CS-448 Sublinear Algorithms for Big Data Analysis February 19, 2025

Lecture 1

Lecturer: Michael Kapralov

1 What this course is about

Historically, the golden standard for efficiency of algorithms has been the notion of linear or near-linear runtime (in the size of the input). Classical examples of such algorithms are sorting ($O(n \log n)$ time using QuickSort, for example) or the Fast Fourier Transform (similarly, $O(n \log n)$ time via FFT of Cooley and Tukey).

The sizes of the data sets that we need to process have been growing very rapidly for a while now, and this trend does not show signs of slowing. As a consequence, even linear time algorithms become prohibitively expensive for many modern applications: sometimes the data is so large that even accessing it in its entirety is not feasible. For example, imagine being given a node in a very large graph (e.g. social network) and wanting to find out which 'communities' the node belongs to. One would like to be able to find these communities 'locally', i.e. by exploring a small neighborhood of the node, as opposed to by running a computation on the entire social network. In other settings the data is too large to be stored in the memory of the computing device. A classical example here is network flow analysis that routers, which possess limited memory, must perform. A similar one is the analysis of query patterns on a major search engine such as Google: how does one find the most frequent queries in the stream of queries on Google? Storing the entire stream of queries is not feasible, but it turns out that the most frequent queries can be found approximately using a small amount of space.

In this course we will study *sublinear* algorithms, which are specifically tailored to processing large datasets using very constrained resources, namely resources *sublinear* in the size of the input. Sublinear algorithms come in various flavors, depending on the computation resource that we would like to minimize:

- Sublinear Time: In this scenario, we wish to evaluate some function f on some data. However, we can't afford reading the entire data set. A sublinear time algorithm is essentially a method of sampling a subset of the dataset and computing (an approximation to) the desired function based on the sample. The above example of finding a community in a social network that a given node belongs to falls into this category, and the sampling is done using a random walk in the social network.

- Sublinear Space: The data is too large to be stored in memory, and we wish to evaluate some function f on the input while using $o(n)$ space. This is known as the *streaming* model of computation. The above example of find the most frequent items in a large data stream falls into this category. The main algorithmic techniques is that achieves sublinear space is a combination of hashing and random projections. The resulting algorithm can be thought of as an approximate hashing scheme.

- Sublinear Communication: The data is too big to be stored on one machine, so the data is divided into chunks and stored on multiple machines. The goal now is to compute some function f on the data, while using $o(n)$ bits of communication between the different machines, usually while minimizing the number of rounds in the computation. This corresponds to the *massively parallel computing* setting.

Sublinear algorithms are typically approximate and randomized, and we will prove formal information theoretic lower bounds that this is necessary. Since randomization is key, we do a quick probability review, and then design a very basic sublinear algorithm for *counting*. The algorithm will be approximate and randomized, but will use an extremely small amount of space.

2 Probability Review

We start by reviewing some probability tools that we will be using in the analysis of the algorithms presented in this course. In particular, we present some bounds on the probability that a random variable deviates from its expected value. For a more complete reference, we refer the reader to the following textbook [MR95].

Let X be a discrete random variable taking values in $S \subseteq \mathbb{R}$, we define its expected value and its variance respectively as follows

$$\mathbb{E}[X] = \sum_{x \in S} x \Pr[X = x]$$

$$Var[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] = E[X^2] - \mathbb{E}[X]^2$$

Theorem 1 *(Markov's Inequality) Let X be a non-negative random variable. Then for all $a > 0$,*

$$\Pr[X \geq a\mathbb{E}[X]] \leq \frac{1}{a}$$

Proof We assume that X takes a positive value with some positive probability, as otherwise the result is trivial.

We then have

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{x < a\mathbb{E}[X]} x \Pr[x] + \sum_{x \geq a\mathbb{E}[X]} x \Pr[x] \\
&\geq \sum_{x \geq a\mathbb{E}[X]} x \Pr[x] \\
&\geq a\mathbb{E}[X] \sum_{x \geq a\mathbb{E}[X]} \Pr[x] \\
&= a\mathbb{E}[X] \Pr[x \geq a\mathbb{E}[X]],
\end{aligned}$$

where the first inequality follows from X being non-negative. Dividing both sides of the equation by $\mathbb{E}[X]$ gives the result. ■

Chebyshev's inequality uses more information, namely the second moment of the random variable:

Theorem 2 *(Chebyshev's Inequality) Let X be a random variable with expectation μ_X and variance σ_X^2. Then for every $\lambda > 0$,*

$$\Pr[|X - \mu_X| > \lambda] \leq \frac{\sigma_X^2}{\lambda^2}$$

Proof We apply Markov's inequality with parameter $a = \lambda^2/\sigma_X^2$ to the non-negative random variable $Y = (X - \mu_X)^2$. Combining this with $\mu_Y = Var[X]$, we prove the theorem. ■

Sums of independent random variables satisfy much tighter tail bounds hold. Let $X_1, X_2 \ldots X_n$ be independent Bernoulli random variables such that $X_i = 1$ with probability p and 0 otherwise, and let $Y = \sum_i X_i$. Then by the Central Limit Theorem $\frac{Y - \mathbb{E}[Y]}{\sqrt{Var(Y)}}$ approaches $\mathcal{N}(0,1)$ (the standard normal distribution) as $n \to \infty$. Therefore, for $t > 0$

$$\Pr[|Y - \mathbb{E}[Y]| > t\sqrt{Var(Y)}] \approx \frac{1}{t} e^{-t^2/2}$$

Chernoff bounds provide non-asympotic upper bounds on the probability of deviation from the mean by a given amount:

Theorem 3 *(Chernoff Bound) Let $Y = \sum_{i=1}^n X_i$ where X_i's are independent Bernoulli random variables, such that $X_i = 1$ with probability p_i and 0 otherwise. Also, let μ_Y denote $\mathbb{E}[Y]$. Then for any $\delta \in (0,1)$*

$$\Pr[|Y - \mu_Y| > \delta\mu_Y] \leq 2e^{-\frac{\delta^2 \mu_Y}{3}}.$$

Proof To prove the theorem, we need to show

1. $\Pr[Y > (1+\delta)\mu_Y] \leq e^{-\frac{\delta^2 \mu_Y}{3}}$
2. $\Pr[Y < (1-\delta)\mu_Y] \leq e^{-\frac{\delta^2 \mu_Y}{3}}$

We start by proving the upper tail inequality: For any $t > 0$, we have

$$\Pr[Y > (1+\delta)\mu_Y] = \Pr[e^{tY} > e^{t(1+\delta)\mu_Y}]$$

due to monotonicity of $\exp(\cdot)$. By Markov's Inequality,

$$\Pr[Y > (1+\delta)\mu_Y] \leq \frac{\mathbb{E}[e^{tY}]}{e^{t(1+\delta)\mu_Y}}$$

To bound $\Pr[Y > (1+\delta)\mu_Y]$ it thus suffices to bound

$$\begin{aligned}
\mathbb{E}\left[e^{tY}\right] = \mathbb{E}\left[e^{t\sum X_i}\right] = \mathbb{E}\left[\prod_{i=1}^n e^{tX_i}\right] &= \prod_{i=1}^n \mathbb{E}[e^{tX_i}] && (1)\\
= \prod_{i=1}^n (e^t p_i + (1-p_i)) &= \prod_{i=1}^n (1 + p_i(e^t - 1)) && (2)\\
\leq \prod_{i=1}^n e^{p_i(e^t-1)} & && (3)
\end{aligned}$$

where the last equality in (1) follows from the independence of the random variables, and (3) follows since $(1+x) \leq e^x$ for any non-negative x. Putting all of this together, we get

$$\begin{aligned}\Pr[Y > (1+\delta)\mu_Y] &\leq \frac{\prod_{i=1}^n e^{p_i(e^t-1)}}{e^{t(1+\delta)\mu_Y}} \\ &= \frac{e^{\sum p_i(e^t-1)}}{e^{t(1+\delta)\mu_Y}} = \frac{e^{\mu_Y(e^t-1)}}{e^{t(1+\delta)\mu_Y}} = \left(\frac{e^{e^t-1}}{e^{t(1+\delta)}}\right)^{\mu_Y}\end{aligned}$$

Setting $t = \ln(1+\delta)$ yields the best bound, which can be easily seen by differentiating the last expression with respect to t and setting it to zero.

$$\Pr[Y > (1+\delta)\mu_Y] \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{\mu_Y}$$

Proof by calculus.

One has $(1+\delta)\ln(1+\delta) \geq \delta + \delta^2/3$, which yields

$$\Pr[Y > (1+\delta)\mu_Y] \leq e^{-\frac{\delta^2 \mu_Y}{3}}$$

as required. The proof of the lower tail can be proved by a similar approach. ■

3 The Counting Problem

The first problem we consider is the *counting problem*. In this problem, a number of events occur, and we want to design a counter for these events. Intuitively, this is the approximate timekeeping task: you listen to a clock tick, and would like to have an estimate of how much time has passed, i.e. how many ticks you have heard, and not spend too much of your memory on this task (i.e. certainly not know exactly how many seconds have passed). A different formulation is: we are given a stream of data $\mathbf{s} = (s_1, s_2, s_3 \cdots s_n)$ one item at a time, and we want to compute the length n of this stream.

The trivial way to do this is to maintain a counter, and increment it whenever we encounter an event. The space required by this algorithm is $\lceil \log_2 n \rceil$, which we write as $O(\log n)$ since we only focus on asymptotic complexity in this course. Note that if we want to compute the count exactly, then we can't do better than that, since $O(\log n)$ bits are necessary to store a number of value n. In order to improve the space requirement, we relax this problem to the *approximate counting problem*, where we allow our answer to be approximate and probabilistic.

3.1 The Approximate Counting Problem

In this version of the problem, we are interested in designing an algorithm that provides an (ϵ, δ)-approximation to the counting problem.

Definition 4 ((ϵ, δ)-approximate counter) *A (randomized) algorithm provides an (ϵ, δ)-approximation $\widehat{n}$ for a counter n if*

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

where the probability is over the internal randomness of the algorithm.

Below we present Morris Algorithm, one of the oldest known streaming algorithm (dating back to 1978), which uses $O(\log\log n)$ space to solve the approximate counting problem[1]. Before doing that, however, let's generate some intuition for why the bound of $O(\log\log n)$ bits is natural to expect. For that, imagine a simpler scenario where you are given the exact value of n, and would like to send a short message to your friend, from which your friend should be able to reconstruct a factor 2 approximation $\widehat{n}$ to n. What would you do? Simply round n to the closest power of two and send the power! Specifically, find $j \geq 1$ such that $2^{j-1} < n \leq 2^j$ and send j to your friend. This takes

We discourage such messaging in an exam setting.

$$\lceil \log j \rceil = \log_2 \log_2 n + O(1) = O(\log\log n)$$

bits. This is of course a simpler setting: we only solved the compression problem, where we are given n and want to write it down succinctly while allowing a factor 2 approximation. The actual counting problem asks us for a way to update such a compression continuously. The Morris counter provides a very clean solution: once you hear a tick of the clock (or, equivalently, a data stream element arrives), increment j by 1 with probability 2^{-j}. This turns out to give an unbiased estimator at all times. The details are presented below.

3.1.1 The Morris Algorithm

The following algorithm provides a $O(1)$-approximation with constant probability.

```
(1) We maintain a counter $X$: $X \leftarrow 0$
(2) For each event, increment $X$ w.p. $2^{-X}$
(3) Output $2^X - 1$
```

The space required by Morris is $O(\log\log n)$ since we are maintaining X, which can be of value at most $O(\log n)$, $O(\log\log n)$ bits suffice to store X.

This assumes free random bits.

Claim 5 *Let X_n be the value of X after the ocurrence of n events, then*

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

Proof We prove the claim by induction of the number of steps n.

Base case: $n = 0$**.** For $n = 0$, X_n is trivially 0, so $\mathbb{E}[2^{X_n}] = n + 1$ holds trivially for the base case.

[1]We will present a basic version of the algorithm that does not quite get optimal bounds, but still illustrates the main ideas. Recent work of [NY22] shows how to achieve asymptotic optimality.

Inductive step: $n \to n+1$**.** We assume that the claim holds for n, and show that it still holds for $n+1$.

$$\begin{aligned}
\mathbb{E}[2^{X_{n+1}}] &= \sum_{j=0}^{\infty} \Pr[X_n = j] \times \mathbb{E}[2^{X_{n+1}} | X_n = j] \\
&= \sum_{j=0}^{\infty} \Pr[X_n = j] \times (2^{-j} 2^{j+1} + (1 - 2^{-j}) 2^j) \\
&= \sum_{j=0}^{\infty} \Pr[X_n = j](1 + 2^j) \\
&= \sum_{j=0}^{\infty} \Pr[X_n = j] 2^j + \sum_{j=0}^{\infty} \Pr[X_n = j] \\
&= \mathbb{E}[2^{X_n}] + 1 \\
&= n + 2
\end{aligned}$$

where the last equality follows from the inductive hypothesis. ■

References

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, NY, USA, 1995.

[NY22] Jelani Nelson and Huacheng Yu. Optimal bounds for approximate counting. In Leonid Libkin and Pablo Barceló, editors, *PODS '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 119–127. ACM, 2022.