Lecture 8: Streaming and Sketching I

"In low-space, nobody can remember your stream."

We will follow for this chapter the (excellent) lecture notes by Amit Chakrabarti [AC], available at `https://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf`.

The Basic Setup

We will specifically focus on one-pass algorithms, unless specified otherwise. m denotes the length of the stream

$$\sigma = \langle a_1, \dots, a_m \rangle$$

where each a_i belongs to the universe $\mathcal{X}$ of size n. We do not impose any bound on the time complexity of our algorithms, but we will enforce that they use very little memory (space), with space complexity denoted by s. We will aim for

Note that this notation is swapped with respect to the previous lectures, in other to match the lecture notes.

$$s = o(\min(m, n))$$

and would love to use much less, ideally

$$s = O(\log m + \log n)$$

or, if not, $s = \mathrm{poly}(\log m, \log n)$. To do so, we will allow for randomised algorithms *and* approximation algorithms, where the quality of the approximation will be controlled by a parameter $\varepsilon > 0$, usually thought of as an (arbitrarily) small fixed constant.

The Majority Problem

Chapter 1 of [AC]

To begin, consider the (seemingly) very simple question of deciding whether there is *one* element that appears at least half the time in the stream: that is, some $j \in [n]$ whose *frequency* f_j, defined as

$$f_j := \sum_{i=1}^{m} \mathbb{1}_{a_i = j}$$

satisfies $f_j \geq \frac{m}{2}$. This is the Majority problem: at first glance, this seems very easy! Yet, spending some time thinking about it, you

Of course, there are *at most* two such elements, and at most one if we define the question as "is there some j such that $f_j > m/2$?" The issue is that we do not know *a priori* which one(s) of the n elements could be the majority element(s).

should convince yourself than it is surprisingly non-trivial to solve it using little memory.

The first algorithm we will see, due to Misra and Gries [42], is quite incredible in that regard: what it does is solving, *deterministically*, a related (and more general) version of this question, which we will return into more detail in the next chapter: the question of *frequency estimation*, which asks to approximate the frequency f_j's, not just decide which ones are at least $m/2$.

[42] Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982

Theorem 39. *The* Misra-Gries *algorithm (Algorithm 15) is a* deterministic *one-pass algorithm which, for any given parameter $\varepsilon \in (0,1]$, provides $\hat{f}_1, \dots, \hat{f}_n$ of all element frequencies such that*

$$f_j - \varepsilon m \leq \hat{f}_j \leq f_j, \qquad j \in [n]$$

with space complexity $s = O(\log(mn)/\varepsilon)$. (In particular, it can be used to solve the Majority *problem in two passes.)*

An interesting observation here is that of course the algorithm cannot compute explicitly n values $\hat{f}_1, \dots, \hat{f}_n$: this by itself would take $\Omega(n)$ space. What it does is *implicitly* do so, by only storing the values $\hat{f}_j$'s that are *non-zero* (and making sure there are very few of them, only $O(1/\varepsilon)$). Which makes sense, since we should not have many more non-zero estimates than this: after all, there can only be at most $1/\varepsilon$ element $j \in [n]$ such that $f_j \geq \varepsilon m$ (*i.e.*, which appear at least an ε fraction of the time in the stream)!

Algorithm 15: The Misra-Gries algorithm. Only store in A: if $A[j]$ does not exist, it is 0. Instead of a BST, one could use a linked list, for instance: this would have the same space complexity, but a larger update time at each step.

```
Input: Parameter $\varepsilon \in (0,1]$
1: $A \leftarrow \emptyset$          ▷ Use, e.g., a self-balancing binary search tree (BST)
2: Set $k \leftarrow \lceil 1/\varepsilon \rceil$
3: for all $1 \leq i \leq m$ do
4:     Get item $a_i = j \in [n]$
5:     if $A[j] > 0$ then                          ▷ $j$ is in the BST
6:         $A[j] \leftarrow A[j] + 1$
7:     else if $A[j] = 0$ and $|A| < k - 1$ then
8:         $A[j] \leftarrow 1$
9:     else if $A[j] = 0$ and $|A| = k - 1$ then
10:        for all $j' \in A$ do        ▷ Loop over all $j'$ such that $A[j'] > 0$
11:            $A[j'] \leftarrow A[j'] - 1$ ▷ If $A[j']$ reaches 0, remove it from $A$
Output: On query $j \in [n]$, return $A[j]$
```

Proof. First, note that since A never stores more $k = O(1/\varepsilon)$ elements, each of them taking $O(\log n + \log m)$ bits (for the index of the element, and its current count), the space s is bounded as

$$s = O\left(\frac{\log(mn)}{\varepsilon}\right)$$

as claimed.

To prove correctness, fix any $j \in [n]$. Since $A[j]$ can only be incremented (on Line 8 or 6) when element j appears in the stream, we have, at the end of the stream,

$$\hat{f}_j = A[j] \leq f_j \,.$$

For the other inequality (the lower bound), we need to get a grasp on the number of times $A[j]$ is decremented, which can only happen in Line 11. Every time this line is reached, this means that (1) nothing else happens in this step (no increment to $A[j]$), and (2) exactly $k-1$ other counters are decremented (in the **for all** loop).

We can see (1), conceptually, as one increment to $A[j]$ immediately followed by a decrement to $A[j]$: thinking of it this way allows us to say that $A[j]$ is incremented every time j appears in the stream – but sometimes, it is decremented immediately after as well, and lets us combine (1) and (2) to say that every time Line 11 is reached, *exactly* k decrements are performed in A. Given that every increment uniquely corresponds to one of the m steps,this means that each execution of Line 11 corresponds to a disjoint chunk of k steps: when the increments to the $A[j']$'s had happened. But there are only m steps in total, so if each decrement "burns" k of them, there can be at most $\frac{m}{k}$ decrements steps! This shows that

$$\hat{f}_j = A[j] \geq f_j - \frac{m}{k}\,,$$

which, given the value of k, implies $\hat{f}_j \geq f_j - \varepsilon m$.

To see how the "In particular" statement follows, consider applying the Misra-Gries algorithm with $\varepsilon = 1/4$, and at the end of the first pass considering the set $S \subseteq [n]$ of elements for which $\hat{f}_j \geq \frac{1}{4}m$. If j^* is a majority element, then

$$\hat{f}_j \geq f_j - \varepsilon m \geq \frac{1}{2}m - \frac{1}{4}m = \frac{1}{4}m\,,$$

so $j^* \in S$. Conversely, if $j \in S$, then

$$f_j \geq \hat{f}_j \geq \frac{1}{4}m\,,$$

and so there can be at most $\frac{m}{(1/4)m} = 4$ elements in S. So keeping S in memory only takes $4 \cdot O(\log n) = O(\log n)$ bits. Then, all that remains to do is, in the *second* pass, count *exactly* the number of times each elements $j \in S$ appears, and check if that's at least $m/2$. Each such counter takes $O(\log m)$ bits, and there are only (at most) 4 counters to maintain now. □

The Approximate Counting Problem

Chapter 4 of [AC]

We will describe and analyse the Morris Counter algorithm, due to, well, Morris [43], which provides a constant-factor estimate of the number of elements of the stream: that is, an F_1 estimator. Put

[43] Robert H. Morris Sr. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978

```
1: $x \leftarrow 0$
2: for all $1 \le i \le m$ do
3:     Get item $a_i \in \{0,1\}$
4:     if $a_i = 1$ then
5:         $r_i \leftarrow \mathrm{Bern}(1/2^x)$        ▷ Independent of previous choices.
6:         $x \leftarrow x + r_i$
7: return $\widehat{d} \leftarrow 2^x - 1$
```

Algorithm 16: The Morris Counter algorithm.

differently, at each time step $1 \le t \le m$, we are told if some event happened ($a_i = 1$) or not ($a_i = 0$): the goal is to estimate how many events happened in total, *i.e.*, the number $d = \sum_{i=1}^{m} a_i$.

The space complexity is a little annoying to bound: we *expect* x to never exceed $\log_2 m$, since d is at most m by definition and we should have $2^x \approx d$. But there is a very, very small chance that all Bernoullis turn out to be 1, in which case x could become as big as m! This would make no sense, and also mean we would need $O(\log m)$ bits to store x, exactly what we do not want to pay. *However*, one can show that with overwhelming probability x remains at most $O(\log m)$, and so the space complexity required is only $s = O(\log\log m)$.

If x ever exceeds this value, the algorithm can just abort: this only adds a vanishing small amount to the probability of error.

The proof of correctness of Algorithm 16 relies on the key lemma below, analysing the expectation and variance of $\widehat{d}$:

Lemma 39.1. *The random variable $\widehat{d}$ defined in Algorithm 16 satisfies*

$$\mathbb{E}\left[\widehat{d}\right] = d$$

and

$$\mathrm{Var}[\widehat{d}] = \frac{d(d-1)}{2}$$

Proof. Define C_i, for $1 \le i \le m$, as the value of 2^x in Algorithm 16 at the end of step i; so that $C_0 = 2^0 = 1$ and $\widehat{d} = C_m - 1$.

For any $1 \le i < m$, we then have

$$C_{i+1} = \begin{cases} 2 \cdot C_i & \text{if } a_{i+1} = 1 \text{ and } r_{i+1} = 1 \\ C_i & \text{otherwise} \end{cases}$$

which we can rewrite as $C_{i+1} = (1 + a_{i+1} r_{i+1}) C_i$. Recalling that $r_{i+1} \sim \mathrm{Bern}(1/C_i)$ gives us

$$\mathbb{E}[\,C_{i+1} \mid C_i\,] = (1 + a_{i+1}\mathbb{E}[\,r_{i+1} \mid C_i\,]) \cdot C_i = \left(1 + \frac{a_{i+1}}{C_i}\right) \cdot C_i = C_i + a_{i+1}$$

and, by the Law of Total Expectation,

$$\mathbb{E}[C_{i+1}] = \mathbb{E}[\mathbb{E}[\,C_{i+1} \mid C_i\,]] = \mathbb{E}[C_i] + a_{i+1}\,.$$

This gives us

$$\mathbb{E}[C_m] = \mathbb{E}[C_0] + \sum_{i=0}^{m-1} (\mathbb{E}[C_{i+1}] - \mathbb{E}[C_i]) = 1 + \sum_{i=0}^{m-1} a_{i+1} = 1 + d$$

showing that $\mathbb{E}\left[\widehat{d}\right] = d$. The above actually showed the more general statement that

$$\mathbb{E}[C_i] = 1 + \sum_{j=1}^{i} a_j, \qquad 1 \leq i \leq m, \tag{59}$$

which we will use very soon.

To compute the variance, we similarly analyse $\mathbb{E}\left[C_m^2\right]$: For any $1 \leq i < m$,

$$\begin{aligned}
\mathbb{E}\left[C_{i+1}^2 \;\middle|\; C_i \right] &= \mathbb{E}\left[(1 + a_{i+1} r_{i+1})^2 \;\middle|\; C_i \right] \cdot C_i^2 \\
&= \left(1 + \frac{a_{i+1}(2 + a_{i+1})}{C_i} \right) \cdot C_i^2 \\
&= C_i^2 + a_{i+1}(2 + a_{i+1}) C_i
\end{aligned}$$

where the second equality follows from expanding the square and computing the expectation. Again, by the Law of Total Expectation,

$$\begin{aligned}
\mathbb{E}\left[C_{i+1}^2\right] &= \mathbb{E}\left[\mathbb{E}\left[C_{i+1}^2 \;\middle|\; C_i \right]\right] = \mathbb{E}\left[C_i^2\right] + a_{i+1}(2 + a_{i+1})\mathbb{E}[C_i] \\
&= \mathbb{E}\left[C_i^2\right] + a_{i+1}(2 + a_{i+1})\left(1 + \sum_{j=1}^{i} a_j\right) \qquad \text{(By Eq. (59))} \\
&= \mathbb{E}\left[C_i^2\right] + 3a_{i+1} \sum_{j=1}^{i+1} a_j
\end{aligned}$$

where that last step is completely magical, but "immediate in hindsight" by checking the two possible cases: $a_{i+1}(2 + a_{i+1})\left(1 + \sum_{j=1}^{i} a_j\right) = 3a_{i+1}\left(a_{i+1} + \sum_{j=1}^{i} a_j\right)$ for both $a_{i+1} = 0$ and $a_{i+1} = 1$. This gives us

$$\begin{aligned}
\mathbb{E}\left[C_m^2\right] &= \mathbb{E}\left[C_0^2\right] + 3 \sum_{i=0}^{m-1} a_{i+1} \sum_{j=1}^{i+1} a_j = 1 + 3 \sum_{i=1}^{m} \sum_{j=1}^{i} a_i a_j \\
&= 1 + 3 \cdot \frac{1}{2} \left(\left(\sum_{i=1}^{m} a_i \right)^2 + \sum_{i=1}^{m} a_i^2 \right) \\
&= 1 + 3 \cdot \frac{1}{2} \left(d^2 + d \right)
\end{aligned}$$

(recalling, for the last step, that $a_i^2 = a_i$ for all i, since $a_i \in \{0, 1\}$). Since $\mathbb{E}[C_m]^2 = d + 1$, we finally get

$$\mathrm{Var}[C_m] = 1 + 3 \cdot \frac{1}{2}\left(d^2 + d\right) - (d+1)^2 = \frac{d^2 - d}{2},$$

as claimed. □

While this $\Theta\left(d^2\right)$ variance by itself is not good enough to obtain an accurate estimate with high constant probability using Chebyshev's inequality, averaging $k = O(1/\varepsilon^2)$ independent copies of the Morris Counter enables us to bring down the variance by this factor, leading to a $(1 + \varepsilon)$-estimate with high (constant) probability. Using the median trick afterwards (running $T = O(\log(1/\delta))$ copies of this improved-variance algorithm, and taking the median result) gives a high-probability result, leading to the following:

Theorem 40. *The medians-of-means version of the* MORRIS COUNTER *is a* randomised *one-pass algorithm which, for any given parameters* $\varepsilon, \delta \in (0,1]$*, provides an estimate* $\widehat{d}$ *of the number* d *of non-zero elements of the stream such that*

$$\Pr\left[(1-\varepsilon)d \leq \widehat{d} \leq (1+\varepsilon)d \right] \geq 1-\delta$$

with space complexity

$$s = O\left(\frac{\log\log m}{\varepsilon^2} \cdot \log\frac{1}{\delta}\right)$$

that is, doubly logarithmic *in* m.

Again, maintaining an exact counter would take $O(\log m)$ bits: this is an exponential improvement! And yet, *we can do better.* Instead of using the median-of-means technique, we can be more careful about the algorithm itself: where we incremented x with probability $1/2^x$ and returned $2^x - 1$, we will, for a suitable choice of $\alpha = \alpha(\varepsilon, \delta) > 0$, increment it with with probability $1/(\alpha(1+\alpha)^x)$ and return $(1+\alpha)^x - 1$. We will see the details in the tutorial, leading to this (much) improved bound:

Theorem 41. *The "careful" version of* MORRIS COUNTER *is a* randomised *one-pass algorithm which, for any given parameters* $\varepsilon, \delta \in (0,1]$*, provides an estimate* $\widehat{d}$ *of the number* d *of non-zero elements of the stream such that*

$$\Pr\left[(1-\varepsilon)d \leq \widehat{d} \leq (1+\varepsilon)d \right] \geq 1-\delta$$

with space complexity

$$s = O\left(\log\log m + \log\frac{1}{\varepsilon} + \log\frac{1}{\delta}\right)$$

that is, doubly logarithmic in m and logarithmic in $1/\varepsilon$.

The Distinct Elements Problem

Chapters 2 and 3 of [AC]

We start this section with the TIDEMARK algorithm, due to Alon, Matias and Szegedy (AMS), which provides a constant-factor estimate of the number of distinct elements of the stream: that is, an F_0 estimator. In this section, we define d as this number of distinct elements, *i.e.*,

$$d = \sum_{j \in [n]} \mathbb{1}_{f_j > 0}$$

In what follows, for a given positive integer k, $\mathrm{zeros}(k)$ denotes the largest power of 2 which divides k, or, equivalently, the number of trailing zeroes in the binary representation of k. The space complexity of Algorithm 17 is

$$s = O(\log n)$$

as all that is needed is storing the hash function ($O(\log n)$ bits, for a suitable strongly universal hash family) and $1 \leq z \leq \log_2 n$

```
1: Pick $h\colon [n] \to [n]$ from a strongly universal hashing family
2: $z \leftarrow 0$
3: for all $1 \leq i \leq m$ do
4:     Get item $a_i \in [n]$
5:     if $\mathrm{zeros}(h(a_i)) \geq z$ then
6:         $z \leftarrow \mathrm{zeros}(h(a_i))$
7: return $\widehat{d} \leftarrow \sqrt{2} \cdot 2^z$
```

Algorithm 17: The TIDEMARK algorithm

(which takes $O(\log \log n)$ bits). To analyse the correctness of the algorithm (and the quality of the estimate it outputs), define the random variables

$$Y_r := \sum_{\substack{j \in [n] \\ f_j > 0}} \mathbb{1}_{\mathrm{zeros}(h(j)) \geq r}, \qquad r \geq 0$$

(where the randomness is over the choice of the hash function h). One can check that, by definition,

$$Y_r \geq 1 \Leftrightarrow z \geq r$$

for every integer $r \geq 0$. Moreover,

$$\mathbb{E}[Y_r] = \sum_{\substack{j \in [n] \\ f_j > 0}} \Pr[\,\mathrm{zeros}(h(j)) \geq r\,] = \sum_{\substack{j \in [n] \\ f_j > 0}} \frac{1}{2^r} = \frac{d}{2^r}$$

where we used the fact that each $h(j)$ is uniformly distributed to write that $\Pr[\,\mathrm{zeros}(h(j)) \geq r\,] = \Pr[\,2^r \text{ divides } h(j)\,] = \frac{1}{2^r}$. Similarly, using pairwise independence,

$$\mathrm{Var}[Y_r] = \sum_{\substack{j \in [n] \\ f_j > 0}} \mathrm{Var}[\mathbb{1}_{\mathrm{zeros}(h(j)) \geq r}] \leq \sum_{\substack{j \in [n] \\ f_j > 0}} \frac{1}{2^r} = \frac{d}{2^r}$$

the inequality using $\mathrm{Var}[X] \leq \mathbb{E}[X^2]$ and the fact that $X^2 = X$ when X is an indicator random variable. Using these two facts, for every $r \geq 0$,

- $\Pr[\,z \geq r\,] = \Pr[\,Y_r \geq 1\,] \leq \mathbb{E}[Y_r] = \frac{d}{2^r}$ by Markov;
- $\Pr[\,z \leq r\,] = \Pr[\,Y_{r+1} = 0\,] \leq \frac{\mathrm{Var}[Y_{r+1}]}{\mathbb{E}[Y_{r+1}]^2} \leq \frac{2^{r+1}}{d}$ by Chebyshev,

using that $\Pr[\,Y_{r+1} = 0\,] \leq \Pr[\,|Y_{r+1} - \mathbb{E}[Y_{r+1}]| \geq \mathbb{E}[Y_{r+1}]\,]$. This is all we need! Setting $C := 3\sqrt{2}$,

$$\Pr\left[\,\widehat{d} \geq C \cdot d\,\right] \leq \Pr\left[\,z \geq \left\lceil \log_2(C \cdot d/\sqrt{2}) \right\rceil\,\right] \leq \frac{\sqrt{2}d}{Cd} = \frac{1}{3}$$

while

$$\Pr\left[\,\widehat{d} \leq d/C\,\right] \leq \Pr\left[\,z \leq \left\lfloor \log_2(d/(\sqrt{2}C)) \right\rfloor\,\right] \leq \frac{2d}{\sqrt{2}Cd} = \frac{1}{3}$$

Combining the above with the median trick, we readily get:

Theorem 42. *The (median trick version of the)* Tidemark *(AMS) algorithm is a* randomised *one-pass algorithm which, for any given parameter $\delta \in (0,1]$, provides an estimate $\widehat{d}$ of the number d of distinct elements of the stream such that, for some absolute constant $C > 0$,*

$$\Pr\left[\frac{1}{C} \cdot d \leq \widehat{d} \leq C \cdot d\right] \geq 1 - \delta$$

with space complexity

$$s = O\left(\log n \cdot \log \frac{1}{\delta}\right).$$

This is not bad, but can we achieve estimation factor arbitrarily close to one, say, $1+\varepsilon$? The answer is yes: the following algorithm, due to Bar-Yossef, Jayram, Kumar, Sivakumar and Trevisan (BJKST), does exactly that.

Algorithm 18: The BJKST algorithm

```
Input: Parameter $\varepsilon \in (0,1]$
1: Set $k \leftarrow O(\log^2 n/\varepsilon^4)$, $T \leftarrow \Theta(1/\varepsilon^2)$
2: Pick $h\colon [n] \to [n]$ from a strongly universal hashing family
3: Pick $g\colon [n] \to [k]$ from a strongly universal hashing family

4: $z \leftarrow 0$, $B \leftarrow \emptyset$
5: for all $1 \leq i \leq m$ do
6:     Get item $a_i \in [n]$
7:     if zeros$(h(a_i)) \geq z$ then
8:         $B \leftarrow B \cup \{(g(a_i), \text{zeros}(h(a_i)))\}$
9:         while $|B| \geq T$ do
10:            $z \leftarrow z + 1$
11:            Remove every $(a,b)$ with $b < z$ from $B$
12: return $|B| \cdot 2^z$
```

Theorem 43. *The (median trick version of the)* BJKST *algorithm is a* randomised *one-pass algorithm which, for any given parameters $\varepsilon, \delta \in (0,1]$, provides an estimate $\widehat{d}$ of the number d of distinct elements of the stream such that, for some absolute constant $C > 0$,*

$$\Pr\left[(1-\varepsilon) \cdot d \leq \widehat{d} \leq (1+\varepsilon)d\right] \geq 1 - \delta$$

with space complexity

$$s = O\left(\left(\log n + \frac{\log(1/\varepsilon) + \log\log n}{\varepsilon^2}\right) \cdot \log \frac{1}{\delta}\right).$$

This is pretty good, but... *Is it optimal?*