NOTES: STREAMING ALGORITHMS

HANI T. DAWOUD

Contents

1. (Turnstile) Model

A stream $\sigma = \langle \sigma_1, \ldots, \sigma_m \rangle$, where $\sigma_i \in [n]$.
Additions/deletions (i, Δ).
$\mathbf{f} = (f_1, \ldots, f_n)$.

Find algorithm $\mathcal{A}$ that approximates the function ϕ.
Use $s = o(\min\{m, n\})$ bits of memory; ideally, $s = O(\log m + \log n)$.
Fast time-per-element processing.

Multiplicative (ϵ, δ)-approximation: $\mathbb{P}\left[|\mathcal{A}(\sigma) - \phi(\sigma)| > \epsilon\phi(\sigma)\right] \le \delta$.
Additive (ϵ, δ)-approximation: $\mathbb{P}\left[|\mathcal{A}(\sigma) - \phi(\sigma)| > \epsilon\right] \le \delta$.

Algorithms assume no advance knowledge of m, while lower bounds hold even if m is known a priori[1].

2. Some Functions Are Easy

Min, Max, Sum, ... are easy. What makes other functions harder to compute?

3. Kadane Algorithm: Maximum Subarray Problem

- Assume an array A with n elements.
- For each index $i \in [n]$, find the maximum subarray *ending* at i, as you go. How?

Claim 3.1. *For each $i \in [n]$,* $\mathsf{MaxHere}$[2] *is the maximum value over the subarrays ending at i.*

Date: August 21, 2015.

[1] When m is unknown, you claim the lower bound is (partly) due to lack of knowledge. When m is known, you have no excuse (i.e., your task of claiming a lower bound is harder).

[2] Two algorithm variants: $\mathsf{MaxHere} \ge 0$: empty array allowed; $\mathsf{MaxHere} \ge A_i$: empty array not allowed.

Proof. By induction. Let $\mathsf{MaxHere}_i$ be $\mathsf{MaxHere}$ at the i-th iteration.

Base case is easy. Now assume $\mathsf{MaxHere}_{i-1}$ is maximum and show $\mathsf{MaxHere}_i$ is optimal too.

Empty: $\mathsf{MaxHere}_{i-1} \geq 0$ by the algorithm. (1) $A_i \geq 0$ and so $\mathsf{MaxHere}_i$ is maximum; (2) $A_i < 0$ and $|A_i| < \mathsf{MaxHere}_{i-1}$ and so $\mathsf{MaxHere}_i$ is maximum; (3) $A_i < 0$ and $|A_i| > \mathsf{MaxHere}_{i-1}$ and so empty is maximum.

Non-empty (at least A_i in): $\mathsf{MaxHere}_{i-1} \geq A_{i-1}$ by the algorithm. (1) $\mathsf{MaxHere}_{i-1} + A_i < A_i$ implies $\mathsf{MaxHere}_{i-1}$ is negative and hence can not be included; (2) $\mathsf{MaxHere}_{i-1} + A_i \geq A_i$ implies $\mathsf{MaxHere}_{i-1}$ is non-negative and hence must be in. □

Corollary 3.2. *After the n-th iteration,* $\mathsf{MaxSoFar}$ *is the value of the maximum subarray.*

Proof. All subarrays can be identified by their ending index, $i \in [n]$. $\mathsf{MaxSoFar}$ tracks the maximum. □

4. Misra-Gries Algorithm: Items Frequencies Deterministically

- $k-1$ counters. The k-th element triggers a "decrement" that includes itself and everyone is kicked.
- $\hat{f}_i$ is *only* incremented upon occurrences of i. So, $\hat{f}_i \leq f_i$.
- How many decrements to a given $i \in [n]$ can we have? At most, i is decremented each time a "decrement" occurs. How many "decrements" can we have?
- Each "decrement" will "touch" k *distinct* items in the multiset, including the k-th element. So, *at most* we have m/k "decrements", as then all elements will have been "touched". Thus, $\hat{f}_i \geq f_i - m/k$.
- The $\hat{m}$[3] remaining items are *never* "touched". This means we have $m - \hat{m}$ "touchable" items. So, *at most* we have $(m-\hat{m})/k$ "decrements". Thus, $\hat{f}_i \geq f_i - (m-\hat{m})/k$.
- Synopsis of MG is mergeable.
  1. Merge, then apply MG on the $\hat{m}(\sigma_1) + \hat{m}(\sigma_2)$ items using $k-1$ counters as before[4].
  2. We have *at most* $(\hat{m}(\sigma_1) + \hat{m}(\sigma_2) - \hat{m}(\sigma_1 \circ \sigma_2))/k$ decrements to a given $i \in [n]$. Done!
  3. Total decrements is *at most*

$$\frac{m(\sigma_1) - \hat{m}(\sigma_1)}{k} + \frac{m(\sigma_2) - \hat{m}(\sigma_2)}{k} + \frac{\hat{m}(\sigma_1) + \hat{m}(\sigma_2) - \hat{m}(\sigma_1 \circ \sigma_2)}{k} = \frac{m(\sigma_1) + m(\sigma_2) - \hat{m}(\sigma_1 \circ \sigma_2)}{k}.$$

- MG works in practice because typical frequency distributions have few heavy elements (Zipf law).

5. Maximum Matching in Unweighted Graph

- Let M' be a maximum matching.
- M is matching. So $|M'| \geq |M|$.
- M is maximal. So, *all* edges in M' share endpoints in M[5]; otherwise maximality of M is contradicted.
- How to maximize edges in M' *provided they all share endpoints in* M? Take an edge from each endpoint in M. We have $2|M|$ endpoints. Thus $|M| \leq |M'| \leq 2|M|$.

6. Frequency Moments

Theorem 6.1. *F_0 & F_2 can be approximated to within $(1 \pm \epsilon)$ factor with probability at least $1-\delta$ in space*

$$O\left(\frac{1}{\epsilon^2}(\log n + \log m)\log\frac{1}{\delta}\right). \tag{6.1}$$

6.1. Key Ideas in Randomized Streaming Algorithms.

1. Find an unbiased estimator.
2. Reduce the variance of the estimator by taking averages and medians. Aggregate many independent copies of the estimator in parallel to get an accurate estimate with high probability.
3. Estimators with large variance necessitate a large number of independent copies to obtain a good approximation

---

[3] $\hat{m}$ is the sum of all counters maintained *after* the algorithm has processed the input stream.

[4] This is more elementary than merging, and then decrementing the $(k+1)$-th largest counter C from all elements. The latter is easier as an algorithm. The former explains why decrementing C works: it ensures at most k counters remain and hence satisfies $\hat{m}(\sigma_1) + \hat{m}(\sigma_2) - \hat{m}(\sigma_1 \circ \sigma_2) \geq Ck$, as in (b).

[5] This may include edges of M themselves.

6.2. F_2 **Algorithm.** Let $h : U \to \{\pm\}$ be a random but consistent hash function. Initialize $Z = 0$. For each $j \in U$, add $h(j)$ to Z. Return $X = Z^2$.

Remark 6.2. *Since h is fixed once and for all before data stream arrives, $j \in U$ is treated consistently every time it shows up. Z is either incremented every time j shows up or is decremented. So j contributes $h(j)f_j$ to the final value of Z.*

6.3. **Analysis.** X is an unbiased estimator,

$$\begin{aligned}
\mathbb{E}X &= \mathbb{E}Z^2 && (6.2)\\
&= \mathbb{E}\left(\sum_{j\in U} h(j)f_j\right)^2 && (6.3)\\
&= \mathbb{E}\left(\sum_{j\in U} h(j)^2 f_j^2 + 2\sum_{j<\ell} h(j)f_j h(\ell)f_\ell\right) && (6.4)\\
&= \sum_{j\in U} f_j^2 + 2\sum_{j<\ell} f_j f_\ell \mathbb{E}\left(h(j)h(\ell)\right) && (6.5)\\
&= F_2. && (6.6)
\end{aligned}$$

Remark 6.3. (6.6) *used the property that h is pairwise-independent. In particular that all four sign patterns for $(h(j), h(\ell))$ are equally likely to give $\mathbb{E}\left(h(j)h(\ell)\right) = 0$. Another possibly simpler argument is $\mathbb{E}\left(h(j)h(\ell)\right) = \mathbb{E}h(j)\mathbb{E}h(\ell)$ because of pairwise-independence.*

Remark 6.4. *$\pm$ in Z ensures the* cross terms *$h(j)f_jh(\ell)f_\ell$ in the estimator X cancel out in expectation.*

X is not guaranteed to be close to $\mathbb{E}X$ with high probability. So take many independent copies of X

$$Y = \frac{1}{t}\sum_{i=1}^{t} X_i. \tag{6.7}$$

Y is still an unbiased estimator,

$$\mathbb{E}Y = \frac{1}{t}\sum_{i=1}^{t} \mathbb{E}X_i = F_2. \tag{6.8}$$

Averaging reduces the variance by a factor equal to the number of copies t

$$\mathrm{Var}\left(Y\right) = \mathrm{Var}\left(\frac{1}{t}\sum_{i=1}^{t} X_i\right) = \frac{1}{t^2}\sum_{i=1}^{t}\mathrm{Var}\left(X_i\right) = \frac{\mathrm{Var}\left(X\right)}{t}. \tag{6.9}$$

The number of copies t we need is governed by $\mathrm{Var}\left(X\right)$. So we have to compute it.

$$\mathrm{Var}\left(X\right) = \mathbb{E}X^2 - \left(\mathbb{E}X\right)^2. \tag{6.10}$$

$$\begin{aligned}
\mathbb{E}X^2 &= \mathbb{E}\left(\sum_{j\in U} h(j)f_j\right)^4 && (6.11)\\
&= \mathbb{E}\left(\sum_{j\in U} h(j)^4 f_j^4 + 6\sum_{j\neq\ell} h(j)^2 f_j^2 h(\ell)^2 f_\ell^2\right) && (6.12)\\
&= \sum_{j\in U} f_j^4 + 6\sum_{j<\ell} f_j^2 f_\ell^2 && (6.13)
\end{aligned}$$

Remark 6.5. (6.12) *also used the* 4*-wise independence of* h. *In particular that for every set of four distinct universe elements, their* 16 *possible sign patterns under* h *are equally likely. Or that* $\mathbb{E}jk\ell p = \mathbb{E}j\mathbb{E}k\mathbb{E}\ell\mathbb{E}p$.

Remark 6.6. *Expanding the right side of* (6.11) *gives* $|U|^4$ *monomials of different types:*

- $(jjjj) \forall j \in U$ *which gives* $\sum_{j\in U} h(j)^4 f_j^4$.
- $(jjkk)$ *arises in* $\binom{4}{2} = 6$ *different ways. Summing over all* $j \neq k \in U$ *gives* $6\sum_{j\neq\ell} h(j)^2 f_j^2 h(\ell)^2 f_\ell^2$.
- *The rest of monomials must contain singleton* j *for some* $j \in U$, *in which case we have* $\mathbb{E}h(j) = 1/2(-1) + 1(1) = 0$ *and the whole monomial is* 0.

Recall that $F_2 = \sum_{j\in U} f_j^2$. So

$$F_2^2 = \sum_{j\in U} f_j^4 + 2\sum_{j\neq\ell} f_j^2 f_\ell^2 \tag{6.14}$$

and hence

$$\mathbb{E}X^2 \leq 3F_2^2. \tag{6.15}$$

Recalling (6.10),

$$\mathrm{Var}\,(X) \leq 2F_2^2. \tag{6.16}$$

This is good. The standard deviation of the basic estimator X is in the same ballpark as its expectation. A constant (depending on ϵ and δ only) number of copies is good enough for our purpose. Now, using Chebyshev's inequality, we show that Y approximates F_2 with high probability.

$$\mathbb{P}\left[|Y - \mathbb{E}Y| > c\right] \leq \frac{\mathrm{Var}\,(Y)}{c^2} \leq \frac{\mathrm{Var}\,(X)}{t\cdot c^2} \leq \frac{2F_2^2}{t\cdot c^2}. \tag{6.17}$$

Since we seek a $(1\pm\epsilon)$-approximation, set $c = \epsilon F_2$. Since we want the right-hand side of (6.17) to be δ, set $t = 2/\epsilon^2\delta$. So we

$$\mathbb{P}\left[Y \in (1\pm\epsilon)F_2\right] \geq 1-\delta. \tag{6.18}$$

Let's make sure we're clear on the final algorithm.

(1) Choose $h_1, \ldots, h_t : U \to \{\pm 1\}$, where $t = \frac{2}{\epsilon^2\delta}$.
(2) Initialize $Z_i = 0$ for $i = 1, 2, \ldots, t$.
(3) For each $j \in U$, add $h_i(j)$ to Z_i for every $i = 1, 2, \ldots, t$.
(4) Return the average of the Z_i^2's.

How much space is required? There's a factor of $\frac{2}{\epsilon^2\delta}$ for running the t copies. How much space does each of these need? Each Z_i requires $O(\log m)$ bits. But we have to also store h_i. Recall that h_i is random and *consistent*. So once we choose a sign $h_i(j)$ for j we need to remember it forever. But then we need to remember one bit for each of the possible $\Omega(n)$ elements.

Fortunately, as noted before, the entire analysis has relied only on 4-wise independence. Happily, there're small families of simple hash functions that possess this property. Such a hash function is specified with $O(\log n)$ bits.

Putting it all together we get a space bound of

$$O\left(\frac{1}{\epsilon^2\delta}\cdot(\log m + \log n)\right). \tag{6.19}$$

6.4. **Further Optimizations.** So far we averaged t copies of X to achieve two conceptually different things: to improve the approximation ratio to $(1 \pm \epsilon)$ for which we suffered an $\frac{1}{\epsilon^2}$ factor, and to improve the success probability to $1-\delta$ for which we suffered an additional $\frac{1}{\delta}$. The smarter implementation first uses $\approx \frac{1}{\epsilon^2}$ copies to obtain an approximation of $(1 \pm \epsilon)$ with probability at least $\frac{2}{3}$ (say). To boost the success probability from $\frac{2}{3}$ to $1-\delta$, it is enough to run $\approx \log \frac{1}{\delta}$ different copies and then take the *median*. Since we expect at least two-thirds of these estimates to lie in the interval $(1 \pm \epsilon)F_2$, it's very likely that the median of them lies in this interval (Chernoff bound).

Remark 6.7. *The* $\log m$ *term in* (6.19) *can be improved to* $\log \log m$. *We don't need to count the* Z_i*'s exactly, only approximately and with high probability. This relaxed counting problem can be solved using Morris's algorithm, which can be implemented as a streaming alorithm that uses* $O(\epsilon^{-2} \log \log m \log \frac{1}{\delta})$.

7. Lower Bounds

- Small-space streaming algorithms imply low-communication 1-way protocols.
- The latter don't exist.

These two steps require a Boolean function that

- Can be reduced to a streaming problem.
- Does not admit a low-communication one-way protocol.