *(Build a $(1+\epsilon)$-near optimal histogram with respect to the then-current dataset $\mathbf{A}$, under workload w.*

4 Lockstep Model

In this section, we briefly describe the lockstep model and algorithms for it.

In the lockstep model for signal $\mathbf{A}$ and workload w, we receive $\mathbf{A}[0], w_0, \mathbf{A}[1], w_1, \mathbf{A}[2], w_2, \ldots$, in that order. Similar to above, we may modify each weight w_i by a factor between 1 and $(1+\epsilon)$, getting w_i' from a set of $O(\log(M)/\epsilon)$ possibilities; for simplicity, assume here that $w_i' = w_i$ and that each w_i' is an integer. Let $U \leq MN$ denote the sum of the w_i''s. We call $[0, N)$ the *original* domain and $[0, U)$ the *expanded* domain.

For any signal $\mathbf{A}$ in the original domain, we define a transformed signal $\mathbf{A}'$ in the expanded-workload domain as following: $\forall i \in \{0, 1, \cdots, N-1\}$, let $s = \sum_{j=0}^{i-1} w_j$. We define $\mathbf{A}'_s, \mathbf{A}'_{s+1} \ldots \mathbf{A}'_{s+w_i-1}$ to be $\mathbf{A}_i$. That is, assuming for illustration that all original-domain signal values are distinct, the i'th run of $\mathbf{A}'$ is a run with value $\mathbf{A}[i]$ and length w_i. Let I_i denote the interval where the i'th run occurs; so $I_i = [\sum_{j<i} w_j, \sum_{j\leq i} w_j)$ is independent of $\mathbf{A}$.

The following are straightforward to check, using standard facts and algorithms about Haar wavelets, where appropriate:

- If $\mathbf{H}$ is a histogram and $\mathbf{H}'$ is its transformation to the expanded workload domain, then $\|\mathbf{A} - \mathbf{H}\|_w = \|\mathbf{A}' - \mathbf{H}'\|$. That is, transformation of signals to the expanded workload domain takes the weighted norm to the uniform norm.
- There are at most $O(N \log(U))$ Haar wavelet terms for $\mathbf{A}'$ with nonzero coefficients.
- From lockstep input, the nonzero Haar wavelet terms can be output by an algorithm that uses space $O(\log(U))$ and time $O(N \log(U))$.

The implications are as follows:

Theorem 2 *In the lockstep model for data of length N and integer weights that sum to U, we can construct a $(1+\epsilon)$-near optimal histogram using space poly$(B, \log N, \log M, 1/\epsilon)$ and time $O(N \log(U))$.*

Note that the time is somewhat worse than the algorithm of Section 3 for stored weights, but the model is somewhat weaker—no space to store the weights is assumed. Also note that the lockstep model algorithm applies also for the case of stored weights. Both the technique of this section and Section 3 are reductions from a non-uniform workload to a uniform workload, but the reduction of this section is simpler.

Both the technique of this section and Section 3 can be modified for the situation of stored weights and dynamic updates (additions or subtractions) to the data, yielding the same qualitative time bound of poly$(B, \log N, \log M, 1/\epsilon)$ per item. Given an update "add a to $\mathbf{A}[i]$" in the dynamic model, we determine I_i from i (say, from a lookup table), then, using standard facts about Haar wavelets, compute the set S of $O(\log(U))$ wavelets whose support includes an endpoint of I_i—exactly this set of wavelet coefficients changes. The update "add a to $\mathbf{A}[i]$" in the original domain transforms to "add a to each wavelet in S" in the expanded workload domain. This concludes the reduction to the uniform workload. (The top wavelet terms, as needed by our histogram algorithm, can then be tracked efficiently using any of several recent "heavy hitters" data structures [22].) The cost of this simple reduction is the factor $O(\log(U))$, which is at most $O(\log(MN))$. Similarly, it is easy to see that the symbol-range-count data structure for the weights suffices, so we can compress the weights losslessly by `LZ78` for the dynamic maintenance model.

5 Lower Bounds

It is easy to see that a histogram algorithm that first reads the data and then is given a workload must store all the data, since the choice of workload and histogram approximation criterion can force the algorithm to recover any data item exactly. This immediately gives

Theorem 3 *Suppose data and workload values are interleaved arbitrarily. For any $B \geq 3$, any algorithm that outputs approximate B-bucket histogram uses space $\Omega(N \log(M))$ bits (enough to store all the data).*

While Theorem 3 is the strongest possible statement about interleaving data and workload values, it says nothing about compressing the workload. Above we showed that, to get a $(1+\epsilon)$-factor approximation, one can round weights to a power of $(1+\epsilon)$ (*i.e.*, discard low-order bits). We now show that, in a sense, this is the only kind of lossy compression that is possible.

Lemma 4 *Suppose an algorithm reads and processes a workload of length N and bound M into an object s of size $|s|$, then discards everything about the workload except s, then reads time series data. If, for any workload, any data, and any sufficiently small $\epsilon > 0$, the algorithm produces, with probability $\gg 1/2$, a $(1+\epsilon)$-approximation to the best 3-bucket* *histogram, then the algorithm can be used as a subroutine to store any value from a vector of positive integer entries bounded by* $M/4$*, of length* $\geq N - O(\log(M)/\epsilon)$*, up to the factor* $(1 + O(\epsilon))$*.*

Proof. Suppose we are given a vector v of entries bounded by $M/4$, of length $N - \log(M)/\epsilon$. Construct a workload as follows. The first $O(\log(M)/\epsilon)$ values are all the powers of $(1+\epsilon)$, in order, called "reference values." The next value is equal to M. Finally, the last $N - \log(M)/\epsilon$ values are the original vector. Run the first part of the algorithm on this workload, producing s. Our goal now is to recover any v_j from s and j. To do this, consider the data that is all zeros except for a 1 at a position corresponding to a reference workload value of $(1 + \epsilon)^k$ and another 1 at a position corresponding to v_j. If $v_j > (1 + \epsilon)^{k+1}$, then the best 3-bucket histogram gets v_j right and is near zero everywhere else (getting the 1 at reference position $(1 + \epsilon)^k$ wrong). Similarly, if $v_j < (1 + \epsilon)^{k-1}$ the best 3-bucket histogram gets v_j wrong and $(1 + \epsilon)^k$ right. It follows that we can learn v_j up to the factor $(1 + \epsilon)^2 = (1 + O(\epsilon))$. ∎

In particular, the lemma above implies:

Theorem 4 *For any* $B \geq 3$ *and any sufficiently small* $\epsilon > 0$*, if an algorithm represents in space* s *a workload* w *of length* N *and bound* M *and finds* $(1 + \epsilon)$*-near-best* B*-bucket histograms with respect to* w*, then* s *is at least the space needed to store* $(N - O(\log(M)/\epsilon))$ *counters of* $\Omega(\log(M)/\epsilon)$ *states.*

6 Compressed Weights

In the previous section, we showed that lossy compression of the workload beyond rounding is not possible, even information-theoretically. In this section, we consider efficient algorithms for manipulating losslessly compressed workloads of rounded weights. We consider principally the famous Lempel-Ziv compression methods. Our goal in this section is to build a symbol-range-count structure R to match the given compression scheme, where, we recall, a symbol-range-count query is the pair (i, j), for which the answer is the number of occurrences of i in $[0, j)$. That is, if the compressed text $C(w')$ has size $|C(w')|$, then, ideally, we want to build R with preprocessing time $O(|C(w')|)$, we want $|R| \leq O(|C(w')|)$, and we want symbol-range-count queries to be as quick as possible—plausible guarantees are $O(\mathrm{poly}(B, \log N, \log M, 1/\epsilon))$ or some function of the compressed string. Thus, the challenge is to be *opportunistic* and design data structures bounded in size by $|C(w')|$. We also discuss building R' of size $|R'| < o(|C(w')|)$ such that R' and $C(w')$ together constitute a symbol-range-query structure, R. This has the advantage that the total size $|R|$ is $(1 + o(1))|C(w')|$ rather than $O(1)|C(w')|$; this is useful if, say, $|C(w')| = |w'|/100$, so that the constant factor in $O(1)$ is significant. Opportunistic data structures are known for indexing a string for full-text substring queries [6], but no previous results are known for our problem of supporting the symbol-range-count query. The results of this section may of interest separately in database and string processing.

Formally, we are given a string $S[1, ..., N]$, with each $S[i]$ in alphabet set of size p (the rounded weight classes in our histogram problem). We compress S using Lempel-Ziv algorithm, denoted LZ78, which works as follows. Say $S[1,...,i]$ has been compressed; a dictionary D of tuples (d_k, l_k) would have been constructed thus far with each $d_k = S[l_k, \ldots, l_k + |d_k| - 1]$. The algorithm iteratively proceeds by finding the longest prefix $S[i+1,..,j]$ that equals some d_k, compressing $S[i+1,...,j+1]$ as $(l_k, |d_k|, S[j+1])$, adding $(S[i+1,...,j+1], i+1)$ to D and continuing. Each such step is called a "parse" and the number of parses is directly related to the size of the compressed representation $C(S)$ of S upto constant factors. Hereafter, we will let $|C(S)|$ be the number of such parses, as is standard in the string compression area, without being specific about how to code each $(l_k, |d_k|, S[j+1])$ in smallest number of bits.

There are many variants of this basic method, depending on whether windowing is used, whether $S[i+1,...,j]$ and $S[l_k,...,l+k+|d_k|-1]$ may overlap or not, how the parses are encoded using bits, etc. We will focus on the basic version above and our results will hold for these other variants as well. A significantly different variant is the LZ77 [29] algorithm in which we add all substrings of $S[i+1,..,j]$ to D. This leads to larger D and hence, fewer parses and smaller $C(S)$. Our algorithm in this section will work with the LZ77 compression method as well (also for run-length encoding methods), but we omit the details in this extended abstract.

Lemma 5 *A string S given in its* LZ78 *compressed form $C(S)$ can be preprocessed in time and space $O(|C(S)|)$ such that a symbol-range-count query (i, j, α) can be answered in time $O(|C[i, j]| \log\log N)$, where $|C[i, j]|$ is the number of* LZ78 *parses overlapping $[i, j]$.*

Proof. We sketch our solution. First, we make a few observations about the dictionary D. In particular, focus on the d_k's. Notice that each new d_k that is added to D is one symbol longer than a different $d'_k \in D$. For example, when $S[i+1,...,j+1]$ is added to D in the description of the algorithm above, $S[i+1,...,j]$ is already in D. Hence, the set of all d'_k in D can be arranged as a rooted trie with the internal nodes labeling the index k; the edge from node k' to k is marked with the additional symbol in k. We label this trie as T. The size of T is proportional to the number of tuples in D, which we denote $|T| = |D|$ and in fact, it is easy to observe that $|T| = |C(S)|$, too.

Our algorithm is as follows. Consider the parses that overlap $[i, j]$ and denote their number by $|C(i, j)|$. There may be a left boundary parse $[l_1, l_2]$ with $l_1 < i$ and $i < l_2 < j$; likewise, there may be a right boundary parse $[r_1, r_2]$ with $i < r_1 < j$ and $j < r_2$. The remainder of the parses are central with both their endpoints inside $[i, j]$. For each central parse, we will compute the number of occurrences of α's in it. For the right boundary $[r_1, j]$, we note that since $[r_1, r_2]$ is in D, then so must be $[r_1, j]$ since D has the property that all the proper prefixes of a d_k in D are also in D. Given the pointer to the node that stands for $[r_1, r_2]$, in $O(1)$ time one can get the node that stands for $[r_1, j]$ using the level ancestors algorithm [3]. We can now treat the right boundary as the parse $[r_1, j]$. A similar trick does *not* work for the left. We need to find the number of occurrences of α's in $[i, l_2]$, but while $[l_1, l_2]$ is a parse, $[i, l_2]$ need not be because D is not suffx-closed, ie., d_k in D does not imply that all its suffixes are in D. We do a simple trick here: we count the number of α's in $[i, l_2]$ as that in $[l_1, l_2]$ minus that in $[l_1, i-1]$ both of which are parses in D because we are given that $[l_1, l_2]$ is, and the other is its prefix. Thus the overall algorithm is to consider each of these parses, at most $|C[i,j]|+1$ of them, and find the number of α's in each such substring.

We now focus on finding the number of α's in a given parse. We need a few definitions. Define T_α to be the induced subtree of T on edges labeled α; *i.e.*, for any (x, y) labeled α with x the parent of y, connect x to v if v is the closest ancestor of x in T such that (u, v) is labeled α and u is the parent of v. It is clear that T_α's partition the edges of T and hence their total size is $O(|T|)$. We preprocess each T_α to answer Least Common Ancestor (LCA) queries between any two nodes in $O(1)$ time as in [13]. Also, we preprocess T_α so that for each node v, we know its depth $d(v)$ in T_α or equivalently the number of α's on the path from to the root to v. Also, we preprocess T by constructing an Euler tour (in-order traversal of the tree but we explicitly write down the node ID each time we visit it), writing down in an array E the sequence of node IDs that are encountered as well as the symbol on the edge leading to that node. It is easy to see that each node u in T has a leftmost and a rightmost position in E corresponding to the first and the last time u was visited; we denote these as L_u and R_u resp.

We can think of the query as counting α's labeling edges on the path in T from the root to the node u given by that parse. We do not want to spend time linear in the length of the parse because that is prohibitive. Our approach is to find the deepest edge (x, y) on the path to u that is labeled α. Then, we would be able to read off $d(y)$ that has been precomputed. We find the edge (x, y) as follows. We determine the closest occurrence a of α on the Euler tour to the left of L_u. Similarly, we determine the closest occurrence b of α to the right of $R(u)$. Each of this takes $O(\log\log N)$ time using the van Emde Boas structure [28]. Now our central claim is: *$LCA(a, b)$ in T_α gives y for u.* The proof involves case analyis which we omit here. ∎

We also present a different method that we call *decimated statistics*. First, for illustration, we will construct a simple data structure of size $O(p|C(w')|)$, where p is the size of the alphabet (the number of weight classes). Recall that, on query (i, j), we want the number of times weight class i occurs in $[0, j)$. We can build a binary search tree of size $O(|C(w')|)$ to find the parse that contains j (*i.e.*, to find a pointer into the trie to the node corresponding to the last symbol in the parse containing j). At each node in the trie, for each weight class i, we can store the answer to the query (i, j') for the case that j' is a position that ends a parse. Next, using a level-ancestor algorithm, we can find a node in the trie corresponding to position j. Finally, at each node in the trie corresponding to some parse prefix $[k, j)$, for each weight class i, we can store the number of occurrences of i in $[k, j)$. It is easy to see that this data structure suffices to answer the query in time $O(\log(N))$ and that the size of the data structure is at most $O(p|C(w')|)$.

There is a simple tradeoff between space and query time. For example, let $P \geq p$ be a parameter. We can build a binary search tree over the $|C(w')|/P$ parses whose position is a multiple of P, rather than building a search tree over all parses. At each such parse, for all weight classes, we store answers to all queries, as above. We can decimate a (sub-optimal) level-ancestor algorithm, *i.e.*, at each node, store $\log(N)$ pointers to ancestors $1, 2, 4, \ldots$ levels up, but only at trie levels that are multiples of $P\log(N)$. Finally, at each node in the trie corresponding to a parse prefix $[k, j)$, where the node is in a level that is a multiple of P, for each weight class i, we store the number of occurrences of i in $[k, j)$.

It is straightforward to check that the size of each data structure is reduced by the factor P. Each part of the query can be answered in time that is greater by at most the factor $P^{O(1)}$, provided we can efficiently walk the underlying data structure, *i.e.*, find a parent in the trie and find the successor parse to any given parse. The underlying data structure can be augmented to allow such walks in time (and space) $O(|C(w')|)$, but, in a plausible implementation of LZ78, the underlying compressed text *already supports these operations*. This concludes the construction.

Recall that $p \leq \log(M)/\epsilon$ is the number of weight classes in our histogram problem, which is the number of different symbols to comress. Taking P to be $\omega(p)$, we get the following:

Lemma 6 *Suppose a rounded workload w' of length N, multiplicative increment $(1 + \epsilon)$, and bound M, is compressed to $C(w')$ by* LZ78*. One can construct from $C(w')$, in time $O(p|C(w')|)$, a structure R', of size $o(|C(w')|)$, such that R' together with $C(w')$ constitute a symbol-range-count data structure of size* $(1+o(1))|C(w')|$*, with query time* $\mathrm{poly}(\log N, \log M, 1/\epsilon)$.

The decimated statistics approach is incomparable with Lemma 5, as the next theorem records. Combining the lemmas with our histogram algorithms we have, analogous to Theorem 1 (a similar result holds analogous to Corollary 1 as well)

Theorem 5 *There is an algorithm that, given parameters B, N, M, ϵ and* LZ78*-compressed text $C(w')$ of rounded weight vector w' with $p \leq O(\log(M)/\epsilon)$ classes, preprocesses $C(w')$ in time and symbol-range-count space $O(|C(w')|)$, reads data $\mathbf{A}$ with $\|\mathbf{A}\|_w^2 \leq M$ in time series, then outputs a B-bucket histogram $\mathbf{H}$ with $\|\mathbf{A} - \mathbf{H}\|_w^2 \leq (1 + O(\epsilon))\|\mathbf{A} - \mathbf{H}_{\mathrm{opt}}\|_w^2$, where $\mathbf{H}_{\mathrm{opt}}$ is the best possible B-bucket histogram representation to $\mathbf{A}$ under weight w. The algorithm uses space $O(poly(B, \log N, \log M, 1/\epsilon))$ in addition to the space associated with w independent of the input. The algorithm uses time $O(N)$ to read the stream of data and post-processing time $|C(w')|poly(B, \log N, \log M, 1/\epsilon)$ to build $\mathbf{H}$.*

*An alternative algorithm is as above except, for any parameter P (*e.g.*, a P that grows slightly faster than $p \approx \log(M)/\epsilon$), the preprocessing time degrades to $O(P|C(w')|)$, the symbol-range-count space improves to $(1+p/P)|C(w')|$, and the post-processing time changes to $poly(B, \log N, \log M, 1/\epsilon)$.*

7 Concluding Remarks

We have shown, for the first time, given a data set of length N and bound M, how to build a $(1 + \epsilon)$-near optimal B-bucket histogram that is provably nearly optimal with respect to a non-uniform workload, where the algorithm runs in nearly linear time $O(N + \mathrm{poly}(B, \log N, \log M, 1/\epsilon))$ and space $\mathrm{poly}(B, \log N, \log M, 1/\epsilon)$ beyond what is needed to store the workload. This algorithm generalizes to the dynamic update streaming model. For both time-series and dynamic maintenance update models, our time and space costs are comparable to those for the uniform workload. We have also shown that lossy compression of the workload is not possible beyond rounding to within the factor $(1+\epsilon)$. Finally, we show how to improve the space cost to essentially the space used to compress losslessly the rounded workload by the Lempel-Ziv algorithm.

Another way to solve the problem under the time-series model is based on the lockstep model, where we are given w_i *only* when $\mathbf{A}[i]$ arrives and we can not archive the entire workload w_i. This is what we call the *lockstep* model. In this model, we are able to give a different algorithm that can be run in time $O(N + \mathrm{poly}(B, \log U, \log M, 1/\epsilon))$ using space $\mathrm{poly}(B, \log U, \log M, 1/\epsilon))$ to construct the $(1+\epsilon)$-approximate histogram, where $U = \sum_i w_i$. This is somewhat weaker than our main result here, but the lockstep streaming model is weaker. So, this result may be of independent interest. We do not know whether near-linear time can be achieved in the lockstep model; this is a natural open question.

Many other open problems remain in the area of tracking synopses such as the histograms taking the workload into account. For example, a natural problem is to extend our work to two and higher dimensional $\mathbf{A}$'s. STholes is a known heuristic for multidimensional histograms that are workload-aware [4], but it will be of interest to study the complexity of this problem. Another natural problem is to not consider the workload on point queries alone, but also consider the workload on *range queries*. Note that for a signal of length N, a workload for range queries has size N^2—it would be interesting to reduce the workload storage space even to size N, even when the data is stored explicitly in space $\Theta(N)$. Database research such as [19, 5] has considered using range query workload to refine histograms; [21] has recently proposed changing the Haar basis to be workload-aware and find B-term wavelet synopsis for range workloads. Again, these results do not provide any theoretical guarantees on complexity and accuracy of the problem of computing $\mathbf{H}_{\mathrm{opt}}$ for range workloads, and provable results are of our interest. For *uniform* workload over the range queries, see results in [23] for the rangesum histogram problem.

References

[1] A. Aboulnaga and S. Chaudhuri. Self-Tuning Histograms: Building Histograms Without Looking at Data. Proc. ACM SIGMOD, 1999.

[2] S. Acharya, P. Gibbons, V. Poosala, S. Ramaswamy. The Aqua Approximate Query Answering System. SIGMOD Conference 1999: 574-576.

[3] O. Berkman and U. Vishkin. Finding level-ancestors in trees. Journal of Computer and System Sciences, 48 (2), 1994, 214 - 230.

[4] N. Bruno, S. Chaudhuri and L. Gravano STHoles: A Multidimensional Workload-Aware Histogram. Proc. ACM SIGMOD 2001.

[5] C. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. SIGMOD 1994.

[6] P. Ferragina, G. Manzini. Opportunistic Data Structures with Applications. FOCS 2000: 390-398.

[7] V. Ganti, M. Lee and R. Ramakrishnan. ICICLES–Self-tuning samples for approximate query answering. VLDB, 2000.

[8] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. STOC 2002: 389-398

[9] S. Guha. A note on wavelet optimization. 2004. http://www.cis.upenn.edu/ sudipto/notes/wavelet.pdf.gz

[10] S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Histogramming Data Streams with Fast Per-Item Processing. Proc 29'th ICALP, 681–692, 2002.

[11] S. Guha, N. Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. ICDE 2002: 567-

[12] S. Guha, N. Koudas, K. Shim. Data-streams and histograms. STOC 2001: 471-475. See also the journal version available at http://www.cis.upenn.edu/ sudipto/mypapers/histjour.pdf.gz.

[13] D. Harel, R. Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. SIAM J. Comput. 13(2): 338-355 (1984).

[14] Y. Ioannidis. The history of histograms (abridged). Proc. VLDB 2003.

[15] Y. Ioannidis, S. Christodoulakis. Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results. ACM Trans. Database Syst. 18(4): 709-748 (1993).

[16] Y. Ioannidis, V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. SIGMOD Conference 1995: 233-244.

[17] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, T. Suel. Optimal Histograms with Quality Guarantees. VLDB 1998: 275-286.

[18] E. Keogh, K. Chakrabarti, S. Mehrotra, M. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. SIGMOD Conference 2001.

[19] A. Konig and G. Weikum. Combining Histograms and Parametric Curve Fitting for Feedback Driven Query Result Size Estimation. Proc. VLDB, 1999.

[20] Y. Matias and D. Urieli. Optimal workload-based wavelet synopses, TR-TAU, February (revised, July), 2004. Also, ICDt 2005.

[21] Y. Matias and D. Urieli. Optimal wavelet synopses for range-sum queries, TR-TAU, July, 2004.

[22] S. Muthukrishnan. Data stream algorithms and applications. http: www.cs.rutgers.edu/ muthu/stream-1-1.ps. 2003.

[23] S. Muthukrishnan and M. Strauss. Rangesum histograms. ACM SODA 2003. 233-242.

[24] S. Muthukrishnan. Nonuniform sparse approximation theory with Haar wavelets. DIMACS Technical Report, 2004.

[25] V. Poosala. Histogram-based estimation techniques in database systems. Ph. D. Thesis, Univ of Wisconsin, 1997.

[26] L. Qiao, D. Agrawal, A. El Abbadi. RHist: adaptive summarization over continuous data streams. CIKM 2002: 469-476.

[27] M. Stillger, G. Lohman, V. Markl and M. Kandil. LEO - DB2's Learning Optimizer. VLDB 2001, 19–28.

[28] P. van Emde Boas. Preserving Order in a Forest in Less Than Logarithmic Time and Linear Space. Inf. Process. Lett. 6(3): 80-82 (1977).

[29] J. Ziv, A. Lempel. A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory 23(3): 337-343 (1977).

[30] J. Ziv, A. Lempel. Compression of Individual Sequences via Variable-Rate Coding. IEEE Transactions on Information Theory 24(5): 530-536 (1978).