COMS E6998-9: Algorithms for Massive Data (Fall'23) Sep 6, 2023

Lecture 1: Introduction, Streaming for Graphs

Instructor: *Alex Andoni* Scribes: *Andrei Coman*

1 Introduction

This is an advanced class on designing algorithms for massive data. The class is theoretical and involves no programming. The focus will be on algorithms that process big datasets as efficiently as possible (e.g. in linear or sub-linear time). Primarily, we will talk about tools and techniques for algorithm design and models of computation (e.g. restrictions on computation, constraints on space and time) on massive datasets.

2 Course Topics

• **Numerical Linear Algebra**: we will talk about designing fast numerical algorithms for linear algebra:

1. **Least Squares Regression (LSR)**: given a matrix A of of n d-dimensional points and a vector b of associated values

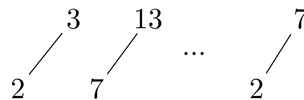
$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1d} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nd} \end{bmatrix} \in \mathcal{M}_{n\times d} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \in \mathcal{M}_{n\times 1}$$

we want to find $\arg\min_{x\in\mathbb{R}^d} \|Ax - b\|_2$. In one dimension, x determines the line closest to the input points $((a_i, b_i))_{i\in[n]}$. This problem admits a classic solution which uses the matrix-inverse of A. If we allow some approximation, can we do this faster?

2. **Compressed Sensing**: measure a high-dimensional signal using few measurements.

3. **Faster Fast Fourier Transform**: The Fast Fourier Transform (FFT) algorithm can be computed faster under certain conditions (e.g. sparsity).

• **Streaming (Sketching) for Graphs**: We are given a graph G with n nodes and m edges. The graph is presented to us edge by edge:



(i.e. access to the graph is restricted to reading it in the streaming order). Given some extra working-space much smaller than the length of the stream, compute a property of the graph (connectivity, distances, and others). Sketching refers to a summary of the input graph (much smaller than the graph itself) which allows the computation of the desired property. This connects to dynamic data structures on graphs.

• **Sublinear-time Algorithms**: the goal is to solve problems in time much smaller than the input size. In particular, we want to determine properties of the input dataset without accessing all the data. Some areas where such algorithms are used include:

1. **Graph problems** (e.g. determining connectivity, counting the number of connected components, estimating the average degree, estimating the cost of the minimum spanning tree). These problems are trivial if we have the entire graph, but are not immediately obvious if we only have query access to the graph.
2. **Distribution testing**. Given access to some distribution π over $[n]$ (which we can only simulate or query), we want to determine properties of this distribution. For example, is π uniform or close to some fixed distribution (e.g. a Gaussian distribution)? How many samples do we need to answer such questions? These topics are covered in more detail in Prof. Xi Chen's "Intro Property Testing" course.

• **Learning-augmented Algorithms**: Our standard approach will be to look at the worst-case analysis of algorithms (however, most datasets are not worst-case, and worst-case analysis is not predictive of the algorithms that run in real life; in these cases, average-case analysis is used). Learning-augmented Algorithms assume a worst-case input (i.e. adversarially generated), but have access to a "helpful" oracle (i.e. ML model). The oracle is useful on average, but cannot be fully trusted.

• **Distributed/Parallel Models of Computation**: These are examples of models of computation that restrict how we process our dataset. We assume the dataset is so large that it does not fit on one machine, so the computation is done by multiple machines.

To provide more efficient algorithms for most of these problems, we are required to give up the guarantee of solving the problem exactly. More specifically, we will consider:

• **Approximate algorithms**: An algorithm $\mathcal{A}$ outputs some answer $\hat{c}$ such that $c \leq \hat{c} \leq \alpha \cdot c$, where c is the real answer and $\alpha \geq 1$ is the approximation-factor.

• **Randomized algorithms**: The algorithm should output an answer within the desired range with some desired probability. Formally, for an approximate algorithm $\mathcal{A}$, we require $\Pr_{\mathcal{A}}[c \leq \hat{c} \leq \alpha \cdot c] \geq 90\%$, where this probability can be adjusted as needed using standard techniques. Note that the probability is taken over the random choices of algorithm $\mathcal{A}$ (such as taking random samples), and not over the input.

3 Administrative Details

For administrative details, consult the course description.

4 Problem: Graph Connectivity

We are given an undirected, unweighted graph G with n nodes and m edges. The m edges are streamed in arbitrary (i.e. possibly adversarial) order. Is G connected? Design an algorithm using $O(n)$ words of space (where, by a word, we understand an $O(\log n)$-bit register).

Algorithm 1: keep spanning forest of what we have seen so far

```
H = φ (the empty spanning forest)
foreach (i,j) ∈ G (in the stream)
    if i and j not connected in H
    then add (i,j) to H
output "connected" iff H is connected
```

Claim 1. *$\forall i,j \in [n]$, i and j are connected in H $\iff$ i and j are connected in G. In other words, H is a graph with the same connectivity information as G.*

Proof. The proof is by construction. We only drop an edge (i,j) when i and j are already connected in the spanning forest H, so there is no loss in dropping the edge. □

Claim 2. $|H| \leq n-1$.

Proof. H is a spanning forest, so there are no cycles by construction. A cycle can only be creating by adding an edge (i,j) that closes a cycle. However, that would require i and j to already be connected in H, in which case the new edge is dropped. □

Therefore, the algorithm uses $|H| \leq (n-1)$ space.

5 Problem: Distances in G

Given the same graph G, output the distance between two nodes x and y in G (i.e. the length of the shortest path between them in G). We will consider approximate algorithms with approximation-factor $\alpha = 2k-1$ for $k \geq 2$ integer. Below, $dist_G(x,y)$ denotes the shortest path distance between x and y in the graph G.

Algorithm 2: keep a graph H which approximates the distances in G

```
H = φ (the empty graph)
foreach (i,j) ∈ G (in the stream)
    if dist_H(i,j) > α
    then add (i, j) to H
output dist_H(x,y)
```

We will prove the following claims in the next lecture.

Claim 3. *$dist_H(x,y)$ is an α-approximation of $dist_G(x,y)$.*

Claim 4. $|H| \leq O(n^{1+1/k}) = O(n^{1+2/(\alpha+1)})$