MA421 Advanced Algorithms   Lent 2019

4 Streaming Algorithms

Reading:

- **Surveys**
  - S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science* 1,2, 117–236, 2005. https://cs.uwaterloo.ca/ david/cs848/Muthu-Survey.pdf
  - G. Cormode. Sketch Techniques for Massive Data. In *Synopses for Massive Data: Samples, Histograms, Wavelets and Sketches.* Foundations and Trends in Databases, vol. 4, no. 1–3, 2011.
- **Courses**
  - P. Indyk. Sketching, Streaming and Sub-linear Space Algorithms. MIT. http://stellar.mit.edu/S/course/6/fa07/6.895/
  - A. Chakrabarti. Data Stream Algorithms. Dartmouth College. `http://www.cs.dartmouth.edu/~ac/Teach/CS49-Fall11/`
  - A. McGregor. Data Streams and Massive Data. University of Massachusetts at Amherst. http://people.cs.umass.edu/ mcgregor/courses/CS711S12/index.html

Parts of Section 4.5 is based on A. Chakrabarti's lecture notes listed above.

4.1 Introduction

Streaming data models are proposed to capture the recent computational challenges posed by the requirements to process high volumes of data. These models try to encapsulate the constraints of computations on massive data sets such as impracticality of random access to the data as well as impossibility of fitting even a small fraction of the data into the working memory. For example, these models suit high-throughput devices and small devices that receive data continuously and have limited processing power and memory.

A streaming algorithm receives its input as a sequence of m elements from a universe of size n ($[n] = \{1, 2, \ldots, n\}$): for example,

$$\langle x_1, x_2, \ldots, x_m \rangle \qquad = \qquad 4, 7, 2, 4, 5, 7, \ldots,$$

where m is possibly unknown to the algorithm. The goal of the algorithm is to compute some function of the stream: e.g., median, number of distinct elements, longest increasing subsequence, etc. The main constraints of the models are (i) limited amount of working memory, sublinear (often desired be polylogarithmic) in n and m, (ii) access to the data only sequentially (possibly, with more than one pass), (iii) requirement to process each data element quickly.

Given these restrictions, the algorithms, provably, have to be both *randomised* and *approximate* for most of the interesting problems. Even though there are a few examples starting from the 70s, streaming algorithms has become more popular in the last two decades because of growing theory and applicability. On the practical side, fast networks and processors, cheap data storage, ubiquitous data logging results in massive data sets to be processed. Applications to network monitoring, query planning, I/O efficiency for massive data, sensor networks aggregation, etc. make streaming models attractive. On the theoretical side, there are easy-to-state but hard-to-solve fundamental problems in these models and many connections to other fields such communication complexity, compressed sensing, metric embeddings, pseudo-random generators, approximation algorithms have been formed.

4.2 Simple problem: (Approximate) Counting the Elements in a Stream

In this section, we will look at a streaming algorithm to count the number of items in a data stream. Obviously, a simple (and, more than often, sufficiently efficient) solution is using a counter of $\lceil \log_2 m \rceil$ bits. Note that, it will usually be more appropriate to study the memory requirements in terms of the number of bits needed. Given that we aim for sublinear (or, even, polylogarithmic) memory usage, it makes sense to do this. For example, a constant-size counter for counting the unknown number of items in the stream is not realistic.

Suppose we are happy with an approximate count. Can we do with less space than $\Omega(\log m)$? In particular, suppose we are happy to have a 2-approximation to the actual count (that is, a value between 50% and 200% of the real count). We can achieve this by keep track of the logarithm of the count instead.

Algorithm **Approximate Count**:

1. Initialise counter c to 1.
2. For each item seen, increment counter c by 1 with probability 2^{-c}.
3. At the end, output estimate $2^c - 2$.

Note that the counter is incremented less often as it gets bigger, which is inline with the growth rate of the logarithm function.

A detailed analysis show that, after seeing m items, the expected value of the counter is $m + 2$. A rather vague intuition behind the correctness analysis is that, in an expected sense, the counter will be incremented once in a window of the stream where the logarithm of the actual count goes up by one. Since it is the logarithm of the count that we keep track of, the space usage is $O(\log \log m)$ bits.

We can use a base smaller than 2 for more accuracy for a modest constant factor increase in the counter size. We can use several parallel (and independent) counters for more confidence (because each counter will be close to the actual count only with some positive probability).

Even though it is a simple and easy-to-describe algorithm, a proper analysis is quite technical. You can refer to the following article for details: Flajolet, P. Approximate Counting: A Detailed Analysis. BIT 25, (1985), 113-134, `http://algo.inria.fr/flajolet/Publications/Flajolet85c.pdf`.

4.3 Sampling

Various sampling problems pose interesting challenges in the context of streaming data. Furthermore, algorithms for different sampling techniques are serve general tools for solving other problems for massive amounts of data. For example, to compute the median, we could just sample the stream elements uniformly at random and use the median of the sample as an estimate. Probabilistic arguments will relate the size of the sample to the accuracy of the estimate.

Here is a challenge: How do you take a sample from a stream of unknown length or from a "sliding window" of stream elements?

- Problem 1: Take a uniform sample s from a stream of unknown length (Reservoir sampling)

- Algorithm:
  - Initially, $s = x_1$.
  - On seeing the tth element, $s \leftarrow x_t$ with probability $1/t$
- Analysis:
  - What is the probability that $s = x_i$ at some time $t \geq i$?

$$\begin{aligned}\Pr[s = x_i] &= \frac{1}{i} \times \left(1 - \frac{1}{i+1}\right) \times \left(1 - \frac{1}{i+2}\right) \times \cdots \times \left(1 - \frac{1}{t}\right)\\ &= \frac{1}{i} \times \frac{i}{i+1} \times \frac{i+1}{i+2} \times \cdots \times \frac{t-1}{t}\\ &= \frac{1}{t}\end{aligned}$$

  - To get k samples, we can use $O(k \log n)$ bits of space.
- Problem 2: Maintain a uniform sample from the last w items.
- Algorithm:
  1. For each x_i, we pick a random value $v_i \in (0, 1)$.
  2. In a window $\langle x_{j-w+1}, x_{j-w+2}, \ldots, x_j \rangle$, return value x_i with minimum v_i.
  3. To achieve this, maintain a set S of all elements x_i in sliding window whose v_i value is minimal among subsequent values. (Note that no other element needs to be saved in the memory.)
- Analysis:
  - The probability that the kth oldest (i.e., x_{j-k+1}) element is in S is $1/k$.

$$\int_0^1 (1-x)^{k-1}\, dx = \left.\frac{-(1-x)^k}{k}\right|_0^1 = 0 + \frac{1}{k} = \frac{1}{k}$$

  - So, the expected number of items in S is

$$1/w + 1/(w-1) + \cdots + 1 = O(\log w).$$

  - Hence, the algorithm uses only $O(\log w \log n)$ bits of memory (say, with probability at least 0.99).

4.4 Sketching

Sketching is another general technique for processing streams: it provides a "summary" of the stream seen so far. The information stored in this summary depends on the problem being solved. For example, sketching is useful for distributing the stream processing or comparing streams at different locations or times. One general technique here is applying a linear projection "on the fly" that takes high-dimensional data to a smaller dimension and post-processing lower dimensional image to estimate the quantities of interest. We will see a simple example of a sketch in the next section.

4.5 Estimating Frequency Moments

We have a stream $\langle x_1, x_2, \ldots, x_m \rangle$, with each $x_j \in [n]$, and this implicitly defines a frequency vector $f = (f_1, f_2, \ldots, f_n)$. Note that $f_1 + \cdots + f_n = m$. The kth frequency moment of the stream, denoted F_k is defined as follows:

$$F_k = \sum_{j=1}^{n} f_j^k = \|f\|_k^k.$$

Note that F_0 is the number of distinct tokens in the stream (using the convention $0^0 = 0$). F_0 can be approximated well using space logarithmic in m and n. And, $F_1 = m$ is easy to compute exactly. Can we say anything for general F_k?

By way of motivation, note that F_2 represents the size of the self join of a relation r in a database, with f_j denoting the frequency of the value j of the join attribute. Imagine that we are in a situation where r is a huge relation and n, the size of the domain of the join attribute, is also huge; the tuples can only be accessed in streaming fashion (or perhaps it is much cheaper to access them this way than to use random access). Can we, with one pass over the relation r, compute a good estimate of the self join size? Estimation of join sizes is a crucial step in database query optimisation. The solution we shall eventually see for this F_2 estimation problem will in fact allow us to estimate arbitrary equijoin sizes (not just self joins).

In what follows, we present two estimation algorithms for F_2 and for arbitrary F_k, where $k \geq 2$, respectively, using space sublinear in m and n. The algorithms we present are due to Alon, Matias, and Szegedy [AMS99].

4.5.1 The AMS Estimator for F_2

In this section, we present an algorithm by Alon, Matias, and Szegedy for estimating F_2. The space complexity of the algorithm is logarithmic in n and m. To make the algorithm's analysis simpler, n random numbers are generated and stored for the duration of the execution. We later discuss how this requirement can be relaxed.

The idea behind the algorithm can be described as follows. With the given space requirement as above, we cannot keep an accurate estimate for f_i for more than a few values of i. We can certainly keep track of $\sum_i f_i$ exactly, which at the end gives the stream length. Squaring $\sum_i f_i$, we would get

$$(\sum_i f_i)^2 = \sum_i f_i^2 + 2\sum_{i<j} f_i f_j,$$

where the second summation gives an overestimation for F_2 and, thus, we would like to avoid. Instead, we will keep track of $\sum_i r_i f_i$, for randomly chosen values of r_i in $\{-1, 1\}$. Therefore, when we square $\sum_i r_i f_i$, we will lose the second summation in an expected sense.

Algorithm 1 A data stream algorithm for estimating $\sum_i f_i^2$

```
1: procedure ESTIMATE-SECOND-FREQ-MOMENT($\langle x_1, x_2, \ldots \rangle$)
2:     Choose random $r_1, r_2, \ldots, r_n$ such that $\Pr[r_i = 1] = \Pr[r_i = -1] = 1/2$.
3:     for each stream element $x_i$ do
4:         $Z \leftarrow Z + r_{x_i}$                         ▷ Note that $Z = \sum_i r_i f_i$
5:     return $Y = Z^2$
```

We will show below that Y approximates $F_2 = \|f\|_2^2$ well, with high probability. First, we show that our estimator is unbiased.

Lemma 4.1 $\mathbb{E}[Z^2] = F_2$.

Proof. Noting that $Z = \sum_i r_i f_i$, we get

$$\mathbb{E}[(\sum_i r_i f_i)^2] = \sum_i \mathbb{E}[r_i^2 f_i^2] + 2\sum_{i<j}\mathbb{E}[r_i r_j f_i f_j] = \sum_i f_i^2 \underbrace{\mathbb{E}[r_i^2]}_{=1} + 2\sum_{i<j} f_i f_j \underbrace{\mathbb{E}[r_i]}_{=0}\underbrace{\mathbb{E}[r_j]}_{=0} = \sum_i f_i^2,$$

where we have used the (pairwise) independence of r_i and r_j, and the expectations of r_i and r_i^2, respectively. □

In order to argue that we can obtain a good estimator, we need to bound the variance of the estimator as well.

Lemma 4.2 $\mathrm{Var}[Z^2] \le 2(F_2)^2$.

Proof. We will use $\mathrm{Var}[Z^2] = \mathbb{E}[Z^4] - \mathbb{E}[Z^2]^2$ to bound the variance.

$$\mathbb{E}[Z^4] = \mathbb{E}[(\sum_i r_i f_i)^4] = \sum_i f_i^4 \mathbb{E}[r_i^4] + 6\sum_{i<j} f_i^2 f_j^2 \mathbb{E}[r_i^2]\mathbb{E}[r_j^2] + \cdots$$

Note that all the omitted terms in the summation above will involve, for some i, $\mathbb{E}[r_i]$, which is 0. Thus, observing $\mathbb{E}[r_i^4] = \mathbb{E}[r_i^2] = 1$, we get $\mathbb{E}[Z^4] = \sum_i f_i^4 + 6\sum_{i<j} f_i^2 f_j^2$.

Finally, using Lemma 4.1,

$$\mathbb{E}[Z^2]^2 = (\sum_i f_i^2)^2 = \sum_i f_i^4 + 2\sum_{i<j} f_i^2 f_j^2.$$

Therefore, we can conclude

$$\mathrm{Var}[Z^2] = \sum_i f_i^4 + 6\sum_{i<j} f_i^2 f_j^2 - (\sum_i f_i^4 + 2\sum_{i<j} f_i^2 f_j^2) = 4\sum_{i<j} f_i^2 f_j^2 = 2\sum_{i\neq j} f_i^2 f_j^2 \le 2(\sum_i f_i^2)(\sum_j f_j^2).$$

□

Now that we have the expectation and the variance of our estimator bounded, we can use Chebyshev's inequality on Z^2 to claim that, for any constant $c > 1$,

$$\Pr[|Z^2 - \mathbb{E}[Z^2]| > c\sqrt{\mathrm{Var}[Z^2]}] = \Pr[|Z^2 - \mathbb{E}[Z^2]| > \sqrt{2}cF_2] \le \frac{1}{c^2}.$$

Note that this will ensure only a deviation larger F_2 with a constant probability. We can use t independent copies of Y to improve accuracy. For example, if $Y' = \frac{1}{t}\sum_{i=1}^t Y_i$, where each Y_i is an independent copy of Z^2, we get

$$\mathbb{E}[Y'] = \mathbb{E}[Y] = F_2 \text{ and } \mathrm{Var}[Y'] = \frac{\mathrm{Var}[Y]}{t} \le \frac{2(F_2)^2}{t}.$$

Setting $c = 10$ and $t = 200/\epsilon^2$, we can claim

$$\Pr[|Y' - \mathbb{E}[Y']| > c\sqrt{(2/t)}F_2] = \Pr[|Y' - \mathbb{E}[Y']| > \epsilon F_2] \le 0.01.$$

In other words, for $t = O(1/\epsilon^2)$, we get a $(1+\epsilon)$-approximation to F_2 with constant probability. We should also note that, running t independent copies of the estimator is simply maintaining $A \cdot f$ for a t-by-n matrix A, where each entry of A is chosen from $\{-1, 1\}$ uniformly at random.

Finally, we observe that we only need to generate independent random numbers such that any group of four them are mutually independent (which was required for the variance bound). Such random variables are called 4-wise independent. It turns out that this can be using $O(\log n)$ truly random bits and space. This argument is left as an exercise for the reader.

4.5.2 The AMS Estimator for F_k

We first describe a surprisingly simple basic estimator that gets the answer right in expectation; i.e., it is an unbiased estimator. Eventually, we shall run many independent copies of this basic estimator in parallel and combine the results to get our final estimator, which will have good error guarantees.

The estimator works as follows. Pick an item from the stream uniformly at random, i.e., pick a position $j \in [m]$ (using reservoir sampling algorithm). Count the length m of the stream and the number r of occurrences of our picked token x_j in the stream from that point on:

$$r = |\{\ell \geq j \mid a_\ell = a_j\}|.$$

The basic estimator is then defined to be $m(r^k - (r-1)^k)$.

The catch is that we do not know m beforehand, and picking a token uniformly at random requires the use of reservoir sampling, as seen in the pseudocode below.

Algorithm 2 A data stream algorithm for estimating $\sum_i f_i^k$

```
   Initialise: $(m, r, a) \leftarrow (0, 0, 0)$

   Process item $j$:
1: $m \leftarrow m + 1$
2: $\beta \leftarrow$ random bit with $\Pr[\beta = 1] = 1/m$
3: if $\beta = 1$ then
4:     $a \leftarrow x_m$
5:     $r \leftarrow 0$
6: if $a = x_m$ then
7:     $r \leftarrow r + 1$

   Output: $m(r^k - (r-1)^k)$
```

This algorithm uses $O(\log m)$ bits to store m and r, plus $\lceil \log n \rceil$ bits to store the token a, for a total space usage of $O(\log m + \log n)$.

4.5.3 Analysis of the Basic Estimator

For the analysis, it will be convenient to think of the algorithm as picking a random token from the stream in two steps, as follows.

1. Pick a random token value, $a \in [n]$, with $\Pr[a = x_j] = f_{x_j}/m$ for each $j \in [n]$.
2. Pick one of the f_a occurrences of a in the stream uniformly at random.

Let random variables A and R denote the (random) values of a and r, respectively, after the algorithm has processed the stream, and let X denote its output. Taking the above viewpoint, let us condition on the event $A = a$, for some particular $a \in [n]$. Under this condition, R is equally likely to be any of the values $\{1, \ldots, f_a\}$, depending on which of the f_a occurrences of a was picked by the algorithm. Therefore,

$$\mathbb{E}[X \mid A = a] = \mathbb{E}[m(R^k - (R-1)^k)|A = a] = \sum_{i=1}^{f_a} \frac{1}{f_a} m(i^k - (i-1)^k) = \frac{m}{f_a}(f_a^k - 0^k).$$

Thus,

$$\mathbb{E}[X] = \sum_{a=1}^{n} \Pr[A = a] \cdot \mathbb{E}[X|A = a] = \sum_{a=1}^{n} \frac{f_a}{m} \cdot \frac{m}{f_a} \cdot f_a^k = F_k.$$

This shows that X is indeed an unbiased estimator for F_k.

We shall now bound $\mathrm{Var}[X]$ from above. Calculating the expectation as before, we have

$$\mathrm{Var}[X] \le \mathbb{E}[X^2] = \sum_{j=1}^{n} \frac{f_j}{m} \sum_{i=1}^{f_j} \frac{1}{f_j} m^2 (i^k - (i-1)^k)^2 = m \sum_{j=1}^{n} \sum_{i=1}^{f_j} (i^k - (i-1)^k)^2. \tag{1}$$

By the Mean Value Theorem, for all $x \ge 1$, there exists $y \in [x-1, x]$ such that

$$x^k - (x-1)^k = ky^{k-1} \le kx^{k-1},$$

where the last step uses $k \ge 1$. Using the bound in Eq. (1), we get

$$\begin{aligned}
\mathrm{Var}\, X &\le m \sum_{j=1}^{n} \sum_{i=1}^{f_j} k i^{k-1} (i^k - (i-1)^k) \\
&\le m \sum_{j=1}^{n} k f_j^{k-1} \sum_{i=1}^{f_j} (i^k - (i-1)^k) \\
&= m \sum_{j=1}^{n} k f_j^{k-1} f_j^k \\
&= k F_1 F_{2k-1}.
\end{aligned} \tag{2}$$

For reasons we shall soon see, it will be convenient to bound $\mathrm{Var}[X]$ by a multiple of $\mathbb{E}[X]$, i.e., F_k^2. To do so, we shall use the following lemma (stated without proof here).

Lemma 4.3 *Let $n > 0$ be an integer and let $w_1, \ldots, w_n \ge 0$ and $k \ge 1$ be reals. Then,*

$$\left(\sum w_i\right)\left(\sum w_i^{2k-1}\right) \le n^{1-1/k} \left(\sum w_i^k\right)^2,$$

where all the summations range over $i \in [n]$.

Using the above lemma in Eq. (2), with $w_j = f_j$, we get

$$\mathrm{Var}[X] \le k F_1 F_{2k-1} \le k n^{1-1/k} F_k^2. \tag{3}$$

We are now ready to present our final F_k estimator, which combines several independent basic estimators.

4.5.4 The Median-of-Means Improvement

Unfortunately, we cannot take the median of a small number of our basic estimator directly to obtain a good estimator. This is because its variance is so large that we are unable to bound the probability of a large relative deviation in the estimator from below. So, we first bring the variance down by averaging a number of independent copies of the basic estimator, and then apply the median trick. The next theorem — a useful general-purpose theorem — quantifies this precisely.

Lemma 4.4 *There is a universal positive constant c such that the following holds. Let X be the distribution of an unbiased estimator for a real quantity Q. Let $\{X_{ij}\}_{i\in[t],j\in[r]}$ be a collection of independent random variables with each X_{ij} distributed identically to X, where*

$$t = c\log\frac{1}{\delta}, \text{ and } r = \frac{3\,\mathrm{Var}[X]}{\epsilon^2\mathbb{E}[X]^2}.$$

Let $Z = \mathit{median}_{i\in[t]}\left(\frac{1}{r}\sum_{j=1}^{r} X_{ij}\right)$. Then, we have $\Pr[|Z-Q|\ge \epsilon Q]\le\delta$.

Proof. For each $i\in[t]$, let $Y_i = \frac{1}{r}\sum_{j=1}^{r} X_{ij}$. Then, by linearity of expectation, we have $\mathbb{E}[Y_i] = Q$. Since the variables X_{ij} are (at least) pairwise independent, we have

$$\mathrm{Var}[Y_i] = \frac{1}{r^2}\sum_{j=1}^{r}\mathrm{Var}[X_{ij}] = \frac{\mathrm{Var}[X]}{r}.$$

Applying Chebyshev's inequality, we obtain

$$\Pr[|Y_i - Q|\ge\epsilon Q]\le\frac{\mathrm{Var}[Y_i]}{\epsilon^2Q^2} = \frac{\mathrm{Var}[X]}{r\epsilon^2\mathbb{E}[X]^2} = \frac{1}{3}.$$

Now an application of a Chernoff bound tells us that for an appropriate choice of c, we have $\Pr[|Z-Q|\ge\epsilon Q]\le\delta$. □

Using this lemma, we can build a final estimator from our basic AMS estimator, X, for F_k: We simply compute the median of $t = c\log(1/\delta)$ intermediate estimators, each of which is the mean of r basic estimators. To make the final estimator a good approximation to F_k, we need

$$r = \frac{3\,\mathrm{Var}[X]}{\epsilon^2\mathbb{E}[X]^2}\le\frac{3kn^{1-1/k}F_k^2}{\epsilon^2F_k^2} = \frac{3k}{\epsilon^2}\cdot n^{1-1/k},$$

where the inequality uses Eq. (3). This leads to a final space bound that is tr times the space usage of a basic estimator, i.e.,

$$\text{space}\le tr\cdot O(\log m+\log n) = O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\cdot kn^{1-1/k}(\log m+\log n)\right).$$

The above bound is good, but not optimal. The optimal bound (up to polylogarithmic factors) is $O(\epsilon^{-2}n^{1-2/k})$ instead; there are known lower bounds of $\Omega(n^{1-2/k})$ and $\Omega(\epsilon^{-2})$.