COMS 4995-8: Advanced Algorithms (Spring'21) Jan 11, 2021

Lecture Lecture 1: Introduction, Approximate Counting

Instructor: *Alex Andoni* Scribes: *Max Daniels*

1 Introduction

In this course, we will study the theory behind many modern, practical algorithms. We will focus on algorithms which are randomized, producing a correct result only with some fixed probability, and which are approximate, producing a result which may be off the true result by at most a constant factor. For example, instead of finding some minimized quantity $a_{\min}$, it may be easier to output a random approximation b with

$$\Pr\left(\text{output } b \text{ has } a_{\min} < b < (1+\epsilon)a_{\min}\right) \geq 90\%.$$

Our main tasks will be to design algorithms and analyze them. Analyzing an algorithm requires proving its correctness and studying its performance. Algorithms in this course may be approximately or non-deterministically correct and we may focus on performance analysis of the time usage, space usage, communication cost, etc.

2 Course Info

Here is the course website: `http://www.cs.columbia.edu/~andoni/advancedS21/index.html`. The course has no textbook but lectures will be scribed, with notes and additional resources for each lecture are available on the course page.

Mathematical Background. We are encouraged to review the self-evaluation test as preparation for the course. Familiarity with the following subjects will be important:

1. Probability: expectation, variance, Markov bounds, Chebyshev bounds.
2. Linear Algebra: span, subpaces, linear transformations.
3. Computer Science: big-O notation, basic algorithms for sorting and searching, graphs.

Grading: here is the grade breakdown.

1. (10%) Scribing Lectures
2. (55%) Homeworks. there are 5 homework assignments, roughly once every 2 weeks. Students can take 5 total late days. Afterwards, assignments receive a 10%-per-day point deduction.
3. (35%) Final Project. Students will work in groups of 2-3 and will submit two reports (5% each) and a final project (25% each). The final project can be reading based, implementation based, or research based.

3 Course Topics

We will discuss the following topics.

1. **Hashing**. We will design hash functions $h : U \to [n]$ which map items in some universe U, a generic set, to keys in the set $[n] = \{1, 2, \ldots, n\}$. We will use them to solve the Dictionary Problem:

   Given $S \subseteq U$, how can we preprocess S so that later,
   given arbitrary $x \in U$, we can cheaply query whether $x \in S$.

   We will use perfect hashing to solve the dictionary problem with $O(1)$ time queries.

2. **Sketching and Streaming**. How can we compress ("sketch") extremely large objects into useful smaller versions? For example, we may compress large objects into digests that can be easily compared for similarities or which provide a synopsis of the original object. As a real-world example, consider the problem of approximately counting the number of unique IP addresses seen by a router under heavy traffic. By embedding data into high dimensional spaces, we will design algorithms that leverage properties of high dimensional geometry.

3. **Nearest Neighbor Searches**. Given a high dimensional set of points, how can we efficiently compute the nearest neighbors of any particular point? This is another question related to high dimensional geometry.

4. **Graph Algorithms, Maximum Flows**. We have previously studied the Ford-Fulkerson algorithm which is not exactly polynomial time. We will study various polynomial time algorithms for max flow. We will also study *scaling algorithms*, which can be used to solve max flow in polynomial time.

5. **Spectral Graph Theory**. Graphs are often represented by matrices, for example the adjacency or degree matrix. We will relate spectral properties of these matrices (ie. properties of their eigenvalues) to important properties of the graph, like connectedness or the presence of clusters of well-connected nodes. Our goal is to study *spectral clustering algorithms*.

6. **Optimization**. How can we solve optimization problems of the form $\min_{x\in C} f(x)$ where x is constrained within $C \subseteq \mathbb{R}^d$? We will focus on *linear programs* (LPs), where $f(x)$ and C are linear. For example, $f(x) = c^T x$ and $C = \{x \in \mathbb{R}^d : Ax \geq b\}$ for $b, c \in \mathbb{R}^d$ and $A \in \mathbb{R}^{d\times d}$.

   We will study *interior point methods* for solving LPs in polynomial time as well as descent algorithms like *Gradient Descent* or *Newton Iteration*. As an application of LPs, we will study *Multiplicative Weight Updates* for learning-from-experts/boosting problems.

7. **Models of Large Scale Computation**. There are many models of computation which incorporate design elements of modern computers like parallel processors, cache memory, and cluster computing. We will study the design of algorithms for various computational models, such as parallel algorithms for MapReduce.

8. **Extra Things**. Perhaps the Fast Fourier Transform?

4 Morris's Algorithm for Approximate Counting

Here is the *counting problem*:

> **Input**: a stream of button presses, observed one at a time.
> **Goal**: output the number of observed button presses.
> **Performance metric**: use as little space as possible.

Any algorithm that exactly counts n distinct items must use at least $\log_2(n)$ bits, so no exact algorithm beats the simple approach of incrementing an integer for each button press. In the next few lectures, we will study a randomized, approximate algorithm that uses only $O(\log_2 \log_2(n))$ bits.

Algorithm 1 (Morris's Algorithm)**.**

```
1. Set $X = 0$.
2. For each input button press:
    (a) with probability $2^{-X}$, set $X = X + 1$.
3. Output $\hat{n} = 2^X - 1$.
```

We will analyze this algorithm next class. Here is some background material that we will use:

1. Expectations. For a random variable X, the expected value of X is $\mathbb{E}[X] = \sum_a a\Pr(X = a)$. Expectations are linear: for $a, b \in R$, $\mathbb{E}[aX_1 + bX_2] = a\mathbb{E}[X_1] + b\mathbb{E}[X_2]$.

2. Variance. The variance of X is $\sigma^2 = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$. The standard deviation σ measures how far typical values of X are from the mean.

A random algorithm should be correct on average, but it should also have small standard deviation. Otherwise, its typical outputs will be far from the average/correct value. Using the mean and variance, we can extract performance guarantees using *concentration bounds*:

1. *Markov Bound.* If X is non-negative, $\Pr(X > \lambda) \leq \frac{\mathbb{E}[X]}{\lambda}$.

2. *Chebyshev Bound.* $\Pr(|X - \mathbb{E}[X]| > \lambda) \leq \frac{\mathrm{Var}[X]}{\lambda^2}$.