Maintaining Time-Decaying Stream Aggregates

Edith Cohen Martin Strauss
AT&T Research Labs
180 Park Ave.
Florham Park, NJ, USA.
{edith,mstrauss}@research.att.com

ABSTRACT

We formalize the problem of maintaining *time-decaying* aggregates and statistics of a data stream: the relative contribution of each data item to the aggregate is scaled down by a factor that depends on, and is non-decreasing with, elapsed time. Time-decaying aggregates are used in applications where the significance of data items decreases over time. We develop storage-efficient algorithms, and establish upper and lower bounds. Surprisingly, even though maintaining decayed aggregates have become a widely-used tool, our work seems to be the first both to explore it formally and to provide storage-efficient algorithms for important families of decay functions, including polynomial decay.

1. INTRODUCTION

The advent of high speed communication networks and massive transient datasets has made the streaming data model and stream management systems the focus of much recent work (see [1] for an excellent survey). In the data stream model, data arrives sequentially and is processed by an algorithm whose workspace is insufficient to store all the data, so the algorithm must process and then discard each data item. The algorithm maintains a *summary* whose storage requirement is considerably smaller than that of the stream. Using this compact summary, the algorithm is able to answer queries about the data stream.

One of the most fundamental queries on a stream of values is a sum, or, equivalently, an average. Consider a stream of nonnegative integers. It is not hard to see that exactly maintaining the sum (or average) requires storage of $\Theta(\log(n))$ bits (where n is the value of the sum). Work of Morris [11] and subsequent folklore work shows how to maintain an approximate value of the sum using only $O(\log\log(n))$ storage bits.

In many applications, however, older data items are less significant than more recent ones, and thus should contribute less. This is because the characteristics or "state" of the data generator may change over time, and, for purposes such as prediction of future behavior or resource allocation, the most recent behavior should be given larger weight. A *decay function* determines the relative weight of each data item, as a function of elapsed time since the item was observed. A *time-decayed sum* or *average* of the stream weighs each item according to its elapsed time.

PODS 2003, June 9–12, 2003. San Diego, California, USA.

1.1 Applications

There has been an abundance of applications work with time-decayed aggregates. We list some examples we are aware of. The listed applications use time-decaying averages to estimate network delay or congestion or connection activity, but many other applications are conceivable.

- **The Random Early Detection (RED) protocol.** RED is a popular protocol deployed in Internet routers for congestion avoidance and control. RED uses the weighted average of previous queue lengths to estimate the impending congestion at the router; the estimate is then used to determine what fraction of packets to discard [9, 6].
- **Holding-time policies for ATM virtual circuits.** Circuit-switched data connections have an associated cost of being kept open; data transfers are bursty and intermittent and, when a data burst arrives, it incurs smaller delay when the circuit is open. Thus, when there are multiple connections, it is important to assess the anticipated idle times (time to the next data burst) of each circuit and to close first those circuits that have longer anticipated idle times. This can be done using a time-decaying weighted average of previous idle times [10]. A similar model applies to maintaining open TCP connections at a busy Web server [3].
- **Internet gateway selection products.** Multiple Internet paths are available to each destination host and the product needs to assess the reliability of each path in order to facilitate a better selection of a path. A time-decayed average of previous measurements can be used as a measure of path quality [2].
- **Maintaining statistics about usage patterns of AT&T telecom customers.** In this application, a summary is maintained per field on each of around 100 million customers; thus, optimal balancing of information value and available storage is very important [4].

1.2 Decay functions

Much of the applications of time-decayed aggregates predate *formal* analysis of stream algorithms, and the most commonly used decay function is Exponential decay, which constituted the only natural alternative for which simple and efficient algorithms were known. The exponentially-decaying weighted average can be maintained easily in a single counter, C, by the formula $C \leftarrow (1-w)\,x + wC$, where $0 < w < 1$ determines the rate of decay and x is the new data value. The required storage for approximately-maintaining this value is $\Theta(\log N)$, where N is elapsed time.[1]

Another natural model for discounting older data is *Sliding Windows*. For a parameter N, we want to maintain the count of 1's (or sum of values) over the last N time units. It is not hard to see that maintaining an exact count requires $\Omega(N)$ storage bits, as the algorithm must track the most recent N data items exactly. Sliding Window decay has recently been addressed in [5], where the authors consider the problem of maintaining an *approximate* count. They give an algorithm (the "exponential histogram," EH) and a matching lower bound for maintaining this count using $\Theta(\log^2(N))$ storage bits. Further work on maintaining sliding window decay for distributed streams was performed by Gibbons and Tirthapura [7].

Note that there is an exponential gap between the storage required for approximately maintaining a non-decaying sum ($\Theta(\log\log N)$) and the storage required for Exponential or Sliding-Window decaying sums. In turn, Sliding Window decay ($\Theta(\log^2 N)$ storage bits) is quadratically worse than Exponential decay ($\Theta(\log N)$ bits).[2] These differences in required storage can be very significant for storage-strained applications.

We have, so far, seen two particular families of decay functions: sliding windows and Exponential decay. Both families are well understood, in terms of storage requirements, and time-decaying aggregates according to these functions can be approximately maintained with polylograithmic storage. We next identify some desired properties of decay functions. We then argue, using examples, that these two families are "insufficient:" First, they do not possess some intuitive properties, and, second, they do not provide a rich-enough class of decay rates, as the desired rate of decay for a particular application depends on the time scales of correlations between values. Both sliding windows and exponential decay discount older data very severely. Intuitively, we expect the significance of an event indeed to decay with elapsed time, but also to be larger for a more severe event. For different applications, we would like to be able to tune the balance of severity with time-decay.

As an illustrative example, we consider a time-decayed aggregate of past performance as a measure of availability of network links. We consider two links (see Figure 1), L1 and L2. Suppose that link L1 fails for a 5-hour duration and, 24 hours later, alternative link L2 experiences a 30-minute long failure. The two links were otherwise reliable and no more failures are observed later on. We would like to compute online simple numerical ratings that capture the relative reliability of the two links. The link L1 experienced a more severe failure event than L2, but less recently than L2. One would imagine that the initial rating of the two links should depend on the application and general properties of the failure patterns and the desired balance between the severity of an event and the rate of time-decay. Thus, one may initially want to deem one of the links as more reliable or deem both of them to be similarly reliable. We would like to have a rich enough range of decay rates to support these possibilities. Regardless of the initial rating, as time progresses, and the time difference between the failure events become small relative to elapsed time, we expect that L2, which experienced a less-severe failure, to emerge eventually as more reliable than L1.

Consider first using a sliding window decay function for our two links. A small window size would completely discount the failure event experienced by L1, thus viewing it as a more reliable link, and later on as an equally-reliable link. A larger window size will provide a view that changes from one that deems L2 to be much more reliable than L1 to one that deems L1 to be much more reliable. This counters our intuition where we expect the relative reliability rating of L2 to increase as time progresses.

We next consider using an Exponential decay function. Exponential decay has the property that the relative contribution of both failure events remains fixed through time, and, thus, its "view" on the relative reliability of links remains fixed as time progresses. So depending on the decay parameter, either L1 or L2 will be consistently viewed as the more reliable link. Again, this view counters our intuition.

In particular, no member in these two families can provide a view where first L1 is rated as more reliable and then L2 emerges as more reliable. We can now ask which decay functions do fulfill these intuitive expectations? A key property is that the weights of two items become closer as time progresses. We noted above that this property does not hold for Exponential decay or sliding windows. Smooth subexponential decays do have this property, and one particularly appealing family of decay functions is polynomial decay, where the weight given to a data item is inverse-polynomial in elapsed time. These functions can also tremendously expand our ability to tune the rate of decay. As far as we know, however, until now, there was no known method to maintain aggregates with these decay functions with polylogarithmic storage.

1.3 Our contributions

We now summarize the main contributions of our work.

- We formulate the problem of maintaining time-decayed aggregates for general decay functions and identify desired properties of the family of available decay functions. We argue that these properties are not satisfied by decay functions with known storage-efficient algorithms and identify new classes of decay functions that fulfill these properties.

- We show that *any* decay function can be approximately tracked using $O(\log^2(N))$ storage bits; this is the first polylogarithmic-storage approximation algorithm known for general decay functions. Our algorithm builds on the Exponential Histogram technique [5]. Interestingly, this result shows that, in a sense, sliding windows, for which a matching lower bound of $\Omega(\log^2 N)$ is known [5], are the "hardest" decay func-

[1] This bound holds for 0/1 streams, but extends in a simple way to streams where items have polynomial-size (in elapsed time). This assumption also holds for bounds that are stated in the sequel.

[2] In the sequel we define N to be the minimum of elapsed time and the minimum value after which the decay function nullifies. Exponential decay and window size are compared under this unifying metric.

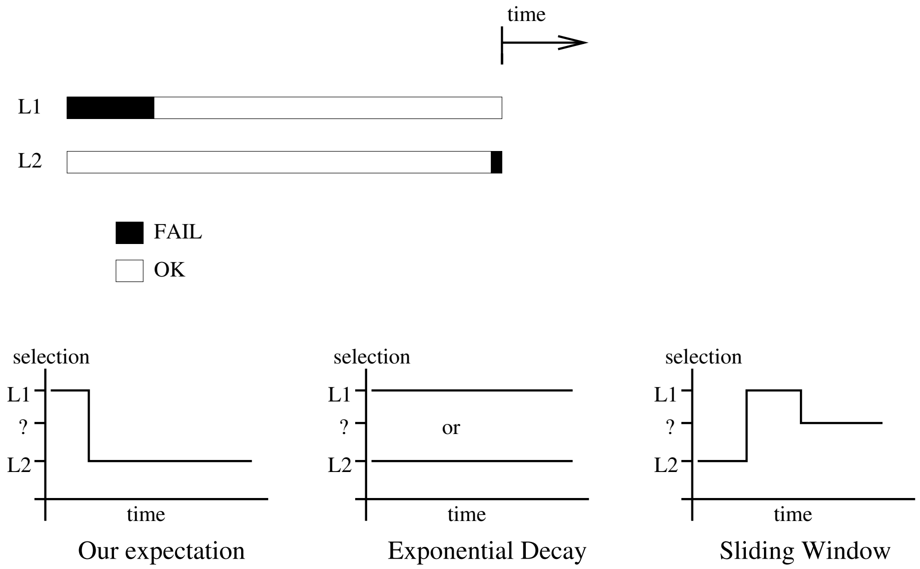


Figure 1: Relative reliability rating for the links L1 and L2. The figure shows the more reliable link as a function of time when using different decay functions.

tions to maintain.

- We give a different algorithm for maintaining time-decayed aggregates for decay functions with certain properties. The algorithm maintains polynomial decay (approximately) with storage of $O(\log(N) \log\log(N))$ bits. This result shows that polynomial decay can be tracked nearly as efficiently as Exponential decay.
- We give a nearly-matching lower bound of $\Omega(\log(N))$ storage bits for maintaining polynomial decay.
- The single-counter tracking of Exponential decay is appealing also because there are no data structures required and because it consumes minimum update time. We generalize the single-counter Exponential decay technique to a large class of decay functions, while keeping the space cost at $O(1)$ counters and time cost at $O(1)$ simple word operations per update.

The paper is structured as follows. We start in Section 2 with preliminaries and then, in Section 3, discuss some specific decay functions. In Section 4 we show how to use cascaded computation with an Exponential Histogram to approximate general decay functions. In Section 5 we develop a different algorithm, weight-based merging histogram (WBMH) and show that it performs better than cascaded Exponential Histograms on some families of decay functions that include polynomial decay. In Section 6 we give the lower bound on the storage requirement of any algorithm that approximates polynomially decaying aggregates. In Section 7 we show how to extend the single counter Exponential decay tracking algorithm to a wider class of decay functions.

2. PRELIMINARIES

Consider a stream of data items where $f(t) \geq 0$ is the value of the item obtained at time t. A *decay function* is a non-increasing $g(x) \geq 0$ defined for $x \geq 0$. We use T to denote current time. At time T, the *weight* of the item obtained at time t is $g(T-t)$ and the *decayed value* of the item is $f(t)g(T-t)$. Below we define the two main decayed aggregates we consider: the decayed-sum and the decayed-average of the stream $f(t)$ under the decay function $g()$.

2.1 Decayed Sum

PROBLEM 2.1. Decayed Sum Problem *(DSP). The input to the problem is the stream $f(t)$. The goal is to estimate, at any current time T, the decayed sum*

$$V_g(T) = \sum_{t<T} f(t)g(T-t) \ .$$

For simplicity of presentation, in the analysis we will assume that t is an integer. We also assume that values $f(t)$ are in $\{0,1\}$, as the results can be easily generalized to polynomial values (as in [5]). We refer to the DSP problem with binary $f(t)$ values as the *Decayed Count Problem* (DCP).

As with traditional streaming applications, our goal is to obtain an estimate of $V_g(T)$ at any time T while using a small amount of memory. We are interested in $(1 \pm \epsilon)$-approximate estimates, where $\epsilon > 0$ is fixed. That is, in obtaining $V'_g(T)$ such that $\frac{V'_g(T)-V_g(T)}{V_g(T)} \leq \epsilon$.

2.2 Decayed-Average

The decayed average of a stream is a time-decayed weighted average of observed values.

PROBLEM 2.2. Decayed Average Problem *(DAP). The input is a stream of data items, where $f(t_i)$ is the value of the item observed at time t_i. The goal is to estimate at any time T, the decayed average of the stream, which is defined as follows*

$$A_g(T) = \frac{\sum_{i|t_i<T} f(t_i)g(T-t_i)}{\sum_{i|t_i<T} g(T-t_i)} \ .$$

Each item in the decayed average is weighted according to the decay function. The denominator is a normalization factor that is the sum of the weights. The Decayed-Average is especially useful if $\sum_t g(t) = \infty$, in which case the Decayed-Count may also go to infinity.

Observe that the numerator of the decayed average is a decayed sum of the stream $f(t)$ (with $f(t) \equiv 0$ for $t \notin \{t_1, t_2, \dots\}$. The denominator of the decayed average is a decayed count of the stream $\hat{f}(t) = 1$ iff $t \in \{t_1, t_2, \dots\}$. It follows that an approximate weighted average can be obtained from the corresponding approximate decayed sum and decayed count. Therefore in the sequel, we focus on algorithms for the more basic decayed count problem.

2.3 Histograms and Metrics

The main performance metric we use is the amount of storage used by the algorithm, as a function of the minimum N of the elapsed time and the maximum value for which the decay function is positive ($\arg\max_x g(x) > 0$).

The algorithms we discuss aggregate the items into a *histogram*. Each bucket of the histogram aggregates all items that were observed in some time interval. Thus, each bucket of the histogram has a start-time and an end-time, and we refer to the difference between the start and end times as the *time-width* of the bucket and to the sum of values of all items in the bucket as the *count-width* of the bucket. Buckets are merged and discarded in a particular way which depends on the algorithm. When two consecutive buckets are merged, the new bucket inherits the start-time of the earlier bucket, the end-time of the later bucket, and count-width which is the sum of the count-widths of the two buckets.

Different algorithms may use time- or count-based criteria to place the boundaries of the buckets. The Exponential Histogram algorithm is such that the count boundaries of the buckets are independent of the stream, but the time boundaries vary. We will present another algorithm, weight-based merging histograms (WBMH), where the time boundaries are independent of the particular stream but the count boundaries vary.

Boundaries that are independent of the stream are useful for reducing the amount of storage needed for the histogram, since the boundary values do not need to be stored for each stream. This distinction between time and count, and which boundaries are stream-independent, turns out to be important. The total storage we require is determined by the number of buckets we use and the storage that is needed for each bucket. The WBMH algorithms we explore can maintain an approximate count instead of the exact count-width of each bucket. In contrast, approximate tracking of time-boundaries seems to be problematic, as a small deviation in a boundary can lead to a large deviation in the value of the decayed sum.

3. DECAY FUNCTIONS

We describe some families of decay functions that are of particular interest.

3.1 Exponential decay $\textsc{ExpD}_\lambda$

For a parameter $\lambda > 0$, $g(x) = \exp(-\lambda x)$. With ExpD, the relative significance of each measurement decreases exponentially with elapsed time. ExpD is commonly used in practice. One of the appeals of ExpD is that it can be maintained easily using

$$V_{\textsc{ExpD}}(t) = f(t) + \exp(-\lambda)V_{\textsc{ExpD}}(t-1) \ . \qquad (1)$$

We now explore the storage requirements of ExpD.

LEMMA 3.1.

- *Exact tracking of* $V_{\textsc{ExpD}}$ *requires* $\Omega(N)$ *storage bits.*
- *Approximate tracking of* $V_{\textsc{ExpD}}$ *requires* $\Theta(\log N)$ *storage bits.*

PROOF. We start with exact tracking. We show that a binary stream of length N (where an item with value either 0 or 1 can be observed every time unit) can have exponentially many distinct values of the decay function. Let $k = \lceil \lambda^{-1} \rceil$ be a constant. Consider now only binary streams with either a 0 or a 1 at times t with $t \bmod k = 0$ (and 0 value or no items if $t \bmod k \neq 0$). Consider now the value of the decayed count N time units after the start of the stream. There are at least $2^{(\lceil N/k \rceil)}$ such streams and each stream corresponds to a unique value of the decayed count. Thus, our algorithm must be able to differentiate all these streams, and thus use at least $\lceil N/k \rceil$ storage bits.

For the lower bound on approximate tracking, consider all streams that consist of a single item of value "1" during elapsed time of N time units. In order to obtain a factor 2 approximation, our algorithm must be able to differentiate the timestamp of the "1" item to within k time units. Thus, it requires at least $\log\lfloor N/2k \rfloor$ bits.

For the upper bound note that it is sufficient to track the time stamps of the C most recent 0/1-valued items,[3] where $C = \lceil -\lambda^{-1} \ln((1 - \exp(-\lambda))\epsilon) \rceil$ is a constant that depends on λ and ϵ. Alternatively, we can use the classic algorithm (1) over floating point arithmetic with a logarithmic number of bits. □

3.2 Sliding Window $\textsc{SliWin}_W$.

Sliding windows are parameterized by the window size W. The corresponding decay function has $g(x) = 1$ for $x \leq W$ and $g(x) = 0$ otherwise. Sliding windows were introduced in [5], which also presented an algorithm that obtains a $(1+\epsilon)$ approximation for any constant ϵ using $O(\epsilon^{-1} \log^2 W)$ bits. With SliWin decay, all data values in a recent window of size W have equal importance and all older data values have 0 weight.

3.3 Polynomial decay $\textsc{PolyD}_\alpha$

For a parameter α, put $g(x) = 1/x^\alpha$. In the introduction we argued that PolyD is often a better choice for a decay function than ExpD. As with ExpD, exactly maintaining decayed count with PolyD seem to require linear storage. In the sequel we present efficient algorithms for approximately maintaining PolyD counts.

3.4 Polyexponential Decay

The next families of decay functions are of special practical interest, since, as we shall see in Section 7, they can be tracked using a small number of single-register Exponential Decay calculations, while allowing for a more moderate rate of decay on recent values.

Polyexp decay with parameter k is decay by the function $g(x) = x^k e^{-x}/k!$. As we'll see below, polyexp decay can

[3]For general (non binary) streams, we record the C "most recent" values, but instead of using actual time stamps we treat an item of values v recieved at time t as an item of value 1 received at time $t + \lambda^{-1} \ln v$ (note that the contribution to the decayed sum is the same).

be computed exactly in space $k+1$ by performing $k+1$ instances of Exponential decay. For $k=2$ ($k=3$), this is known as Brown's double (triple) exponential smoothing [8], from around 1960, and is still in use to model the data by a line or quadratic. Below we will use Polyexp decay, in a new way, as a building block to approximate Chordal decay.

3.5 Chordal and Polygonal Decay

We call decay by the function $\chi_W(x) = 1 - x/W$, for $0 \leq x \leq W$ and 0 otherwise, *chordal decay.* (See Figure 2.) More generally, Polygonal decay is decay by a decreasing polygonal function, which can be regarded as a sum of chords.

4. DOMINATION-BASED AGGREGATION

We quickly review Exponential Histograms and then show how cascaded computation can be used to extend them to general decay functions. We define N as the minimum of elapsed time since the arrival of the first data value and $N(g()) = \max\{x | g(x) > 0\}$ and consider the dependence of the required number of storage bits on N.

4.1 Exponential Histograms

Exponential Histograms (EHs) were introduced by Datar et al [5] as a method to estimate SLIWIN decayed counts; Datar et al provided tight bounds where $\log^2(N)$ bits are sufficient and necessary for $(1+\epsilon)$ approximate estimates.[4]

EH maintain buckets over ranges of data points. All data seen in a time range $(w_{i-1}, w_i]$ are aggregated into the ith bucket, and the EH maintains the values w_i and the count

$$C_i = \sum_{w_{i-1} < t \leq w_i} f(t)$$

for each bucket. Buckets where $w_i < T - W$ are discarded. When a new data point arrives, it is placed in its own new bucket. Buckets are then merged in a certain way such that there is always a logarithmic $O(\log N)$ number of buckets. At any given point T, the estimate on the decayed count $V_{\text{SLIWIN}_N}(T)$ can be obtained from

$$V'_{\text{SLIWIN}_N}(T) = \sum_i C_i \ .$$

A characterization of the merging process of EH is that two consecutive buckets are merged if the combined count of the merged buckets is dominated by the total count of all more-recent buckets. We refer to this aggregation process as *domination-based*. (Note that the factor by which the preceding buckets need to dominate depends on the desired approximation factor.) Furthermore, the sequence of bucket counts is a non-decreasing sequence of powers of 2, and, for some $k = \Theta(1/\epsilon)$ and some P, for each possible count $2^p < 2^P$, there are exactly k or $k+1$ buckets having count 2^p, there are at most $k+1$ buckets having count 2^P, and no buckets have counts greater than 2^P.

We will next argue that EHs can be used to obtain a decayed count for general decay functions. To do so we use the following immediate property of the EH data structure.

[4]Note that for sliding windows, N is the minimum of elapsed time since the arrival of the first data value and the window size W.

LEMMA 4.1. *EH for window size N can be used to estimate* SLIWIN *DCs for all window sizes smaller than N ($V_{\text{SLIWIN}_i}(T)$ for all $i \leq N$).*

PROOF. Let the points

$$T - N \leq w_0 < w_1 < w_2 \cdots < w_k < T$$

be the end times of buckets in the EH. Let j be the minimum such that $w_j \geq T - i$. Observe that the buckets $j, \ldots, k$ are the buckets of an EH for SLIWIN_i, since the domination criterion for merging The estimate for $V_{\text{SLIWIN}_{N-i}}(T)$ produced by the EH is thus

$$V'_{\text{SLIWIN}_i}(T) = \sum_{\ell=j}^{k} C_\ell \ . \qquad (2)$$

□

4.2 Cascaded Exponential Histograms (CEH)

We now consider general decay function.

THEOREM 1. *DCP with any decay function can be estimated using an Exponential Histogram with window size N.*

PROOF. Using summation by parts, we can rewrite the DC of g as

$$\begin{aligned}
V_g(T) &= \sum_{T-N \leq t < T} f(t) g(T-t) \\
&= g(N) \sum_{T-N \leq t < T} f(t) + \\
&\quad \sum_{i=1}^{N-1} (g(N-i) - g(N+1-i)) \sum_{T-N+i \leq t < T} f(t) \\
&= g(N) V_{\text{SLIWIN}_N}(T) + \qquad (3) \\
&\quad \sum_{i=1}^{N-1} (g(N-i) - g(N+1-i)) V_{\text{SLIWIN}_{N-i}}(T)
\end{aligned}$$

Equation (3) represents the DC $V_g(T)$ as a positively weighted sum of SLIWIN DCs. Since a $(1+\epsilon)$-accurate estimate of each DC $V_{\text{SLIWIN}_{N-i}}(T)$ can be obtained from the EH, we obtain a $(1+\epsilon)$-estimate for the DC $V_g(T)$. □

We now discuss the computation involved in maintaining the approximate count $V'_g(T)$ given the EH. Straightforward use of (3) suggests an $O(N)$ time computation in each time step. It is not hard to see, however, that, using the partition of the EH into buckets, we can considerably reduce the computation: To do so, we first substitute the approximations supplied by Equation (2) for the DCs $V_{\text{SLIWIN}_{N-i}}(T)$ in Expression (3). We obtain that the estimated DC is equal to

$$V'_g(T) = g(T-w_0) C_0 + \sum_{j \geq 1} (g(T-w_j) - g(T-w_{j-1})) C_j \ . \qquad (4)$$

Direct re-computation of Equation (4) in each time T requires $O(k) = O(\log N)$ steps. It is not hard to show, however, (using similar arguments as in [5]) that the amortized update time can be brought down to a constant. The total memory (words) required for this estimate is $\log(N)$.

For example, suppose consecutive weights in g are $8, 5, 3, 2$. Then, at time $T = 4$, the decayed count is $8f(3) + 5f(2) + 3f(1) + 2f(0)$. This can be rewritten as $2[f(0) + f(1) + f(2) + f(3)] + (3-2)[f(1) + f(2) + f(3)] + (5-3)[f(2) + f(3)] + (8-5)[f(3)]$. Each expression in square brackets is a sliding window decayed count of f; all of these can be approximated with good relative error by a single instance of the EH method. Many of these approximations will be the same; we suppose that $f(1) + f(2) + f(3)$ and $f(2) + f(3)$ are each approximated by $f(2) + f(3)$. Then the decayed count is approximated by $2[f(0) + f(1) + f(2) + f(3)] + (5-2)[f(2) + f(3)] + (8-5)[f(3)]$.

It is shown in [5] that EHs are essentially optimal for SliWin, that is, that a logarithmic number of counters is indeed necessary. The result above thus shows that SliWin decay is in the sense the "hardest" decay function, as any other decay function can be estimated using the EH data structure.

In the sequel, we refer to cascaded use of an EH applied to general decay functions as CEH (Cascaded Exponential Histogram).

5. WEIGHT-BASED AGGREGATION

We develop another class of histograms, which we refer to as **Weight-Based Merging** histograms (WBMH). WBMH histograms also aggregate values into buckets but the bucket boundaries are determined by the decay function and the current time and are independent of the particular input stream. This feature is important for giving storage-efficient algorithms, since the boundaries do not need to be explicitly stored per stream.

Each WBMH bucket is such that if t_s, t_f are the start and end time of the bucket, and T is the time the bucket is formed, we have that $g(T+x-t_f) \leq (1+\epsilon)g(T+x-t_s)$ for all $x \geq 0$. What this means is that all items aggregated in the bucket are assigned similar weight by the decay function at the time the bucket is formed and at any later time.

WBMH are applicable to decay functions with the following property: for any fixed time window Δ, the ratio $g(x)/g(x+\Delta)$ is non-increasing with x. (Note that it suffices to require this for $\Delta = 1$.) Stated in words, this property implies that the ratio of the "weights" of two items remains fixed or becomes closer to 1 as time progresses. As argued in the introduction, this is a natural intuitive requirement. This property is possessed by many families of decay functions including Exponential and Polynomial decay. Note that, for Exponential decay, $g(x)/g(x + \Delta)$ is constant. So, intuitively, WBMH are applicable to decay functions that drop off like exponentials or more slowly.

Let $D(g()) = g(1)/g(N)$ be the weight ratio between the initial weight of an item and the weight of the oldest item for which the decay function assigns positive weight. We now outline a deterministic process that determines bucket boundaries as time progresses. Let b_1 be the maximum such that $(1 + \epsilon)g(b_1 - 1) \geq g(1)$. Similarly, for $i > 1$, let b_i be maximum such that $(1 + \epsilon)g(b_i - 1) \geq g(b_{i-1})$. The "current" bucket is sealed and a new bucket is started at times T such that $T \bmod b_1 = 0$. Whenever there is an i and two consecutive buckets $[a_s, a_t]$ and $[c_s = a_t+1, c_t]$ such that $b_i \leq T - c_t$ and $T - a_s \leq b_{i+1} - 1$, the two buckets are merged into a single bucket $[a_s, c_t]$.

Observe that the process above is independent of the stream (the count in each bucket depends on the stream, but the boundaries of each bucket do not). We refer to a range $[b_i, b_{i+1} - 1]$ as a *region*. Observe that the total number of regions is $\lceil \log_{1+\epsilon} D(g) \rceil$, and there can be at most two buckets per region, on average (1/2 bucket shared with each neighboring region and 1 bucket contained in the region), so the total number of buckets is $O(\epsilon^{-1} \log D(g))$. If the decay function has the property that $g(x)/g(x + 1)$ is non-increasing with x, we can further argue that the ratio of weights of two items aggregated into the same bucket is at most $1 + \epsilon$.

For example, consider decay $g(x) = 1/x^2$, and suppose $(1 + \epsilon) = 5$. Then the regions are determined by $b_1 = 3$, $b_2 = 7$, $b_3 = 16$, and so on. The weights associated with "spots" in the region are thus

$$\left(1, \frac{1}{4}\right); \left(\frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\right); \left(\frac{1}{49}, \ldots, \frac{1}{225}\right); \ldots$$

(the weights within each pair of parentheses corresponds to a region). This is correct because the weights in each region span a range at most $5 = (1+\epsilon)$. Consider now the formation of buckets starting at time $T = 1$. Initially at $T = 1$ there is a single bucket formed with corresponding weights (1). At $T = 2$ the new item is appended to the current bucket and it is sealed as $(1, \frac{1}{4})$. At $T = 3$ we have, $(1); (\frac{1}{4}, \frac{1}{9})$, at $T = 4$ we have $(1, \frac{1}{4}); (\frac{1}{9}, \frac{1}{16})$, at $T = 6$ we have

$$\left(1, \frac{1}{4}\right); \left(\frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\right),$$

at $T = 8$ we have

$$\left(1, \frac{1}{4}\right); \left(\frac{1}{9}, \frac{1}{16}\right); \left(\frac{1}{25}, \frac{1}{36}, \frac{1}{49}, \frac{1}{64}\right),$$

and at $T = 10$

$$\left(1, \frac{1}{4}\right); \left(\frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\right); \left(\frac{1}{49}, \frac{1}{64}, \frac{1}{81}, \frac{1}{100}\right).$$

Note that the bucket of most recent items always alternates between time-width 1 and time-width 2.

The information stored per stream is the total count of items that arrived within the time boundaries of each bucket. Since each such number is at most N, each number requires $\log(N)$ bits to store exactly, for a total of $O(\log^2(N))$ bits over the $O(\log(N))$ buckets. But the counts need only be stored approximately. In floating point, the exponent requires $\log\log(N)$ bits. We will also store the most significant $\log(1/\beta) = \log(1/\epsilon) + \log\log(N)$ bits of the expansion, where $\beta = \epsilon/\log(N)$. Note that the count in each bucket is obtained by summing exact counts of 1 or 0 in a summation tree of depth $d \leq O(\log(N))$. Recursively assume that, at level i of the summation tree, the values are good to the factor $(1+\beta)^i$. We can add two such counts, getting a sum also good to the factor $(1 + \beta)^i$. We next round the sum to the most significant $\log(\beta)$ bits, which is equivalent to multiplying by a number between 1 and $(1 + \beta)$. Thus the rounded sum is good to the factor $(1 + \beta)^{i+1}$, as desired. Thus, at the top of the tree, the rounded sum is good to $(1 + \beta)^{\log(N)} \approx (1 + \beta\log(N)) = (1 + \epsilon)$, by definition of β.

The foregoing assumes that we know N in advance. More generally, at level i, rounding is equivalent to multiplying by a number between 1 and $1 + \beta_i$, where the β_i's are not necessarily all equal. We want $\prod_i (1 + \beta_i) \approx \left(1 + \sum_{i=1}^{\log(N)} \beta_i\right)$

to be at most $(1+\epsilon)$, or $\sum_{i=1}^{\log(N)} \beta_i < \epsilon$. For this, it suffices to put $\beta_i \approx \epsilon/i^2$, so that the number of storage bits at level i is $\log(1/\beta_i) = \log(1/\epsilon) + 2\log(i)$, for $i \leq \log(N)$. This way N does not need to be known in advance.

We have established the following:

LEMMA 5.1. *Let g be a decay function such that $g(x)/g(x+1)$ is non-increasing with x. The WBMH algorithm uses $O(\log\log(N)\log D(g))$ storage bits and approximates the decayed count to within $1+\epsilon$.*

For which decay functions does the WBMH algorithms outperform CEHs ? It is not hard to see that for EXPD, $\log D(g) = O(N)$, and thus WBMH requires a linear number of buckets and CEH is more efficient. For POLYD, $\log D(g) = O(\log N)$, WBMH uses only a logarithmic number of buckets, and requires considerably fewer bits per bucket than CEH. In total, it uses $O(\log N \log\log N)$ storage bits versus $O(\log^2 N)$ bits used by CEH; thus, there is a quadratic gap. More generally, WBMH beats CEHs also for sub-polynomial decay, as the number of buckets of WBMH is sub-logarithmic in elapsed time; WBMH also beats CEH for slightly super-polynomial decays.

6. LOWER BOUND FOR POLYNOMIALLY-DECAYING COUNTS

We now establish that a logarithmic number of bits are necessary for maintaining polynomially-decaying counts.

THEOREM 2. *A logarithmic number of bits (in elapsed time) is necessary for approximately maintaining decay by $g(x) = 1/x^\alpha$.*

PROOF. We first give the proof for decayed sums (of data that can take non-negative integer values other than 0's and 1's). Later, we extend the proof to 0/1-valued data.

Consider the decay function $g(x) = 1/x^\alpha$. Let k be a constant such that $\frac{2}{k}\cdot\frac{k+1}{k-1} < \frac{1}{4}$; *e.g.*, $k = 10$. We will consider a time interval of length N, from $-N/2$ to $N/2$. Let $r = \lfloor \frac{\alpha}{2\log k}\log(N/2)\rfloor$.

Consider the following family of streams: for $i \leq r$, at time $t = -k^{2i/\alpha}$, we receive a burst of count $C_i \in \{k^i, 2k^i\}$. The streams differ by the size of the burst and thus there are 2^r different streams in our family. Note that the size of the bursts decreases with time (increases with i) and no more data arrives after time $t = -1$.

We now consider the time interval $[0, N/2]$. During this time no more data arrives, but we need to "remember" enough about the stream we had seen so that we can estimate the value of the decayed count, to within some constant accuracy $\epsilon < 1/4$, at all times $t \in [0, N/2]$.

Let $n_i = C_i/k^i \in \{1, 2\}$. A particular stream in the family is determined by the values n_i $(i = 1, \dots, r)$. We will show that we will need to know all the values n_i in order to estimate the decayed count correctly. Thus, at time 0, we will need a storage of at least r bits.

Consider the decayed count value $V_g(t)$ at time $t_i = k^{2i/\alpha}$. We have

$$V_g(t_i) = \sum_{j=1}^{r} g(k^{2i/\alpha} + k^{2j/\alpha})k^j n_j \ .$$

We now compute upper bounds on the contribution of the prefix and the suffix of this sum. For the prefix of the sum up to term $i-1$ we obtain

$$\begin{aligned}\sum_{j=1}^{i-1} g(k^{2i/\alpha} + k^{2j/\alpha})k^j n_j &\leq 2\sum_{j=1}^{i-1} g(2k^{2i/\alpha})k^j \\ &= \frac{2^{1-\alpha}}{k^{2i}}\sum_{j=1}^{i-1} k^j \\ &\leq \frac{2^{1-\alpha}}{k^{i+1}}\frac{k}{k-1} \qquad (5)\end{aligned}$$

For the suffix of the sum, starting at term $i+1$, we obtain

$$\begin{aligned}\sum_{j=i+1}^{r} g(k^{2i/\alpha} + k^{2j/\alpha})k^j n_j &\leq 2\sum_{j=i+1}^{r} g(2k^{2j/\alpha})k^j \\ &\leq 2^{1-\alpha}\sum_{j=i+1}^{\infty} 1/k^j \\ &\leq \frac{2^{1-\alpha}}{k^{i+1}}\frac{1}{k-1} \qquad (6)\end{aligned}$$

By combining the bounds (5) and (6) we obtain that

$$\begin{aligned}\sum_{j\neq i} g(k^{2i/\alpha} + k^{2j/\alpha})k^j n_j &\leq 2^{1-\alpha}k^{-(i+1)}\frac{k+1}{k-1} \\ &\leq \frac{2}{k}\cdot\frac{k+1}{k-1}2^{-\alpha}k^{-i}.\end{aligned}$$

Observe now that the ith term is equal to

$$g(2k^{2i/\alpha})k^i n_i = 2^{-\alpha}k^{-i}n_i \ .$$

Since $n_i \geq 1$ and $\frac{2}{k}\cdot\frac{k+1}{k-1} < \frac{1}{4}$, we obtain that the contribution of the prefix and suffix is at most the fraction 1/4 of the contribution of the ith term. It follows that (independently of the rest of the stream) we must know the value of n_i in order to estimate the decayed count with $\epsilon < 1/4$ at time t_i.

Now suppose we require that the data takes values 0 or 1 and arrives at equally-spaced time steps. Note that the above proof goes through, essentially unchanged, if $\alpha < 2$, since a single burst of count $C_i = \Theta(k^i)$ at time $-k^{2i/\alpha}$ can be spread out to C_i 1's. At time $k^{2i/\alpha}$, the burst will have occured $2k^{2i/\alpha}$ to $2k^{2i/\alpha} + k^i$ time units in the past, and so all 1's in the burst will have approximately the same weight. If $\alpha \geq 2$, however, then the proof needs modification, since there isn't time to read k^i 1's from time $-k^{2i/\alpha}$ to 0 (we actually need to absorb the 1's between consecutive bursts). Instead, we proceed as follows. Instead of using bursts of size k^i, use bursts of size $k^i/N^{1/\alpha}$, and only use some of the r slots—the ones for which the size of the burst is at least 1 and at most $\frac{2\epsilon}{\alpha}k^{2i/\alpha}$. First note that, if we divide all the bursts by the same number (still concentrating each burst at a single time instant), the relative contributions of the prefix, suffix, and i'th terms remains the same. Next note that, by spreading out a burst as several 1's into the past, all contributions decrease, compared with a burst occuring at a single time instant. A decrease in contributions from prefixes and suffixes can only help. If the i'th burst has size at most $\frac{2\epsilon}{\alpha}k^{2i/\alpha}$, then, at time $k^{2i/\alpha}$, the weight of data in this burst varies from $g(2k^{2i/\alpha})$ to $g(2k^{2i/\alpha} + \frac{2\epsilon}{\alpha}k^{2i/\alpha})$; *i.e.*, the weight can decrease by the factor $1/(1+\epsilon/\alpha)^\alpha$, which, assuming $\alpha \geq 2$, is approximately $e^{-\epsilon} \approx (1-\epsilon)$, a tolerable amount. Next, we show that there are $\Omega(\log(N))$ such slots. The requirement $k^i/N^{1/\alpha} \geq 1$ is equivalent to $i \geq \log_k(N)/\alpha$. The requirement $k^i/N^{1/\alpha} \le \frac{2\epsilon}{\alpha} k^{2i/\alpha}$ is equivalent to

$$i \le \frac{\log_k(N)}{\alpha - 2} + \frac{\alpha \log(2\epsilon/\alpha)}{\alpha - 2},$$

i.e., $i \le \frac{\log_k(N)}{\alpha-2} - \Theta(1)$, when $\alpha > 2$, and vacuous otherwise. Thus the number of allowable i's is $(1/(\alpha-2) - 1/\alpha)\log_k(N) - \Theta(1) = \Theta(\log(N))$, as desired. Finally, for this to work, we also need that the i'th and $(i+1)$'st bursts are disjoint. That is, we need $\frac{2\epsilon}{\alpha} k^{2i/\alpha} < k^{2(i+1)/\alpha} - k^{2i/\alpha} = (k^{2/\alpha} - 1)k^{2i/\alpha}$. Choose k large enough that $(k^{2/\alpha} - 1) \ge 1$. Then, since $\alpha \ge 2$ and $\epsilon = 1/4$, $\frac{2\epsilon}{\alpha} k^{2i/\alpha} \le k^{2i/\alpha} \le (k^{2/\alpha} - 1)k^{2i/\alpha}$, as desired. Thus, for $\epsilon = 1/4$, given arbitrary α, $\Omega(\log(N))$ bits of storage (depending on α) is required to track $1/x^\alpha$ decay of 0/1-valued data to within the factor $(1+\epsilon)$. □

7. ALGORITHMS FOR POLYEXPONENTIAL, CHORDAL, AND POLYGONAL DECAY

In this section, we generalize Exponential decay, while keeping some of its attractive properties—space $O(1)$ words and time $O(1)$ simple word operations.

The generalization is in two steps. First, we generalize the class of exactly trackable decay functions from exponentials $e^{-\lambda x}$ to polyexponential $x^k e^{-\lambda x}/k!$ (with cost $O(k)$). Any sum of these decay functions can also be tracked exactly. We next show that chordal decay functions χ_W (and, more generally, polygonal decay) can be approximated uniformly by the sum of few polyexponential functions. That is, the sum S of a few polyexp functions, for all x, satisfies $|S(x) - \chi_W(x)| \le \epsilon$.

The guarantee implied here for chordal decayed *values* is strictly weaker than relative error of factor $(1+\epsilon)$ and strictly stronger than additive error of ϵW. For example, consider a single data item of age $W-2$, which should have weight $2/W$. A relative error result would be $(1\pm\epsilon)\cdot(2/W)$, additive error would be $2/W \pm \epsilon W$, and our guarantee would be $2/W \pm \epsilon$. Below we present a situation in which our ultimate result is as good as relative error, and better than additive error.

Our result is useful in several ways, as we discuss next. First, in some applications, using exactly the right shape of the decay function is not as important as using little space. Previously, such applications were stuck with using Exponential decay; we provide a somewhat wider variety of decay functions to use while keeping the memory under control. Indeed, space of around 4 words gives a decay function much closer to chordal decay than to Exponential decay. (See Figure 2.)

If the data is 0/1 valued, then values decayed by g can approximated with additive error approximately ϵW by our techniques, using space approximately $1/\epsilon$. (The ϵ of this section corresponds to ϵ^2 in Section 7.2.) In some applications, a decayed value is compared against a threshold, and our approximation guarantee is good enough for that. For example, consider chordal decay $\chi_W(x)$, equal to $(1 - x/W)$ for $0 \le x \le W$ and zero for $x > W$. Suppose an application has 0/1-valued data and wants to know whether the chordally-decayed value exceeds a threshold, τW, where $0 \le \tau \le 1/2$. For example, a decayed value of $W/2$ corresponds to saturation by all 1 data and a decayed value of $W/4$ (*i.e.*, $\tau = 1/4$) indicates data density half the saturation value. The approximate version of this problem, using a relative error approximation, returns true if the decayed value exceeds $(1+\epsilon)\tau W$, false if the decayed value is less than τW, and gives no guarantee, otherwise. Our technique will give a decayed value accurate to within $\epsilon\tau W$, additively, in space approximately $1/(\epsilon\tau)$. Thus our technique gives the same ultimate guarantee as one gets from relative-error approximation, but using word space independent of N. Also, we show that the exact chordal threshold problem for, say, $\tau = 1/8$, requires space $\Omega(W)$, so the exact threshold problem is not any easier than the problem of returning the exact decayed value.

In fact, our guarantee is strictly stronger than additive error in the sense that it adapts to unknown data density. Here we give just an example; we defer a formal statement and proof to the final version of this paper. Consider chordal decay with parameter W. Suppose that, for some random 0/1 data source, between one and three 1s arrive in any period of $W^{1/3}$ steps. Observe that the decayed count remains between about $\frac{1}{2}\cdot 1 \cdot W^{2/3}$ and $\frac{1}{2}\cdot 3\cdot W^{2/3}$. Additive error of ϵW would overwhelm these values. Our technique, however, would give relative error $(1+\epsilon')$, using $\epsilon < \epsilon'/6$. This is because our technique only contributes error for non-zero data items. In this example, the error of our technique over data in each period of length $W^{1/3}$ is at most 3ϵ, for a total of $3\epsilon W^{2/3} < \epsilon' W^{2/3}/2$, while the decayed count is at least $W^{2/3}/2$. Note that the baseline density $W^{-1/3}$ can be replaced by any other density $\gg 1/W$; the cost depends on the ratio $3:1$ in which the density varies. Our technique will be better than additive error provided the density is much less than 1; our technique will be better in terms of space than the exact algorithm of simply storing all data more recent than W if the density is much greater than $1/W$.

7.1 Polyexp Decay

Polyexp decay with parameters k and θ is decay by the function $g(x) = x^k e^{-\theta x}/k!$. Henceforth we fix $\theta = 1$ to focus on other aspects.

THEOREM 3. *Polyexp decay can be computed exactly in space $k+1$ (by peforming $k+1$ pipelined instances of Exponential decay).*

PROOF SKETCH. We will show how to track decay by $\exp(-x)$, $x\exp(-x)$, $x^2\exp(-x)/2!$, ..., $x^k\exp(-x)/k!$ simultaneously in space $k+1$. Note that the sum of the count of f decayed by g_1 and g_2 equals, by linearity, the count of f decayed by $(g_1 + g_2)$. Thus tracking decay by $\exp(-x)$, $x\exp(-x)$, $x^2\exp(-x)/2!$, ..., $x^k\exp(-x)/k!$ is equivalent to tracking decay by $p_0\exp$, $p_1\exp$, ..., $p_k\exp$, where each p_j is a polynomial of degree j (with non-zero leading coefficient), since the polynomials $p_0, p1, \ldots, p_k$ are linear combinations of $1, x, \ldots, x^k$, and vice versa. (For example, $\sum_t f(t) 3e^{-(T-t)}$ and $\sum_t f(t)(2(T-t)+5)e^{-(T-t)}$ can be recovered from $\sum_t f(t)e^{-(T-t)}$ and $\sum_t f(t)(T-t)e^{-(T-t)}$, and vice versa.) Next, note that decaying a function, f, by another function, g, is a convolution $f \star g$, and convolution is associative. Thus k-times pipelined exponential decay, $((\cdots(f\star\exp)\star\exp)\cdots\star\exp)$, can be rewritten as $f\star(\exp\star\exp\star\cdots\star\exp)$, where $\exp(x)$ equals e^{-x} for $x \ge 0$ and $\exp(x) = 0$, otherwise. It suffices to show that the k-times convolution product of exp is $p_k\exp$ for some polynomial p_k of degree k. By induction, we need to show that, if p_{k-1} is any polynomial of degree $k-1$, then $p_{k-1}\exp$ decayed by exp is $p_k \exp$ for some polynomial p_k of degree k. Finally,

$$\sum_{0\le t<T} p_{k-1}(t)e^{-t}e^{-(T-t)} = e^{-T}\sum_{0\le t<T} p_{k-1}(t) = e^{-T}p_k(T),$$

as desired. $\square$

One could also reduce Polyexp decay to Exponential decay by $xe^{-\theta x} = -\frac{d}{d\theta}e^{-\theta x} \approx -\frac{e^{-(\theta+\delta)x}-e^{-(\theta-\delta)x}}{2\delta}$, for $\delta \ll \theta$, and similarly for higher derivatives. That is, simply perform ordinary exponential decay for $k+1$ equally-spaced parameters. We can get all k derivatives at once from the same $k+1$ space, by noting, *e.g.*, that $e^{-\theta x} \approx \frac{e^{-(\theta+\delta)x}+e^{-(\theta-\delta)x}}{2}$.

In the final version of this paper, we will give a full study of the storage required in bits per word to insure accurate computations. It turns out that a number of bits polynomial in k and logarithmic in N suffices, and, in practice, k in the range 4–8 already gives nontrivial results.

Below, we will use polyexp decay as building blocks.

7.2 Chordal Decay

We call decay by the function $\chi_W(x) = 1 - x/W$, for $0 \le x \le W$ and 0 otherwise, *chordal decay.*

THEOREM 4. *Let $T_k(x) = (1+x+x^2/2+\cdots+x^{k-1}/(k-1)!)$ be the k-term Taylor approximation to e^x. For some $k = \epsilon^{-(2+o(1))}$ specified below, let $g(x) = (1-x/k)T_k(x)e^{-x}$. Then*

- *$|\chi_W(x) - g(kx/W)| < \epsilon$, uniformly for all x.*
- *$\int x^j g(kx/W)\,dx < \infty$ for all j (so $g(kx/W)$ can be normalized for total mass, spread, etc.).*
- *For 0/1-valued data, a value decayed by $\chi_W(x)$ differs from a value decayed by $g(kx/W)$ by at most $O(\epsilon^{2+o(1)}W)$, additively.*
- *Decay by $g(kx/W)$ can be computed in word space $k+1 \approx 1/\epsilon^2$, independent of N.*

PROOF SKETCH. We will show that $T_k(x)e^{-x}$ approximates $\text{SLIWIN}_k(x)$ for $x \not\approx k$. Then $g(x) = (1-x/k)T_k(x)e^{-x}$ approximates $(1-x/k)$ for $x \not\approx k$. For $x \approx k$, the factor $(1-x/k)$ is small and $T_k(x)e^{-x}$ is at most 1. It follows that we can approximate χ_W uniformly for any W, since stretching in the x direction does not affect the y direction.

Given ϵ, we need to find $k \approx 1/\epsilon^2$ so that

$$\left|(1-x/k)T_k(x)e^{-x} - \chi_k\right| < \epsilon.$$

We will consider separately x's in the ranges $[0, k/2]$, $[k/2, k-a\sqrt{k}]$, $[k-a\sqrt{k}, k+a\sqrt{k}]$, $[k+a\sqrt{k}, 3k/2]$, and $[3k/2, \infty]$, for small a to be determined below. The following analysis is elementary (similar to the derivation of Stirling's formula); we include it for completeness to justify $k \approx 1/\epsilon^2$.

First note that, on $[0, k/2]$, we have $|e^x - T_k(x)| < e^{k/2}x^k/k!$, using a common formula for the remainder of a Taylor series. Using Stirling's formula for the factorial, this is approximately $e^{k/2}x^k \cdot \sqrt{2\pi k}k^k/e^k < \sqrt{2\pi k}\left(\frac{e^{1/2}}{2}\right)^k$, which is less than ϵ provided k is sufficiently large compared with ϵ (other bounds on k below will be tighter). Since $e^{-x} \le 1$ on this interval, it follows that $\left|T_k(x)e^{-x} - \text{SLIWIN}_k\right| < \epsilon$, so that $\left|(1-x/k)T_k(x)e^{-x} - \chi_k\right| < \epsilon$ here.

Next, consider $[k/2, k-a\sqrt{k}]$. (Here we analyze T_{k+1} instead of T_k to make the expressions a bit nicer.) Note that $\left(T_{k+1}(x)e^{-x}\right)' = -\frac{x^k}{k!}e^{-x}$. Using Stirling's formula,

$$\left(T_{k+1}(x)e^{-x}\right)' = -\frac{e^k}{k^k\sqrt{2\pi k}}x^k e^{-x}\left(1+O\left(\frac{1}{k}\right)\right).$$

Put $x = k - y\sqrt{k} \ge k/2$; we get

$$\left(T_{k+1}(x)e^{-x}\right)' = -\frac{e^{y\sqrt{k}}}{\sqrt{2\pi k}}\left(1-\frac{y}{\sqrt{k}}\right)^k\left(1+O\left(\frac{1}{k}\right)\right).$$

Consider $e^{y\sqrt{k}}\left(1-\frac{y}{\sqrt{k}}\right)^k$. We have

$$\begin{aligned}
\log\left(e^{y\sqrt{k}}\left(1-\frac{y}{\sqrt{k}}\right)^k\right) &= \sqrt{k}(y+\sqrt{k}\log(1-y/\sqrt{k})) \\
&\le \sqrt{k}\left(y-\sqrt{k}\left(\frac{y}{\sqrt{k}}+\frac{y^2}{2k}\right)\right) \\
&= -y^2/2.
\end{aligned}$$

Thus, for $x = k - y\sqrt{k} \ge k/2$, we get $\left(T_{k+1}(x)e^{-x}\right)' \le \frac{e^{-y^2/2}}{\sqrt{k}}$ (since $\sqrt{2\pi} > 1$), which is less than ϵ/k provided $y > a \ge \sqrt{\log(\sqrt{k}/\epsilon)}$. Now $e^0T_{k+1}(0) = 1$, and $|\left(T_{k+1}(x)e^{-x}\right)'| \le O(\epsilon/k)$ for $x \le k - y\sqrt{k}$. Integrating from 0 to $k - a\sqrt{k}$, it follows that $e^{-x}T_{k+1}(x) \ge 1 - \epsilon$ for $x \le k - y\sqrt{k}$. Also, the derivative is always negative except at 0, so $e^{-x}T_{k+1}(x)$ is always decreasing; *i.e.*, $1 - O(\epsilon) \le e^{-x}T_{k+1}(x) \le 1$. It follows that $\left|(1-x/k)T_k(x)e^{-x} - \chi_k\right| < \epsilon$ on $[k/2, k-a\sqrt{k}]$.

On the region $[k-a\sqrt{k}, k+a\sqrt{k}]$, $|1-x/k|$ is bounded by $a/\sqrt{k}$ and $e^{-x}T_k(x) < 1$, so, provided $a/\sqrt{k} \le \epsilon$, we conclude that $|g(x)| \le \epsilon$ and $|\chi_k(x)| \le \epsilon$, whence $|g(x) - \chi_k(x)| \le O(\epsilon)$, uniformly.

Similar to the analysis above for $[k/2, k-a\sqrt{k}]$, one can show that $\frac{x^k e^{-x}}{k!} \le e^{-y^2/4}/\sqrt{k}$ for $x = k + y\sqrt{k}$, where $x \le 3k/2$ and $y > a > 0$ for some small a. Now, on the range $[k+a\sqrt{k}, 3k/2]$, $T_k(x)$ is the sum of k terms, each of which is at most $x^k/k!$. It follows that

$$T_k(x)e^{-x} \le kx^k e^{-x}/k! \le ke^{-y^2/4}/\sqrt{k}.$$

Thus

$$\begin{aligned}
|g(x)| &\le (x-k)e^{-y^2/4}/\sqrt{k} \\
&\le (k/2)e^{-y^2/4}/\sqrt{k},
\end{aligned}$$

which is less than ϵ provided $y > a > 2\sqrt{\log(\sqrt{k}/(2\epsilon))}$.

Finally, on $[3k/2, \infty)$, it is easy to see that $g(x) < 0$. We now show that $g'(x) > 0$, so $|g(x)|$ decreases, from an acceptable value derived for the range $[k+a\sqrt{k}, 3k/2]$. We have

$$\begin{aligned}
g'(x) &= \left((1-x/k)T_k(x)e^{-x}\right)' \\
&= -\frac{1}{k}T_k(x)e^{-x} - (1-x/k)\frac{x^{k-1}}{(k-1)!}e^{-x} \\
&\ge \left(-\frac{T_k(x)}{k} + \frac{x^{k-1}}{2(k-1)!}\right)e^{-x},
\end{aligned}$$

using the fact that $-(1-x/k) \geq 1/2$. Now, $\frac{T_k(x)}{k}$ is the average of the terms in $T_k(x)$ and $\frac{x^{k-1}}{(k-1)!}$ is the last term. But the j'th term is bigger than the $(j-1)$'st by the factor $(x^j/j!)/(x^{j-1}/(j-1)!) = x/j \geq x/k \geq 3/2$. Thus the terms in $T_k(x)$ grow at least as fast as a geometric series; it follows that the last term is more than twice the average term, for sufficiently large k, so $g'(x) > 0$ for $x \geq 3k/2$.

Summarizing all the above constraints, we need

$$\begin{cases} a \geq \sqrt{\log(\sqrt{k}/\epsilon)} \\ a/\sqrt{k} \leq \epsilon \\ a > 2\sqrt{\log(\sqrt{k}/(2\epsilon))} \end{cases}$$

Given ϵ, first put $k = 1/\epsilon^{2+\iota}$ for small constant $\iota > 0$ (in fact, ι can be $o(1)$ as $k \to \infty$). Then put $a = \epsilon\sqrt{k}$, satisfying the second constraint. That is, $a = 1/\epsilon^{\iota/2} \approx k^{\iota/4} \gg \sqrt{\log(k/\epsilon)}$.

Next, consider integrating $g(x)$. Clearly $\int_0^\infty p(x)e^{-x} < \infty$ for any polynomial p, so $\int x^j g(x) < \infty$.

Now, suppose the original data is 0/1-valued. Similar to the above, data in the range $[0, W - aW/\sqrt{k}] \cup [W + aW/\sqrt{k}, \infty)$ contributes negligible error, since the error in approximating χ_W drops off exponentially in $W(x-k)/\sqrt{k}$. For x in the range $[W - aW/\sqrt{k}, W + aW/\sqrt{k}]$, the error is at most $O(\epsilon a W/\sqrt{k}) \approx O(\epsilon^2 W)$.

Finally, the word space required for $g(kx/W)$ is one more than the degree of the polynomial, $(1-x/k)T_k(x)$, multiplying the exponential, *i.e.*, space $k+1$, as required. □

7.3 Threshold Problem

Next we define the chordal threshold problem and its approximate version as follows:

DEFINITION 7.1. *Fix parameters W and τ. We are given a stream $f(t)$ of 0/1-valued data. At any time instant, we need to decide whether or not $V_{\chi_W} > \tau W$.*

In the approximate version, we also fix a parameter ϵ. We need to answer "yes" if $V_{\chi_W} > (1+\epsilon)\tau W$, no if $V_{\chi_W} < \tau W$, and may answer arbitrarily otherwise.

Then we have

THEOREM 5. *The (τ, W, ϵ) approximate chordal threshold problem can be solved in word space $(\tau\epsilon)^{-(1+o(1))}$.*

PROOF. To solve the problem, we need additive error $\epsilon\tau W$. To achieve this, put $\epsilon' = \sqrt{\epsilon\tau}$ and track approximate chordal decay with parameter ϵ'. By Theorem 4, this will achieve additive error $(\epsilon')^2 W$ in space $(\epsilon')^{-(2+o(1))}$, as desired. □

On the other hand,

THEOREM 6. *Fix $\tau = 1/8$. The (τ, W) exact chordal threshold problem requires $\Omega(N)$ space.*

PROOF. It suffices to show that $\Omega(W)$ space is required; that is, we show the lower bound on streams of length W.

We will show that any bit string s of length $W/4$ can be encoded in the memory of any algorithm for the (τ, W) exact chordal threshold problem. Enter s into the memory as ordinary data, putting the algorithm into some configuration C. We will show how to recover s.

Suppose we allow $3W/4 - 1$ time steps to elapse. Then the most recent bit s_0 of s gets weight $1/W$ and all the other bits get weight zero. Now suppose that, as the $3W/4 - 1$ time steps elapse, we enter data, either 0 or 1, so that the decayed value of the data we enter is exactly τW (we'll show how to do this below). Then the (τ, W) exact chordal threshold problem will return yes or no, depending on whether $s_0 = 1$ or $s_0 = 0$.

In general, inductively, we suppose that we have determined the most recent j bits of s. Return to configuration C, then let $3W/4 - j - 1$ time units elapse, so that the $(j+1)$'st bit of s gets weight $1/W$ and all older bits of s get weight 0. We will add data so that the total decayed value that we add together with the decayed value of the most recent j bits of s totals exactly τW. At this point, we'll be able to learn s_{j+1}.

It remains only to show how to add our own data. First, we show how to add data to make the decayed value exactly τW. There's some number $c \approx .07$ such that, if the most recent cW data items are 1 and the others are 0, then the decayed value is less than or equal to $\tau W = W/8$; but, if the most recent $cW + 1$ data items are 1, then the decayed value is greater than τW. Tentatively let the most recent $cW + 1$ items be 1; the decayed value is between τW and $\tau W + (1-c)$. Now observe that, by shifting the oldest 1 one unit into the past, the decayed value decreases by exactly $1/W$, the finest unit of change needed, so, by shifting all the 1's one unit into the past, the value decreases by c. Shifting all the 1's 20 units into the past decreases the decayed value by $20c > 1 > (1-c)$, the largest amount of decrease needed. It's not hard to see that some sequence of data points of the form $0^{a_1}1^{a_2}0^{a_3}1^{a_4}0^{a_5}$, with $a_1 + a_2 + a_3 + a_4 \leq c + O(1) < W/10$ will have decayed value exactly τW.

Similarly, the j most recent bits of s, $j < W/4$, will have total decayed value at most $W/32$ when they are aged so that the $(j+1)$'st most recent bit becomes the most recent with weight zero. This decayed value gets added to the decayed value that we artificially introduce, and we want the total to be exactly τW, so we need to decrease the decayed value we introduce by an offsetting amount. We can decrease that decayed value by up to $W/32$ by using less than all the most recent cW slots (followed by some fine adjustments). Thus the data we add uses approximately the most recent $.07W \lll 3W/4$ slots, so it does not interfere with the $W/4$ aged data points in s. □

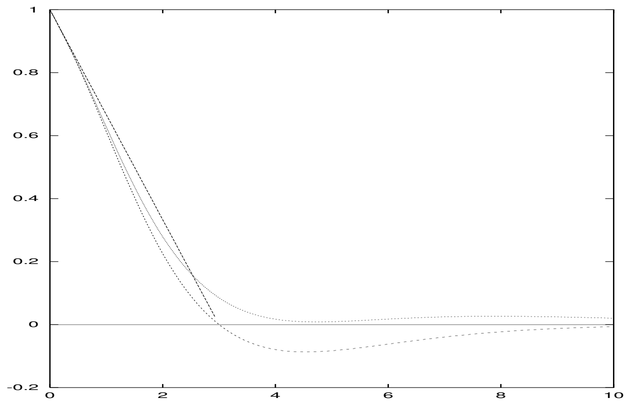
7.4 Extensions and variations.

- The approximation to chordal decay χ_W becomes negative. This can be prevented, if desired, by approximating $\chi_W(x) + \Theta(1/\sqrt{k})$, instead. (The exponential decay makes all curves go to zero asymptotically.) An example is given in Figure 2 (with hand-tuned parameters).
- Any decreasing polygonal function can be regarded as a sum of chords. Since each chord can be approximated, so can any polygonal function, with cost depending to the complexity of the function and desired distortion ϵ, but independent of elapsed time and size of support of the decay function.
- Sliding Window decay can be approximated by decay by $T_k(kx/W)e^{-kx/W}$. Unlike the case of chordal decay, the sliding window is not continuous, so the sliding

window decay function is not approximated uniformly by a polyexp function. Using the polyexp approximation, decayed values are tracked, in space $\epsilon^{-(2+o(1))}$, with additive error ϵW instead of $\epsilon^2 W$.

- Decay of the form $p(x)$ for $x \in [0, W]$ and 0 elsewhere, where p is a low degree polynomial with $p(W) = 0$, can be approximated as $p(x)T_k(kx/W)e^{-kx/W}$. The word space will depend on the desired distortion, the degree of p, and the slope of p near W, but not directly on W.

Figure 2: Some plots showing Polyexp approximation to Chordal decay. The line segment $(1 - x/3)$ is ideal. The non-negative smooth curve is $(.78 - x/3)(1 + x + x^2/2)e^{-x} + .22(1 + x/2 + (x/2)^2/2)e^{-x/2}$ and uses space 7. The sometimes-negative curve is $(1 - x/3)(1 + x + x^2/2)e^{-x}$ and uses space 4. These coefficients were hand-tuned for the space allowed.



8. SUMMARY

Decay function are important tools for summarizing streams of data values. The use of time-decaying weighted aggreagates predates the theory of stream algorithms, but was essentially limited to Exponential decays, for which there was a known simple and efficient algorithm that requires only $O(\log N)$ storage bits. Recent work of Datar et al [5] considered sliding-window decay and developed algorithms for approximately maintaining aggregates using $O(\log^2 N)$ storage.

We argue that Exponential and sliding window decay lack some important intuitive properties and do not provide sufficient flexibility in tuning the rate of decay. Thus, it is important to develop efficient algorithms for other families of decay functions. One very appealing such family is polynomial decay.

We then extend the technique of [5] and present algorithms for maintaining time-decayed aggregates under general decay functions using $O(\log^2 N)$ storage, in particular, showing that in a sense, sliding windows are the "hardest" decay functions in terms of required storage. We continue and develop a different algorithm which beats the $O(\log^2 N)$ storage bound for some families of decay functions. In particular, polynomially-decaying aggregates can be maintained using $O(\log N \log\log N)$ storage bits. We thus show that, interestingly, polynomial decay can be maintained almost as efficiently as Exponential decay and considerably more efficiently than sliding window decay. We also provide a lower bound of $\Omega(\log N)$ for polynomially-decaying aggregates.

This work is complemented by another set of techniques, showing how the classic single-register Exponential decay tracking can be extended to handle a wider class of decay functions.

Beyond the theoretical and conceptual contributions, our work introduces a powerful new tool for a range of applications involving large data streams.

9. REFERENCES

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of the 2002 ACM Symp. on Principles of Database Systems (PODS 2002)*. ACM, 2002.

[2] A. Bremler-Barr, E. Cohen, H. Kaplan, and Y. Mansour. Predicting and bypassing internet end-to-end service degradations. In *Proc. 2nd ACM-SIGCOMM Internet Measurement Workshop*. ACM, 2002.

[3] E. Cohen, H. Kaplan, and J. D. Oldham. Managing TCP connections under persistent HTTP. *Computer Networks*, 31:1709–1723, 1999.

[4] Corinna Cortes and Daryl Pregibon. Giga-mining. In *Proceedings of KDD*, New York, August 1998.

[5] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 2002.

[6] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transaction on Networking*, 1(4), 1993.

[7] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proc. of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 63–72. ACM, 2002.

[8] Warren Gilchrist. *Statistical Forecsting*. Wiley, 1976.

[9] V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM'88 Conference*, August 1988.

[10] S. Keshav, C. Lund, S. Phillips, N. Reingold, and H. Saran. An empirical evaluation of virtual circuit holding time policies in IP-over-ATM networks. *IEEE J. on Selected Areas in Communication*, 13, 1995.

[11] R. Morris. Counting large numbers of events in small regisgers. *CACM*, 21:840–842, 1978.