CS 395T: Sublinear Algorithms, Fall 2020 27 Aug 2020

Lecture 1: Course Introduction

Prof. Eric Price *Scribe: Devvrit*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

In this course, we focus on the following three main areas of computer science

1. Streaming Algorithm - when there's a constraint on space availability. Typically considered when data arrives in streams and we want to compute some property of the data using $o(n)$ space.
2. Compressed Sensing - constraint on the number of measurements we can make. When making complete observation on data is costly, we aim to make $o(n)$ measurements of data and compute some property.
3. Property testing - constraint on time. When we have enormous amount of data but we can't look at the entire data. So we aim to quickly find some property by querying a few points in $o(n)$ time

2 Property Testing

A few examples of property testing include:

1. Given a function, determine if it's monotonic
2. Given a graph, find if it's bipartite
3. Given a distribution, determine if it's uniform

Generally, the worst case sample complexity bound for determining the required property requires $\Omega(n)$ points to be sampled. Hence, we re-define our goal as follows. We aim to ascertain one of the two properties

1. f satisfies the property P with high probability; OR
2. f is ϵ-far from satisfying P with high probability

For any specific problem, we need to define the function f, the property P, and the definition of ϵ-far.

2.1 Testing function monotonicity

Given a function $f : [n] \to [0, 1]$, our goal is to find if f is monotonically increasing. That is, to check whether the following property holds

$$f(x) \le f(y)\ \forall x < y, \text{ where } x, y \in [n]$$

Observation: In the worst case we need to query all the points. For example, if only one point $x \in [n]$ is out of the monotonic order. Therefore, worst case sample complexity lower bound is $\Omega(n)$.

We thus rather aim to determine if $f : [n] \to [0, 1]$ is monotonically increasing (with high probability), or it requires $\ge \epsilon n$ values to be changed to become monotonic.

To understand the problem better, we consider a few algorithms and instances.
Algorithm 1: We sample $k = O(1)$ random points and check monotinicity among them.
Instance-1: All the points except ϵn fraction follow the monotonic property. For example, the first $(1 - \epsilon)n$ points are monotonically increasing, while the last ϵn points are decreasing. For this instance, the above proposed *Algorithm 1* fails.
Instance-2: Consider the function which is monotonic on all even numbers and odd numbers separately, and satisfies $f(2r + 1) < f(2r)\ \forall r \in [\lfloor n/2 \rfloor]$. For example, $f(x)$ for $x \in [10]$ having values $\{2, 1, 4, 3, 6, 5, 8, 7, 10, 9\}$. This function is 1/2-far from being monotonic. And any algorithm would need to sample at least one pair of consecutive numbers. The number of samples k needed for such an event to happen is $O(\sqrt{n})$ (refer Birthday paradox)

We'll later see in the course an algorithm which requires only $O(log(n)/\epsilon)$ samples and gives error probability at most $O\left(\frac{1}{log(n)}\right)$

2.2 Distribution Testing

Given n samples $\{X_1, \cdots, X_n\}$ from a distribution D, we need to determine if D is uniform or ϵ-far in Total Variation from uniform. Total Variation from Uniform distribution is defined as $||p - \mathcal{U}_n||_{TV} = \sum_{i=1}^{n} |p_i - \frac{1}{n}|$.

We again consider a few instances to understand the problem better.
Instance: Consider set $S \subseteq [n]$ of size $n/2$ chosen at random. Let D' be a uniform distribution on S. For this instance, we won't be able to distinguish between D' and a uniform distribution on $[n]$ unless we see at least one collision. Hence, again by Birthday paradox we need at least $\Omega(\sqrt{n})$ samples.
Algorithm: Count the number of collisions in the sample (of size $m = O(\sqrt{n})$). If it's greater than the expected number of collisions by some quantity, say, r (to be determined later), then output "Non-uniform". Else, output "Uniform".

We analyze the above algorithm. Consider a distribution $P = [p_1, p_2, \cdots, p_n]$ on the n points.

Considering m samples,

$$\mathbb{E}[\#collisions] = \binom{m}{2}\mathbb{P}[\text{first and second sample collide}]$$
$$= \binom{m}{2}\left(p_1^2 + \cdots p_n^2\right)$$

Note that $\left(p_1^2 + \cdots p_n^2\right)$ is convex, and minimized at $p_i = \frac{1}{n}\ \forall i$.

$$\begin{aligned}
\mathbb{E}[\#collisions] &= \binom{m}{2}\left(p_1^2 + \cdots + p_n^2\right) \\
&= \binom{m}{2}\sum_i \left(\left(p_i - \frac{1}{n}\right)^2 + \frac{2}{n}p_i - \frac{1}{n^2}\right) \\
&= \binom{m}{2}\sum_i \left(p_i - \frac{1}{n}\right)^2 + \frac{\binom{m}{2}}{n} \\
&= \binom{m}{2}||p - \mathcal{U}_n||_2^2 + \frac{\binom{m}{2}}{n} \\
&\geq \frac{\binom{m}{2}}{n}\left(1 + ||p - \mathcal{U}_n||_2^2\right) \\
&\geq \frac{\binom{m}{2}}{n}\left(1 + ||p - \mathcal{U}_n||_1^2\right)
\end{aligned}$$

where $\mathcal{U}_n$ represents uniform distribution on n items. And the last inequality comes from the fact that l_2-norm squared is at least as large as l_1-norm squared. Note that $||p - \mathcal{S}||_1^2 = ||p - \mathcal{S}||_{TV}^2$. Therefore, for a distribution D' which is ϵ-far from $\mathcal{U}_n$, the expected number of collisions is at least $(1 + \epsilon^2)\mu_{\mathcal{U}}$. We'll later see in the course using the above result, concentration inequality, and measuring the variance of #collisions to come up with an algorithm which uses $O(\frac{\sqrt{n}}{\epsilon^2})$ samples.

An intuition is as follows. We'll measure the number of collisions observed. If the #collisions are much more than the expected #collisions for uniform case, then the distribution is not uniform with high probability. Else it is uniform with high probability.
Let the #collisions be a sum of independent $\{0, 1\}$ variables (this assumption is just to give an intuition. In actual case, they are dependent). Let $\{X_1, \cdots, X_n\}$ denote these variables. In this case, the expectation and variance would be:

$$\mu = \mathbb{E}\left[\sum_i X_i\right] = \sum_i \mathbb{E}[X_i] = \sum_i \mathbb{P}[X_i]$$

$$\sigma^2 = Var\left(\sum_i X_i\right) = \sum_i var(X_i) = \sum_i \mathbb{P}[X_i](1 - \mathbb{P}[X_i]) \leq \mu$$

We want the gap in the number of observed collisions and expected collisions ($\epsilon^2\mu$) to be large, say, $> 10\sigma$ (if we want to bound the error probability using Chebyshev inequality). Which implies

$$\epsilon^2\mu \geq 10\sqrt{\mu}$$
$$\implies \mu \geq \frac{100}{\epsilon^4}$$

Also recall that the expected number of collisions for uniform case is $\frac{\binom{m}{2}}{n}$. Therefore,

$$\frac{\binom{m}{2}}{n} = \frac{100}{\epsilon^4}$$
$$\implies m = O\left(\frac{\sqrt{n}}{\epsilon^2}\right)$$

3 Streaming Algorithms

An example could be an order processing system. Where orders are being streamed, and the required information could be stored on disk/external memory. In such a scenario, we can't store all the stream, as we don't know how long the stream is, etc. Therefore, the goal is to compute some function of the data stream in $o(n)$ memory, like $O(log^c(n))$ or ideally $O(n^\epsilon)$ memory.

3.1 Counting distinct elements

We see a stream $\{X_1, X_2, \cdots, X_n\} \in U$ of n elements, where U is some giant universe. Our goal is to estimate the number of distinct values.

An exact, deterministic algorithm would require $\Omega(n)$ words space. We'll see later in the course that in order to get $(1 \pm \epsilon)$ approximate solution, there exists randomized algorithms which requires $O\left(\frac{log(n)}{\epsilon}\right)$. We'll also see another algorithm which further improves this to $O\left(\frac{1}{\epsilon^2} loglog(n)\right)$

4 Compressed Sensing

An example includes taking M.R.I of the brain. Rather than observing every pixel, only observe a few projections and include some prior information to get more information from the limited observations.