Estimating Rarity and Similarity over Data Stream Windows

Mayur Datar*[1] and S Muthukrishnan[2]

[1] Department of Computer Science, Stanford University, Stanford, CA 94305.
datar@cs.stanford.edu
[2] AT&T Research, Florham Park NJ, USA.
muthu@research.att.com

Abstract. In the *windowed data stream model*, we observe items coming in over time. At any time t, we consider the *window* of the last N observations $a_{t-(N-1)}, a_{t-(N-2)}, \ldots, a_t$, each $a_i \in \{1, \ldots, u\}$; we are required to support queries about the data in the window. A crucial restriction is that we are only allowed $o(N)$ (often polylogarithmic in N) storage space, so not all items within the window can be archived.

We study two basic problems in the windowed data stream model. The first is the estimation of the rarity of items in the window. Our second problem is one of estimating similarity between two data stream windows using the Jacard's coefficient. The problems of estimating rarity and similarity have many applications in mining massive data sets. We present novel, simple algorithms for estimating rarity and similarity on windowed data streams, accurate up to factor $1 \pm \epsilon$ using space only logarithmic in the window size.

1 Introduction

Consider a *stream* of data elements (possibly infinite) $\ldots, a_{t-2}, a_{t-1}, a_t, \ldots$, where a_t represents the data element observed at time t. In the *windowed data stream model*, at any time t we consider the *window* of the last N observations $a_{t-(N-1)}, a_{t-(N-2)}, \ldots, a_t$, each $a_i \in \{1, \ldots, u\}$. We are allowed to query this data for any function of interest – say, we wish to compute the minimum or the median of these items. Intuitively, in this model, we consider only a recent window within a continuing stream of observations, letting aged data disappear from the computation. The first crucial aspect of this model is that the data under consideration changes over time as new observations get added to the "right" and old observations to the "left" get deleted because the window "moves" past the old items. The second crucial aspect is that in scenarios where this model is applicable, storage is at premium and we can not store the entire window in memory. In particular, only $o(N)$ and $o(u)$ memory (often only memory polylogarithmic in N and u) may be used. Clearly, computing most functions *exactly* is impossible in this model[8], and hence, the focus is on approximating functions.

* This work was done while the author was a DIMACS visitor.

This *windowed data stream model* was formalized in a recent paper by Datar et al. [8].

While the above model may, at first glance, appear limiting, it however arises quite naturally in several applications. Data stream of observations are now fundamental to many data processing applications. For example, telecommunication network elements such as switches and Internet routers periodically generate records of their traffic — telephone calls, Internet packet and flow traces — which are data streams [15]. Atmospheric observations — weather measurements, lightning stroke data and satellite imagery — also produce multiple data streams [23]. Emerging sensor networks produce many streams of observations, e.g., highway traffic conditions [20]. Sources of data streams — large scale transactions, web clicks, ticker tape updates of stock quotes, toll booth observations — are ubiquitous in daily life. For more applications and a survey of the recent work in the databases and algorithms community in data streams, refer to Babcock et al. [3]. Two observations hold in practice:

- In many of these applications, despite the exponential growth in the capacity of storage devices, it is not common for such streams to be stored. Instead they must be processed "on the fly" as they are produced, because, the data is produced at a very high rate and it is challenging (if not impossible) to store and manipulate them offline.
- In many of these scenarios, data analysis often involves asking questions about recent time and the goal is to make decisions based on the statistics or models gathered over the "recently observed" data elements. For example: How many distinct customers made a call through a given switch in the past 24 hours? Thus we need to discard "stale data" while gathering statistics and building models.

These two observations naturally lead to the *windowed data stream model* that we focus on in this paper. We study two fundamental problems in this model. The first requires estimating a distributional parameter (rarity) in a windowed stream and the other involves comparing two windowed streams (similarity). Both these problems are novel in this context, and they are well motivated. We present the first known streaming algorithms for both these problems: our algorithms take time and space only logarithmic in N, and provide $1 \pm \epsilon$ approximation of rarity as well as similarity. Our results thus add to the growing body of data stream algorithms. In what follows, we will define the problem and describe our results in more detail.

1.1 Problems

We define the two problems below. Recall that at any time t, we consider the window $a_{t-N-1}, a_{t-N-2}, \ldots, a_t$ of N observations. This is a multiset of items from some Universe set $[u] = \{1, \ldots, u\}$.

1. *(Rarity)* We say an item $i \in a_{t-(N-1)}, a_{t-(N-2)}, \ldots, a_t$ is α-*rare* for integer α if i appears precisely α times, i.e., $a_j = i$ for precisely α indexes j, for $j \in \{t-$

$(N-1), \ldots, t\}$. Let $\#\alpha-rare$ denote the number of such items. Likewise, let $\#distinct$ denote the number of distinct items in $a_{t-(N-1)}, a_{t-(N-2)}, \ldots, a_t$. We define *$\alpha$-rarity* ρ_α as the ratio: $\boldsymbol{\rho_\alpha = \frac{\#\alpha-rare}{\#distinct}}$.
The problem is to estimate α-rarity at any time t. Thus, 1-rarity is the fraction of items that appear uniquely in the window. More generally, α-rarity is a measure of number of items that repeat exactly α times within the window.
Previous works have studied various distributional parameters like estimating the number of distinct items or the most frequent item etc. But our measure above is more general, being able to capture the "inverse" (how many items appear x items as opposed to how many times an item i appears) relation completely, for different values of α.

2. (*Similarity*) We study the classical Jacard coefficient of similarity between two streams a and b. Let X_t be the set of distinct items in a windowed data stream $a_{t-(N-1)}, a_{t-(N-2)}, \ldots, a_t$, and let Y_t be that in windowed data stream $b_{t-(N-1)}, b_{t-(N-2)}, \ldots, b_t$. The Jacard similarity between these two windowed data streams at time t is defined as $\boldsymbol{\sigma_t = \frac{|X_t \bigcap Y_t|}{|X_t \bigcup Y_t|}}$.
The problem is to estimate the similarity σ_t at any time t.
This is the classical notion of similarity between two sets. It has been useful in estimating transitive closures [6], web page duplicate detection [2] and data mining [7], among other things.

Rarity ρ finds many data mining applications. For example, consider the data stream of IP-addresses that access any online store like Amazon. The set of rare IP-address (for the appropriate α) in a reasonable time window denotes the set of users who accessed the store website, but did not explore it in depth, for perhaps they were unhappy with their customer experience. A different, somewhat unusual application of rarity is as an indicator of a potential Denial of Service attack. Consider the data stream of IP *flows* (loosely, contiguous packets that comprise a session) that are sent to or received from a particular subnet (range of IP addresses). If the number of flows with a single packet (only the first SYN packet) is large it indicates that there is a Denial of Service attack on one or more of the servers in the subnet.

Similarly, σ too has many data mining applications. For example, let X_t (Y_t) denote the set of distinct IP-addresses that access a particular news website A (B, respectively). Estimating the similarity between these two sets in a short time window indicates if a lot of the readers of the two newspapers look to both for their news coverage. This example can be extended to any other online service.

An important observation about both ρ and σ is that in most applications, we are not interested in calculating them precisely but rather interested in only checking if their values are significantly large. In particular, we are interested in estimating them accurately provided they exceed a certain threshold value. Thus we will assume that ρ and σ need only be estimated accurately to some additive, prespecified precision p which is a fixed fraction. Another observation is that in applications where ρ_α finds use, α is often small, so we will focus on the case of α being some constant, for most part.

1.2 Our Results

No results are known in windowed stream model for estimating either ρ or σ. We present first known results for these problems. Our results are as follows.

We present an algorithm that uses only $O(\log N + \log u)$ space, and per-item processing time $O(\log \log N)$ and estimates ρ by $\hat{\rho}$ such that $\hat{\rho} \in [1 \pm \epsilon]\rho + \epsilon p$, for any given fraction ϵ, with high probability. For estimating σ we get an identical result.

Notice that if one additionally wished to estimate just $\#\alpha - rare$ or $|X_t \bigcap Y_t|$ (numerators of ρ and σ respectively), then we can use known results for estimating the number of distinct elements ($\#distinct$) [8, 12] and union size of streams ($|X_t \bigcup Y_t|$) [13] in addition to our estimates of $\hat{\rho}$, and $\hat{\sigma}$ and get $1 \pm \epsilon$ approximation for them as well, up to relative precision p.

1.3 Related Work

There has been very little work on algorithms for the windowed data stream model. This model was defined and studied in [8] where the authors presented algorithms for keeping various aggregates (the number of 1's, sums etc) as well as the L_p norm of the window. In [5], the authors studied the problem of maintaining a uniform random sample of the items within the current window. These results do not give any bound for estimating rarity or similarity.

A related data stream model is one in which the window size is unbounded. In other words, N is the total number of items we have seen so far, from the beginning of the time. So there is no notion of recency, and the entire past is considered in estimating distributions as well as similarity. We call this the *unbounded data stream model.* Substantial amount of work has been done in this model. This includes computing L_1, L_2 norms [1, 11, 18], computing similarity [7][3], clustering [16], maintaining decision trees [9], maintaining histograms and other synopsis structures [15, 14] etc. For a survey of data stream algorithms, refer to [3]. Our windowed data stream model proves to be more challenging than the unbounded data stream model. In particular, consider the problem of exactly calculating the minimum of the items on the data stream. In the unbounded data stream model, we can maintain the current minimum as new data items arrive, trivially. In contrast, this is provably not possible in the windowed data stream model (see [8]).

There is another data stream model which is based on transactions, wherein items may be inserted or deleted arbitrarily, and we need to estimate functions in presence of these updates. This is called the *cash register model* in [15]. Here, a delete operation is explicitly specified as an input; in contrast, in the windowed

[3] While the algorithm is not described as a streaming algorithm, it applies to the unbounded streaming model but not for windowed model

model, the delete corresponding to the element that "falls off" of the window is implicit. Unless we buffer the entire window we have no means to explicitly figure out, at every instant, the element that is deleted in the windowed stream model.

1.4 Overview of Techniques

First consider the problem of estimating ρ. For now focus on the unbounded data stream model. One may be tempted to randomly sample each item, but we need to coordinate the sampling of an item when it applies multiple times, and this calls for sampling from the universe, rather than the individual items in the data stream.[4] We use min-wise hashing, which was introduced in [6, 2], to sample from the universe, and are able to derive an unbiased estimator for rarity. Min-wise hashing, per se, is expensive to compute, but suitable approximations can be generated [19, 21] and we show that these suffice to solve our problem.

Now let us consider the additional complication in the windowed model. Here the crux of the problem can be abstracted as follows. For each item in the data stream, we compute a hash function and we need to maintain the minimum of the hash values. Maintaining the minimum of a data stream, as we noted above, is easy for the unbounded data stream model, but impossible in the windowed model using limited (sublinear) memory. However, we use the observation that the hash values are nearly random and therefore we can keep the entire "chain" of consecutive minimums amongst these hash values in the time increasing order within our space bounds, and obtain our result. The same approach works for σ estimation as well.

We begin with preliminaries in the next section, moving on to prove our main results in the subsequent sections.

2 Preliminaries

A central tool to all our algorithms is the concept of min-wise hashing (also called min-hashing). Let π be a randomly chosen permutation over $[u] = \{1, \ldots, u\}$. For a subset $A \subseteq [u]$ the *min-hash* of A for the given permutation π ($h_\pi(A)$) is defined as : $h_\pi(A) = min_{i \in A} \pi(i)$.

Thus the min-hash of any subset is the minimum element of that subset after we have applied the given permutation to its elements. The crucial (simple to prove) property of min-hash functions is: For any pair of subsets $A, B \subseteq [u]$,

$$Pr[h_\pi(A) = h_\pi(B)] = \frac{|A \bigcap B|}{|A \bigcup B|},$$

where the probability above is defined over the random choice of the permutation π. Applications of min-hashing use this property. The following lemma, that is based on this property, is taken from [7]:

[4] It is quite easy to see using techniques in [22] that random sampling directly will not work.

Lemma 1. *Let $h_1(A), h_2(A), \ldots, h_k(A)$ $(h_1(B), h_2(B), \ldots, h_k(B))$ be k independent min-hash values for the set A (B, respectively). Let $\hat{S}(A,B)$ be the fraction of the min-hash values that they agree on, i.e.,*

$$\hat{S}(A,B) = |\{l | 1 \leq l \leq k, h_l(A) = h_l(B)\}|/k.$$

For $0 < \epsilon < 1$, $1 > p > 0$, $k \geq 2\epsilon^{-3}p^{-1}\log \tau^{-1}$,

$$\hat{S}(A,B) \in (1 \pm \epsilon)\frac{|A \bigcap B|}{|A \bigcup B|} + \epsilon p$$

for a prespecified precision p, with success probability at least $1 - \tau$.

The ideal family of min-hash functions is defined by the set of all permutations over $[u]$. However it requires $O(u \log u)$ bits to represent any random permutation from this family; hence, they are not useful for data stream applications. In [21, 19], a family of *approximate* min-hash functions is presented. A family of hash functions, $\mathcal{H} \subset [n] \to [n]$ (where $[n] = \{1, \ldots, n\}$), is called $\epsilon' - min - wise\ independent$ if for any $X \subset [n]$ and $x \in [n] - X$ we have

$$Pr_{h \in \mathcal{H}}[h(x) < min\ h(X)] = \frac{1}{|X|+1}(1 \pm \epsilon')$$

In particular, [19] presents a family of $\epsilon' - min - wise\ independent$ hash functions such that any function from this family can be represented using $O(\log u \log(1/\epsilon'))$ bits only and each hash function can be computed efficiently in $O(\log(1/\epsilon'))$ time; in addition, any hash function h chosen randomly from this family satisfies

$$Pr[h(A) = h(B)] = \frac{|A \bigcap B|}{|A \bigcup B|} \pm \epsilon'$$

In fact, the family proposed in [19] is the set of polynomials over $GF(u)$[5] of degree $O(\log(1/\epsilon'))$. It is now straightforward to observe that Lemma 1 still holds true if we used ϵ'-min-hash functions instead of min-hash functions; only the value of k will be appropriately adjusted in terms of ϵ' and the additive error ϵp changes to $\epsilon p + \epsilon'$. The advantage is that we only use $O(\log u \log(1/\epsilon'))$ bits for storing each hash function and $O(k)$ hashes are needed for the entire estimation which is very efficient in terms of space.

Our results that follow will crucially use min-hashing. We will describe all our results in terms of "perfect" min-hash functions. But it will be implicit that our results can be made to work with ϵ'-min-hash functions with small change in the number of hash functions needed, instead of the perfect min-hash functions; hence, the space bound will be only $O(\log u \log(1/\epsilon'))$ for generating the required hashes [21, 19]. With this implicit understanding, we will work henceforth with perfect min-wise hashing and as a result, the presentation should be simpler.

---

[5] We assume that u is a prime. If not we choose the next biggest prime.

3 Estimating Rarity in Unbounded Data Stream Model

In what follows we will present an algorithm for estimating $\rho_\alpha(t)$ up to precision p, at any time instant t, in the unbounded data stream model.

Consider a multiset S. Let $mult_S(i)$ denote the multiplicity of an element i in the multiset S. Also let D denote the set of distinct elements in the multiset S. For any integer α we can represent D as a union of two disjoint subsets, $D = R_\alpha \bigcup \overline{R_\alpha}$ where $R_\alpha = \{i | i \in D, mult_S(i) = \alpha\}$ and $\overline{R_\alpha} = D - R_\alpha$. In other words R_α is the set of *α-rare* elements in S. Our aim is to accurately estimate the ratio $\rho_\alpha = |R_\alpha|/|D|$.

We make the following simple observations: Since $R_\alpha \subseteq D$, $\frac{|R_\alpha \bigcap D|}{|R_\alpha \bigcup D|} = \frac{|R_\alpha|}{|D|}$. Using the property of min-hash functions we have $Pr[h(R_\alpha) = h(D)] = \rho_\alpha$. Thus one idea would be to maintain the min-hash values for R_α and D and estimate this ratio. Moreover, $h(R_\alpha) = h(D)$ if and only if the element corresponding to the min-value (after hashing) in D belongs to R_α. This observation precludes the need to maintain a separate min-hash value for both R_α and D. We only need to maintain a min-hash value for D and check if the count of the element corresponding to min-value is no more than α.

Based on these observations we get the following simple algorithm: We choose k min-hash functions $h_1, h_2, \ldots, h_k$ for some parameter k to be determined later. We maintain $h_i^*(t) = min_{a_j,\, j \le t} h_i(a_j)$ at each time t, that is, min-hash value of the set $\{\ldots, a_{t-2}, a_{t-1}, a_t\}$. We also maintain k counters $(c_1(t), c_2(t), \ldots, c_k(t))$, one counter corresponding to each min-hash value $h_i^*(t)$. The min-hash values are all initialized to ∞ and counters are all initialized to zero, at $t = 0$. When the new element a_{t+1} is seen, we update these values as follows. For each i, we compute $h_i(a_{t+1})$. If $h_i(a_{t+1})$ is strictly smaller than $h_i^*(t)$, we update the new minimum to be $h_i^*(t+1) = h_i(a_{t+1})$ and set the counter $c_i(t+1)$ to one. If $h_i(a_{t+1}) = h_i^*(t)$, we increment the counter $c_i(t)$ by one to get $c_i(t+1)$. Else we do no further processing for element a_{t+1}, $h_i^*(t+1) = h_i^*$ and $c_i(t+1) = c_i(t)$, and consider the next item in the data stream. We have the following claim whose proof follows trivially from the preceding discussion:

Lemma 2. *Let $\hat{\rho}_\alpha(t)$ be the fraction of the counters $c_i(t)$ that equal α, i.e.,*

$$\hat{\rho}_\alpha(t) = |\{l | 1 \le l \le k, c_i(t) = \alpha\}|/k.$$

For $0 < \epsilon < 1$, $1 > p > 0$, $k \ge 2\epsilon^{-3}p^{-1}\log\tau^{-1}$,

$$\hat{\rho}_\alpha(t) \in (1 \pm \epsilon)\rho_\alpha(t) + \epsilon p$$

for a prespecified precision p, with success probability at least $1 - \tau$.

The remainder of the algorithm trivially follows from the claim. We merely return estimate $\hat{\rho_\alpha}(t)$ at time t. The space used is $O(k)$ for the hash values as well as the counters. In addition, we need to store $O(k)$ seeds for the ϵ'-min-hash functions h_i each of which takes $O(\log u \log(1/\epsilon'))$ bits as discussed earlier. Processing each item takes time to update the k hash values and counters which takes time $O(k \log(1/\epsilon'))$ using results in [19]. The appropriate value of k can be determined in terms of ϵ' using the preceding Lemma, but for setting all parameters to be desired constants, $k = O(1)$. That gives the claimed result for estimating $\rho_\alpha(t)$ at any time t, for the unbounded data stream model.

Some additional observations are as follows: Keeping exact counters $c_i(t)$, as described above has the nice property that α need not be fixed apriori. Instead, it can be specified by the user at query time and changed for different queries. In addition, our technique above can work for estimating the size (ratio of sizes) of any "special" subset of the distinct elements provided we can efficiently check if the element corresponding to the min-value belongs to this subset [6]. Moreover, our algorithm is well suited for a distributed scenario. Say we saw a stream A at one site and another stream B at another site and wished to consider the stream $C = A||B$ where $||$ stands for concatenation of the stream. Then, it is easy to see that if we maintain the min-hashes $(h_i^*(A),\, h_i^*(B))$ along with the counters $(c_i(A),\, c_i(B))$ separately at the two sites, we can compute the min-hashes and counters for the concatenated (union) stream. The amount of communication required (and the time required for computing) is the size of the min-hashes plus counters, which as we saw is $O(k) = O(1)$.

4 Similarity Estimation

We now consider the similarity estimation problem. Recall from Section 2 that to estimate similarity between two sets X and Y, we need to only compare k min-hash values for each. In the unbounded data stream model, it is trivial to maintain the k min-hash values as new items arrive. However, the challenge for the windowed data stream model, is to maintain $h_i^*(t)$'s for the window $t-(N-1), \ldots, t$ as t progresses. Consider the task of maintaining any single $h_i^*(t)$. At any time t, we would like to maintain the minimum value over the window $t-(N-1), \ldots, t$ for the "hashed" stream $h_i(a_j)$. An item a_j is *active* at time t iff $j \in \{t-(N-1), \ldots, t\}$. At time t, consider two active data items d_1 and d_2 with arrival times t_1, t_2 such that $t_1 < t_2$. If it is the case that $h_i(d_1) \geq h_i(d_2)$ then we say that item d_2 *dominates* item d_1. At any time t, we need not store the hash value for d_1 if there exists such a dominating element d_2. This is because the hash value of d_2 is no greater than that of d_1 and for all times in the future whenever d_1 is active, d_2 will also be active since $t_1 < t_2$. Thus $h_i^*(t)$ is not affected by the hash value for d_1 which can be safely discarded.

Based on the observation above, we propose the use of the following linked list $\mathcal{L}_i(t)$. Every element of $\mathcal{L}_i(t)$ consists of the pair $(h_i(a_j), j)$ for some active data item a_j at time t. Thus $\mathcal{L}_i(t)$ looks like

$$\{(h_i(a_{j_1}), j_1), (h_i(a_{j_2}), j_2), \ldots, (h_i(a_{j_l}), j_l)\},$$

[6] As a special case, the same algorithm can be used, even if we slightly change the definition of rarity to include all elements with multiplicity less than α (and at least 1).

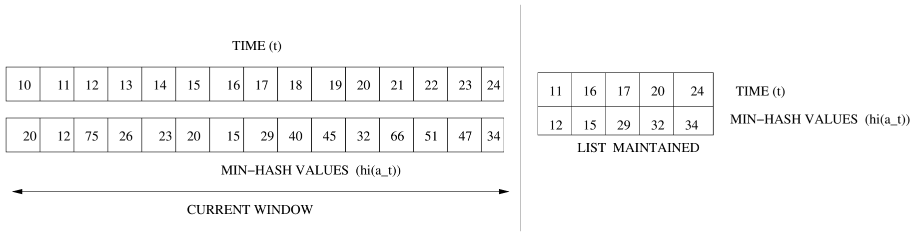

Fig. 1. An example of linked list maintained by the algorithm.

where each j_m is the arrival time of active item a_{j_m}. Please refer to Figure 1 for an example. The list $\mathcal{L}_i(t)$ satisfies the property that both the hash values and the arrival times are strictly increasing in the list from left to right. In other words,

$$j_1 < j_2 < \ldots < j_l \;\;\&\;\; h_i(a_{j_1}) < h_i(a_{j_2}) < \ldots < h_i(a_{j_l}).$$

Then clearly $h_i^*(t) = h_i(a_{j_1})$.

The algorithm to maintain list $\mathcal{L}_i(t)$ is simple. When we see item a_{t+1}, we compute its hash value $h_i(a_{t+1})$. We find the largest β such that $h_i(a_{j_\beta}) < h_i(a_{t+1})$; here $h_i(a_{j_\beta})$ is from the list $\mathcal{L}_i(t)$. We perform the following steps:

1. We delete all $(h_i(a_{j_k}), j_k)$ from $\mathcal{L}_i(t)$ for which $k > \beta$. Now, $(h_i(a_{j_\beta}), j_\beta)$ is the rightmost item in the list $\mathcal{L}_i(t)$.
2. We insert $(h_i(a_{t+1}), t+1)$ after $(h_i(a_{j_\beta}), j_\beta)$ in list $\mathcal{L}_i(t)$. Hence, $(h_i(a_{t+1}), t+1)$ is now the rightmost element in the list.
3. If the leftmost pair $(h_i(a_{j_1}), j_1)$ is not active at time $t+1$, i.e., $j_1 \notin \{t+1-(N-1), \ldots, t+1\}$, then the leftmost item is deleted, and the second item from left on the list becomes the leftmost.

The list thus obtained is now $\mathcal{L}_i(t+1)$. (We have omitted the description of the boundary cases since they are obvious.)

The difficulty with the algorithm we have described above is that we are explicitly storing $\mathcal{L}_i(t)$ at any time t, and hence, the space used is its length, $|\mathcal{L}_i(t)|$. However,

Lemma 3. *With high probability, over the random choice of min-hash function h_i, we have $|\mathcal{L}_i(t)| = \Theta(H_N)$, where H_N is the Nth Harmonic number, given by $1 + 1/2 + 1/3 + \cdots + 1/N = \Theta(\log N)$.*

Here is a simple, high level argument for proving the lemma above. Consider building a treap on the last N elements where the arrival times are fully ordered (in-order traversal gives increasing arrival times) and hash values ($h_i(a_j)'s$) are heap ordered. Then the list $\mathcal{L}_i(t)$ is the *right spine* of this treap. If the hash values are fully independent, i.e. every hash function is a random permutation, then it is well known (refer [22])that the expected length of the right spine is H_N. In fact, it was proved in [24], the length is $\Theta(\log N)$ with very high probability. However, the hash function families that we propose using are not fully independent, instead have limited independence. In fact, as seen earlier, we propose using a random polynomial of constant degree ($O(\log 1/\epsilon')$) which is $\epsilon' - min - wise\ independent$([19]). It was proved in [24] that for a hash function that is chosen randomly from such a family, the expected length of the right spine and hence that of our list $\mathcal{L}_i(t)$, is $\Theta(\log N)$. From the preceding discussions (using binary search for finding β with an appropriate data structure), we can conclude

Theorem 1. *There is an algorithm for estimating similarity σ between any two windowed data streams, with pre-specified precision p and up to $1 \pm \epsilon$ factor with high probability; the algorithm uses $O(\log N + \log u)$ words of space (or $O((\log N)(\log u))$ bits of space) and takes time $O(\log \log N)$ per data item, with high probability.*

Note that the success probability in the theorem above is predicated on the random choice of the min-hash functions, and *not* over the distribution of the input. In other words, the theorem above holds for an arbitrary (worst case) input.

5 Estimating Rarity Over Windowed Data Streams

In this section, we return to the problem of estimating α-rarity, but now we work in the windowed data stream model. Recall that the algorithm for estimating α-rarity in the unbounded data stream model required us to maintain min-hash and counter pairs for k independent min-hash functions. In the windowed data stream model, we follow the same overall algorithm, but modify it as follows.

The main idea here is simple: we maintain for each min-hash value, several (not merely latest) arrival times of items which map to that min-hash value. In particular, we maintain a list $\mathcal{L}_i^t$ of pairs $(h_i(a_{j_k}), list_{i,j_k}^t)$. Here, $list_{i,j_k}^t$ is an ordered list of the latest $\alpha + 1$ time instances ℓ such that $h_i(a_\ell) = h_i(a_{j_k})$; "latest" refers to largest indexes ℓ. (If there are fewer than α such ℓ, we store them all.) Thus $\mathcal{L}_i^t$ now comprises $\{(h_i(a_{j_1}), list_{i,j_1}^t), (h_i(a_{j_2}), list_{i,j_2}^t), \ldots, (h_i(a_{j_l}), list_{i,j_l}^t)\}$. A property of $list$'s worth noting: if we concatenate $list_{i,j_k}^t$'s in the increasing order of j_k (i.e., in the order they appear in $\mathcal{L}_i^t$), then the elements will be strictly increasing. This data structure differs from the one in Section 3 in that here we store $list_{i,j_k}^t$'s, while previously, we only stored a single value for each $h_i(a_{j_1})$, which one can observe will be the largest index in $list_{i,j_k}^t$'s.

When the next item in the data stream, i.e., a_{t+1}, is read, we need to update the lists. This update procedure is similar to the one in Section 4, but there are additional cases to consider. When we see item a_{t+1}, We compute its hash value $h_i(a_{t+1})$. We find the largest β such that $h_i(a_{j_\beta}) \leq h_i(a_{t+1})$, for $h_i(a_{j_\beta})$ in $\mathcal{L}_i(t)$. We perform the following steps:

1. We delete all $(h_i(a_{j_k}), list_{i,j_k}^t)$ pairs from $\mathcal{L}_i(t)$ for which $k > \beta$. Now, $(h_i(a_{j_\beta}), list_{i,j_\beta}^t)$ is the rightmost item in the list $\mathcal{L}_i(t)$.

2. If $h_i(a_{j_\beta}) \neq h_i(a_{t+1})$, we insert $(h_i(a_{t+1}), list_{i,t+1})$ after $(h_i(a_{j_\beta}), list^t_{i,j_\beta})$ in list $\mathcal{L}_i(t)$. Here, $list_{i,t+1}$ consists of the singleton item $t+1$. Hence, $(h_i(a_{t+1}), list_{i,t+1})$ is now the rightmost element in the list $\mathcal{L}_i(t)$.
3. If $h_i(a_{j_\beta}) = h_i(a_{t+1})$, we append $t+1$ to $list^t_{i,j_\beta}$; if the resulting list has more than $\alpha + 1$ items, we delete the smallest index in it.
4. Let x be the smallest index in $list^t_{i,j_1}$; if a_x is not active at time $t+1$, we remove x from $list^t_{i,j_1}$. As a result if $list^t_{i,j_1}$ becomes empty, we remove the pair $(h_i(a_{j_1}), list^t_{i,j_1})$ from $\mathcal{L}_i(t)$.

The list $\mathcal{L}_i(t)$ that remains after the operations above is the list $\mathcal{L}_i(t+1)$. (Again, as in Section 4, the boundary cases have been omitted.) We can argue as in Section 4 that the total size of $\mathcal{L}_i(t)$ is $O(\alpha \log N)$ with high probability, which is $O(\log N)$ for α's of our interest (small constants). Again, as before, we can conclude that this algorithm yields an $1 \pm \epsilon$ approximation, up to precision p, in bounds identical to Theorem 1 for the problem of estimating α-rarity in windowed data streams, giving our result.

Note that in the preceding discussion we assumed that α is a small constant; this is the case that is applicable in the mining scenarios we described in Section 1. However, if one really wishes to solve the problem of estimating ρ_α, for arbitrarily large α, over windowed data streams, we can reduce the memory requirement as follows: Instead of maintaining an ordered list of the latest α time instances we maintain the data structure from [8] for each min-hash value. This data structure can be used to approximately count the number of instances of the hash value in the current window. It uses space $\frac{1}{\tau} \log \alpha$ and maintains a count up to α with accuracy $1 \pm \tau$.

6 Concluding Remarks

Much remains to be done in the windowed data stream model. Our technique of keeping the min-hash value together with a count or list of occurrences has been recently applied to a quite different problem of mining the database structure and building data quality browser [10]. We expect our techniques to find other applications in practice.

References

1. N. Alon, Y. Matias, M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. Twenty-Eighth Annual ACM Symp. on Theory of Computing*, 1996.
2. A. Broder. Filtering Near-Duplicate Documents. In *Proc. of FUN*, 1998.
3. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proc. of Principles of Database Systems,* Madison, Wisconsin, June 3 – 5, 2002.
4. A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise Independent Permutations. In *Proc. of STOC*, 1998.

5. B. Babcock, M. Datar, and R. Motwani. Sampling from a Moving Window over Streaming Data. In *Proc. of Thirteenth Annual ACM-SIAM Symp. on Discrete Algorithms,* 2002
6. E. Cohen. Size-Estimation Framework with Applications to Transitive Closure and Reachability. *Journal of Computer and System Sciences* 55 (1997): 441–453.
7. E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang. Finding Interesting Associations without Support Pruning. In *Proc. of the 16th International Conference on Data Engineering, San Diego, USA, 2000.*
8. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. In *Proc. of Thirteenth Annual ACM-SIAM Symp. on Discrete Algorithms.*
9. P. Domingos, G. Hulten, and L. Spencer. Mining time-changing data streams. In *Proc. of the 7th International Conference on Knowledge Discovery and Data Mining*, 2001.
10. T. Dasu, T. Johnson, S. Muthukrishnan and V. Shkapenyuk Mining database structure, or to How to build a data quality browser. In *Proc. of the SIGMOD*, 2002.
11. J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan. An Approximate L1-Difference Algorithm for Massive Data Streams. In *Proc. 40th IEEE Symp. on Foundations of Computer Science*, 1999.
12. P. Flajolet, G. Martin. Probabilistic Counting. In *Proc. 24th IEEE Symp. on Foundations of Computer Science*, 1983.
13. P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Symp. on Parallel Algorithms and Architectures*, 2001.
14. A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, Small-Space Algorithms for Approximate Histogram Maintenance. In *ACM Symp on Theory of Computing (STOC)*, 2002.
15. A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. of VLDB*, 2001.
16. S. Guha, N. Mishra, R. Motwani, L. O'Callaghan. Clustering data streams. In *Proc. 2000 Annual IEEE Symp. on Foundations of Computer Science*, pages 359–366, 2000.
17. M. R. Henzinger, P. Raghavan, S. Rajagopalan. Computing on data streams. Technical Report TR 1998-011, Compaq Systems Research Center, Palo Alto, California, May 1998.
18. P. Indyk. Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, 2000.
19. P. Indyk A Small Approximately Min-Wise Independent Family of Hash Functions. In *Journal of Algorithms 38(1): 84-90 (2001).*
20. S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of ICDE*, 2002.
21. K. Mulmuley An Introduction through Randomized Algorithms. Prentice Hall, 1993.
22. R. Motwani, P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
23. NOAA. U.S. national weather service. `http://www.nws.noaa.gov/` .
24. R. Seidel, and C. Aragon. Randomized Search Trees. In *Algorithmica (1996) 16*, pp 464–497.