Lecture 9: Streaming Algorithms

1 Introduction to Streaming Algorithms

This lecture provides an introduction to streaming, sampling, and estimation algorithms.

1.1 Setting

We are given a set of objects or tokens that arrive one by one in a stream (online fashion):

$$e_1, e_2, \ldots, e_m$$

Examples of stream elements e_i:

- A number from a set $[n] = \{1, \ldots, n\}$.
- An edge (u, v) in a graph.
- A vector or point in $\mathbb{R}^d$.
- A row or column of a matrix.
- A matrix itself.

Assumption: The stream length m is very large and possibly unknown. We cannot afford to store all elements $e_1, \ldots, e_m$ in memory. We have a limited memory space, say B units (where one token takes one unit of space).

Question: What functions of the input stream can we compute? This depends on the available space B. There are various trade-offs between efficiency, accuracy, etc.

1.2 Frequency Vectors

A simple but powerful setting is when each element e_i is an integer from a large range, say $[n] = \{0, 1, \ldots, n-1\}$. For example, consider a stream: $5, 3, 2, 2, 10, 5, 90, \ldots$ This stream implicitly defines a frequency vector $f \in \mathbb{N}^n$ that starts at all zeros. For each incoming element e_i, we effectively perform an update $f_{e_i} \leftarrow f_{e_i} + 1$. After the stream $5, 3, 2, 2, 10, 5, 90$, the frequency vector would have $f_2 = 2, f_3 = 1, f_5 = 2, f_{10} = 1, f_{90} = 1$, and all other entries would be zero.

2 Frequency Moment Estimation

Given a stream $e_1, \ldots, e_m$, let f_i be the frequency of element $i \in [n]$ at the end of the stream. We want to estimate or compute the k^{th} frequency moment, denoted by F_k.

$$F_k = \sum_{i=1}^{n} f_i^k$$

(Note: Sometimes this is defined as the L_k-norm of the frequency vector, i.e., $(\sum_{i=1}^{n} f_i^k)^{1/k}$).

- **k=0:** $F_0 = \sum_{f_i > 0} 1$. This is the number of **distinct elements** in the stream.
- **k=1:** $F_1 = \sum f_i = m$. This is the total length of the stream.
- **k=2:** $F_2 = \sum f_i^2$. This is related to the squared Euclidean norm (L_2 norm) of the frequency vector and is very important in many applications.

- **k=∞:** $F_\infty = \max_i f_i$. This is the frequency of the most frequent element, used in finding "heavy hitters".

If we can store the entire stream or the frequency vector f, these problems are trivial. The main question is: can we compute or estimate these moments with memory $B \ll n$? The answer is yes, often with $B = \tilde{O}(1)$ (polylogarithmic space), but this requires randomization and approximation. It can be shown that without one of these, it's not feasible with sub-linear ($o(n)$) memory.

3 Reservoir Sampling

3.1 Sampling One Element

Given a stream $e_1, \ldots, e_m$ where m is unknown, we want to pick one sample uniformly at random.

Algorithm 1 Reservoir Sampling (1 element)

```
m ← 0
s ← null
while stream has next element e do
    m ← m + 1
    With probability 1/m, set s ← e.
end while
output s
```

3.2 Sampling k Elements

To get k samples without replacement, we can extend the algorithm.

Algorithm 2 Reservoir Sampling (k elements)

```
S ← ∅
m ← 0
while stream has next element e_m do
    m ← m + 1
    if m ≤ k then
        Add e_m to S.
    else
        With probability k/m:
        Replace a random element of S with e_m.
    end if
end while
output S
```

Exercise 1. *Prove the correctness of the k-element Reservoir Sampling algorithm.*

4 Distinct Element Estimation (F_0)

The problem is to find the number of distinct elements in a stream $e_1, \ldots, e_m$ where each $e_i \in [n]$.

- **Example:** In the stream $1, 2, 5, 2, 3, 5, 5, 1$, the distinct elements are $\{1, 2, 3, 5\}$, so $F_0 = 4$.
- **Application:** In a high-speed internet switch, packets are tuples of (source, destination, content). Source and destination are IP addresses (e.g., 128-bit values, so $n = 2^{128}$). A key question is: how many distinct source addresses have sent packets?

An offline solution could use a hash table or dictionary to store distinct elements seen so far. This would require $\Theta(F_0)$ space, which can be very large and is not suitable for a streaming model where F_0 is unknown and potentially massive.

It can be shown that any deterministic algorithm that gives a $(1 \pm \epsilon)$ approximation requires $\Omega(n)$ space. However, with randomization, we can get a $(1 \pm \epsilon)$ estimate with probability at least $(1 - \delta)$ in $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} \cdot \text{polylog}(n))$ space.

4.1 Hashing-Based Algorithm (Flajolet-Martin)

Assume we have access to an "ideal" hash function $h : [n] \to [0, 1]$ that maps each item to a uniformly random real number.

Algorithm 3 Basic Distinct Elements Hashing

1: Pick a random hash function $h : [n] \to [0, 1]$.
2: $z \leftarrow \infty$.
3: **while** stream has next element e_i **do**
4: $\quad z \leftarrow \min(h(e_i), z)$.
5: **end while**
6: **output** $\frac{1}{z} - 1$.

This algorithm uses memory for only one number, z.

Lemma 1. *Suppose $X_1, X_2, \ldots, X_k$ are independent random variables uniformly distributed in $[0, 1]$. Let $Y = \min_{i=1,\ldots,k} X_i$. Then:*

1. $E[Y] = \frac{1}{k+1}$
2. $Var(Y) = \frac{k}{(k+1)^2(k+2)}$

Proof Sketch. Let $F_Y(t)$ and $f_Y(t)$ be the CDF and PDF of Y. For $t \in (0, 1)$:

$$F_Y(t) = P(Y \leq t) = 1 - P(Y > t) = 1 - P(X_1 > t, \ldots, X_k > t) = 1 - (1 - t)^k$$

The PDF is the derivative:

$$f_Y(t) = k(1 - t)^{k-1}$$

The expectation is:

$$E[Y] = \int_0^1 t \cdot f_Y(t)dt = \int_0^1 tk(1 - t)^{k-1}dt = \frac{1}{k + 1}$$

Similarly, one can calculate $E[Y^2] = \frac{2}{(k+1)(k+2)}$, which gives:

$$\mathrm{Var}(Y) = E[Y^2] - (E[Y])^2 = \frac{2}{(k + 1)(k + 2)} - \frac{1}{(k + 1)^2} = \frac{k}{(k + 1)^2(k + 2)}$$

□

If $F_0 = k$, our estimator Z is the minimum of k uniform random variables. The algorithm's output $\frac{1}{Z} - 1$ is an unbiased estimator for k, since $E[\frac{1}{Z} - 1] \approx \frac{1}{E[Z]} - 1 = \frac{1}{1/(k+1)} - 1 = k$. However, the variance is too high for a good approximation with a single estimate. Using Chebyshev's inequality shows that the probability of being close to the mean is not high enough.

4.2 Variance Reduction

4.2.1 Averaging

To reduce variance, we can average multiple independent estimators. Let $Y_1, \ldots, Y_h$ be i.i.d. random variables with mean μ and variance σ^2. Let their average be $\bar{Y} = \frac{1}{h} \sum_{i=1}^{h} Y_i$. Then:

$$E[\bar{Y}] = \mu \quad \text{and} \quad \mathrm{Var}(\bar{Y}) = \frac{\sigma^2}{h}$$

The variance is reduced by a factor of h.

We apply this to our problem. We run the basic algorithm h times in parallel, each with an independent hash function h_j. This gives us h minimum values $z_1, \ldots, z_h$. Let $Z = \frac{1}{h} \sum_{j=1}^{h} z_j$. Our final output is $Y = \frac{1}{Z} - 1$.

- $E[Z] = \frac{1}{k+1}$
- $\mathrm{Var}(Z) = \frac{\mathrm{Var}(z_j)}{h} \leq \frac{1}{h(k+1)(k+2)}$

Using Chebyshev's inequality on Z:

$$P\left(|Z - E[Z]| \geq \delta \cdot E[Z]\right) \leq \frac{\text{Var}(Z)}{(\delta E[Z])^2}$$

To get a $(1 \pm \epsilon)$ approximation for $F_0 = k$, we need Z to be within a certain range of its mean. A calculation shows that if we set $h = O(1/\epsilon^2)$, we can get a good estimate with constant probability.

4.2.2 Median Trick

To boost the success probability from a constant to $(1 - \delta)$, we can use the median trick. This provides a better dependency on δ than simple averaging.

1. Create a "base estimator" that is a $(1 \pm \epsilon)$-approximation with probability at least $3/4$. We can do this by averaging $h = O(1/\epsilon^2)$ basic estimators as described above.
2. Run this base estimator l times independently to get estimates $Y_1, Y_2, \ldots, Y_l$.
3. Output $Y = \text{median}(Y_1, \ldots, Y_l)$.

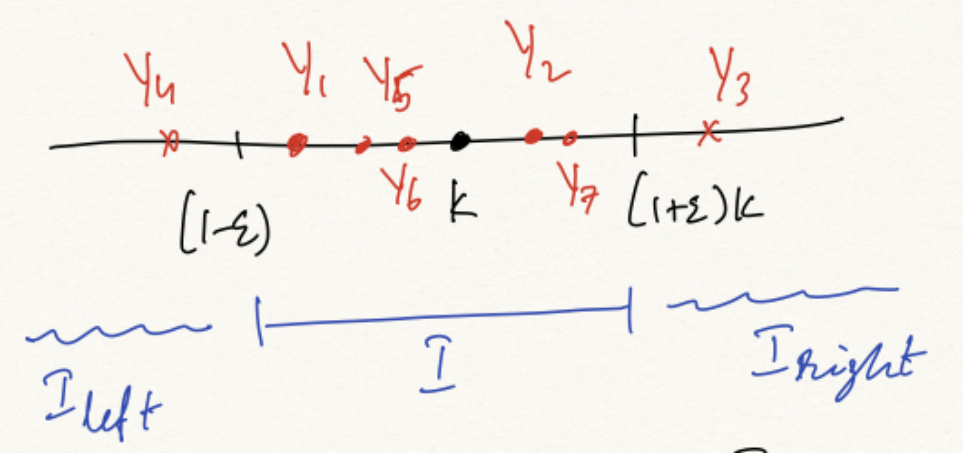

Figure 1: Illustration of the median trick from the lecture notes. The true value is k. The interval $[(1 - \epsilon)k, (1 + \epsilon)k]$ contains most estimates $(Y_1, Y_5, Y_6, Y_2, Y_7)$. Outliers like Y_3 and Y_4 are ignored by taking the median, which is much more likely to fall within the desired interval.

Lemma 2. *Let Y_i be independent estimators such that $P[(1 - \epsilon)k \leq Y_i \leq (1 + \epsilon)k] \geq 3/4$. If we take $l = O(\log(1/\delta))$, then the median Y is a $(1 \pm \epsilon)$ approximation of k with probability at least $1 - \delta$.*

Proof Sketch using Chernoff Bounds. Let A_i be the event that Y_i is a "good" estimate (within the desired interval). $P(A_i) \geq 3/4$. For the median to be "bad" (outside the interval), at least $l/2$ of the estimators Y_i must be "bad". Let B_i be an indicator variable for Y_i being "bad". $P(B_i = 1) \leq 1/4$. Let $B = \sum B_i$. $E[B] \leq l/4$. We want to bound the probability $P(B \geq l/2)$. By Chernoff bounds, this probability is exponentially small in l.

$$P(B \geq l/2) = P(B \geq 2E[B]) \leq e^{-c \cdot l/4}$$

By setting $l = O(\log(1/\delta))$, we can make this probability less than δ. □

4.3 Final Hashing Algorithm and Analysis

The full algorithm combines both techniques.

Algorithm 4 Full Distinct Elements Algorithm

Given ϵ, δ. Let $h = O(1/\epsilon^2)$ and $l = O(\log(1/\delta))$.
for $i = 1$ to l **do**
    Run h parallel instances of the basic hashing algorithm with independent hash functions $h_{i,1}, \ldots, h_{i,h}$ to get minimums $z_{i,1}, \ldots, z_{i,h}$.
    $X_i \leftarrow \frac{1}{h} \sum_{j=1}^{h} z_{i,j}$.
    $Y_i \leftarrow \frac{1}{X_i} - 1$.
end for
output median$(Y_1, \ldots, Y_l)$.

This algorithm uses $O(h \cdot l) = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ space (for storing the minimums). It outputs a $(1 \pm \epsilon)$ approximation with probability at least $(1 - \delta)$.

Practical consideration: The "ideal" hash function to $[0, 1]$ can be approximated by hashing to a large integer range, like $[n^3]$, and then normalizing. Pairwise independent hash functions are often sufficient.

5 A Simpler Sampling-Based Algorithm

A more recent and surprisingly simple algorithm is based on sampling. **Idea:** Suppose we sample each of the F_0 distinct elements with a fixed probability p. If we end up with a sample B, then $E[|B|] = p \cdot F_0$. So, $|B|/p$ is an unbiased estimator for F_0.

Challenges:

1. How do we sample from the set of *distinct* elements when the stream contains duplicates?
2. How do we choose p? If p is too small, $|B|$ will be small and the estimate inaccurate. If p is too large, the sample set B might be too large to store in memory.

Solution to (1): We can adapt the sampling procedure. Maintain a sample set B. For each new element e_m:

1. If e_m is already in B, remove it.
2. Add e_m to B with probability p.

This procedure ensures that at the end of the stream, each distinct element is present in the final set B with probability p.

Solution to (2): The key idea is to adapt the sampling probability p dynamically. Start with $p = 1$ and decrease it whenever the sample set B grows too large.

Algorithm 5 Adaptive Distinct Element Sampling

Given ϵ. Choose a threshold $\tau = O(\frac{\log n}{\epsilon^2})$.
$p \leftarrow 1$, $B \leftarrow \emptyset$.
while stream has next element e_m **do**
    $B \leftarrow B \setminus \{e_m\}$ *(remove if present)*
    With probability p, add e_m to B.
    if $|B| \geq \tau$ **then**
        $p \leftarrow p/2$.
        Subsample B: for each element in B, discard it with probability $1/2$.
    end if
end while
output $|B|/p$.

Theorem 1. *The adaptive sampling algorithm uses $O(\frac{\log n}{\epsilon^2})$ words of memory and outputs an estimate that is within $(1 \pm \epsilon)F_0$ with high probability.*