DIMACS Technical Report 2002-21
May 2002

Estimating Rarity and Similarity over Data Stream Windows.

by

Mayur Datar[1] S Muthukrishnan[2]

[1]Department of Computer Science, Stanford University, Stanford, CA 94305. Email:datar@cs.stanford.edu. This work was done while the author was visiting AT&T Research Labs on a DIMACS sponsored visit.
[2]AT&T Research, Florham Park NJ, USA. Email:muthu@research.att.com.

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore).

DIMACS was founded as an NSF Science and Technology Center, and also receives support from the New Jersey Commission on Science and Technology.

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 allowed to ask queries about the data in the window, say, we wish to compute the minimum or the median of the items in the window. A crucial restriction is that we are only allowed $o(N)$ (often polylogarithmic in N) storage space, that is, space smaller than the window size, so the items within the window can not be archived. Window data stream model arose out of the need to formally reason about the underlying data analyses problems in applications like inter-networking and transactions processing.

In this paper, we study two basic problems in the windowed data stream model. The first is the estimation of the rarity of items in the window. While previous work has studied simple distributional parameters such as the number of distinct items in the window, no prior work has addressed the general problem of estimating the rarity. Our second problem is one of estimating similarity between two data stream windows using the Jacard's coefficient. Prior work has focused on L_p norms and set similarity measures such as the Jacard's coefficient have been studied before in the windowed data stream model. The problems of estimating rarity and similarity have many applications in mining massive data.

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. In both cases, our solutions are based on modifications of the powerful min-wise hashing technique. We expect our solutions to find applications in practice.

1 Introduction

Consider a *stream* of data elements (possibly infinite) indexed by time $\ldots, a_{t-2}, a_{t-1}, a_t, \ldots$, 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[1], and hence, the focus is on approximating functions.

The windowed data stream model may, at first glance, appear limiting. However, it 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 [28]. Atmospheric observations — weather measurements, lightning stroke data and satellite imagery — also produce multiple data streams [43]. Emerging sensor networks produce many streams of observations, e.g., highway traffic conditions [40]. Sources of data streams — large scale transactions, web clicks, ticker tape updates of stock quotes, toll booth observations — are ubiquitous in daily life. Two observations hold in practice:

- 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? What was the most number of times a web server was accessed by a single client in the past 1 hour? What was the largest difference in the speed between any two cars that crossed an intersection in the past 10 minutes? Etc.

- 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. For example, in sensors [40], resources are heavily constrained and often they have small memory, only intermittent connectivity, and can admit little time for processing per data item seen. As another example, consider high-speed, backbone Internet routers that route several Gbit/s and therefore process 10's of

[1]For example, consider computing the minimum of the items in the window $a_{t-(N-1)}, a_{t-(N-2)}, \ldots, a_t$. Say we store the minimum $\min_{i=0}^{N-1} a_{t-i}$ in the current window. As a new item a_{t+1} is appended to the right, we can easily update the minimum of $N+1$ items $\max_{i=0}^{N} a_{t+1-i}$ but we need to delete the effect of $a_{t-(N-1)}$. We need to know $a_{t-(N-1)}$ at this step, and inductively all of a_{t-i} as new items get appended to the right. In other words, we need to be able to store each item in the window exactly to be able to compute the minimum value exactly at each step, which is impossible for the worst case input since there exist inputs that can not be compressed at all. Please refer to [15] for a mathematically formal version of the same argument.

millions of packets per second on average. Storing the log of these packets locally even for an hour or two will call for several Mbytes of fast memory which is expensive, and alternatively, moving it to central warehouse would consume a sizeable portion of the bandwidth of the network itself [22, 16]. (Even if the data is all archived, it may not be desirable since the cost of any simple processing – even just sorting the data – would be too great.) Hence, it is typical to analyze the data "on the fly" while the logs are generated, with a small processing time per element, and only a summary or synopsis is stored at the routers.

In applied areas of research — networking and databases, for example — the phenomena of data streams, and windowed queries has been discussed extensively in the past few years. There has been a growing need to develop a framework for reasoning about algorithms in this context. Very recently, the *windowed data stream model* was formalized in the algorithms community [15] as a framework for designing algorithms with guaranteed performance bounds, addressing this need.

In this paper, we study two fundamental problems in the windowed data stream 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, i.e., our algorithms take time and space only logarithmic in N and provide $1+\epsilon$ 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 study 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 α-*rarity* ρ_α as the ratio:

$$\rho_\alpha = \frac{\#\alpha - rare}{\#distinct}.$$

   The problem is to estimate α-rarity at any time t.

   1-rarity is the fraction of items that appear uniquely in the window, and thus is the measure of the items not repeating within the window. More generally α-rarity is a measure of number of items that repeat 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 $a_{t-(N-1)}, a_{t-(N-2)}, \ldots, a_t$, and let Y_t be that in $b_{t-(N-1)}, b_{t-(N-2)}, \ldots, b_t$. The Jacard similarity between these two windowed data streams at time t is defined as

$$\sigma_t = \frac{|X_t \bigcap Y_t|}{|X_t \bigcup Y_t|}.$$

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 [11], web page duplicate detection [4] and data mining [12, 35], among other things.

Rarity ρ finds many data mining applications. For example, consider the data stream of IP-addresses that access any online service like a search engine, online-store like Amazon, online-newspapers etc. The set of rare IP-address (for the appropriate value of α) in a reasonable time window denotes the set of users who accessed the service and did not explore the service in depth, perhaps because they were not happy with their customer experience. Estimating the number of such unhappy customers is useful for any service provider. A 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). A typical flow consists of a lot of packets, in particular at least more than one, since the first packet is a SYN packet that establishes the flow. However if we observe that the number of flows with a single packet (rare flows) is large in a small time window in the recent past, then it indicates that either lot of flows are being dropped (do not receive ACK) in the subnet or that there is a Denial of Service attack on one or more of the servers in the subnet. See [26] for more on this application.

Similarly, σ too has many data mining applications. Consider the data stream generated by purchases at a super market. For any two items A and B, let X_t and Y_t respectively denote the set of distinct market-baskets that these two items belong to. The similarity between X_t and Y_t within a short time window, denotes the correlation between these two data items. For example, in the context of data mining, the sets corresponding to the distinct market baskets that contain items bread and butter, would have similarity very close to 1. As another example, let X_t (Y_t) denote the set of distinct IP-addresses that access a particular news website A(B). 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 typically use them to only check if they are significant. In particular, we are interested in estimating them accurately provided they exceed a certain threshold value and hence it suffices if we get their value to some fixed fraction accuracy, say one in 100 or one in 1000. In other words, 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 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.[2] Thus, we obtain an $1 + \epsilon$ approximation since ρ needs only to be estimated up to precision p.

For estimating σ we get an identical approximation guarantee, the space used is $O(\log N + \log u)$ and the per-item processing time is $O(\log \log N)$.

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 ($\#distinct$) [15, 23] and union size of streams ($|X_t \bigcup Y_t|$) [25] 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 [15] 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 [7], 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 on this model. Some of the early papers on data streams have looked at the problem of estimating the L_1 norm [2, 21] and L_2 norm (Euclidean norm) [37] over large vectors. [12] considers the problem of computing the similarity (as measured by the Jacard coefficient) between various items in a market-basket data set. There techniques directly work in this unbounded streaming model (but not for our windowed data streaming model). There has been work on the problem of one pass clustering of data streams [34]. [17, 18] studied the problem of maintaining decision trees on data streams. There has been lot of work [32, 33, 27, 28, 29, 31] on maintaining histograms and other synopsis structures like top wavelet coefficients over data streams. For a survey of the recent work in data stream algorithms, refer to [5]. 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 as we argued earlier.

[2] Here, with high probability means with probability exceeding any given constant fraction. We can repeat this logarithmic times and drive down the failure probability to inverse polynomial in N if needed. This detail has been omitted in the rest of the paper.

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 [28]. Our windowed data stream model again differs substantially from the cash register model. When the window moves off an item, one may think of it as a "delete" operation and hence one may be tempted to think of the windowed data stream model as special case of the cash register model. However, there is a subtlety. In the cash register model, a delete operation explicitly specifies the item to be deleted; in the windowed data stream model, the item to be deleted $a_{t-(N-1)}$ cannot be stored for all possible shifts because of our space constraint, and hence, the delete is implicit! This complicates algorithm design in our model.

1.4 Overview of Techniques

First consider the problem of estimating ρ. For now focus on the unbounded data stream model where we do not face the complication of deleting an item when the window passes over it. 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.[3] We use min-wise hashing, which was introduced in [11, 4] 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 [38, 41] and we show that these suffice to solve our problem.

Now let us consider the additional complication in the windows data stream 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, which as we noted above is easy for the unbounded data stream model, but impossible in the windowed data stream model (using limited memory) when items (and their hash values) get deleted. It is important to note that the lower bound for the windowed data stream model applies to worst case pathological data distributions. 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.

1.5 Map

In Section 3, we show how to estimate rarity in the unbounded data stream model. In Section 4, we show how to estimate similarity in the windowed data stream model. Finally, in Section 5, we show how to estimate rarity on a windowed data stream model. Concluding remarks are in Section 6.

2 Preliminaries

In this section, we will review the tools we would later use. In particular, we need the concept of min-wise hashing (also called min-hashing).

[3] It is quite easy to see using techniques in [42] that random sampling directly will not work.

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 after permuting the elements of that subset. 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. For example, it follows from [12] that

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 [41, 38], a family of *approximate* min-hash functions is presented. A family of hash functions, $\mathcal{H} \subset [n] \rightarrow [n]$ (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, they present 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 (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

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

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 the hash function and $O(k)$ hashes in addition 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. As such they would need a large amount of space to store these 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 [41, 38]. With this implicit understanding, we will work henceforth with perfect min-wise hashing and as a result, the presentation should be simpler.

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:

1. Since $R_\alpha \subseteq D$, $\frac{|R_\alpha \bigcap D|}{|R_\alpha \bigcup D|} = \frac{|R_\alpha|}{|D|}$. Thus one idea would be to maintain the min-hash values for R_α and D and estimate the ratio.

2. $h(R_\alpha) = h(D)$ 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 we consider the next item in the data stream. We have the claim:

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 proof comes from the fact that $Pr[c_i(t) = \alpha] = \rho_\alpha(t)$. This is because $Pr[c_i(t) = \alpha] = Pr[h_i^*(t) = h_i(x) \mid x \in R_\alpha(t)]$, which in return equals $\frac{|R_\alpha(t)|}{|D_t|}$ where D_t is the number of distinct items seen up to a_t (this follows from the definition of min-hash function in Section 2).

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 to store [41, 38]. Processing each item takes time to update the k hash values and counters which takes time $O(k \log(1/\epsilon'))$ using results in [41, 38]. The value of k will now be determined in terms of ϵ' as well, 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. Of course if we knew α ahead of time, we do not have to maintain the counters $c_i(t)$ exactly, once they exceed α. We are only concerned if the final counter equals α. However, keeping the counters exactly as described has the nice property that at some point during the execution of the streaming algorithm, if user wishes to estimate $\rho_\alpha(t)$, they have the choice of α; the estimation as claimed above will work for any α determined at the query time t.

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. In our case we could check if the min-value element belonged to R_α by simply maintaining a counter corresponding to it. In general if the membership to the subset can be checked efficiently (using small space and small time) then the technique above can be applied[4].

Also, our algorithm above has the additional property that it is distributed. 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. Let $h_i^*(A)$ ($h_i^*(B)$) denote the hashes and $c_i(A)$ ($c_i(B)$), the counters, respectively for streams A (B, respectively) at time t when we perform the concatenation. We can determine the hash and counters for the combined stream as follows. For some i, say $h_i^*(A) < h_i^*(B)$. Then, we have $h_i^*(C) = h_i^*(A)$ and $c_i(C) = c_i(A)$ (and analogously if $h_i^*(B) < h_i^*(A)$. If $h_i^*(A) = h_i^*(B)$, then we have $h_i^*(C) = h_i^*(A)$ and $c_i(C) = c_i(A) + c_i(B)$. This is because for min-hash function h as in Section 2, we have $h(X \bigcup Y) = \min\{h(X), h(Y)\}$. Thus the concatenation takes only $O(k) = O(1)$ time.

Finally, if we wish to estimate $|R_\alpha(t)|$ rather than $\rho_\alpha(t)$, we can get an estimate for the number of distinct items seen so far using [23] as described in Section 2 and multiply that by $\hat{\rho_\alpha}$; the result will be a $1 \pm \epsilon$ approximation to $|R_\alpha(t)|$ up to precision p. This may be desired in some cases.

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. Say after seeing item a_{t-1}, we maintain $h_i^*(t-1) = \min_{j \le t-1} h_i(a_j)$. When item

[4] As a special case it is easy to see that the exact same algorithm can be used, even if we slightly change the definition of rarity to term all elements with multiplicity less than α (and at least 1) as α-rare.

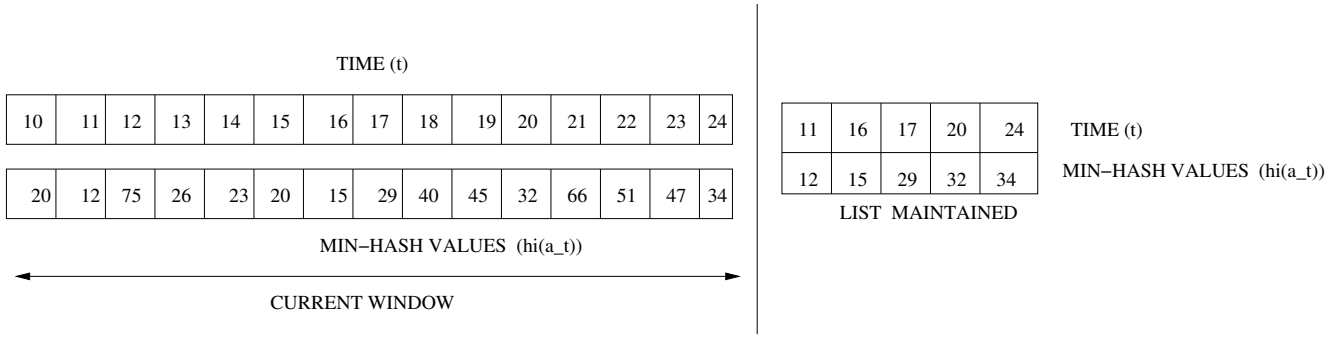

Figure 1: An example of linked list maintained by the algorithm.

a_t is seen, we compute $h_i(a_t)$ and set $h_i^*(t)$ to be $\min\{h_i(a_t), h_i^*(t-1)\}$. Thus, the problem of estimating similarity on unbounded data stream model is trivial.

We focus on the windowed data stream model. Here, the main challenge is to maintain $h_i^*(t)$'s for the window $t-(N-1), \ldots, t$ as t progresses. Consider the "hashed" stream $h_i(a_j)$. Our task is equivalent to maintaining the minimum value in this hashed data stream for the window $t-(N-1), \ldots, t$ as t progresses. In general, maintaining the minimum in a windowed data stream model is difficult as we discussed in Section 1. However, we use a property of the hashed data stream to perform our task as we explain below.

We say an item a_j is *active* at time t if $j \in \{t-(N-1), \ldots, t\}$. Consider maintaining a single min-hash value h_i^* over the window. As observed before, at each time t, we would like to maintain the minimum value in the hashed data stream over the window $t-(N-1), \ldots, t$. 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 of an active item d_1 if there is an active item d_2 that dominates it. This is because for all time intervals in the future whenever d_1 is active, d_2 will also be active and it would have a hash value no greater than d_1. Hence $h_i^*(t)$ is not affected by the hash value of an active item that is dominated.

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

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

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 $h_i(a_{j_\beta}) \leq h_i(a_{t+1})$; 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. If $h_i(a_{j_\beta}) \neq h_i(a_{t+1})$, 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 $h_i(a_{j_\beta}) = h_i(a_{t+1})$, we update $(h_i(a_{j_\beta}), j_\beta)$ to be $(h_i(a_{j_\beta}), t+1)$.

4. 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 spline* of this heap. Since the hash values are randomly chosen (in fact, they are defined by a random permutation in the case of a perfect min-wise hash family), we can use the property of treaps to bound the length of the link list. By an argument in [3], the expected length of the list $\mathcal{L}_i(t)$ (right spline) is $\Theta(H_N)$, the Nth harmonic number, from which the $\Theta(\log N)$ bound follows. [41]

From the preceding discussions (using binary search for finding α 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 random min-hash functions, and *not* over the distribution of the input. In other words, the theorem above holds for an arbitrary (worst case) input. The theorem is also unusual in the sense that in typical data stream results, the space used is deterministically bounded and the success probability is only over the quality of approximation. In contrast, in our result above, the space used is guaranteed only with high probability.

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 one) 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 α 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 $(h_i(a_{j_\beta}), list_{i,j_\beta}^t)$ 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 $list_{i,j_\beta}^t$; if the resulting list has more than α items, we delete the smallest index in it.

4. Let x be the smallest index in $list_{i,j_1}^t$; if a_x is not active at time $t+1$, we remove x from $list_{i,j_1}^t$. As a result if $list_{i,j_1}^t$ becomes empty, we remove the pair $(h_i(a_{j_1}), list_{i,j_1}^t)$ 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 for each min-hash value we maintain the data structure from [15] 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 $1 \pm \tau$.

As an aside, if we wished to estimate $|R_\alpha|$ rather than α-rarity over the windowed data stream, we need to estimate the number of distinct items in the window at any time t. An algorithm for that problem is known [15], hence, $|R_\alpha|$ can be approximated to factor $1 \pm \epsilon$ in the windowed data stream model (as always, up to precision p).

6 Concluding Remarks

Windowed data stream model is very natural for query processing in networking and massive database applications. In this paper, presented efficient algorithms for two basic problems in this model, one is estimating distributional characteristic, i.e., rarity and the other is a similarity characteristic, i.e., the Jacard's coefficient. While previous research in this area has focused on L_p norms and simple, specific distributional parameters such as the number of distinct items, our problems are novel and quite general. Furthermore, they are well motivated in the data stream context.

Our algorithms are actually quite simple. If we settle for pairwise independent hash functions rather than ϵ'-min-hash functions, we do not get the provable guarantees in this paper, but the algorithms will be simpler to implement and we expect them to work quite well in practice. We hope to explore this in the future.

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 [19]. We expect our techniques to find other applications in practice.

References

[1] Amazon. Amazon project. `http://www.cs.cornell.edu/database/amazon/amazon.htm`.

[2] N. Alon, Y. Matias, M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. Twenty-Eighth Annual ACM Symposium on Theory of Computing*, 1996.

[3] C. Aragon, and R. Seidel. Randomized search trees. In *Proc. 30th IEEE FOCS, 1989.*

[4] A. Broder. Filtering Near-Duplicate Documents. In *Proceedings of FUN*, 1998.

[5] 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.

[6] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise Independent Permutations. In *Proc. of STOC*, 1998.

[7] B. Babcock, M. Datar, and R. Motwani. Sampling from a Moving Window over Streaming Data. to appear in *Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms.*

[8] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. *Computer Networks and ISDN Systems*, 29(8–13):1157–1166, 1997.

[9] S. Babu, L. Subramanian, and J. Widom. A data stream management system for network traffic management. In *Proceedings of Workshop on Network-Related Data Management (NRDM)*, 2001.

[10] S. Chaudhuri, R. Motwani, V. R. Narasayya. On random sampling over joins. In *Proc. 1999 ACM SIGMOD*, pp. 263–274, 1999.

[11] E. Cohen. Size-Estimation Framework with Applications to Transitive Closure and Reachability. *Journal of Computer and System Sciences* 55 (1997): 441–453.

[12] 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.*

[13] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of tabular data via approximate distance computations. In *Proceedings of the International Conference on Data Engineering*, 2002.

[14] C. Cortes, K. Fisher, D. Pregibon, A. Rogers. Hancock: a language for extracting signatures from data streams. In *Proc. 2000 ACM SIGKDD*, pp. 9–17, 2000.

[15] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. to appear in *Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms.*

[16] DIMACS. Dimacs workshop on streaming data analysis and mining. `http://dimacs.rutgers.edu/Workshops/Streaming/`.

[17] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, 2000.

[18] P. Domingos, G. Hulten, and L. Spencer. Mining time-changing data streams. In *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*, 2001.

[19] T. Dasu, T. Johnson, S. Muthukrishnan and V. Shkapenyuk Mining database structure, or to How to build a data quality browser. To appear in *Proc. of the SIGMOD*, 2002.

[20] M. Fang, H. Garcia-Molina, R. Motwani, N. Shivakumar, J.D. Ullman. Computing iceberg queries efficiently. In *Proc. 24th International Conference on Very Large Data Bases (VLDB)*, 1998.

[21] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan. An Approximate L1-Difference Algorithm for Massive Data Streams. In *Proc. 40th Symposium on Foundations of Computer Science*, 1999.

[22] A. Feldmann, A. Greenberg, C. Lund, N. Reingold and J. Rexford, NetScope: Traffic engineering for IP networks. In *IEEE Network Magazine*, special issue on Internet Traffic Engineering, 2000.

[23] P. Flajolet, G. Martin. Probabilistic Counting. In *Proc. 24th Symposium on Foundations of Computer Science*, 1983.

[24] C. Fraleigh, S. Moon, C. Diot, B. Lyles, F. Tobagi. Architecture of a passive monitoring system for backbone IP networks. Technical Report TR00-ATL-101-801, Sprint Labs, 2000.

[25] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Symposium on Parallel Algorithms and Architectures*, 2001.

[26] T. Gil and M. Poletto. MULTIOPS: a data structure for bandwidth attach detection. IN *USENIX Conference*, 2001.

[27] 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.

[28] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of VLDB*, 2001.

[29] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. QuickSAND: Quick Summary and Analysis of Network Data. *DIMACS Technical Report*, 2001-43.

[30] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proc. of VLDB'99.*

[31] S. Guha, P. Indyk, N. Koudas, and N. Thaper. Dynamic Multidimensional Histograms. In *Proc. ACM Symp on Mgmt of Data (SIGMOD)*, 2002.

[32] S. Guha, and N. Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. IN *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, 2002.*

[33] S. Guha, N. Koudas, K. Shim. Data-Streams and Histograms. In *Proc. Thirty-Third Annual ACM Symposium on Theory of Computing*, 2001.

[34] 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.

[35] T. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating Strategies for Similarity Search on the Web. To appear in *Proc. of the Eleventh International World Wide Web Conference,* 2002.

[36] 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.

[37] P. Indyk. Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation. In *Proc. 41st Symposium on Foundations of Computer Science*, 2000.

[38] P. Indyk A Small Approximately Min-Wise Independent Family of Hash Functions. In *Journal of Algorithms 38(1): 84-90 (2001).*

[39] H. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. In *Proc. 24th International Conference on Very Large Data Bases (VLDB)*, 1998.

[40] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proceedings of ICDE*, 2002.

[41] K. Mulmuley An Introduction through Randomized Algorithms. Prentice Hall, 1993.

[42] R. Motwani, P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[43] NOAA. U.S. national weather service. `http://www.nws.noaa.gov/` .

[44] *Netflow Services and Applications.* Whitepaper, Cisco Systems, 2000. Available at http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm.

[45] Stanford. Stanford stream data management. `http://www-db.stanford.edu/stream`.