No Repetition

Fast Streaming with Highly Concentrated Hashing

Aamand, Anders; Das, Debarati; Kipouridis, Evangelos; Knudsen, Jakob Bæk Tejs; Rasmussen, Peter Michael Reichstein; Thorup, Mikkel

Publication date:
2021

Document version
Publisher's PDF, also known as Version of record

No Repetition: Fast Streaming with Highly Concentrated Hashing

Anders Aamand* Debarati Das* Evangelos Kipouridis* Jakob B. T. Knudsen*
Peter M. R. Rasmussen* Mikkel Thorup*

April 3, 2020

Abstract

To get estimators that work within a certain error bound with high probability, a common strategy is to design one that works with constant probability, and then boost the probability using independent repetitions. Important examples of this approach are small space algorithms for estimating the number of distinct elements in a stream, or estimating the set similarity between large sets. Using standard strongly universal hashing to process each element, we get a sketch based estimator where the probability of a too large error is, say, 1/4. By performing r independent repetitions and taking the median of the estimators, the error probability falls exponentially in r. However, running r independent experiments increases the processing time by a factor r.

Here we make the point that if we have a hash function with strong concentration bounds, then we get the same high probability bounds without any need for repetitions. Instead of r independent sketches, we have a single sketch that is r times bigger, so the total space is the same. However, we only apply a single hash function, so we save a factor r in time, and the overall algorithms just get simpler.

Fast practical hash functions with strong concentration bounds were recently proposed by Aamand *et al.* (to appear in *STOC 2020*). Using their hashing schemes, the algorithms thus become very fast and practical, suitable for online processing of high volume data streams.

*Basic Algorithms Research Copenhagen (BARC), University of Copenhagen.

1 Introduction

To get estimators that work within a certain error bound with high probability, a common strategy is to design one that works with constant probability, and then boost the probability using independent repetitions. A classic example of this approach is the algorithm of Bar-Yossef *et al.* [3] to estimate the number of distinct elements in a stream. Using standard strongly universal hashing to process each element, we get an estimator where the probability of a too large error is, say, $1/4$. By performing r independent repetitions and taking the median of the estimators, the error probability falls exponentially in r. However, running r independent experiments increases the processing time by a factor r.

Here we make the point that if we have a hash function with strong concentration bounds, then we get the same high probability bounds without any need for repetitions. Instead of r independent sketches, we have a single sketch that is $\Theta(r)$ times bigger, so the total space is essentially the same. However, we only apply a single hash function, processing each element in constant time regardless of r, and the overall algorithms just get simpler.

Fast practical hash functions with strong concentration bounds were recently proposed by Aamand *et al.* [1]. Using their hashing schemes, we get a very fast implementation of the above streaming algorithm, suitable for online processing of high volume data streams.

To illustrate a streaming scenario where the constant in the processing time is critical, consider the Internet. Suppose we want to process packets passing through a high-end Internet router. Each application only gets very limited time to look at the packet before it is forwarded. If it is not done in time, the information is lost. Since processors and routers use some of the same technology, we never expect to have more than a few instructions available. Slowing down the Internet is typically not an option. The papers of Krishnamurthy *et al.* [19] and Thorup and Zhang [25] explain in more detail how high speed hashing is necessary for their Internet traffic analysis. Incidentally, the hash function we use from [1] is a bit faster than the ones from [19, 25], which do not provide Chernoff-style concentration bounds.

The idea is generic and can be applied to other algorithms. We will also apply it to Broder's original min-hash algorithm [7] to estimate set similarity, which can now be implemented efficiently, giving the desired estimates with high probability.

Concentration Let us now be more specific about the algorithmic context. We have a key universe U, e.g., 64-bit keys, and a random hash function h mapping U uniformly into $R = (0, 1]$.

For some input set S and some fraction $p \in [0, 1)$, we want to know the number X of keys from S that hash below p. Here p could be an unknown function of S, but p should be independent of the random hash function h. Then the mean μ is $\mathbb{E}[X] = |S|p$.

If the hash function h is fully random, we get the classic Chernoff bounds on X (see, e.g, [20]):

$$\Pr[X \geq (1+\varepsilon)\mu] \leq \exp(-\varepsilon^2\mu/3) \text{ for } 0 \leq \varepsilon \leq 1, \tag{1}$$

$$\Pr[X \leq (1-\varepsilon)\mu] \leq \exp(-\varepsilon^2\mu/2) \text{ for } 0 \leq \varepsilon \leq 1. \tag{2}$$

Unfortunately, we cannot implement fully random hash functions as it requires space as big as the universe.

To get something implementable in practice, Wegman and Carter [26] proposed strongly universal hashing. The random hash function $h : U \to R$ is *strongly universal* if for any given distinct keys $x, y \in U$, $(h(x), h(y))$ is uniform in R^2. The standard implementation of a strongly universal hash function into $[0, 1)$ is to pick large prime $\wp$ and two uniformly random numbers $a, b \in \mathbb{Z}_\wp$. Then $h_{a,b}(x) = ((ax + b) \bmod \wp)/\wp$ is strongly universal from $U \subseteq \mathbb{Z}_\wp$ to $R = \{i/\wp | i \in Z_\wp\} \subset [0, 1)$. Obviously it is not uniform in $[0, 1)$, but for any $p \in [0, 1)$, we have $\Pr[h(x) < p] \approx p$ with equality if $p \in R$. Below we ignore this deviation from uniformity in $[0, 1)$.

Assuming we have a strongly universal hash function $h : U \to [0, 1)$, we again let X be the number of elements from S that hash below p. Then $\mu = \mathbb{E}[X] = |S|p$ and because the hash values are 2-independent, we have $\mathrm{Var}[X] \leq \mathbb{E}[X] = \mu$. Therefore, by Chebyshev's inequality,

$$\Pr[|X - \mu| \geq \varepsilon\mu] < 1/(\varepsilon^2\mu).$$

As $\varepsilon^2\mu$ gets large, we see that the concentration we get with strongly universal hashing is much weaker than the Chernoff bounds with fully random hashing. However, Chebyshev is fine if we just aim at a constant error probability like $1/4$, and then we can use the median over independent repetitions to reduce the error probability.

In this paper we discuss benefits of having hash functions with strong concentration akin to that of fully random hashing:

Definition 1. *A hash function $h : U \to [0,1)$ is* strongly concentrated *with added error probability $\mathcal{E}$ if for any set $S \subseteq U$ and $p \in [0,1)$, if X is the number of elements from S hashing below p, $\mu = p|S|$ and $\varepsilon \leq 1$, then*

$$\Pr\left[|X - \mu| \geq \varepsilon\mu\right] = 2\exp(-\Omega(\varepsilon^2\mu)) + \mathcal{E}.$$

If $\mathcal{E} = 0$, we simply say that h is strongly concentrated*.*

Another way of viewing the added error probability $\mathcal{E}$ is as follows. We have strong concentration as long as we do not aim for error probabilities below $\mathcal{E}$, so if $\mathcal{E}$ is sufficiently low, we can simply ignore it.

What makes this definition interesting in practice is that Aamand *et al.* [1] recently presented a fast practical small constant time hash function that for $U = [u] = \{0, \ldots, u-1\}$ is strongly concentrated with added error probability $u^{-\gamma}$ for any constant γ. This term is so small that we can ignore it in all our applications. The speed is obtained using certain character tables in cache that we will discuss later.

Next we consider our two streaming applications, distinct elements and set-similarity, showing how strongly concentrated hashing eliminates the need for time consuming independent repetitions. We stress that in streaming algorithms on high volume data streams, speed is of critical importance. If the data is not processed quickly, the information is lost.

Distinct elements is the simplest case, and here we will also discuss the ramifications of employing the strongly concentrated hashing of Aamand *et al.* [1] as well as possible alternatives.

2 Counting distinct elements in a data stream

We consider a sequence of keys $x_1, \ldots, x_s \in [u]$ where each element may appear multiple times. Using only little space, we wish to estimate the number n of distinct keys. We are given parameters ε and δ, and the goal is to create an estimator, $\hat{n}$, such that $(1-\varepsilon)n \leq \hat{n} \leq (1+\varepsilon)n$ with probability at least $1-\delta$.

Following the classic approach of Bar-Yossef *et al.* [3], we use a strongly universal hash function $h : U \to (0,1]$. For simplicity, we assume h to be collision free over U.

For some $k > 1$, we maintain the k smallest distinct hash values of the stream. We assume for simplicity that $k \leq n$. The space required is thus $O(k)$, so we want k to be small. Let $x_{(k)}$ be the key having the k'th smallest hash value under h and let $h_{(k)} = h(x_{(k)})$. As in [3], we use $\hat{n} = k/h_{(k)}$ as an estimator for n (we note that [3] suggests several other estimators, but the points we will make below apply to all of them).

The point in using a hash function h is that all occurrences of a given key x in the stream get the same hash value, so if S is the set of distinct keys, $h_{(k)}$ is just the k smallest hash value from S. In particular, $\hat{n}$ depends only on S, not on the frequencies of the elements of the stream. Assuming no collisions, we will often identify the elements with the hash values, so x_i is smaller than x_j if $h(x_i) \leq h(x_j)$.

We would like $1/h_{(k)}$ to be concentrated around n/k. For any probability $p \in [0,1]$, let $X^{<p}$ denote the number of elements from S that hash below p. Let $p_- = k/((1+\varepsilon)n)$ and $p_+ = k/((1-\varepsilon)n)$. Note that both p_- and p_+ are independent of the random hash function h. Now

$$\begin{aligned} 1/h_{(k)} \leq (1-\varepsilon)n/k &\iff X^{<p_+} < k = (1-\varepsilon)\mathbb{E}\left[X^{<p_+}\right] \\ 1/h_{(k)} > (1+\varepsilon)n/k &\iff X^{<p_-} \geq k = (1+\varepsilon)\mathbb{E}\left[X^{<p_-}\right], \end{aligned}$$

and these observations form a good starting point for applying probabilistic tail bounds as we now describe.

2.1 Strong universality and independent repetitions

Since h is strongly universal, the hash values of any two keys are independent, so for any p, we have $\mathrm{Var}\left[X^{<p}\right] \leq \mathbb{E}\left[X^{<p}\right]$, and so by Chebyshev's inequality,

$$\Pr\left[1/h_{(k)} \leq (1-\varepsilon)n/k\right] < (1-\varepsilon)/(k\varepsilon^2)$$
$$\Pr\left[1/h_{(k)} > (1+\varepsilon)n/k\right] \leq (1+\varepsilon)/(k\varepsilon^2).$$

Assuming $\varepsilon \leq 1$, we thus get that

$$\Pr\left[|\hat{n}-n| > \varepsilon n\right] = \Pr\left[\left|1/h_{(k)} - n/k\right| > \varepsilon n/k\right] \leq 2/(k\varepsilon^2).$$

To get the desired error probability δ, we could now set $k = 2/(\delta\varepsilon^2)$, but if δ is small, e.g. $\delta = 1/u$, k becomes way too large. As in [3] we instead start by aiming for a constant error probability, δ_0, say $\delta_0 = 1/4$. For this value of δ_0, it suffices to set $k_0 = 8/\varepsilon^2$. We now run r (to be determined) independent experiments with this value of k_0, obtaining independent estimators for n, $\hat{n}_1, \ldots, \hat{n}_r$. Finally, as our final estimator, $\hat{n}$, we return the median of $\hat{n}_1, \ldots, \hat{n}_r$. Now for each $1 \leq i \leq r$, $\Pr[|\hat{n}_i - n| > \varepsilon n] \leq 1/4$ and these events are independent. If $|\hat{n}-n| \geq \varepsilon n$, then $|\hat{n}_i - n| \geq \varepsilon n$ for at least half of the $1 \leq i \leq r$. By the standard Chernoff bound (1), this probability can be bounded by

$$\Pr\left[|\hat{n}-n| > \varepsilon n\right] \leq \exp(-(r/4)/3) = \exp(-r/12).$$

Setting $r = 12\ln(1/\delta)$, we get the desired error probability $1/\delta$. The total number of hash values stored is $k_0 r = (8/\varepsilon^2)(12\ln(\delta)) = 96\ln(1/\delta)/\varepsilon^2$.

2.2 A better world with fully random hashing

Suppose instead that $h : [u] \to (0,1]$ is a fully random hash function. In this case, the standard Chernoff bounds (1) and (2) with $\varepsilon \leq 1$ yield

$$\Pr\left[1/h_{(k)} \leq (1-\varepsilon)n/k\right] < \exp(-(k/(1-\varepsilon))\varepsilon^2/2)$$
$$\Pr\left[1/h_{(k)} > (1+\varepsilon)n/k\right] \leq \exp(-(k/(1+\varepsilon))\varepsilon^2/3).$$

Hence

$$\Pr\left[|\hat{n}-n| > \varepsilon n\right] = \Pr\left[|1/h_{(k)} - n/k| \geq \varepsilon n/k\right] \leq 2\exp(-k\varepsilon^2/6). \tag{3}$$

Thus, to get error probability δ, we just use $k = 6\ln(2/\delta)/\varepsilon^2$. There are several reasons why this is much better than the above approach using 2-independence and independent repetitions.

- It avoids the independent repetitions, so instead of applying $r = \Theta(\log(1/\delta))$ hash functions to each key we just need one. We thus save a factor of $\Theta(\log(1/\delta))$ in speed.
- Overall we store fewer hash values: $k = 6\ln(2/\delta)/\varepsilon^2$ instead of $96\ln(1/\delta)/\varepsilon^2$.
- With independent repetitions, we are tuning the algorithm depending on ε and δ, whereas with a fully-random hash function, we get the concentration from (3) for every $\varepsilon \leq 1$.

The only caveat is that fully-random hash functions cannot be implemented.

2.3 Using hashing with strong concentration bounds

We now discuss the effect of relaxing the abstract full-random hashing to hashing with strong concentration bounds and added error probability $\mathcal{E}$. Then for $\varepsilon \leq 1$,

$$\Pr\left[1/h_{(k)} \leq (1-\varepsilon)n/k\right] = 2\exp(-\Omega(k/(1-\varepsilon))\varepsilon^2) + \mathcal{E}$$
$$\Pr\left[1/h_{(k)} > (1+\varepsilon)n/k\right] = 2\exp(-\Omega(k/(1+\varepsilon))\varepsilon^2) + \mathcal{E}.$$

so

$$\Pr\left[|\hat{n} - n| \geq \varepsilon n\right] = \Pr\left[|1/h_{(k)} - n/k| \geq \varepsilon n/k\right] \leq 2\exp(-\Omega(k\varepsilon^2)) + O(\mathcal{E}). \quad (4)$$

To obtain the error probability $\delta = \omega(\mathcal{E})$, we again need to store $k = O(\log(1/\delta)/\varepsilon^2)$ hash values. Within a constant factor this means that we use the same total number using 2-independence and independent repetitions, and we still retain the following advantages from the fully random case.

- With no independent repetitions we avoid applying $r = \Theta(\log(1/\delta))$ hash functions to each key, so we basically save a factor $\Theta(\log(1/\delta))$ in speed.
- With independent repetitions, we only address a given $\varepsilon \leq 1$ and δ, while with a fully-random hash function we get the concentration from (3) for every $\varepsilon \leq 1$.

2.4 Implementation and alternatives

We briefly discuss how to maintain the k smallest elements/hash values. The most obvious method is using a priority queue, but this takes $O(\log k)$ time per element, dominating the cost of evaluating the hash function. However, we can get down to constant time per element if we have a buffer for k. When the buffer gets full, we find the median in linear time with (randomized) selection and discard the bigger elements. This is standard to de-amortize if needed.

A different, and more efficient, sketch from [3] identifies the smallest b such that the number $X^{<1/2^b}$ of keys hashing below $1/2^b$ is at most k. For the online processing of the stream, this means that we increment b whenever $X^{<1/2^b} > k$. At the end, we return $2^b X^{<1/2^b}$. The analysis of this alternative sketch is similar to the one above, and we get the same advantage of avoiding independent repetitions using strongly concentrated hashing, that is, for error probability δ, in [3], they run $O(\log(1/\delta))$ independent experiments with independent hash functions, each storing up to $k = O(1/\varepsilon^2)$ hash values, whereas we run only a single experiment with a single strongly concentrated hash function storing $k = O(\log(1/\delta)/\varepsilon^2)$ hash values. The total number of hash values stored is the same, but asymptotically, we save a factor $\log(1/\delta)$ in time.

Other alternatives Estimating the number of distinct elements in a stream began with the work of Flajolet and Martin [13] and has continued with a long line of research [2, 3, 4, 5, 8, 9, 11, 12, 13, 14, 15, 16, 17, 27]. In particular, there has been a lot of focus on minimizing the sketch size. Theoretically speaking, the problem finally found an asymptotically optimal, both in time and in space, solution by Kane, Nelson and Woodruff [18], assuming we only need $\frac{2}{3}$ probability of success. The optimal space, including that of the hash function, is $O(\varepsilon^{-2} + \log n)$ bits, improving the $O(\varepsilon^{-2} \cdot \log n)$ bits needed by Bar-Yossef *et al.* [3] to store $O(\varepsilon^{-2})$ hash values. Both [3] and [18], suggest using $O(\log(1/\delta))$ independent repetitions to reduce the error probability to $1/\delta$, but then both time and space blow up by a factor $O(\log(1/\delta))$.

Recently Blasiok [6] found a space optimal algorithm for the case of small error probability $1/\delta$. In this case, the bound from [18] with independent repetitions was $O(\log(1/\delta)(\varepsilon^{-2} + \log n))$ which he reduces to $O(\log(1/\delta)\varepsilon^{-2} + \log n)$, again including the space for hash functions. He no longer has $O(\log(1/\delta))$ hash functions, but this only helps his space, not his processing time, which he states as polynomial in $\log(1/\delta)$ and $\log n$.

The above space optimal algorithms [6, 18] are very interesting, but fairly complicated, seemingly involving some quite large constants. However, here our focus is to get a fast practical algorithm to handle a high volume data stream online, not worrying as much about space. Assuming fast strongly concentrated hashing, it is then much better to use our implementation of the simple algorithm of Bar-Yossef *et al.* [3] using $k = O(\varepsilon^{-2}\log(1/\delta))$.

2.5 Implementing Hashing with Strong Concentration

As mentioned earlier, Aamand *et al.* [1] recently presented a fast practical small constant time hash function, Tabulation-1Permutation, that for $U = [u] = \{0, \ldots, u-1\}$ is strongly concentrated with additive error $u^{-\gamma}$ for any constant γ. The scheme obtains its power and speed using certain character tables in cache.

More specifically, we view keys as consisting of a small number c of characters from some alphabet Σ, that is, $U = \Sigma^c$. For 64-bit keys, this could be $c = 8$ characters of 8 bits each. Let's say that hash values are also from U, but viewed as bit strings representing fractions in $[0, 1)$.

Tabulation-1Permutation needs $c + 1$ character tables mapping characters to hash values. To compute the hash value of a key, we need to look up $c + 1$ characters in these tables. In addition we need $O(c)$ fast AC^0 operations to extract the characters and xor the hash values. The character tables can be populated with an $O(\log n)$ independent pseudo-random number generator, needing a random seed of $O((\log n)(\log u))$ bits.

Computer dependent versus problem dependent view of resources for hashing We view the resources used for Tabulation-1Permutation as computer dependent rather than problem dependent. When you buy a new computer you can decide how much cache you want to allocate for your hash functions. In the experiments performed in [1], using 8-bit characters and $c = 8$ for 64-bit keys was very efficient. On two computers, it was found that tabulation-1permutation was less than 3 times slower than the fastest known strongly universal hashing scheme; namely Dietzfelbinger's [10] which does just one multiplication and one shift. Also, Tabulation-1Permutation was more than 50 times faster than the fastest known highly independent hashing scheme; namely Thorup's [24] double tabulation scheme which, in theory also works in constant time.

In total, the space used by all the character tables is $9 \times 2^8 \times 64$ bits which is less than 20 KB, which indeed fits in very fast cache. We note that when we have first populated the tables with hash values, they are not overwritten. This means that the cache does not get dirty, that is different computer cores can access the tables and not worry about consistency.

This is different than the work space used to maintain the sketch of the number of distinct keys represented via $k = O(\varepsilon^{-2} \log(1/\delta))$ hash values, but let's compare anyway with real numbers. Even with a fully random hash function with perfect Chernoff bounds, we needed $k = 6 \ln(2/\delta)/\varepsilon^2$, so with, say, $\delta = 1/2^{30}$ and $\varepsilon = 1\%$, we get $k > 2^{20}$, which is much more than the 9×2^8 hash values stored in the character tables for the hash functions. Of course, we would be happy with a much smaller k so that everything is small and fits in fast cache.

We note that any $k > |\Sigma| = 2^8$ rules out the concentration of previous tabulation schemes such a simple tabulation [21] and twisted tabulation [22]. The reader is referred to [1] for a thorough discussion of the alternatives.

Finally, we relate our strong concentration from Definition 1 to the exact concentration result from [1]:

Theorem 1. *Let $h\colon [u] \to [r]$ be a tabulation-1permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Consider a key/ball set $S \subseteq [u]$ of size $n = |S|$ where each ball $x \in S$ is assigned a weight $w_x \in [0, 1]$. Choose arbitrary hash values $y_1, y_2 \in [r]$ with $y_1 \le y_2$. Define $X = \sum_{x \in S} w_x \cdot [y_1 \le h(x) < y_2]$ to be the total weight of balls hashing to the interval $[y_1, y_2)$. Write $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathrm{Var}[X]$. Then for any constant γ and every $t > 0$,*

$$\Pr[|X - \mu| \ge t] \le 2\exp(-\Omega(\sigma^2\, \mathcal{C}(t/\sigma^2))) + 1/u^\gamma. \tag{5}$$

Here $\mathcal{C} : (-1, \infty) \to [0, \infty)$ is given by $\mathcal{C}(x) = (x + 1)\ln(x + 1) - x$, so $\exp(-\mathcal{C}(x)) = \frac{e^x}{(1+x)^{(1+x)}}$. The above also holds if we condition the random hash function h on a distinguished query key q having a specific hash value.

The above statement is far more general than what we need. All our weights are unit weights. We fix $r = u$ and $y_1 = 0$. Viewing hash values as fractions in $[0, 1)$, the random variable X is the number of items hashing below $p = y_2/u$. Also, since $\mathrm{Var}[X] \le \mathbb{E}[X]$, (5) implies the same statement with μ instead of σ^2. Moreover, our $\varepsilon \le 1$ corresponds to $t = \varepsilon\mu \le \mu$, and then we get

$$\Pr[|X - \mu| \ge \varepsilon\mu] \le 2\exp(-\Omega(\mu\, \mathcal{C}(\varepsilon))) + 1/u^\gamma \le 2\exp(-\Omega(\mu\varepsilon^2)) + 1/u^\gamma.$$

which is exactly as in our Definition 1. Only remaining difference is that Definition 1 should work for *any* $p \in [0, 1)$ while the bound we get only works for p that are multiples of $1/u$. However, this suffices by the following general lemma:

Lemma 2. *Suppose we have a hash function $h : [u] \to [0,1)$ such that for any set $S \subseteq U$ and for any $p \in [0,1)$ that is a multiple of $1/u$, for the number $X^{<p}$ of elements from S that hash below p, with $\mu_p = p|S|$ and $\varepsilon \leq 1$, it holds that*

$$\Pr\left[|X^{<p} - \mu_p| \geq \varepsilon\mu_p\right] \leq 2\exp(-\Omega(\varepsilon^2\mu_p)) + O(\mathcal{E}).$$

Then the same statement holds for all $p \in [0,1)$

Proof. First we note that the statement is trivially true if $\varepsilon^2\mu_p = O(1)$, so we can assume $\varepsilon^2\mu_p = \omega(1)$. Since $\varepsilon \leq 1$, we also have $\mu_p = \omega(1)$.

We are given an arbitrary $p \in [0,1)$. Let $p_+ = i/u$ be the nearest higher multiple of $1/u$. Since $|S| \leq u$ and $\mu_p = p|S|$ we have $i \geq \mu_p$, implying $i = \omega(1)$. We also let $p_- = (i-1)/u$.

It is now clear that since $p_- < p \leq p_+$, it holds that $X^{<p_-} \leq X^{<p} \leq X^{<p_+}$. We first show that

$$X^{<p} \leq (1-\varepsilon)\mu_p \implies X^{<p_-} \leq (1-\varepsilon/2)\mu_{p_-}.$$

Indeed, $X^{<p} \leq (1-\varepsilon)\mu_p$ implies $X^{<p_-} \leq (1-\varepsilon)p|S| \leq (1-\varepsilon)(p_- + 1/u)|S| = \mu_{p_-} - \varepsilon\mu_{p_-} + (1-\varepsilon)|S|/u$.

But $|S| \leq u$ and $(1-\varepsilon) < 1$, so $X^{<p_-} \leq \mu_{p_-} - \varepsilon\mu_{p_-} + 1 \leq (1-\varepsilon/2)\mu_{p_-}$. The last follows from the fact that $(\varepsilon/2)\mu_{p_-} \geq (\varepsilon/2)\mu_p - (\varepsilon/2)|S|/u \geq (\varepsilon^2/2)\mu_p - 1$, but $\varepsilon^2\mu_p = \omega(1)$ and so $(\varepsilon/2)\mu_{p_-} = \omega(1)$.

The exact same reasoning gives

$$X^{<p} \geq (1+\varepsilon)\mu_p \implies X^{<p_+} \geq (1+\varepsilon/2)\mu_{p_+}.$$

But then

$$\Pr\left[|X^{<p} - \mu_p| \geq \varepsilon\mu_p\right] = \Pr\left[X^{<p} \leq (1-\varepsilon)\mu_p\right] + \Pr\left[X^{<p} \geq (1+\varepsilon)\mu_p\right] \leq$$

$$\Pr\left[X^{<p_-} \leq (1-\varepsilon/2)\mu_{p_-}\right] + \Pr\left[X^{<p_+} \geq (1+\varepsilon/2)\mu_{p_+}\right] \leq$$

$$\Pr\left[|X^{<p_-} - \mu_{p_-}| \geq (\varepsilon/2)\mu_{p_-}\right] + \Pr\left[|X^{<p_+} - \mu_{p_+}| \geq (\varepsilon/2)\mu_{p_+}\right] \leq$$

Notice that $\mu_p - 1 \leq \mu_{p_-} \leq \mu_{p_+}$, and p_- and p_+ are multiples of $1/u$, so we can use the bounds of the statement. Thus $\Pr[|X^{<p} - \mu_p| \geq \varepsilon\mu_p]$ is upper bounded by

$$4\exp(-\Omega((\varepsilon/2)^2(\mu_p - 1))) + O(\mathcal{E}) = 2\exp(-\Omega(\varepsilon^2\mu_p)) + O(\mathcal{E})$$

□

We note that [1] also presents a slightly slower scheme, Tabulation-Permutation, which offers far more general concentration bounds than those for Tabulation-1Permutation in Theorem 1. However, Tabulation-1Permutation is faster and sufficient for the strong concentration needed for our streaming applications.

3 Set similarity

We now consider Broder's [7] original algorithm for set similarity. As above, it uses a hash function $h : [u] \to [0,1]$ which we assume to be collision free. The bottom-k sample $\mathrm{MIN}_k(S)$ of a set $S \subseteq [u]$ consists of the k elements with the smallest hash values. If h is fully random then $\mathrm{MIN}_k(S)$ is a uniformly random subset of k distinct elements from $\mathrm{MIN}_k(S)$. We assume here that $k \leq n = |S|$. With $\mathrm{MIN}_k(S)$, we can estimate the frequency $f = |T|/|S|$ of any subset $T \subseteq S$ as $|\mathrm{MIN}_k(S) \cap T|/k$.

Broder's main application is the estimation of the Jaccard similarity $f = |A \cap B|/|A \cup B|$ between sets A and B. Given the bottom-k samples from A and B, we may construct the bottom-k sample of their union as $\mathrm{MIN}_k(A \cup B) = \mathrm{MIN}_k(\mathrm{MIN}_k(A) \cup \mathrm{MIN}_k(B))$, and then the similarity is estimated as $|\mathrm{MIN}_k(A \cup B) \cap \mathrm{MIN}_k(A) \cap \mathrm{MIN}_k(B)|/k$.

We note again the crucial importance of having a common hash function h. In a distributed setting, samples $\mathrm{MIN}_k(A)$ and $\mathrm{MIN}_k(B)$ can be generated by different entities. As long as they agree on h, they

only need to communicate the samples to estimate the Jaccard similarity of A and B. As noted before, for Tabulation-1Permutation h can be shared by exchanging a random seed of $O((\log n)(\log u))$ bits.

For the hash function h, Broder [7] first considers fully random hashing. Then $\mathrm{MIN}_k(S)$ is a fully random sample of k distinct elements from S, which is very well understood.

Broder also sketches some alternatives with realistic hash functions, but Thorup [23] showed that even if we just use 2-independence, we get the same expected error as with fully random hashing, but here we want strong concentration. Our analysis follows the simple union-bound approach from [23].

For the analysis, it is simpler to study the case where we are sampling from a set S and want to estimate the frequency $f = |T|/|S|$ of a subset $T \subseteq S$. Let $h_{(k)}$ be the kth smallest hash value from S as in the above algorithm for estimating distinct elements. For any p let $Y^{\leq p}$ be the number of elements from T with hash value at most p. Then $|T \cap \mathrm{MIN}_k(S)| = Y^{\leq h_{(k)}}$ which is our estimator for fk.

Theorem 3. *For $\varepsilon \leq 1$, if h is strongly concentrated with added error probability $\mathcal{E}$, then*

$$\Pr\left[|Y^{\leq h_{(k)}} - fk| > \varepsilon fk\right] = 2\exp(-\Omega(fk\varepsilon^2)) + O(\mathcal{E}). \tag{6}$$

Proof. Let $n = |S|$. We already saw in (4) that for any $\varepsilon_S \leq 1$, $P_S = \Pr\left[|1/h_{(k)} - n/k| \geq \varepsilon_S n/k\right] \leq 2\exp(-\Omega(k\varepsilon_S^2)) + O(\mathcal{E})$. Thus, with $p_- = k/((1+\varepsilon_S)n)$ and $p_+ = k/((1-\varepsilon_S)n)$, we have $h_{(k)} \in [p_-, p_+]$ with probability $1 - P_S$, and in that case, $Y^{\leq p_-} \leq Y^{\leq h_{(k)}} \leq Y^{\leq p_+}$.

Let $\mu^- = \mathbb{E}\left[Y^{\leq p_-}\right] = fk/(1+\varepsilon_S) \geq fk/2$. By strong concentration, for any $\varepsilon_T \leq 1$, we get that

$$P_T^- = \Pr\left[Y^{\leq p_-} \leq (1-\varepsilon_T)\mu_-\right] \leq 2\exp(-\Omega(\mu_-\varepsilon_T^2)) + \mathcal{E} = 2\exp(-\Omega(fk\varepsilon_T^2)) + \mathcal{E}.$$

Thus

$$\Pr\left[Y^{\leq h_{(k)}} \leq \frac{1-\varepsilon_T}{1+\varepsilon_S} fk\right] \leq P_T^- + P_S.$$

Likewise, with $\mu^+ = \mathbb{E}\left[Y^{\leq p_+}\right] = fk/(1-\varepsilon_S)$, for any ε_T, we get that

$$P_T^+ = \Pr\left[Y^{\leq p_+} \geq (1+\varepsilon_T)\mu_+\right] \leq 2\exp(-\Omega(\mu_+\varepsilon_T^2)) + \mathcal{E} = 2\exp(-\Omega(fk\varepsilon_T^2)) + \mathcal{E},$$

and

$$\Pr\left[Y^{\leq h_{(k)}} \geq \frac{1+\varepsilon_T}{1-\varepsilon_S} fk\right] \leq P_T^+ + P_S.$$

To prove the theorem for $\varepsilon \leq 1$, we set $\varepsilon_S = \varepsilon_T = \varepsilon/3$. Then $\frac{1+\varepsilon_T}{1-\varepsilon_S} \leq 1+\varepsilon$ and $\frac{1-\varepsilon_T}{1+\varepsilon_S} \geq 1-\varepsilon$. Therefore

$$\Pr\left[|Y^{\leq h_{(k)}} - fk| \geq \varepsilon fk\right] \leq P_T^- + P_T^+ + 2P_S \leq 8\exp(-\Omega(fk\varepsilon_T^2)) + O(\mathcal{E}) = 2\exp(-\Omega(fk\varepsilon_T^2)) + O(\mathcal{E}).$$

This completes the proof of (6). □

As for the problem of counting distinct elements in a stream, in the online setting we may again modify the algorithm above to obtain a more efficient sketch. Assuming that the elements from S appear in a stream, we again identify the smallest b such that the number of keys from S hashing below $1/2^b$, $X^{\leq 1/2^b}$, is at most k. We increment b by one whenever $X^{\leq 1/2^b} > k$ and in the end we return $Y^{\leq 1/2^b}/X^{\leq 1/2^b}$ as an estimator for f. The analysis of this modified algorithm is similar to the analysis provided above.

Acknowledgements

Research of all authors partly supported by Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation. Evangelos Kipouridis has also received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 801199.

References

[1] Aamand, A., Knudsen, J. B. T., Knudsen, M. B. T., Rasmussen, P. M. R., and Thorup, M. Fast hashing with strong concentration bounds. *CoRR abs/1905.00369* (2019). Accepted for STOC'20.

[2] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences 58*, 1 (1999), 209–223. Announced at STOC'96.

[3] Bar-Yossef, Z., Jayram, T. S., Kumar, R., Sivakumar, D., and Trevisan, L. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)* (2002), pp. 1–10.

[4] Bar-Yossef, Z., Kumar, R., and Sivakumar, D. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2002), pp. 623–632.

[5] Beyer, K. S., Haas, P. J., Reinwald, B., Sismanis, Y., and Gemulla, R. On synopses for distinct-value estimation under multiset operations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007* (2007), pp. 199–210.

[6] Blasiok, J. Optimal streaming and tracking distinct elements with high probability. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018* (2018), pp. 2432–2448.

[7] Broder, A. Z. On the resemblance and containment of documents. In Proc. Compression and Complexity of Sequences (SEQUENCES) (1997), pp. 21–29.

[8] Brody, J., and Chakrabarti, A. A multi-round communication lower bound for gap hamming and some consequences. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009* (2009), pp. 358–368.

[9] Cohen, E. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences 55*, 3 (1997), 441–453. Announced at STOC'94.

[10] Dietzfelbinger, M. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proc. 13th Symposium on Theoretical Aspects of Computer Science (STACS)* (1996), pp. 569–580.

[11] Durand, M., and Flajolet, P. Loglog counting of large cardinalities (extended abstract). In *Proc. 11th European Symposium on Algorithms (ESA)* (2003), pp. 605–617.

[12] Estan, C., Varghese, G., and Fisk, M. E. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw. 14*, 5 (2006), 925–937.

[13] Flajolet, P., and Martin, G. N. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences 31*, 2 (1985), 182–209.

[14] Flajolet, P., Éric Fusy, Gandouet, O., and Meunier, F. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In AOFA '07: Proceedings of the 2007 International Conference on Analysis of Algorithms* (2007), pp. 127–146.

[15] Gibbons, P. B. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy* (2001), pp. 541–550.

[16] Gibbons, P. B., and Tirthapura, S. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001, Heraklion, Crete Island, Greece, July 4-6, 2001* (2001), pp. 281–291.

[17] Indyk, P., and Woodruff, D. P. Tight lower bounds for the distinct elements problem. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings* (2003), pp. 283–288.

[18] Kane, D. M., Nelson, J., and Woodruff, D. P. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA* (2010), pp. 41–52.

[19] Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference, IMC 2003, Miami Beach, FL, USA, October 27-29, 2003* (2003), pp. 234–247.

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

[21] Pătraşcu, M., and Thorup, M. The power of simple tabulation-based hashing. *Journal of the ACM 59*, 3 (2012), Article 14. Announced at STOC'11.

[22] Pătraşcu, M., and Thorup, M. Twisted tabulation hashing. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2013), pp. 209–228.

[23] Thorup, M. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proc. 45th ACM Symposium on Theory of Computing (STOC)* (2013).

[24] Thorup, M. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proc. 54th IEEE Symposium on Foundations of Computer Science (FOCS)* (2013), pp. 90–99.

[25] Thorup, M., and Zhang, Y. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing 41*, 2 (2012), 293–331. Announced at SODA'04 and ALENEX'10.

[26] Wegman, M. N., and Carter, L. New classes and applications of hash functions. *Journal of Computer and System Sciences 22*, 3 (1981), 265–279. Announced at FOCS'79.

[27] Woodruff, D. P. Optimal space lower bounds for all frequency moments. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2004), pp. 167–175.