The Frequent Items Problem, under Polynomial Decay, in the Streaming Model

Guy Feigenblat, Ofra Itzhaki, and Ely Porat

Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel
{feigeng,yzhakio,porately}@cs.biu.ac.il

Abstract. We consider the problem of estimating the frequency count of data stream elements under polynomial decay functions. In these settings every element arrives in the stream is assigned with a time decreasing weight, using a non increasing polynomial function. Decay functions are used in applications where older data is less significant \ interesting \ reliable than recent data. We propose 3 poly-logarithmic algorithms for the problem. The first one, deterministic, uses $O(\frac{1}{\epsilon^2}\log N(\log\log N+\log U))$ bits. The second one, probabilistic, uses $O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon\delta}\log N)$ bits and the third one, deterministic in the stochastic model, uses $O(\frac{1}{\epsilon^2}\log N)$ bits. In addition we show that using additional additive error can improve, in some cases, the space bounds. This variant of the problem is important and has many applications. To our knowledge it was never studied before.

1 Introduction

Processing large data is a significant subject in nowadays research. In the internet epoch, we have mass amount of data to process, in the size of Terabytes or even Petabytes. Many applications, such as IP communication management, audio and video streaming, sensor reading and stock exchange tracking are characterized by data arriving in mass amounts, sequentially and rapidly. Most of the frameworks for these applications don't have enough workspace to process the data and store it entirely. Thus, most of the algorithms typically store a synopsis of the data using much less space than the amount of the arriving data. Using the synopsis, the algorithm should be able to answer queries regarding the data and clearly, there is a tradeoff between the size of the synopsis and the precision of the returned answers. As an outcome, there is a constant need to develop new algorithms, more accurate and efficient, for online management and monitoring of these applications. These settings introduced, a decade ago, a model of computation called data stream (Streaming).

In the streaming model each arriving data item has the same relative contribution, i.e. weights the same. However, in many applications older data is less significant \ interesting \ reliable than recent data. Therefore, we would like to weight newer data more heavily than older data. This can be done using decay functions, which are non increasing functions, that assign weight to each arriving item. Each weight is constantly updated as a function of the time elapsed since

J. Karlgren, J. Tarhio, and H. Hyyrö (Eds.): SPIRE 2009, LNCS 5721, pp. 266–276, 2009.
© Springer-Verlag Berlin Heidelberg 2009

the item was first observed in the stream. There are few types of commonly used decay functions, such as Sliding window, Polynomial and Exponential decay. When we consider Sliding window decay, each data item in the window is assigned with one unit weight, while all the others, outside the window, are discounted. Other types of decay functions assign each item with different weight, diminishes as a function of the time elapsed.

For many applications needs, both sliding window and exponential decay aren't sufficient. The advantage of polynomial decay is that the weights decrease in a smoother way. Namely, the ratio between two different elements is constant, as will be elaborated in the sequel.

In this work we propose 3 sketch based algorithms for estimating data stream elements decayed frequencies under polynomial decay. The first one, is deterministic, the second one is probabilistic and the third one, is deterministic in the stochastic model. Our work serves both theoretical and practical aspects. Specifically, we address a variant of the frequency count problem that to our knowledge was never studied before, in an efficient and simple way. Moreover, multiple sketches of the deterministic algorithm can be joined together easily, which is an advantage for the distributed streaming scenario. In addition, this variant can be suitable for applications where other types of decay are too rigid.

One application that can utilize this work is caching system for web servers. There are many replacements policies for caching, where the most commonly used are the LRU (Least Recently Used) policies. However, it isn't the case for caching web servers [1,2]. Results of researches show that using LFU (Least Frequently Used) replacement policies with aging mechanism (i.e. frequency count with decay) perform much better [1]. There are more frequency based caching algorithms like in [3].

1.1 Decay Functions

We represent the definitions given by Cohen and Strauss in [4]. Consider a stream where $f(t) \geq 0$ is the item value of the stream obtained at time t. For simplicity we assume our stream only receives values at discrete times, and therefore, t is integral. We define a decay function $g(x) \geq 0$ for $x \geq 0$ to be a non-increasing function. At time T the weight of an item that arrived at time $t \leq T$ is $g(T-t)$ and the decayed value of that item is $f(t)g(T-t)$. The decayed sum (DSP) under the decay function $g(x)$ is defined as: $V_g(T) = \sum_{t \leq T} f(t)g(T-t)$. When $f(t)$ receives only binary values, we refer to $V_g(T)$ as decay count (DCP), since it aggregates under decay the number of positive bits.

As mentioned above, there are 3 types of commonly used decay functions: Sliding Window, Exponential and Polynomial Decay. Formally, the Sliding Window decay for a window size W, assigns weights $\forall\ 0 \leq x \leq W\ g(x) = 1$ and $g(x) = 0$ otherwise. Exponential Decay (ExpD) for a given parameter $\lambda > 0$, assigns weights $g(x) = \exp(-\lambda x)$. Polynomial Decay (PolyD) for a given parameter $\alpha > 0$, assigns weights $g(x) = \frac{1}{x^{\alpha}}$.

1.2 Model and Problem Definition

Consider a data stream of N elements drawn from a universe U. Each is assigned at arrival with a time decreasing weight. Elements constantly arrive and due to the size of the stream, it is only allowed to perform one pass over the data. Furthermore, the storage available is poly-logarithmic in N and the data should be processed in minimum time.

Let $\widetilde{N}$ be the sum of weights assigned to the elements, such that $\widetilde{N} = \sum_{i=1}^{N} g(i)$, and $\epsilon \in (0,1)$ be the error factor. Our goal is to approximate, up to error of $\epsilon\widetilde{N}$, the decayed frequency of each element observed in the stream, i.e. we would like to approximate the decayed count (DCP) of each element with error less then $\epsilon\widetilde{N}$.

1.3 Related and Previous Work

As was mentioned above, to our knowledge, this variant of the problem was not studied before.

Cohen and Strauss in [4] introduced the time decay sum and time decay average under general decay function. In addition, they developed a data structure (sketch) - Weight Based Merging Histograms (WBMH) - that guarantees $(1 \pm \epsilon)$ multiplicative approximation for the sum of values of the elements observed in the stream, under polynomial decay. This structure uses at most $O(\frac{1}{\epsilon} \log N \log\log N)$ bits of space, where $\epsilon \in (0,1)$ and N denotes the length of the stream. They also showed a lower bound of $\Omega(\log N)$ bits for this problem. Kopelowitz and Porat in [5] proposed an improved algorithm - Altered Exponential Histograms (AEH) - matching the lower bound from [4]. Their algorithm combines WBMH [4] and Exponential Histograms (EH) [6]. They also proposed another model where additive error is allowed. For more details see the sequel.

Cormode et al [7] proposed a time decay sketch technique for summarizing decayed streaming data. The sketch can give estimates for various decayed aggregates. It is mainly targeted to the distributed streaming scenarios, such as sensor networks, since it is duplicate insensitive - meaning that re-insertion of the same data will not affect the estimates of the aggregates. Furthermore, multiple sketches computed over distributed data can be combined without losing accuracy.

In another paper, Cormode et al [8] proposed deterministic algorithm for approximating several decayed aggregates in out of order streams. Under these settings, elements do not necessarily arrive in the order of their appearance in the stream. The algorithm works with Sliding window, Polynomial and Exponential decay.

Not a lot of work was done on estimating decayed stream aggregates, however, many algorithms were proposed for approximating stream aggregates in general and frequency count in particular. For a more detailed description of the work that was done see [9].

The first deterministic algorithm, which provides additive approximation for the frequencies of data streams elements, is the Misra-Gries Algorithm [10]. This algorithm uses m counters, where the size of m depends on the approximation accuracy, and provides $O(1)$ amortized processing time per element. The same algorithm was rediscovered by Demaine et al [11] and Karp et al [12], who reduced the processing time to $O(1)$ in the worst case. In order to get approximation with maximum error ϵN where N denotes the length of the stream, we set $m = \lceil \frac{1}{\epsilon} \rceil$. Demaine et al [11] also developed algorithm for the stochastic model.

Manku and Motwaniet [13] proposed 2 algorithms for computing frequencies of elements. The first one, *StickySampling*, is probabilistic algorithm which identifies all items that their true frequency exceeds $(s - \epsilon)N$ with probability $1 - \delta$, where N denotes the length of the stream, $s \in (0, 1)$ is a user specified threshold and $\epsilon \in (0, 1)$ is the maximum error. The expected number of counters is at most $O(\frac{1}{\epsilon} \log(s^{-1}\delta^{-1}))$. The second algorithm, *LossyCounting* is a deterministic algorithm. It guarantees the same precision using at most $O(\frac{1}{\epsilon} \log(\epsilon N))$ counters, regardless of s.

Arasu and Manku [14] proposed 2 algorithms for calculating frequency count over Sliding Window decay. The first one, deterministic, uses $O(\frac{1}{\epsilon} \log^2(\frac{1}{\epsilon}))$ counters and the second one, probabilistic, provides approximation with probability at least $(1 - \delta)$ by using $O(\frac{1}{\epsilon} \log(\epsilon\delta)^{-1})$ counters.

Cormode and Muthukrishnan in [15] introduced the Count-Min sketch. They developed a poly-logarithmic data structure for summarizing data streams, utilizing pair-wise independent hash functions. Each arriving element is mapped from the universe U to entries in the sketch. At query time, the result of the query is the minimum value of the entries mapped to the element. The size of the sketch is $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$ and the error guarantee per query is ϵL_1 with probability $1 - \delta$, where L_1 denotes the sum of frequencies of the elements observed. For further details see the sequel.

2 Deterministic Algorithm

2.1 Weight Based Merging Histograms

The Weight Based Merging Histograms were introduced by Cohen and Strauss in [4]. These histograms provide $\epsilon-$ multiplicative approximation for Poly Decay count using $O(\frac{1}{\epsilon} \log N \log \log N)$ bits of space (where N denotes the stream's length).

Weight Based Histograms (WBMH) as other types of histograms, such as Exponential histograms [6], aggregate values into buckets. The main difference is that in WBMH the boundaries of the buckets are dependent on the decay function, and not on a particular stream instance. This means that the buckets' timestamps don't need to be stored in the buckets explicitly.

WBMH utilize the fact that if a decay function has the property that $g(x)/g(x + \Delta)$ is non-increasing with x for any time frame Δ, then the ratio of two items remains fixed, or approaches one as time advances. This means that as time progresses, elements in larger vicinities have the same decayed weight

up to a multiplicative factor. WBMH utilize this property by grouping together in the same bucket, values with similar weights. Buckets boundaries are determined in the following way. Let $b_0 = 1$ and b_1 be the maximum value such that $(1+\epsilon)g(b_1-1) \geq g(b_0)$. In the same manner, let b_i be the maximum value such that $(1+\epsilon)g(b_i-1) \geq g(b_{i-1})$. Furthermore, the number of elements in each bucket depends on the stream, but the boundaries don't. The range $[b_i, b_{i+1}-1]$ is referred as a region.

In order to approximate the decay count (DCP) of the stream, WBMH work in the following way. When a new element arrives it is added to the "current" (first) bucket. At any time T where $T \equiv 0(mod\ b_1)$ the "current" bucket is sealed, and a new one is opened. Whenever there exist an i and two buckets, such that the two buckets are within a region $[b_i, b_{i+1}-1]$, they are merged into one. The DCP is the sum of the buckets multiplied by their region boundary.

Notice that the decay weight of each element in the same bucket is within $1 \pm \epsilon$ factor. Thus, the decayed count can be computed by multiplying the number of non-zero values in the bucket, with the bucket's corresponding region boundary. Since keeping an exact counter for the number of non-zero values in the bucket is costly, an estimated counter in the size of $O(\log\log N)$ bits is used. The approximated decayed count of the stream can be computed by summing the approximated decayed count of every bucket. It was showed in [4] that under Poly Decay there are $O(\frac{1}{\epsilon}\log N)$ different b_is for every function. Since 2 buckets within the same region are merged, the total number of buckets is $O(\frac{1}{\epsilon}\log N)$.

2.2 Algorithm

In this section, we describe a deterministic process that approximates the decayed frequency counts of data stream elements, under Poly Decay, with maximum error of $\epsilon\widetilde{N}$ per element. The space used in this process is $O(\frac{1}{\epsilon^2}\log N(\log\log N + \log U))$ storage bits, where N denotes the length of the stream.

For this process we will exploit the property that polynomial decay divides the stream to $O(\frac{1}{\epsilon}\log N)$ boundaries and the way buckets are constructed in [4] as was explained above. Since elements in a bucket have almost the same weight, the approximated decayed frequency of a single element in a bucket, is the number of times it appears multiply by the bucket's corresponding region boundary. Thus, the approximated decayed frequency of an element over the entire stream, is the sum of the approximated decayed frequencies over all the buckets. Formally, denote F_e as the DCP of element e, $\widetilde{f_e^i}$ as the approximated number of appearances of element e in bucket i and w_i as the weight corresponding to the region of bucket i. The approximated decayed frequency of element e over the stream is $\sum_{i=1}^{O(\frac{1}{\epsilon}\log N)} \widetilde{f_e^i} w_i$. Notice that we suffer from two approximation factors.

Our algorithm works as follows. We approximate in each bucket, the frequencies of the elements observed by it. In order to do so we utilize the Misra-Garies algorithm [10](or others as described in [11,12]) as a black box in each bucket. Notice that since we require approximation of ϵ, there should be $O(\frac{1}{\epsilon})$ counters in each black box. The buckets are constructed the same way as in WBMH.

Whenever a new element e arrives, it is added to the current bucket by sending it to the bucket's corresponding black box. At any time T where $T \equiv 0(mod\ b1)$ the current bucket is sealed and a new one is opened.

Whenever there exist an i and two buckets that are within a region $[b_i, b_{i+1}-1]$ they are merged into one. Merge operation is preformed in the following manner. Suppose we need to merge buckets i and j. Denote by N_i the elements observed by bucket i and by N_j the elements observed by bucket j. In each bucket there are $O(\frac{1}{\epsilon})$ counters corresponding to the $O(\frac{1}{\epsilon})$ elements with the highest frequencies. In the new bucket, created during the merge, we keep the $O(\frac{1}{\epsilon})$ elements, with the highest frequencies from both buckets. We can merge the buckets in a straight forward way (without considering the decay), since both buckets are in the same region and thus the elements' weights are roughly the same. Notice that, the number of elements allegedly observed by the new bucket is $N_i + N_j$ and the maximum errors in the old buckets i and j are ϵN_i and ϵN_j respectively. We get that the maximum error in the new merged bucket is $\epsilon(N_i + N_j)$, therefore, the ϵ approximation guarantee is preserved.

Whenever we are asked to retrieve the elements decayed frequencies we scan the buckets and for each element we sum it's frequencies multiplied by the bucket corresponding boundary.

Theorem 1. *Let $g(.)$ be a polynomial decay function such that $g(x)/g(x+1)$ is non increasing with x. The data structure uses $O(\frac{1}{\epsilon^2}log^2 N)$ storage bits and provides approximation with maximum error of $\epsilon\widetilde{N}$, for the elements decayed frequencies.*

Proof. In each bucket we use an instance of the Misra-Garies algorithm as a "black box" with error parameter ϵ'. In addition, we pick our boundaries using the given decay function with error parameter ϵ'', formally $\forall i\ (1+\epsilon'')g(b_i - 1) \geq g(b_{i-1})$. Since in each boundary there can be at most 2 buckets, we get that the total number of buckets is $O(\frac{1}{\epsilon''}\log N)$. Adding the "black box" to each bucket yields a total of $O(\frac{1}{\epsilon'\epsilon''}log^2 N)$ storage bits. Notice that for each element monitored by a counter, we need to maintain it's ID using $O(\log U)$ bits, but usually $\log U \leq \log N$ and thus bounded by $O(\log N)$.

Recall that our algorithm suffers from two approximation factors: the first one is from the way we approximate the frequencies in each bucket and the second is from using boundaries instead of exact decay weights. Combing these approximations together we get $(1 \pm \epsilon'')(F_e \pm \epsilon'\widetilde{N}) = F_e \pm \epsilon'\widetilde{N} \pm \epsilon'' F_e \pm \epsilon''\epsilon'\widetilde{N}$. Notice that $\epsilon'' F_e \leq \epsilon''\widetilde{N}$. Choosing $\epsilon \geq \epsilon' + \epsilon'' + \epsilon'\epsilon''$ by setting $\epsilon' = \epsilon'' = \frac{\epsilon}{3}$ provides the desired approximation guarantee and yields total of $O(\frac{1}{\epsilon^2}\log^2 N)$ storage bits. □

Lemma 1. *Under polynomial decay, $O(1)$ processing operations are required in amortized per element observed in the stream.*

Proof. We use amortization argument for proving the lemma. Let $c = \lceil \frac{1}{\epsilon^2}\log N \rceil$. We hold a buffer of size c and build the sketch in steps. At the first step, when c becomes full we iterate over it and build the sketch structure. The

cost of this operation is c. The number of buckets created in the sketch is $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon^2}\log N)) = O(\frac{1}{\epsilon}\log\log N)$. We do the same at every round, but in addition we have to append and merge the old sketch at the end of the new sketch. Since the old sketch has $O(\frac{1}{\epsilon}\log\log N)$ buckets each of size $O(\frac{1}{\epsilon})$, the merge operation is bounded by $O(\frac{1}{\epsilon^2}\log\log N)$. Thus, the processing time is bounded by the size of the buffer, namely, $O(\frac{1}{\epsilon^2}\log N)$. Since we do this operation every c time units, we get that the amortized processing time is $O(1)$. □

We can further reduce the space by allowing additional multiplicative approximation. It can be done by using estimated counters, in each black box. It is sufficient to save the exponent, using $O(\log\log N)$ bits and in addition the most significant $O(\log\frac{1}{\epsilon}+\log\log N)$ bits, for each counter. This extra relaxation is possible since the number of merge operations, for each bucket, bounded by $O(\log N)$. This reduces the overall size of the structure to $O(\frac{1}{\epsilon^2}\log N(\log\log N+\log U))$. Due to lack of space, further details are omitted, see [4] for a similar counter method.

Using this structure, multiple histograms can be joined together quiet easily without losing approximation guarantees. Joining two histograms can be done by iterating over the regions and merging 2 buckets at a time (merging buckets as explained above). This property is an advantage when considering distributed streams.

3 Probabilistic Algorithm

In this section, we describe a probabilistic algorithm that approximates with high probability $(1-\delta)$, the decayed frequency count under Poly Decay, with maximum error $\epsilon\widetilde{N}$. The space used by this algorithm is $O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon\delta}\log N)$ bits, where N denotes the current length of the stream and $\epsilon,\delta\in(0,1)$. Our algorithm utilizes Altered Exponential Histograms [5] and Count-Min sketches [15].

3.1 Count-Min Sketch

As mentioned above, the Count-Min sketch is a poly-logarithmic space data structure for summarizing data streams. The idea of the sketch is to model the stream as a vector X of $|U|$ dimensions, where U denotes the universe. The current state of the vector (stream) at time t is $X(t)=[X_1(t),...,X_i(t),...,X_{|U|}(t)]$. Initially, X is the zero vector: $\forall i\ X_i(0)=0$. When element $i\in U$ arrives in the stream at time t, it is modeled as if the vector's i'th entry is updated (incremented). The value of the i'th entry at time t, i.e. $X_i(t)$, is the frequency of element i. The user specifies 2 parameters (ϵ,δ), where ϵ denotes the error parameter and δ is the probability of failure. Since it isn't possible to maintain the entire vector, a vector sketch is constructed. The sketch is represented as a two-dimensional array of size wd, denotes by $count[\ ,\ ]$, where $w=\lceil\frac{e}{\epsilon}\rceil$ and $d=\lceil\ln\frac{1}{\epsilon\delta}\rceil$. The accuracy estimates for individual query in the sketch depends on the L_1 norm of vector X at any time t.

One of the queries to the sketch is the point query, denotes by $Q(i)$. It returns the approximated frequency of element $i\in U$ at any time t. Formally, for any

time t, $X_i(t)$ is the frequency of element i and let $\widetilde{X_i}(t)$ be the approximated frequency of element i. The query retrieves frequency estimation such that $X_i \leq \widetilde{X_i}$ and with probability at least $1-\delta$, $\widetilde{X_i} \leq X_i + \epsilon||X(t)||_1$.

3.2 Algorithm

Our idea adapts the Count-Min sketch structure and combines Alter Exponential Histograms (AEH) as a black box in each sketch entry. The function of the AEH is to calculate each entry value under the Poly decay function. The sketch uses wd two-dimensional array, initially set to zero. In addition, d hash functions : $h_1...h_d : 1...|U| \rightarrow 1...w$, are chosen uniformly at random from a pairwise-independent family.

When an element i arrives in the stream at time t, the data structure is updated by adding "1" bit, to the AEH corresponding to each entry mapped to the element. Formally, $\forall\ 1 \leq j \leq d,\ count[j, h_j(i)].\ AEH.\ increment(1)$.

In order to retrieve decayed frequency estimation of element i, the entry with the minimum value of all the entries mapped to the element is returned. Formally, $Q(i) = min_j\ count[j, h_j(i)].\ AEH.\ value()$.

Notice that under these settings X_i denotes the decayed frequency of element i and $\widetilde{X_i}$ denotes the approximated decayed frequency of element i. In addition, recall that we are calculating the frequency by estimating the DCP for each observed element. For any polynomial decay function $g(.)$, we set the accuracy estimates to be dependent on $\widetilde{N}$, since this is the sum of the decayed weights of the elements. Thus, the maximum frequency error expected per element, is set to be $\epsilon\widetilde{N}$ with high probability.

Theorem 1. *Let $g(.)$ be a polynomial decay function such that $g(x)/g(x+1)$ is non increasing with x. The data structure uses $O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon\delta}\log N)$ storage bits and provides approximation for single element decayed frequency (Point Query), such that $X_i \leq \widetilde{X_i}$; and with probability at least $(1-\epsilon\delta)$, $\widetilde{X_i} \leq X_i + \epsilon\widetilde{N}$. The processing time per element is $O(\log\frac{1}{\epsilon\delta})$.*

Proof. We use a Count-Min sketch with parameters ϵ', δ. We put in each entry of the sketch an instance of AEH with error parameter ϵ''. ϵ', ϵ'' will be determined later. Under these settings, the total space consumption is $O(\frac{1}{\epsilon'\epsilon''}\log\frac{1}{\epsilon'\delta}\log N)$ bits.

First we consider the error in each Point Query, overlooking the error factor from the AEH. The query proof is actually an adaptation of the proof from [15] and therefore we only sketch it. By pairwise independence of the hash functions we get that the probability of collision, for each entry in a row, is less then $1/range(h_j) = \frac{\epsilon'}{e}$. In [15] the authors showed that the expected error in each sketch entry is less then $\frac{\epsilon'}{e}L_1$, which equal $\frac{\epsilon'}{e}\widetilde{N}$ in these settings. In addition, by pairwise independence of h_j and linearity of expectations, it was shown that $Pr[\widetilde{X_i} \geq X_i + \epsilon' L_1] \leq \epsilon'\delta$.

Combing the AEH, we suffer from 2 errors. The first one, multiplicative error of $(1 \pm \epsilon'')$ from the AEH and the second one, additive error of $\epsilon'\widetilde{N}$ from the

Count-Min structure. Setting $\epsilon' = \epsilon'' = \frac{\epsilon}{3}$, provides the desired approximation guarantee and a total space consumption of $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon\delta} \log N)$ bits.

As for the processing time, whenever an element arrives we update $O(\log \frac{1}{\epsilon'\delta})$ rows in the two-dimensional array. In each update we increment the histogram which cost $O(1)$ in amortized per update [5]. □

Using the above theorem we can now present an algorithm for decayed frequency count, which follows an idea from [16,15]. For each element, we use the Count-Min data structure to estimate its count, and keep a heap of the top $\lceil \frac{1}{\epsilon} \rceil$ elements seen so far.

Given a data stream, for each element i observed at time t we do the following:

1. Update the entries mapped to i in the Count-Min sketch
2. Retrieve the decayed frequency of element i by running $Q(i)$ query
3. If i is in the heap, increment its count (by adding "1" to it's histogram)
4. Else, if $Q(i)$ is greater then the smallest value in the heap then,
   (a) Generate AEH instance equal the the histogram corresponding to $Q(i)$
   (b) Pop the heap (remove the smallest value)
   (c) Add the newly created AEH instance to the heap

At query time the heap is scanned and all elements in the heap with estimated count above $\epsilon\widetilde{N}$ are output. The probability that an element will not be properly estimated during a point query is less than $\epsilon\delta$. Since an improper estimation of an element is only when the approximation is above the $\epsilon\widetilde{N}$ threshold, an improper estimation in the decayed frequency count algorithm can be caused only by one of the $\frac{1}{\epsilon}$ elements in the heap. Applying union bound shows that the total probability of error in the algorithm is bounded by δ.

In the heap we keep $\lceil \frac{1}{\epsilon} \rceil$ elements. Each element is associated with AEH of size $O(\frac{1}{\epsilon} \log N)$ bits and $O(\log U)$ bits for it's ID. Since usually $\log U \leq \log N$, the total space of the heap is $O(\frac{1}{\epsilon^2} \log N)$ bits. We conclude that the total space consumption of the algorithm is $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon\delta} \log N)$ bits.

4 Deterministic Algorithm in the Stochastic Model

In this section we present an algorithm for decayed frequency count in the stochastic model. In this model, an arbitrary probability distribution specifies the relative frequencies of the elements and the order the elements occur in the stream is uniformly random.

Algorithm for frequency count in the stochastic model was proposed by Demaine et al in [11]. The algorithm divides the stream into rounds. At the beginning of each round, the first distinct m elements are sampled, which is equivalent to sampling m elements uniformly at random. At the end of the first round the top $\frac{m}{2}$ elements (with highest frequencies) are saved and the others are discounted. At the next rounds, only $\frac{m}{2}$ counters are used for sampling. At the end of each round, total of $\frac{m}{2}$ elements with the highest frequencies, from the current and the previous rounds are saved. Applying Chernoff bounds show that

the counts obtained during a round are close to the actual frequencies of the elements. For further details and completeness see [11].

We adapt this idea but instead of using m binary counters we use histograms, namely the AEH histograms. As was mentioned above, each histogram is of size $O(\frac{1}{\epsilon}\log N)$ bits. We get that the total size of the structure is $O(m\frac{1}{\epsilon}\log N)$. It is sufficient to set $m = \lceil\frac{1}{\epsilon}\rceil$ for getting good approximation with high probability.

5 Decay with Additional Additive Error

The AEH approximates the DCP by $(1 \pm \epsilon)$ multiplicative approximation. We now consider another model in which in addition to the multiplicative error, an additive error of $\epsilon_2 \in (0, 1)$ is allowed (in the AEH). This kind of relaxation, was discussed in [5]. It was shown that when considering binary streams with polynomial decay, there is a need to differentiate between 2 cases. In the first case where $\alpha > 1$, it is sufficient to maintain only the last $\frac{1}{\epsilon_2}$ elements observed by the stream. It can be done by saving them in AEH and therefore reduce the space to $O(\log(\frac{1}{\epsilon_2}))$ per histogram. In the second case where $0 \leq \alpha \leq 1$, a lower bound was proved showing that it isn't possible to do better.

Since we can calculate the decayed frequencies using the AEH, allowing this extra relaxation can reduce the space consumption. Namely, using this method in the probabilistic algorithm (see section 3), reduces the space consumption to $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon_2}\log\frac{1}{\epsilon\delta}\log N)$ bits. Similarly can be done to the algorithm in the stochastic model.

References

1. Arlitt, M.F., Williamson, C.L.: Trace-driven simulation of document caching strategies for internet web servers. Simulation Journal 68, 23–33 (1996)
2. Cao, P., Irani, S.: Cost-aware www proxy caching algorithms. In: Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, pp. 193–206 (1997)
3. Friedrich, R., Arlitt, M., Arlitt, M., Cherkasova, L., Cherkasova, L., Dilley, J., Friedrich, R., Jin, T.: Evaluating content management techniques for web proxy caches. In: Proceedings of the 2nd Workshop on Internet Server Performance (WISP 1999), Atlanta GA (1999)
4. Cohen, E., Strauss, M.J.: Maintaining time-decaying stream aggregates. J. Algorithms 59(1), 19–36 (2006)
5. Kopelowitz, T., Porat, E.: Improved algorithms for polynomial-time decay and time-decay with additive error. Theor. Comp. Sys. 42(3), 349–365 (2008)
6. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows (extended abstract). In: SODA 2002: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, pp. 635–644 (2002)
7. Cormode, G., Tirthapura, S., Xu, B.: Time-decaying sketches for sensor data aggregation. In: PODC 2007: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, pp. 215–224. ACM, New York (2007)

8. Cormode, G., Korn, F., Tirthapura, S.: Time-decaying aggregates in out-of-order streams. In: PODS 2008: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 89–98. ACM, New York (2008)
9. Muthukrishnan, S.: Data streams: Algorithms and applications. Foundations and Trends in Theoretical Computer Science 1(2) (2005)
10. Misra, J., Gries, D.: Finding repeated elements. Technical report (1982)
11. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 348–360. Springer, Heidelberg (2002)
12. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. 28(1), 51–55 (2003)
13. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Bressan, S., Chaudhri, A.B., Li Lee, M., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, pp. 346–357. Springer, Heidelberg (2003)
14. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: PODS 2004: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 286–296. ACM, New York (2004)
15. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms 55(1), 58–75 (2005)
16. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002)