SEEM5020 Algorithms for Big Data

Streaming Algorithms (I)

Sibo WANG

Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong

Streaming Data

Definition 1 (Data Stream Model)

The data streaming model involves processing a finite sequence of n integers drawn from a finite domain of size m. However, unlike traditional datasets, this sequence is not readily available for random access. Instead, the data arrives incrementally in the form of a continuous 'stream,' with each integer being presented one at a time.

Main challenges:

- In the data streaming model, accessing the input sequence typically allows for only a small number of passes, most likely just once.
- The streaming algorithms are restricted to use a space that is *logarithmic or polylogarithmic* in m and n.

Applications

Here, we list several applications.

- Query streams: Google/ChatGPT wants to know which queries are more frequent today than yesterday.
- Click streams: Wikipedia wants to know which pages have received unusual hits in the past hour.
- Social network post/update streams: trending topics on Twitter, TikTok, Facebook, etc.
- IP packet monitoring at a switch: Gather package receiving speed info and optimize the routing. Identify DDoS attacks.
- Sensor data collection: Identify the number of abnormal results.
- $\cdots$

Table of Contents

Uniform Sampling in a Stream

Problem 1 (Uniform sampling in a stream)

Given a stream of elements from the universe $[m]$*, where* $[m]$ *means the set of integers* $\{1, 2, 3, \cdots m\}$*, sample* k *elements from the stream uniformly at random.*

Assume that we have n elements in the sequence. Each element has k/n probability of being sampled. However, the main challenge is: the size n of the stream is usually unknown.

Reservoir Sampling

The reservoir sampling algorithm achieves this goal without knowing the number n of elements in this sequence. It works as follows:

Algorithm 1 (Reservoir Sampling)

1. Initialize an array A of size k (array index starting from 1) and include the first k elements in the stream. This array is the reservoir;
2. When the i-th element a_i comes ($i > k$), we draw a random integer r; if $r \leq k$, we update $A[i]$ and set $A[r] = a_i$;
3. When the stream ends, return A;

It takes $O(k \log m)$ bits to store the array.

Correctness of Reservoir Sampling

Theorem 1

Algorithm 1 returns each element with a probability $\frac{k}{n}$.

Proof.

We prove this by induction. When $n = k$. It naturally holds.
Inductive hypothesis: When $n > k$, assume that the sample set A contains each element seen so far with probability $\frac{k}{n}$.
Inductive step: Now a new element e comes, we aim to prove that each element seen so far is sampled with probability $\frac{k}{n+1}$.

- For the new element e_{n+1}, according to Algorithm 1, it is added into A with probability $\frac{k}{n+1}$.
- For the remaining elements $e_1, e_2, \cdots, e_n$, the probability is:

$$\mathbb{P}[e_i \in A_{n+1}] = \mathbb{P}[e_i \in A_n] \cdot \frac{n}{n+1} = \frac{k}{n} \cdot \frac{n}{n+1} = \frac{k}{n+1}.$$

Proof done. □

Table of Contents

Counting in Streams

Problem 2 (Counting Problem)

Given a stream of events, the counting problem aims to count the number of events that occur thus far with as little space as possible.

A straightforward solution: By maintaining a counter with $O(\log n)$ bits. That is already the best we can do if we want to maintain the exact count.

Can we do better if we allow for some approximation?

Approximate Counting: Morris Algorithm [1]

Definition 2 ((ϵ, δ)-approximation)

Let μ be the value of interest, e.g. the number of events in previous slides. Let $\hat{\mu}$ be an estimation of μ. Then, we say $\hat{\mu}$ is an (ϵ, δ)-approximation of μ if the following holds:

$$\mathbb{P}[|\mu - \hat{\mu}| > \epsilon \cdot \mu] \leq \delta.$$

A classic solution for approximate counting: Morris algorithm.

Algorithm 2 (Morris Algorithm)

1. *Initialize X as zero;*
2. *For each update, increment X with probability $\frac{1}{2^X}$;*
3. *For a counting query, return $\hat{n} = 2^X - 1$;*

Analysis of Morris Algorithm

Lemma 1

$\mathbb{E}[2^{X_n}] = n + 1.$

Proof.

We prove this by induction. It is easy to verify the base case. Assume that it holds for $j \leq n$. Then, taking the condition on X_n, we have that:

$$
\begin{aligned}
\mathbb{E}[2^{X_{n+1}}] &= \sum_{j=0}^{+\infty} \mathbb{P}[X_n = j] \cdot \mathbb{E}[2^{X_{n+1}} | X_n = j] \\
&= \sum_{j=0}^{+\infty} \mathbb{P}[X_n = j] \cdot \left(2^j (1 - \frac{1}{2^j}) + 2^{j+1} \cdot \frac{1}{2^j} \right) \\
&= \sum_{j=0}^{+\infty} \mathbb{P}[X_n = j] \cdot 2^j + \sum_{j=0}^{+\infty} \mathbb{P}[X_n = j] = \mathbb{E}[2^{X_n}] + 1 = n + 2.
\end{aligned}
$$

Proof done. □

Analysis of Morris Algorithm (Cont.)

We will further derive the variance of the Morris algorithm so as to apply the Chebyshev's inequality. Let $\hat{n} = 2^{X_n} - 1$ The variance $\mathrm{Var}[\hat{n}]$ is:

$$\mathbb{E}[(2^{X_n} - 1 - n)^2] = \mathbb{E}[2^{2X_n}] - (n+1)^2.$$

Lemma 2

$\mathbb{E}[2^{2X_n}] = \frac{3}{2}n^2 + \frac{3}{2}n + 1.$

The proof is left as a self exercise. Accordingly, $\mathrm{Var}[\hat{n}] = \frac{n(n-1)}{2}$. Then, by applying the Chebyshev's inequality, we have that:

$$\mathbb{P}[|\hat{n} - n| > \epsilon \cdot n] \leq \frac{\mathrm{Var}[\hat{n}]}{(\epsilon n)^2} < \frac{n^2/2}{\epsilon^2 n^2} = \frac{1}{2 \cdot \epsilon^2}.$$

However, this bound is loose and the inequality is almost useless if $\epsilon \leq 1/\sqrt{2}$.

Morris+

Can we achieve a meaningful probability, say $\delta = 1/3$?

- Key observation: the sum of Independent and identically distributed (i.i.d) random variables has degraded variance.
- Solution: Take t trials of Morris algorithm. Let their estimations be $\hat{n}_1, \hat{n}_2, \cdots, \hat{n}_t$. Then, take the average of the t trials: $\frac{\sum_{i=1}^{t} \hat{n}_i}{t}$.
  - Variance: $\frac{n(n-1)}{2t}$.
  - Applying the Chebshev's inequality again, we have:

$$\mathbb{P}[|\frac{\sum_{i=1}^{t} \hat{n}_i}{t} - n| > \epsilon \cdot n] \leq \frac{\mathrm{Var}[\hat{n}]}{t \cdot (\epsilon n)^2} < \frac{1}{2t\epsilon^2}$$

- By setting $t = \frac{3}{2\epsilon^2}$, we have that $\delta = \frac{1}{3}$.

Morris++

- To have a probability of δ, Morris+ needs $O(1/\delta)$ trials of Morris algorithm. Can we reduce the dependency to δ?
- Solution: Morris++ via Median trick.
  - Apply Morris+ s times. Then, take the median as the estimation. The solution is Morris++.
  - Let Y be the random variable to indicate whether Morris+ succeeds. If it does succeed, $Y = 1$ and otherwise 0. To make Morris++ fail, at least half of the estimators fail. Think why?
  - Then, Morris++ fails indicates $\sum_{j=1}^{s} Y_i < \frac{s}{2}$.

$$\mathbb{P}[\sum_{j=1}^{s} Y_i < \frac{s}{2}] = \mathbb{P}[\sum_{j=1}^{s} Y_i < (1 - \frac{1}{4}) \cdot \frac{2s}{3}] \leq \mathbb{P}[\sum_{j=1}^{s} Y_i < (1 - \frac{1}{4})\mu]$$

Applying Chernoff bound, we have that:

$$\mathbb{P}[\sum_{j=1}^{s} Y_i < (1 - \frac{1}{4})\mu] \leq e^{-\mu(\frac{1}{4})^2/2} \leq e^{-\mu(\frac{1}{4})^2/2} \leq e^{-\frac{s}{48}}.$$

  - Setting $s = 48\log(1/\delta)$, the dependency reduces to $O(\log(1/\delta))$.

Space Cost of Morris++

What is the space cost to achieve (ϵ, δ)-guarantee for the Morris++ algorithm?

- Notice that the counter is a random variable. So is the space cost. Can we bound the space cost with a high probability?

Consider the case when $X_n \geq c \cdot \log_2(n+1)$. We have that:

$$\mathbb{P}[X_n \geq c \cdot \log_2(n+1)] = \mathbb{P}[2^{X_n} \geq 2^{c \cdot \log_2(n+1)}]$$
$$\leq \frac{\mathbb{E}[2^{X_n}]}{2^{c \cdot \log_2(n+1)}} = \frac{n+1}{(n+1)^c} \leq \frac{1}{(n+1)^{c-1}}$$

We have $\frac{3}{2\epsilon^2} \cdot 48 \log(1/\delta)$ Morris counter. It is easy to bound the probability to $\frac{72 \log(1/\delta)}{\epsilon^2 (n+1)^{c-1}}$. With an appropriate c, this can be bounded with high probability. Thus, the space cost is bounded by $O(\frac{\log(1/\delta) \log \log n}{\epsilon^2})$ with high probability.

Exercise

The above Morris++ algorithm is not effective in practice when ϵ is no larger than 0.25. Also, the dependency to $\log(1/\delta)$ is a multiplication factor over $\log\log n$. Can we reduce this dependency?

More generally, we can adopt a Morris(a) algorithm, where we increment the counter by the $\frac{1}{(1+a)^X}$ probability. Then, we estimate $\hat{N}$ as $\frac{(1+a)^X-1}{a}$. If we set $a = 0$, we actually provide the actual count. If we set $a = 1$, we have the previous Morris algorithm. By choosing a between $(0, 1)$, we are gaining a balance between the accuracy and the space cost.

- Prove that $\hat{N} = \frac{(1+a)^X-1}{a}$ is an unbiased estimation of N and the variance is $\frac{aN(N-1)}{2}$.
- The space cost is bounded by $O(\log\log n + \log(1/\delta) + \log(1/\epsilon))$ with high probability by setting $a = 2\epsilon^2\delta$.

Table of Contents

Distinct Element Counting

Definition 3 (Distinct Element Counting)

Given a stream of integers $e_1, e_2, \cdots, e_n$ from $[m]$ where elements might appear more than once in the stream, the goal of the distinct element counting problem is to output the number of *distinct* elements.

Example 1

Assume that $m = 5$ and the stream is $1, 2, 2, 1, 5, 4, 2, 2, 1$. Then, the number of distinct elements (NDE) is 4.

Two naive solutions

- Store the entire universe with $O(m)$ bits. Mark the i-th bit as 1 if i appears in the stream.
- Store a hash table or binary search tree to keep the distinct elements. This requires $O(n \cdot \log m)$ bits, or more precisely $O(NDE \cdot \log m)$ bits.

MinHash Algorithm [2]

We now assume that we have the following idealized hash function $h : [n] \to [0, 1]$ with an equal probability of hashing to each value in $[0, 1]$.

- Actually, we cannot afford a truly randomized hash function as it cannot be maintained in $o(n)$ bits.

Algorithm 3 (MinHash Algorithm)

1. When an element e_i arrives, derive the hash value $h(e_i)$;
2. Maintain the minimum hash value X_{min} (Initially set as 1). If $h(e_i) < X_{min}$, update X_{min} as $h(e_i)$;
3. When the stream ends, return $\frac{1}{X_{min}} - 1$;

Analysis of MinHash Algorithm

Lemma 3

$\mathbb{E}[X_{min}] = \frac{1}{NDE+1}$.

Recap that for a continuous random variable X, we have the concept of probability density function $f_X(x)$ and cumulative distribution function $F_X(x)$. The expectation of such a random variable is

$$\mathbb{E}[X] = \int_{-\infty}^{+\infty} x \cdot f_X(x)\, dx.$$

For non-negative random variables, we further have that

$$\mathbb{E}[X] = \int_{0}^{+\infty} (1 - F_X(x))\, dx.$$

Analysis of MinHash Algorithm (Cont.)

Lemma 4

$\mathbb{E}[X_{min}] = \frac{1}{NDE+1}$.

Proof.

Let $F_X(x)$ be the cumulative distribution function $F_X(x)$ that X_{min} is not larger than x. Then $1 - F_X(x)$ is the probability that X_{min} is larger than x, which happens only when all distinct elements have a hash value greater than x. The probability is $(1-x)^{NDE}$ for $0 \le x \le 1$. Thus,

$$\mathbb{E}[X_{min}] = \int_0^{+\infty} (1 - F_X(x))\, dx = \int_0^1 (1-x)^{NDE}\, dx$$
$$= \frac{-(1-x)^{NDE}}{NDE+1}\Bigg|_0^1 = \frac{1}{NDE+1}.$$

Proof done. □

Analysis of MinHash Algorithm (Cont.)

Lemma 5

$\mathbb{E}[X_{min}^2] = \frac{2}{(NDE+1)(NDE+2)}$.

The proof is left as a self-exercise. We can derive the variance of X_{min}:

$$\mathrm{Var}[X_{min}] = \mathbb{E}[X_{min}^2] - (\mathbb{E}[X_{min}])^2 = \frac{NDE}{(NDE+1)^2(NDE+2)}.$$

Denote MinHash+ as the algorithm by applying s such hash functions and taking the average of these s results of X_{min}. The expectation is still unbiased. The variance is reduced to

$$\frac{NDE}{s(NDE+1)^2(NDE+2)} \leq \frac{1}{s(1+NDE)^2}$$

Setting $s = \frac{3}{\epsilon^2}$ and applying the Chebshev's inequality, we have:

$$\mathbb{P}[|\frac{1}{s}\sum_{j=1}^{s} X_{min,j} - \frac{1}{1+NDE}| \geq \frac{\epsilon}{1+NDE}] \leq \frac{1}{3} \quad (1)$$

Analysis of MinHash Algorithm (Cont.)

It is easy to verify that if $NDE \geq 2$ (which easily holds) and $\epsilon < 0.5$, the following holds:

$$\mathbb{P}[|\frac{1}{\frac{1}{s}\sum_{j=1}^{s} X_{min,j}} - 1 - NDE| \geq 2\epsilon NDE] \leq \frac{1}{3}$$

To reduce the failure probability from constant (here $\frac{1}{3}$) to an arbitrarily small value δ, we can apply the median trick as we have applied in Morris++ algorithm. This can be achieved with $O(\log(1/\delta))$ trials of MinHash+. The final space is $O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$.

Table of Contents

Alternative to MinHash Algorithm: Bottom-k Sketch [3]

The MinHash Algorithm applies multiple hash functions to reduce the variance. Any alternative method?

- The smallest value tends to have a large variance. Can we use other statistics?
- Intuition: As shown in previous solutions, the median tends to be a stable variable. However, maintaining the median is expensive.
- As an alternative, we maintain the k smallest hash values.
- Let X_{kth} be the k-th smallest hash value, we return $\frac{k}{X_{kth}}$ as the estimation.

Is there any relationship between the k-th smallest hash value and NDE?

- Unfortunately, the relationship is not as explicit as that in MinHash[1].
- We will need to analyze this in a different approach.

[1] The k-th order statistics: https://en.wikipedia.org/wiki/Order_statistic

Bottom-k Sketch

Instead of a truly randomized hash function, bottom-k sketch only needs a hash function that is 2-wise independent, a.k.a, pairwise independent.

Definition 4 (k-wise independent hash family)

A family $\mathcal{H}$ of hash functions mapping from $[a] \to [b]$ is k-wise independent if for any $j_1, \cdots j_k \in [b]$ and any distinct $i_1, \cdots i_k \in [a]$,

$$\mathbb{P}_{h \in \mathcal{H}}[h(i_1) = j_1 \wedge h(i_2) = j_2 \wedge \cdots \wedge h(i_j) = j_k] = 1/b^k.$$

There are efficient solutions in finding a k-wise independent hash function.

Example 2

Let P be a prime number greater than a (the input domain of h). Choose a_i randomly from $[P]$. The following hash function is k-wise independent:

$$H(v) = \Big((a_0 + a_1 \cdot v^1 + \cdots + a_{k-1} \cdot v^k) \bmod p\Big) \bmod m$$

Bottom-k Sketch (Cont.)

How bottom-k sketch algorithm works with pairwise independent hashing:

- First, set the range $[b]$ with $b = n^3$ so that we will have no collision with at least $1 - \frac{1}{n}$ probability.
- When an element e comes, we compute the hash value $h(e)$ of e.
- We maintain the set S_k of the k smallest hash values and denote the k-th smallest hash value as h_{kth}. If $h(e)$ is smaller than h_{kth}, we remove h_{kth} from S_k and add $h(e)$ to S_k.
- At the end, we retrieve h_{kth} from S_k and return an estimate of $\frac{b \cdot k}{h_{kth}}$.

Analysis of Bottom-k Sketch Algorithm

Our goal: $(1-\epsilon)NDE < \frac{b\cdot k}{h_{kth}} < (1+\epsilon)NDE$

- We analyze for the probability of the bad event: $\frac{b\cdot k}{h_{kth}} \geq (1+\epsilon)NDE$. The other side can be analyzed in a similar way.
- We define $Y_i = 1$ if the i-th element has a hash value no larger than $\frac{b\cdot k}{(1+\epsilon)NDE}$ and otherwise 0. Define $Y = \sum_{i=1}^{NDE} Y_i$.
- Observe that:

$$\frac{b\cdot k}{h_{kth}} \geq (1+\epsilon)NDE \Leftrightarrow \frac{b\cdot k}{(1+\epsilon)NDE} \geq h_{kth} \Leftrightarrow \sum_{i=1}^{NDE} Y_i \geq k.$$

The goal is to bound the probability that $\sum_{i=1}^{NDE} Y_i > k$.

- $\mathbb{E}[Y_i] = \lfloor \frac{\cdot k}{(1+\epsilon)NDE} \rfloor$ according to its definition. Thus:

$$\mathbb{E}[Y] = \mathbb{E}[\sum_{i=1}^{NDE} Y_i] = NDE \cdot \lfloor \frac{k}{(1+\epsilon)NDE} \rfloor \leq \frac{k}{(1+\epsilon)}.$$

Analysis of Bottom-k Sketch Algorithm (Cont.)

The variance $\mathrm{Var}[Y]$ of Y is exactly $\sum_{i=1}^{NDE} \mathrm{Var}[Y_i]$ since Y_i are pairwise independent. More specifically:

$$\mathrm{Var}[\sum_{i=1}^{NDE} Y_i] = \mathbb{E}[(\sum_{i=1}^{NDE} Y_i - \mathbb{E}[Y_i])^2] = \sum_{i=1}^{NDE} E[(Y_i - \mathbb{E}[Y_i])^2] +$$

$$2\mathbb{E}[\sum_{1 \le i < j \le NDE} (Y_i - \mathbb{E}[Y_i])(Y_j - \mathbb{E}[Y_j])]$$

The variance $\mathrm{Var}[Y]$ of Y is exactly $\sum_{i=1}^{NDE} \mathrm{Var}[Y_i]$ since Y_i are pairwise independent. More specifically:

$$\mathrm{Var}[\sum_{i=1}^{NDE} Y_i] = \mathbb{E}[(\sum_{i=1}^{NDE} Y_i - \mathbb{E}[Y_i])^2] = \sum_{i=1}^{NDE} E[(Y_i - \mathbb{E}[Y_i])^2] + 2\mathbb{E}[\sum_{1 \le i < j \le NDE} (Y_i - \mathbb{E}[Y_i])(Y_j - \mathbb{E}[Y_j])]$$

As Y_i and Y_j are pairwise independent,

$$\mathbb{E}[(Y_i - \mathbb{E}[Y_i])(Y_j - \mathbb{E}[Y_j])] = \mathbb{E}[(Y_i - \mathbb{E}[Y_i])]\mathbb{E}[(Y_j - \mathbb{E}[Y_j])] = 0.$$

Thus,

$$\mathrm{Var}[\sum_{i=1}^{NDE} Y_i] = \sum_{i=1}^{NDE} E[(Y_i - \mathbb{E}[Y_i])^2] = \sum_{i=1}^{NDE} \mathrm{Var}[Y_i] \le \frac{k}{1+\epsilon}.$$

We further apply the Chebshev's inequality.

$$\mathbb{P}[Y \geq k] = \mathbb{P}[Y - \mathbb{E}[Y] \geq k - \mathbb{E}[Y]] \leq \mathbb{P}[|Y - \mathbb{E}[Y]| \geq k - \mathbb{E}[Y]]$$
$$\leq \mathbb{P}[|Y - \mathbb{E}[Y]| \geq k - \frac{k}{(1+\epsilon)}] = \mathbb{P}[|Y - \mathbb{E}[Y]| \geq \frac{k\epsilon}{(1+\epsilon)}]$$
$$\leq \frac{\mathrm{Var}[Y]}{(k\epsilon/(1+\epsilon))^2} \leq \frac{k}{1+\epsilon} \cdot \frac{(1+\epsilon)^2}{k^2\epsilon^2} = \frac{1+\epsilon}{k\epsilon^2}$$

Setting $k = \lceil \frac{12}{\epsilon^2} \rceil$, we have

$$\mathbb{P}[Y \geq k] \leq \frac{1+\epsilon}{12} \leq \frac{1}{6}.$$

The proof of the other side is left as a self-exercise. You may assume that $\epsilon < \frac{1}{2}$ and the fact that $b = n^3 \gg NDE \cdot \epsilon$.

Analysis of Bottom-k Sketch Algorithm (Cont.)

Thus, we have that

$$\mathbb{P}[(1-\epsilon)NDE < \frac{b \cdot k}{h_{kth}} < (1+\epsilon)NDE] \geq \frac{2}{3}$$

We can further apply the median trick with $\log(1/\delta)$ copies of the bottom-k sketch to achieve a success probability of $1-\delta$. The total space complexity is thus: $O(k \cdot \log(1/\delta)) = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$.

Table of Contents

More Solutions for Distinct Element Counting

FM Algorithm [4]: First, we need a hash function $h : [n] \to [2^L - 1]$ that maps the key x to each value in the range $[2^L - 1]$ uniformly at random.

- For each element e, compute the hash value $h(e)$. Then, let $r(e)$ be the number of trailing 0's in the binary representation of $h(e)$.
  - For example, $h(e) = 12$ and $12 = (1100)_2$ in binary. So, $r(e) = 2$ since there are two zeros at the end of the binary representation of $h(e)$.
- When the stream ends, let R be the maximum of $r(e)$ we have seen.
- Flajolet and Martin [4] prove that $\mathbb{E}[R] \approx \log_2 \phi \cdot n$, where $\phi \approx 0.77351$. The proof is rather involved and hence is omitted.
- According to the above analysis, FM algorithm estimates the number of distinct elements as $2^R/\phi$.
- How to derive more accurate results?

HyperLogLog [5]: Extension of FM algorithm by splitting the stream into numerous sub-streams. Use harmonic mean to derive the estimation.

References

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

Edith Cohen.
Size-estimation framework with applications to transitive closure and reachability.
J. Comput. Syst. Sci., 55(3):441–453, 1997.

Edith Cohen and Haim Kaplan.
Summarizing data using bottom-k sketches.
In *PODC*, pages 225–234, 2007.

Philippe Flajolet and G. Nigel Martin.
Probabilistic counting algorithms for data base applications.
J. Comput. Syst. Sci., 31(2):182–209, 1985.

Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier.
Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm.
Discrete mathematics & theoretical computer science, 2007.