Theoretical Computer Science 411 (2010) 3048–3054

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

The frequent items problem, under polynomial decay, in the streaming model

Guy Feigenblat*, Ofra Itzhaki, Ely Porat

Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel

ARTICLE INFO

Article history:
Received 15 June 2009
Received in revised form 13 March 2010
Accepted 12 April 2010
Communicated by G. Italiano

Keywords:
Streaming
Frequency count
Polynomial decay functions
Algorithms

ABSTRACT

We consider the problem of estimating the frequency count of data stream elements under polynomial decay functions. In these settings every element 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, less interesting or even less reliable than recent data. Consider a data stream of N elements drawn from a universe U. We propose three poly-logarithmic algorithms for the problem. The first one, deterministic, uses $O(\frac{1}{\epsilon^2}\log N(\log\log N + \log U))$ bits, where $\epsilon \in (0, 1)$ is the approximation parameter. The second one, probabilistic, uses $O(\frac{1}{\epsilon^2}\log\frac{N}{\delta}\log\frac{1}{\epsilon})$ bits or $O(\frac{1}{\epsilon^2}\log\frac{N}{\delta}\log N)$ bits, depending on the decay function parameter, where $\delta \in (0, 1)$ is the probability of failure. The third one, deterministic in the stochastic model, uses $O(\frac{1}{\epsilon}\log U)$ bits or $O(\frac{1}{\epsilon^2}\log N)$ bits, also depending on the decay parameter as will be described in this paper. This variant of the problem is important and has many applications. To our knowledge, it has never been studied before.

1. Introduction

Processing large data is a significant subject in modern research. In the internet era, 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 do not 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., is given the same weight. In many applications however older data is less significant, less interesting or even less 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 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 unit weight, while all the others, outside the window, are ignored. Other types of decay functions assign each item a different weight, diminishes as a function of the time elapsed.

* Corresponding author. Tel.: +972 3 531 8866; fax: +972 3 7384056.
E-mail addresses: feigeng@cs.biu.ac.il (G. Feigenblat), yzhakio@cs.biu.ac.il (O. Itzhaki), porately@cs.biu.ac.il (E. Porat).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.04.029

For many applications' needs, both sliding window and exponential decay are not sufficient. The advantage of polynomial decay is that the weights decrease in a smoother way. Namely, when using a polynomial decay function the ratio between two different elements is constant. For more details, see [1].

In this work we propose three 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 has never been studied before, in an efficient and simple way. Moreover, the deterministic algorithm sketches of multiple streams, 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 functions are too rigid.

One application that can utilize this work is caching system for web servers' proxies. There are many replacements policies for caching, where the most commonly used are the LRU (Least Recently Used) policies. However, it is not the case for caching web servers' proxies [2,3]. Results of researches show that using the LFU (Least Frequently Used) replacement policies with aging mechanism (i.e. frequency count with decay) perform much better [2]. There are more frequency based caching algorithms like in [4].

1.1. Decay functions

We present the definitions given by Cohen and Strauss in [1]. 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 the item is $f(t)g(T-t)$. The decayed sum (DSP) under the decay function $g(x)$ is defined as: $V_g(T) = \sum_{t \leq T} f(t)g(T-t)$. In case where $f(t)$ receives only binary values, we refer to $V_g(T)$ as decay count (DCP), since it aggregates the number of positive bits under a decay function.

As mentioned above, there are three types of commonly used decay functions: Sliding Window, Exponential and Polynomial decay. Formally, the Sliding Window decay for a window size W, assigns the weights $g(x) = 1 \; \forall \; 0 \leq x \leq W$ and $g(x) = 0$ otherwise. Exponential Decay (ExpD) for a given parameter $\lambda > 0$, assigns the weights $g(x) = \exp(-\lambda x)$. Polynomial Decay (PolyD) for a given parameter $\alpha > 0$, assigns the 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 element is assigned upon arrival a time-decreasing weight. Elements constantly arrive and, due to the size of the stream, the algorithm 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 has never been studied before.

Cohen and Strauss in [1] 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. For more details see Section 2.1. Kopelowitz and Porat in [5] proposed an improved algorithm – Altered Exponential Histograms (AEH) – matching the lower bound from [1]. Their algorithm combines WBMH and Exponential Histograms (EH) [6]. They also proposed another model, where an additive error is allowed in addition to the multiplicative error in the AEH, and by that reducing the space.

Exponential Histograms (EH) [6] were proposed by Datar et al. in order to estimate the number of positive bits within $1 + \epsilon$ factor under Sliding Window decay. The size of the histogram is $O(\frac{1}{\epsilon} \log^2 N)$ bits, and the processing and query time is constant, per element, in amortized. In addition they give a matching lower bound for any deterministic or randomized algorithm.

Cormode et al. [7] proposed a probabilistic sketch technique for summarizing data streams under general decay functions. They developed a sample based technique in a model that is duplicate insensitive and supports asynchronous arrivals. In these settings re-insertion of the same data does not affect the estimates of the aggregates and the arrival order of the elements is not guaranteed. It is mainly targeted to the distributed streaming scenarios, such as sensor networks, where the original order of the elements is not necessarily preserved, for instance, due to network delays. Their sketch technique is used for calculating the decayed sum, median, quantiles, and frequent elements. The sketch size is $O(\frac{1}{\epsilon^2} \log^2 N)$ bits and the frequency count query time is $O(\frac{1}{\epsilon^2} \log N (\log\log N)^2)$.

In another paper, Cormode et al. [8] proposed a deterministic algorithm for estimating decayed aggregates in the out-of-order streams model (asynchronous arrivals). Their algorithm estimates range queries, quantiles, and frequent elements under Sliding window, Exponential and Polynomial decay. Its space consumption is $O(\frac{1}{\epsilon^2}\log U\log^2 N\log(\frac{\epsilon N}{\log N}))$ bits, the update time is $O(\log(\frac{\epsilon N}{\log N})\log N\log\log U)$ and the query time is linear in the space used.

In our paper we consider a different model, a simpler variant, than those discussed in [7,8]. Our model is neither duplicate insensitive nor supports asynchronous arrivals. Nonetheless, for cases where these properties are not needed we give better space and time bounds.

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 an extensive survey 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 amortized cost of $O(1)$ per 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 an algorithm for the stochastic model. Bose et al. [13] give a lower bound on the accuracy of any deterministic packet counting algorithm, which implies the Misra–Gries algorithm is nearly optimal.

Manku and Motwani [14] proposed two algorithms for computing frequencies of elements. The first one, *StickySampling*, is a probabilistic algorithm which identifies all items whose 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 $O(\frac{1}{\epsilon}\log(\frac{1}{s\delta}))$. 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 [15] had proposed two 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. They also suggested an algorithm for frequency count in a variable-size sliding window, i.e., where there exists an adversary that has the ability to vary the number of elements in the window.

Golab et al. [16] presented a simple deterministic algorithm for identifying the frequent items in the Sliding Window model, over online data streams, and estimating their true frequencies. Their algorithm requires constant amortized processing time but they do not give a tight space bound.

Various problems have been studied in the sliding window model, such as, approximating quantiles [15] and estimating the variance of data streams [17]. Solving the latter, Zhang and Guan [17] proposed an optimal algorithm, which requires $O(1)$ processing time in worst case and the $O(\frac{1}{\epsilon}\log N)$ counters.

Cormode and Muthukrishnan introduced in [18] the Count-Min sketch. They developed a poly-logarithmic data structure for summarizing data streams, utilizing pairwise independent hash functions. They improved the best previous known results using sketches of [19]. In the Count-Min 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 $O(\frac{1}{\epsilon}\log\frac{1}{\delta})$ words 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 Section 3.1.

2. Deterministic algorithm

2.1. Weight Based Merging Histograms

The Weight Based Merging Histograms were introduced by Cohen and Strauss in [1]. These histograms provide ϵ-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 Merging 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 do not 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 do not. 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 $T \equiv 0 \pmod{b_1}$ 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 size of the buckets multiplied by g applied to the region boundary of the bucket.

Notice that the decayed 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 g applied to 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.

Lemma 2.1. *Let g be a polynomial decay function such that $g(x)/g(x+1)$ is non-increasing with x. Let b_i (boundary i) be the maximum value such that: $(1+\epsilon)g(b_i - 1) \geq g(b_{i-1})$. The number of boundaries is $O(\frac{1}{\epsilon}\log(N))$.*

Proof. See Lemma 5.1 in [1]. □

Since two 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. Consider a data stream of N elements drawn from a universe U, the space used in this process is $O(\frac{1}{\epsilon^2}\log N(\log\log N + \log U))$ storage bits.

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 that buckets are constructed in [1] 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 multiplied by g applied to 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–Gries algorithm [10] (or others as described in [11,12]) as a black box in each bucket. Notice that, since we require approximation to within 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 $T \equiv 0 \pmod{b1}$ 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 performed 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 straightforward 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 its frequencies multiplied by g applied to the bucket's corresponding boundary.

Theorem 2.1. *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 an approximation with maximum error of $\epsilon\widetilde{N}$, for the elements' decayed frequencies.*

Proof. In each bucket we use an instance of the Misra–Gries 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, we have $\forall i\ (1+\epsilon'')g(b_i - 1) \geq g(b_{i-1})$. Since in each region there can be at most two 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 its "ID" using $O(\log U)$ bits, but usually $\log U \leq \log N$ and thus $\log U$ is 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 decayed 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 2.2. *Under polynomial decay, $O(1)$ processing operations are required in an amortized sense per element observed in the stream.*

Proof. We use an 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.

The old sketch has $O(\frac{1}{\epsilon}\log N)$ buckets each of size $O(\frac{1}{\epsilon})$. In the merge operation we merge two consecutive buckets that are within the same region. We charge the "older" bucket with the cost of the merge operation, i.e., $O(\frac{1}{\epsilon})$. Notice that each bucket is charged at most once, as buckets cannot be separated after being merged. Thus the processing time is bounded by $O(\frac{1}{\epsilon^2}\log N)$. 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, is bounded by $O(\log N)$. This reduces the overall size of the structure to $O(\frac{1}{\epsilon^2}\log N(\log\log N+\log U))$. Further details are omitted; see [1] for a similar counter method.

Using this structure, multiple histograms can be joined together quite easily without losing approximation guarantees. Joining two histograms can be done by iterating over the regions and merging two 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 our algorithm is dependent on the decay parameter α. When $\alpha > 1$, it uses $O(\frac{1}{\epsilon^2}\log\frac{N}{\delta}\log\frac{1}{\epsilon})$ bits. Otherwise, when $0 < \alpha \leq 1$, it uses $O(\frac{1}{\epsilon^2}\log\frac{N}{\delta}\log N)$ bits. The algorithm utilizes Altered Exponential Histograms [5] and Count-Min sketches [18].

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 length $|U|$, where U denotes the universe. The current state of the vector (stream) at time t is $X(t) = [X_1(t),\ldots,X_i(t),\ldots,X_{|U|}(t)]$. 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 two parameters (ϵ,δ), where ϵ denotes the error parameter and δ is the probability of failure. Since it is not possible to maintain the entire vector, a vector sketch is constructed. The sketch is represented as a w-by-d array, denoted by $count[\ ,\ ]$, where $w = \lceil\frac{e}{\epsilon}\rceil$ and $d = \lceil\ln\frac{N}{\delta}\rceil$. The accuracy estimates for an 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, denoted 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.

Lemma 3.1. *The estimate $\widetilde{X}_i$ (Point Query) has the following guarantees: $X_i \leq \widetilde{X}_i$; and, with probability at least $1-\frac{\delta}{N}$, $\widetilde{X}_i \leq X_i+\epsilon L_1$*

Proof. See Theorem 1 and 6 in [18]. □

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 a w-by-d array, which is initially set to zero. In addition, d hash functions: $h_1\ldots h_d : 1\ldots|U| \to 1\ldots w$, 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 from the element. Formally, $\forall\ 1 \leq j \leq d,\ count[j,h_j(i)].AEH.increment(1)$.

In order to retrieve a 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, for each observed element, we are calculating its frequency by estimating its DCP. 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 3.1. *Let $g(.)$ be a polynomial decay function such that $g(x)/g(x+1)$ is non-increasing with x. The data structure uses $O(\frac{1}{\epsilon^2}\log\frac{N}{\delta}\log N)$ 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 $(1-\frac{\delta}{N})$, $\widetilde{X}_i \leq X_i+\epsilon\widetilde{N}$. The processing time per element is $O(\log\frac{N}{\delta})$.*

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 $O(\frac{1}{\epsilon'\epsilon''}\log\frac{N}{\delta}\log N)$ 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 Lemma 3.1 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 $1/range(h_j) = \frac{\epsilon'}{e}$. The expected error in each sketch entry

is less then $\frac{\epsilon'}{e}L_1$, which equal $\frac{\epsilon'}{e}\widetilde{N}$ in our settings. In addition, by pairwise independence of h_j and linearity of expectations, $Pr[\widetilde{X_i} > X_i + \epsilon' L_1] \leq \frac{\delta}{N}$.

Combing the AEH, we suffer from two 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 $O(\frac{1}{\epsilon^2}\log\frac{N}{\delta}\log N)$ bits.

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

Using the above theorem we can now present an algorithm for decayed frequency count, which follows an idea from [20,18]. 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 from 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 its histogram)
4. Else, if $Q(i)$ is greater than the smallest value in the heap, then
   (a) Generate AEH instance equal 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 $\frac{\delta}{N}$. Applying union bound over at most N different elements, shows that the total probability of error in the algorithm is less then 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 its "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 $O(\frac{1}{\epsilon^2}\log\frac{N}{\delta}\log N)$ bits.

3.3. Additional additive error

We can further reduce the space of the algorithm, by allowing an additive error of $\epsilon''' \in (0, 1)$ to each AEH, in addition to the multiplicative error. 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 two cases. In the first case, where $\alpha > 1$, it is sufficient to maintain only the last $O(\frac{1}{\epsilon'''})$ elements observed by the stream. It can be done by saving them in AEH and therefore reduce the space per histogram to $O(\frac{1}{\epsilon'''}\log\frac{1}{\epsilon'''})$. By setting $\epsilon' = \epsilon'' = \epsilon''' = \frac{\epsilon}{7}$ the approximation guarantee is preserved and the overall space of our algorithm is reduced to $O(\frac{1}{\epsilon^2}\log\frac{N}{\delta}\log\frac{1}{\epsilon})$ bits. In the second case, where $0 < \alpha \leq 1$, there is a lower bound stating that in order to estimate the DCP of single element, with both multiplicative and additive error, $O(\frac{1}{\epsilon}\log N)$ bits are required.

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.

An algorithm for estimating frequency count with additive error 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 m distinct 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 ignored. 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 shows 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 obtaining good approximation with high probability.

In a similar way to Section 3.3, we can utilize AEH with additive error in order to reduce the space when we use a decay function with parameter $\alpha > 1$. This reduces the overall space to $O(\frac{1}{\epsilon}(\frac{1}{\epsilon}\log\frac{1}{\epsilon} + \log U))$ bits.

Acknowledgements

We would like to thank the anonymous referees for their helpful review.

References

[1] E. Cohen, M.J. Strauss, Maintaining time-decaying stream aggregates, J. Algorithms 59 (1) (2006) 19–36.
[2] M.F. Arlitt, C.L. Williamson, Trace-driven simulation of document caching strategies for internet web servers, Simulation J. 68 (1996) 23–33.
[3] P. Cao, S. Irani, Cost-aware www proxy caching algorithms, in: USENIX '97: Proceedings of the Symposium on Internet Technology and Systems, 1997, pp. 193–206.
[4] R. Friedrich, M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, T. Jin, Evaluating content management techniques for web proxy caches, in: WISP '99: Proceedings of the 2nd Workshop on Internet Server Performance, Atlanta GA, 1999.
[5] T. Kopelowitz, E. Porat, Improved algorithms for polynomial-time decay and time-decay with additive error, Theoret. Comput. Syst. 42 (3) (2008) 349–365.
[6] M. Datar, A. Gionis, P. Indyk, R. Motwani, Maintaining stream statistics over sliding windows: (extended abstract), in: In: SODA '02: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002, pp. 635–644.
[7] G. Cormode, S. Tirthapura, B. Xu, Time-decaying sketches for sensor data aggregation, in: PODC '07: Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, ACM, New York, NY, USA, 2007, pp. 215–224.
[8] G. Cormode, F. Korn, S. Tirthapura, Time-decaying aggregates in out-of-order streams, in: PODS '08: Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems, ACM, New York, NY, USA, 2008, pp. 89–98.
[9] S. Muthukrishnan, Data streams: algorithms and applications, Found. Trends Theor. Comput. Sci. 1 (2) (2005).
[10] J. Misra, D. Gries, Finding repeated elements, Sci. Comput. Program. 2 (2) (1982) 143–152.
[11] E.D. Demaine, A. López-Ortiz, J.I. Munro, Frequency estimation of internet packet streams with limited space, in: ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms, Springer-Verlag, London, UK, 2002, pp. 348–360.
[12] R.M. Karp, S. Shenker, C.H. Papadimitriou, A simple algorithm for finding frequent elements in streams and bags, ACM Trans. Database Syst. 28 (1) (2003) 51–55.
[13] P. Bose, E. Kranakis, P. Morin, Y. Tang, Bounds for frequency estimation of packet streams, in: SIROCCO, 2003, pp. 33–42.
[14] G.S. Manku, R. Motwani, Approximate frequency counts over data streams, in: VLDB '02: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB Endowment, 2002, pp. 346–357.
[15] A. Arasu, G.S. Manku, Approximate counts and quantiles over sliding windows, in: PODS '04: Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, New York, NY, USA, 2004, pp. 286–296.
[16] L. Golab, D. DeHaan, E.D. Demaine, A. Lopez-Ortiz, J.I. Munro, Identifying frequent items in sliding windows over on-line packet streams, in: IMC '03: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, ACM, New York, NY, USA, 2003, pp. 173–178.
[17] L. Zhang, Y. Guan, Variance estimation over sliding windows, in: PODS '07: Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, New York, NY, USA, 2007, pp. 225–232.
[18] G. Cormode, S. Muthukrishnan, An improved data stream summary: the count-min sketch and its applications, J. Algorithms 55 (1) (2005) 58–75.
[19] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, in: ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming, Springer-Verlag, London, UK, 2002, pp. 693–703.
[20] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, in: ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming, Springer-Verlag, London, UK, 2002, pp. 693–703.

Update

Theoretical Computer Science

Volume 648, Issue , 4 October 2016, Page 116

DOI: https://doi.org/10.1016/j.tcs.2016.09.001

Theoretical Computer Science 648 (2016) 116

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Corrigendum

Corrigendum to "The frequent items problem, under polynomial decay, in the streaming model" [Theoret. Comput. Sci. 411(34–36) (2010) 3048–3054]

Guy Feigenblat *, Ofra Itzhaki, Ely Porat

Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel

The authors would like to add the following footnote in the paper:

This research was carried out as part of the PhD thesis of Guy Feigenblat under supervision of Professor Ely Porat from Bar-Ilan University.

DOI of original article: http://dx.doi.org/10.1016/j.tcs.2010.04.029.

* Corresponding author. Fax: +972 3 7384056.
E-mail address: feigeng@cs.biu.ac.il (G. Feigenblat).

http://dx.doi.org/10.1016/j.tcs.2016.09.001

Theoretical Computer Science 648 (2016) 116

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Corrigendum

Corrigendum to "The frequent items problem, under polynomial decay, in the streaming model" [Theoret. Comput. Sci. 411(34–36) (2010) 3048–3054]

Guy Feigenblat *, Ofra Itzhaki, Ely Porat

Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel

The authors would like to add the following footnote in the paper:

This research was carried out as part of the PhD thesis of Guy Feigenblat under supervision of Professor Ely Porat from Bar-Ilan University.

DOI of original article: http://dx.doi.org/10.1016/j.tcs.2010.04.029.
* Corresponding author. Fax: +972 3 7384056.
E-mail address: feigeng@cs.biu.ac.il (G. Feigenblat).

http://dx.doi.org/10.1016/j.tcs.2016.09.001