Workload-Optimal Histograms on Streams

S. Muthukrishnan* M. Strauss† X. Zheng‡

December 8, 2004

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.

We present an algorithm for constructing a nearly-optimal histogram in nearly linear time and polylogarithmic space, in one pass. In the more general cash register model where data is streamed as a series of updates, we can build a histogram using polylogarithmic space, polylogarithmic time to process each item, and polylogarithmic post-processing time to build the histogram. These are the first known algorithmic results with provable guarantees for workload-optimal histogram construction, and rely on a notion of *linear robustness* we introduce here. All these results need the workload to be explicitly stored since we show that if the workload is summarized in small space lossily, algorithmic results such as above do not exist. 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, run-length encoding or a universal compression scheme of Lempel-Ziv.

1 Introduction

A histogram is a piecewise-constant approximation of an observed data distribution. A histograms 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].

*Supported by NSF ITR 0220280. Rutgers University. muthu@cs.rutgers.edu.

†Supported by NSF DMS 0354600. Dept. of Mathematics, University of Michigan, 2074 East Hall, 525 E. University Ave., Ann Arbor, Michigan, 48109-1109. Fax: +1-734-763-0937. martinjs@umich.edu

‡University of Michigan. xuanzh@eecs.umich.edu

Formally, 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 that is, the unique j with $b_j \leq i < b_{j+1}$. 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 choices are known [25] for b_j's and h_j's, but the 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 $\mathbf{H}_{\text{opt}}$ that minimizes the total square error. This algorithm has two drawbacks: first, it is expensive—quadratic in N; second, 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 first drawback, focus has been on $(1+\epsilon)$-*approximations*, that is, algorithms to find a histogram $\mathbf{H}$ such that $\|\mathbf{A} - \mathbf{H}\|^2 \leq (1+\epsilon)\|\mathbf{A} - \mathbf{H}_{\text{opt}}\|^2$. 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 sublinear 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]. 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 $c_1 N + \left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$ (for constants c_1 and c_2) in the time series model using no more than $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ space enroute [12, 11, 10]. In the dynamic maintenance model, [8] presents an algorithm that uses time per update, space, and post-processing time time only $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$. 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, then 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, given an input signal $\mathbf{A}[0 \cdots N-1]$ and *workload* $w[0 \cdots N-1]$, $0 \leq w_i \leq 1$, $\sum_i w(i) = 1$, the *workload-optimal* B-bucket histogram $\mathbf{H}_{\text{opt}}$ 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 $\mathbf{H}_{\text{opt}}$ is interesting on stored or streamed signals as well as stored or streamed workloads.

The database community has proposed methods not merely to synposize 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 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 $\mathbf{H}_{\text{opt}}$. There has 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 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, and not streamed. In such a scenario however, the dynamic programming from [17] immediately gives an $O(N^2B)$ time algorithm for finding the optimal $\mathbf{H}_{\mathrm{opt}}$, so the challenge in [24, 9] arises from working with the Haar wavelet basis and does not reflect on the difficulty in constructing $\mathbf{H}_{\mathrm{opt}}$. The significant open problem with finding $\mathbf{H}_{\mathrm{opt}}$ is when either the signal or the workload is streamed, or both are steamed.

In this paper, we address the problem of computing $\mathbf{H}_{\mathrm{opt}}$ on data streams from a foundational point of view. Our primary question is, do the powerful theoretical results known for uniform histogram construction in one pass and on data streams hold for the workload-aware case as well? Is there a difference in streaming the signal versus the streaming the workload? How accurately can we track the workload in polylogarithmic space? What is the information-content of the workload and how does it affect the complexity of histogram construction? We initiate the formal study of the complexity of workload-optimal histogram problems, and prove first-known theoretical results.

Our contributions are as follows. Suppose the data items are (positive, negative, or zero) integers, and the weights are positive integers between the minimum weight, $w_{\min}$, and the maximum weight, $w_{\max}$. Let $M = \max\{||A||^2, \frac{w_{\max}}{w_{\min}}\}$ and let c_1 and c_2 denote constants.

- (Stored $\mathbf{A}$, stored w) We present an $c_1 N + \left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$-time algorithm to compute a B-bucket histogram $\mathbf{H}$ with $\|\mathbf{A} - \mathbf{H}\|_w^2 \leq (1+\epsilon)\|\mathbf{A} - \mathbf{H}_{\mathrm{opt}}\|_w^2$ where $\mathbf{H}_{\mathrm{opt}}$ is the workload-optimal B-bucket histogram, with respect to arbitrary w. This is the first near-linear[1] time algorithm for approximating $\mathbf{H}_{\mathrm{opt}}$ under non-uniform workloads.

- (Stored w, streaming $\mathbf{A}$) The above algorithm can be run in the time series model taking only $O(1)$ time per new item and using $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ space and time to construct the $1+\epsilon$ approximate histogram. Under the more general dynamic maintenance mode, the above algorithm can be modified using previously known techniques so that the time per update, total space used, and postprocessing time are all $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$. This is the first known 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 $\mathbf{H}_{\mathrm{opt}}$ histograms for any w.

- (Streaming w, $\mathbf{A}$.) To get a $(1+\epsilon)$-factor approximation, one can round each workload weight to within the factor $(1+\Omega(\epsilon))$. But this is essentially all the lossy compression one can do, as our (straightforward) lower bounds show.

- (Compressed w, streaming $\mathbf{A}$.) We show that, if the vector w' of rounded weights is losslessly compressible to a structure $C(w')$ of size $|C(w')|$ by run-length encoding or 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

[1]Note that, for moderate values of the parameters other than N, the run time is dominated by $c_1 N$. In this paper, we use the term "near-linear" for this type of cost.

and time $C(w)\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$. This is an advantage for highly compressible workload w's, where $|C(w)| \ll o(|w|)$.

Section 2 has the definitions and preliminaries. Our $c_1 N + \left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$ time algorithm illustrates the main new ideas in this paper, and it is presented in 3. The natural starting point for this algorithm is to round the workload $w(i)$'s so there are only a few workload "classes", within each of which we build a robust histogram, a concept borrowed from [8]. The crux of the contribution here is to prove that combining the individual robust histograms gives a robust histogram for the overall data and workload. Extensions of this result to the stream models of the signal are straightforward and are sketched in Section 4. The lower bound on space when the workload is streamed is in Section 5 and is fairly simple. In 6, we describe methods for managing a compressed workload w and computing $\mathbf{H}_{\text{opt}}$ on a data stream of $\mathbf{A}$ efficiently. Our work initiates a number of open problems and directions which we list in Section 7.

In this extended abstract, proofs are omitted. They can be found in an appendix.

2 Preliminaries

2.1 Definitions and Lemmas from previous work

Definition 1 *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 $\|\mathbf{A}\|_w^2 = \langle \mathbf{A}, \mathbf{A} \rangle_w$ 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.*

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

Lemma 3 ([10]) *Given B, N, ϵ_r and M, for any integer-valued signal $\mathbf{A}$ with $\|\mathbf{A}\| \leq M$, there exists a $B' \leq \left(\frac{B \log N \log M}{\epsilon_r}\right)^{O(1)}$ and a $(B')^{O(1)}$ time algorithm to find a (B, ϵ_r)-robust approximation to $\mathbf{A}$ presented in the time-series model.*

For completeness, we sketch the construction from [10]. Convert the time series $\mathbf{A}$ to a time series of its *Haar wavelet* decomposition, in time $O(N)$ and spaced $O(\log(N))$. Using a buffer of size $O(B'')$ and a selection algorithm, find the Haar wavelet terms with largest coefficients, in time $O(N)$ and space $O(B'')$. Finally, use a greedy algorithm with appropriate stopping condition to select the B' largest wavelet terms, which can be regarded as the desired $O(B')$-bucket robust histogram.

Lemma 4 *Given B, N, and ϵ, there exists $\epsilon_r \geq (\epsilon/B)^{O(1)}$ such that a nearly-optimal representation $\mathbf{H}$ to a (B, ϵ_r)-robust representation $\mathbf{H}_{\text{r}}$ is also $\left(1 + \epsilon^{\Omega(1)}\right)$-nearly optimal to $\mathbf{A}$.*

3 Algorithm

In this section, we give an algorithm for time series data.

At a high level, our algorithm proceeds as follows. We will regard the weights as rounded to a power of $(1+\epsilon)$; there is a small number $p = \log_{1+\epsilon}(M)$ of these classes. We multiplex the incoming time series into

p new time series, according to the associated weight. For each subseries, we create a robust representation, as in [10]. Combining the robust representations gives a linearly-robust representation $\mathbf{H}_\mathrm{r}$, that we define below. Finally, a near-best B-bucket representation to $\mathbf{H}_\mathrm{r}$ can be constructed efficiently and will also be near-best to the original data, $\mathbf{A}$.

3.1 Notation and Basics

We consider signals of length N, with weights $w_1, \cdots, w_N$, and such that $\|\mathbf{A}\|^2 \leq M$. We will assume that data items are integers (positive, zero, or negative) and that weights are positive integers in the range $w_{\min} = 1$ to some $w_{\max} \leq M$ (so we abandon the normalization $\sum_i w_i = 1$.)

Definition 5 *Rounded weights with respect to original weights: 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., $\forall i \in \{1, \cdots, N\}, \exists j \in \{1, \cdots, p\}$, such that $w^j = w'_i$ and*

$$w_i \leq (1+\epsilon) w'_i \leq (1+\epsilon) w_i. \tag{1}$$

We use $w' = (w'_1 \cdots w'_N)$ to represent the length-N rounded weight vector.

In the sequel, any rounding scheme will work if it satisfies Equation (1) and gives just a moderate number of possible different rounded weight values.

Lemma 6 *Fix a single $\mathbf{A}$ of dimension N. Then $\|\mathbf{A} - \mathbf{H}'_{\mathrm{opt}}\|_w^2 \leq (1+\epsilon)\|\mathbf{A} - \mathbf{H}_{\mathrm{opt}}\|_w^2$, 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'.*

Definition 7 *A* symbol-range-count structure *for a rounded weight vector w' supports the following operations:*

- *For all $j < N$, we can recover w'_j in constant time per query, provided we are queried with all j's in order, $j = 0, 1, 2, \ldots, N-1$.*
- *For all $j < N$, we can recover the number of $k < j$ such that $w'_k = w'_j$.*

The time for the second operation will be called the (symbol-range-count) *query* time, and must be at most $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$. Besides this, we will be interested in the *size* of the structure and the *pre-processing* time needed to construct the structure. (All implementations we consider will support the first query in the time indicated.)

It is straightforward to build such structure R of size $O(N)$, in pre-processing time $O(N)$, that answers both queries in constant time (using, say, perfect hashing), for any order of queries. In Section 6, we will be interested in smaller R's and we will exploit the particular requirements listed here.

3.2 Linearly Robust Representations

Definition 8 *Linearly Robust Representations with respect to weights: Fix a signal $\mathbf{A}$ and rounded weight vector w'. A representation $\mathbf{H}_\mathrm{r}$ is called a (B, ϵ)-robust approximation to $\mathbf{A}$ under weight w', if, for any B-bucket histogram H_B and any scalars a and b, we have*

$$(1-\epsilon)\|\mathbf{A} - \mathbf{H}_\mathrm{r}\|_w^2 \leq \|\mathbf{A} - (a\mathbf{H}_\mathrm{r} + b\mathbf{H}_B)\|_w^2.$$

Definition 9 *Partition and combination of representations: Given a stream* $\mathbf{A}$ *and given a partition* $\mathcal{P} = \{P_0, P_1, \ldots, P_{m-1}\}$ *of* $[0, N)$*, for each* $i \in [0, m)$*, define* $\mathbf{A}_i^{\mathcal{P}}$ *to be* $\mathbf{A}$ *restricted to* P_i*, viewed as a substream. Define* $\#_{i\in[0,m)}\mathbf{A}_i^{\mathcal{P}}$ *as the inverse (recombination) operator, so, for example,* $\mathbf{A} = \#_{i\in[0,m)}\mathbf{A}_i^{\mathcal{P}}$ *or, briefly,* $\mathbf{A} = \#\mathbf{A}_i^{\mathcal{P}}$*.*

Definition 10 *Given a stream* $\mathbf{A}$ *and vector* w' *of rounded weights, define the partition* $\mathcal{P}$ *of* $[0, N)$ *as follows: For each* $i \in [0, N)$*, if* $w'_i = w^j$*, put* i *into the* j*'th group. Let* $\mathbf{H}^i$ *be a* (B, ϵ)*-robust representation for* $\mathbf{A}_i^{\mathcal{P}}$*. Define* $\mathbf{H}_\mathrm{r}$ *to be* $\#\mathbf{H}^i$*.*

Lemma 11 *The representation* $\mathbf{H}_\mathrm{r}$ *is a* (B, ϵ)*-robust representation to* $\mathbf{A}$*, under weight* w'*.*

Lemma 12 *There are two constants* c_1 *and* c_2*, s.t.,* $\mathbf{H}_\mathrm{r}$ *can be computed in* $c_1 N + \left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$ *time.*

Note that we need constant time per item to multiplex each data item in the original stream into the corresponding substream, according to $\mathcal{P}$. We then run an algorithm on each substream that takes time linear plus $\left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$. Since the substreams' total length is N, the result follows.

We use an array S_i to store each $\mathbf{H}^i$ in the form $(w^i : h_1, l_1, h_2, l_2 \cdots, h_{B'}, l_{B'})$, where h_i is the height and l_i is the right boundary. By lemma 3, $\mathbf{H}_\mathrm{r}$ has some $B' \leq \left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ buckets. Thus, on input j, we can find $\mathbf{H}_\mathrm{r}[j]$ in time $O(\log(B'))$ by using a simple index structure. (Other data structures are possible.)

Lemma 13 *Given* $\mathbf{H}_\mathrm{r}$*, there is an algorithm that takes as input parameters* p, B', N*, and* M *and histogram* $\mathbf{H}_\mathrm{r} = \#\mathbf{H}^i$ *equal to the #-combination of* p B'*-bucket histograms with respect to a partition* $\mathcal{P}$*, and, on query* $[i, j)$*, computes the best height* h *to* $\mathbf{H}_\mathrm{r}$ *on the interval* $[i, j)$ *and the associated error* $\sum_{k\in[i,j)} |\mathbf{H}_\mathrm{r}[k] - h|^2$*. The algorithm's runtime is at most the time to perform* $O(pB')$ *symbol-range-count queries.*

(Recall that a simple symbol-range-count structure requires preprocessing time and space $O(N)$ and supports queries in constant time.)

3.3 Histogram extracted from robust representation

As in [10], our strategy is to compute a nearly-optimal representation $\mathbf{H}$ to $\mathbf{H}_\mathrm{r}$ and show that it is also nearly optimal for $\mathbf{A}$.

Lemma 14 *Fix a signal* $\mathbf{A}$ *with rounded weight* w'*, and let* $\mathbf{H}_\mathrm{r}$ *be a* (B, ϵ)*-robust approximation to* $\mathbf{A}$ *under weight* w'*. Let* $\mathbf{H}$ *be a* B*-bucket* $(1 + \epsilon)$*-nearly optimal histogram for* $\mathbf{H}_\mathrm{r}$*. Then*

$$\|\mathbf{A} - \mathbf{H}\|_{w'}^2 \leq (1 + O(\epsilon))\|\mathbf{A} - \mathbf{H}'_{\mathrm{opt}}\|_{w'}^2$$

where $\mathbf{H}'_{\mathrm{opt}}$ *is the optimal* B*-bucket representation to* $\mathbf{A}$ *under weight* w'*.*

Now we consider efficient computation of $\mathbf{H}$.

Lemma 15 *Given parameters* B, N, M, p, w'*, and* ϵ*, and robust representations* $\mathbf{H}^1, \cdots, \mathbf{H}^p$ *of length* N *with each* $\|\mathbf{H}^i\| \leq M$*, using a symbol-range-count structure with query time in* $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$*, and* $O(B^2/\epsilon)$ *additional space, we can output a* B*-bucket representation* $\mathbf{H}$ *in* $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ *time, with*

$$\|\mathbf{H}_\mathrm{r} - \mathbf{H}\|_{w'}^2 \leq (1 + \epsilon)\|\mathbf{H}_\mathrm{r} - \widehat{\mathbf{H}}\|_{w'}^2$$

where $\widehat{\mathbf{H}}$ *is the best* B*-bucket representation to* $\mathbf{H}_\mathrm{r}$*.*

3.4 Main Results, Stored Weights

Thus we have, using a simple symbol-range-count structure,

Theorem 16 *There is an algorithm that, given parameters B, N, M, ϵ, weight vector w, preprocess 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 $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ 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 $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ to build $\mathbf{H}$.*

4 Streaming Data

In this section, we consider dynamic or "turnstile" data. Specifically, we are presented with a stream of updates of the form (i, v), meaning, "add v to $\mathbf{A}[i]$." (Here v may be positive or negative.) We have the following from [8]:

Lemma 17 *For parameters N, M, B, ϵ_r, there is a randomized data structure for an array $\mathbf{A}$ that requires space $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ and supports the following operations in time $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ (some only with overwhelming probability):*

- *update: Add v to $\mathbf{A}[i]$*
- *build: Build a robust representation to the then-current dataset $\mathbf{A}$ with respect to the uniform workload.*

Thus we immediately have:

Theorem 18 *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 $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$, and supports the following operations in time $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ (some only with overwhelming probability):*

- *update: Add v to $\mathbf{A}[i]$*
- *build: Build a $(1 + \epsilon)$-near optimal histogram with respect to the then-current dataset $\mathbf{A}$, under workload w.*

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. A similar bound can be shown if we read the workload first. Above we showed that, to get a $(1 + \epsilon)$-factor approximation, one can round weights to a power of $(1 + \epsilon)$. We now show that, in a sense, this is the only kind of lossy compression that is possible.

Theorem 19 *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 $\approx N$, up to the factor $(1 + O(\epsilon))$.*

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 two types of compression, run-length encoding and Lempel-Ziv-78. Our goal in this section is to build a symbol-range-count structure R to match the given compression scheme. 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 $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ 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. This has the advantage that the total size $|R|$ can be compared directly to $|C(w')|$, without giving up any constant factors; this is useful if, say, $|C(w')| = |w'|/100$.

6.1 Run-length Encoding

A *run* of a sequence is a maximal substring that uses just a single character. Run-length encoding is a simple scheme whereby each run is replaced by a single copy of the character used and the length of the run. We'll denote the run-length encoding of the rounded workload w' by $C(w')$ and its length by $|C(w')|$.

Next, consider building a symbol-range-count structure to match the run-length encoding bound. It is easy to see that the run-length encoding $C(w')$ itself lets us recover w'_j in constant time for queries $j = 0, 1, 2, \ldots, N-1$ in order. For all $j < N$, we can recover the number of $k < j$ such that $w'_k = w'_j$ in time polynomial in $|C(w')|$. For some workloads, $|C(w')|$ is so small that this straightforward approach is reasonable. For example, theoretically, $|C(w')|$ can be constant-sized. Alternatively, in some applications, the workload is monotonic, so that more recent data is queried more often than older data. In that case, there is exactly one run per rounded weight, so $|C(w')| = \log(M)/\epsilon$, which is often small.

In other situations, however, $|C(w')|$ is larger, and we would want to do better in terms of symbol-range-count query time. First, for any j, we can find, in time $O(\log(|C(w')|)$, the run containing j by using an index structure of size $|C(w')|$. (The classic van Emde Boas data structure [28] for the "predecessor query" improves this time to $O(\log\log(N))$.) Then store, at each run r, for all possible rounded weights v, the number of k's preceding the run such that $w'_k = v$. Unfortunately, the space requirement for this is $O(|C(w')| \log(M)/\epsilon)$, which is (theoretically) unacceptable. A simple alternative is to store these statistics only for one out of every $\log(M)/\epsilon$ runs, so the space is now acceptably $O(|C(w')|)$. The query time is now $\log(M)/\epsilon$ rather than constant time, but this multiplies (and, therefore, blends into) the comparable expression $(B \log(N) \log(M)/\epsilon)^{O(1)}$. We call this technique "decimated statistics." Finally, we note that one can smoothly tradeoff query time and additional space. That is, one can store statistics somewhat more sparsely than one out of every $\log(M)/\epsilon$ runs, getting an auxiliary structure R' of size $|R'| < o(|C(w')|)$. The other index structures can similarly be decimated appropriately. Thus we have:

Theorem 20 *Given a vector w' over an alphabet Σ (here, of rounded weights) with run-length encoding $C(w')$, one can, in time $O(|C(w')|)$, construct an R' of size $|R'| < o(|C(w')|)$, such that $C(w')$ together with R' constitute a symbol-range-count structure with query time $(\log(N)|\Sigma|)^{O(1)}$.*

6.2 Lempel-Ziv

In this section, we consider the Lempel-Ziv algorithm. 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.

We are given a string $S[1, ..., N]$, with each $S[i]$ in alphabet set of size m given by $\{1, 2, ..., m\}$. 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, but we omit the details in this extended abstract.

Theorem 21 *A string S given in its* LZ78 *compressed form $C(S)$ can be preprocessed in time and space $O(|C(S)|)$. 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 $[i, j]$.*

Finally, we note that the decimated statistics technique applies also to LZ78 and to other dictionary/trie compression techniques. We need to store decimated statistics for prefixes of rounded weights within parses and also for prefixes of parses within the text. We also need to decimate the level-ancestor algorithm and other indices.[2]

This result is incomparable with Theorem 21, since Theorem 21 gives query time $|C[i, j]| \log \log(N)$ for range $[i, j]$ whereas the decimation technique gives time polynomial in $\log(M)/\epsilon$; either of these may be bigger than the other. We note, however, that, in the context of our algorithm, we will query on a set of ranges $[i, j]$ that (several times) cover $[0, N)$, so every parse gets touched. Since the minimum number of parses in LZ78 is $\sqrt{N}$, it follows that the overall contribution to the runtime of Theorem 21 is at least $\sqrt{N}$. Also note that the decimation technique appears to be the only technique that works for certain other dictionary/trie compression schemes, like LZ77.

The decimated statistics technique can also be used with LZ78 to give sublinear auxiliary space cost. That is:

[2]Note that, in our context, we can accept query time $(\log(N) \log(M)/\epsilon)^{O(1)}$, so level-ancestor and predecessor algorithms can be implemented in a manner that is much more straightforward than some of the optimal implementations. No new data structure is needed beyond basic engineering.

Theorem 22 *Given vector w' over alphabet Σ with* LZ78 *$C(w')$ under a plausible implementation, one can, in time $O(|C(w')|)$, construct an R' of size $|R'| < o(|C(w')|)$, such that $C(w')$ together with R' constitute a symbol-range-count structure with query time $(\log(N)|\Sigma|)^{O(1)}$.*

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 also bounded by M, where the algorithm runs in nearly linear time $c_1 N + \left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$ and space $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ beyond what is needed to store the workload. This algorithm generalizes to the dynamic additive update model. For both time-series and addtive 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 either run-length encoding or the Lempel-Ziv algorithm.

Many open problems remain in the area of tracking synopsis 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*. 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 $\mathbf{H}_{\text{opt}}$ 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.

A Appendix—Proofs

Lemma 6 *Fix a single* $\mathbf{A}$ *of dimension* N*. Then* $\|\mathbf{A} - \mathbf{H}'_{\rm opt}\|_w^2 \leq (1+\epsilon)\|\mathbf{A} - \mathbf{H}_{\rm opt}\|_w^2$ *, where* $\mathbf{H}_{\rm opt}$ *is the optimal* B *bucket representation to* $\mathbf{A}$ *under weight* w*, and* $\mathbf{H}'_{\rm opt}$ *is the optimal* B *bucket representation to* $\mathbf{A}$ *under weight* w'*.*

Proof. We have

$$\begin{aligned}
\|\mathbf{A} - \mathbf{H}'_{\rm opt}\|_w^2 &= \sum_{i=1}^{N} [\mathbf{A}_i - \mathbf{H}'_{\rm opt}(i)]^2 w_i \\
&\leq (1+\epsilon) \sum_{i=1}^{N} [\mathbf{A}_i - \mathbf{H}'_{\rm opt}(i)]^2 w'_i \\
&\leq (1+\epsilon) \sum_{i=1}^{N} [\mathbf{A}_i - \mathbf{H}_{\rm opt}(i)]^2 w'_i \\
&\leq (1+\epsilon) \sum_{i=1}^{N} [\mathbf{A}_i - \mathbf{H}_{\rm opt}(i)]^2 w_i \\
&= (1+\epsilon)\|\mathbf{A} - \mathbf{H}_{\rm opt}\|_w^2.
\end{aligned}$$

∎

Lemma 11 *The representation* $\mathbf{H}_{\rm r}$ *is a* (B, ϵ)*-robust representation to* $\mathbf{A}$*, under weight* w'*.*

Proof. The proof is derived from the definition of robustness.

Given scalars a and b and a B-bucket representation $\mathbf{H}_B$, multiplex $a\mathbf{H}_{\rm r} + b\mathbf{H}_B$ into p sub-streams $\mathbf{H}_1^*, \cdots, \mathbf{H}_p^*$, based on the partition $\mathcal{P}$ derived from w'. Notice that the boundaries of each $\mathbf{H}_i^*$ will fall in the boundaries of corresponding $\mathbf{H}^i$ and no more than $B-1$ additional boundaries respectively, since the representation $\mathbf{H}_B$ itself takes no more than $B-1$ additional boundaries. According to the definition of robustness with respect to uniform workloads, we have for the i'th stream:

$$\|\mathbf{A}_i^{\mathcal{P}} - \mathbf{H}_i^*\|^2 \geq (1-\epsilon)\|\mathbf{A}_i^{\mathcal{P}} - \mathbf{H}^i\|^2$$

By scaling, we have for the i'th stream:

$$(1+\epsilon)^{i-1}\|\mathbf{A}_i^{\mathcal{P}} - \mathbf{H}_i^*\|^2 \geq (1-\epsilon)(1+\epsilon)^{i-1}\|\mathbf{A}_i^{\mathcal{P}} - \mathbf{H}^i\|^2.$$

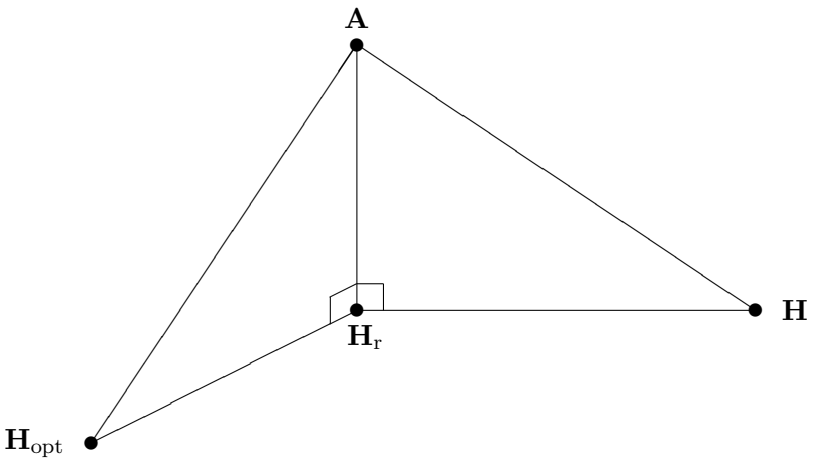
By summing the inequalities for all the sub-streams together, we have

$$\|\mathbf{A} - \mathbf{H}^*\|_{w'}^2 \geq (1-\epsilon)\|\mathbf{A} - \mathbf{H}_{\rm r}\|_{w'}^2$$

So $\mathbf{H}_{\rm r}$ is a (B, ϵ)-robust representation to $\mathbf{A}$ under weight w'. ∎

Lemma 13 *Given* $\mathbf{H}_{\rm r}$*, there is an algorithm that takes as input parameters* p, B', N*, and* M *and histogram* $\mathbf{H}_{\rm r} = \#\mathbf{H}^i$ *equal to the* $\#$*-combination of* p B'*-bucket histograms with respect to a partition* $\mathcal{P}$*, and, on query* $[i, j)$*, computes the best height* h *to* $\mathbf{H}_{\rm r}$ *on the interval* $[i, j)$ *and the associated error* $\sum_{k\in[i,j)} |\mathbf{H}_{\rm r}[k] - h|^2$*. The algorithm's runtime is at most the time to perform* $O(pB')$ *symbol-range-count queries.*

Figure 1: Illustration of histograms in Lemma 14. By optimality of $\mathbf{H}_{\mathrm{r}}$, there are near right angles as indicated.



Proof. Let $w_{\mathrm{sum}} = \sum_{k=i}^{j} w'_k$. Define a random variable X by letting $X = \mathbf{H}_{\mathrm{r}}[k]$ with probability $\frac{w'_k}{w_{\mathrm{sum}}}$, for $k \in [i, j)$ at random. Then $E[X] = h$ is the height of the optimal one-bucket representation to $\mathbf{H}_{\mathrm{r}}$ between i and j. Therefore, $h = \frac{\sum_{k=i}^{j} w'_k \mathbf{H}_{\mathrm{r}}[k]}{\sum_{k=i}^{j}(w'_k)} = \frac{\sum_{k=i}^{j} w'_k \mathbf{H}_{\mathrm{r}}[k]}{w_{\mathrm{sum}}}$.

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 $[i, j)$ into p substreams and at most B' buckets from the corresponding $\mathbf{H}^i$. For each such bucket, we need to make a single symbol-range-count query.

Computing the error, $\sum_{k=i}^{j} w'_k(\mathbf{H}_{\mathrm{r}}[k] - h)^2$, is similar. ∎

Lemma 14 *Fix a signal* $\mathbf{A}$ *with rounded weight* w'*, and let* $\mathbf{H}_{\mathrm{r}}$ *be a* (B, ϵ)*-robust approximation to* $\mathbf{A}$ *under weight* w'*. Let* $\mathbf{H}$ *be a* B*-bucket* $(1 + \epsilon)$*-nearly optimal histogram for* $\mathbf{H}_{\mathrm{r}}$*. Then*

$$\|\mathbf{A} - \mathbf{H}\|_{w'}^2 \leq (1 + O(\epsilon))\|\mathbf{A} - \mathbf{H}'_{\mathrm{opt}}\|_{w'}^2$$

where $\mathbf{H}'_{\mathrm{opt}}$ *is the optimal* B*-bucket representation to* $\mathbf{A}$ *under weight* w'*.*

Proof. Extend the proof from [8, 10] under uniform weight to weight w'. Notice that, the proof under uniform weight uses robustness, the triangle inequality and the Pythagorean theorem.[3] Since all these properties hold under weight w', we can get the above result. Roughly speaking, the linear robustness property insures that there is are near-right angles at $\mathbf{A}$-$\mathbf{H}_{\mathrm{r}}$-$\mathbf{H}_{\mathrm{opt}}$ and $\mathbf{A}$-$\mathbf{H}_{\mathrm{r}}$-$\mathbf{H}$. Since the leg $\mathbf{A}$-$\mathbf{H}_{\mathrm{r}}$ is the same in the two triangles and since $\mathbf{H}$-$\mathbf{H}_{\mathrm{r}}$ is shorter than $\mathbf{H}_{\mathrm{opt}}$-$\mathbf{H}_{\mathrm{r}}$, it follows that $\mathbf{H}$-$\mathbf{A}$ is not much longer than $\mathbf{H}_{\mathrm{opt}}$-$\mathbf{A}$. See Figure 1. ∎

Lemma 15 *Given parameters* B, N, M, p, w'*, and* ϵ*, and robust representations* $\mathbf{H}^1, \cdots, \mathbf{H}^p$ *of length* N *with each* $\|\mathbf{H}^i\| \leq M$*, using a symbol-range-count structure with query time in* $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$*, and* $O(B^2/\epsilon)$ *additional space, we can output a* B*-bucket representation* $\mathbf{H}$ *in* $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ *time, with*

$$\|\mathbf{H}_{\mathrm{r}} - \mathbf{H}\|_{w'}^2 \leq (1 + \epsilon)\|\mathbf{H}_{\mathrm{r}} - \widehat{\mathbf{H}}\|_{w'}^2$$

[3] In this context, the Pythagorean Theorem says that if $\langle \mathbf{C} - \mathbf{A}, \mathbf{B} - \mathbf{A} \rangle_w = 0$, then $\|\mathbf{C} - \mathbf{B}\|_w^2 = \|\mathbf{C} - \mathbf{A}\|_w^2 + \|\mathbf{B} - \mathbf{A}\|_w^2$. The vectors $\mathbf{C} - \mathbf{A}$ and $\mathbf{B} - \mathbf{A}$ have a near-right angle if $\langle \mathbf{C} - \mathbf{A}, \mathbf{B} - \mathbf{A} \rangle_w \leq \epsilon^{O(1)} \|\mathbf{C} - \mathbf{A}\| \|\mathbf{B} - \mathbf{A}\|$.

where $\widehat{\mathbf{H}}$ is the best B-bucket representation to $\mathbf{H}_{\mathrm{r}}$.

Proof. Our proof is as in [10] (similar to [12, 11]), so we give just a brief sketch. We use dynamic programming to get a B-bucket representation $\mathbf{H}$ to $\mathbf{H}_{\mathrm{r}}$. First assume that we know an approximation E to the optimal error E_{opt}, satisfying $E_{\mathrm{opt}} \leq E \leq 2E_{\mathrm{opt}}$, where $E_{\mathrm{opt}} = \|\mathbf{H}_{\mathrm{r}} - \widehat{\mathbf{H}}\|_{w'}^2$, and $\widehat{\mathbf{H}}$ is the optimal representation to $\mathbf{H}_{\mathrm{r}}$ under weight w'. Define $\mathrm{Far}[j,l]$ to be a position x such that some j-bucket histogram on $[0,x)$ has error at most $(\ell+j+1)\frac{\epsilon}{2B}E$ but no j-bucket histogram on $[0,x+1)$ has error at most $(\ell+j)\frac{\epsilon}{2B}E$. We build a $O(B^2/\epsilon)$-sized table T to store the information, for each $j \leq B$ and each $l \leq O(B/\epsilon)$. For each entry of T, we compute:

$$T[j][l] = \mathrm{Far}[j,l] = \max_{l_1+l_2=l+1} x : \mathrm{cost}(\mathrm{Far}[j-1,l_1],x) \leq l_2$$

where $\mathrm{cost}(l,r)$ is the difference of $\mathbf{H}_{\mathrm{r}}$ and the optimal one-bucket representation between index l and index r to it under weight w'.

For some optimal histogram, let e_j be the error in the j'th bucket, and let $m_j = \lceil e_j * 2B/(\epsilon E)\rceil$. Then, inductively, our dynamic program will do at least as well as using error bound $m_j\frac{\epsilon}{2B}E$ for the j'th bucket, which means the boundaries it finds will all be equal to or to the right of the corresponding boundaries in the optimal histogram. The overall error is suboptimal by $\frac{\epsilon}{2B}E$ per bucket, which is at most $\epsilon E_{\mathrm{opt}}$ overall. Thus the overall error is $(1+\epsilon)E_{\mathrm{opt}}$, as desired.

Note that, given $1 \leq E_{\mathrm{opt}} \leq M$, we could use O($\log\log M$) time to find proper E by binary search. ∎

Theorem 19 *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 $\approx N$, 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 position corresponding to v_j. If $v_j > (1+\epsilon)^{k+1}$, then the best 3-bucket histogram gets v_j right and is 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 $(1+\epsilon)^2 = (1+O(\epsilon))$. ∎

Theorem 21 *A string S given in its* LZ78 *compressed form $C(S)$ can be preprocessed in time and space $O(|C(S)|)$. 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. Now our central claim is: $LCA(a, b)$ *in* T_α *gives* y *for* u. The proof involves case analyis which we omit here. ∎

Theorem 22 *Given vector w' over alphabet Σ with* LZ78 *$C(w')$ under a plausible implementation, one can, in time $O(|C(w')|)$, construct an R' of size $|R'| < o(|C(w')|)$, such that $C(w')$ together with R' constitute a symbol-range-count structure with query time $(\log(N)|\Sigma|)^{O(1)}$.*

Proof. The proof assumes that $C(w)$ itself supports the following operations:

- Given a pointer into the decompression trie, find a pointer to the parent in time $O(1)$.
- Given a trie node n that corresponds to the end of some parse π, find, in time $O(1)$, the node that corresponds to the end of the next parse.

These are plausible assumptions about a LZ78 implementation. We omit the details of the algorithm engineering involved in decimation. ∎

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).