ECS 289A Sublinear Algorithms for Big Data **Winter 2026**

Lecture 1: Overview and Introduction to Property Testing

Lecturer: Jasper Lee *Scribe: Jasper Lee*

1 Course Overview

In the study of algorithms, we aim to understand and quantify the amount of computational resources required to solve a problem. Commonly studied resources include:

- Time
- Space
- Randomness
- Qubits
- Communication

and more...

This course addresses the following central question:

> **What can we achieve with computational resources that are *sublinear* in the input size?**

We focus mostly on the resources of time and space. Surprisingly, we can do a lot even under severe resource restrictions. In particular, with sublinear time, we cannot even afford to read the entire input (not even a constant fraction). With sublinear space, while we may be allowed to read the entire input, we cannot store all of it and can instead rely on a small sketch of what we have seen. Throughout the semester, we will survey fundamental algorithms and algorithmic techniques in the area.

I designed the course to enable you gaining the following skills and knowledge by the end of the semester:

Formulating sublinear algorithmic problems You will gain intuition in what makes a computational problem tractable with sublinear resources.

Reading research papers You will learn the basic tools so you can read and understand sublinear algorithmic results, and be able to implement them.

Communicating technical ideas The coursework is designed as practice for you to communicate sophisticated technical ideas.

As a secondary goal, you will also gain familiarity and intuition with randomness, and understand that how randomness can in fact be very well-behaved. Furthermore, for those of you going onto graduate studies in theoretical computer science, the skills you gain from this course will (hopefully) form part of your research toolkit.

Please read the course missive for details on logistics, assessment and various course policies.

2 Preliminaries

You have in fact seen sublinear algorithms in previous courses before. Here are a few examples:

- Binary search
- Breadth-first search, if you are computing the hop distance between two vertices that are close to each other, then you only need to look at the vertices within their neighbourhoods.
- Various statistical estimation algorithms, such as estimating the mean of a population. Except for critical applications, you would probably just sample from the population instead of going through the entire population.

The first two examples are sublinear algorithms on combinatorial structures (lists and graphs) with strong assumptions about the input (especially sortedness for binary search), and the last example is on probability distributions.

Let us focus on the combinatorial examples for now. What is the key feature about these algorithms that enable their time sublinearity, different from other algorithms that are linear or worse? A important insight (that we also had earlier) is that they read only a small portion of the input, and we observe that it takes time to read each bit of the input. This introduces us to more notions of computational resources that we should develop a theory for:

Queries You can think of a combinatorial structure as a collection of "cells", for example a list is a list of cells and a graph (in an adjacency matrix representation) is a quadratic table of Boolean cells. The *query complexity* of an algorithm measures the number of cells it opens and reads from the input.

Samples For statistical algorithms working on probability distributions, the analogous measure is *sample complexity*. We count the required number of samples drawn from the underlying distribution, in order to achieve certain accuracy guarantees for the statistical task at hand.

Both the query complexity and the sample complexity of an algorithm lower bounds its time complexity, as we observed earlier. The measures also capture the amount of information needed from the input in order to solve a problem. They are central notions in the study of sublinear algorithms.

3 Introduction to Property Testing

In order to introduce the framework of *property testing*, consider the following toy problems.

Problem 1.1 (Toy Problem 1) Given a 0/1 list A of length n, test whether

- A is all 0s, versus
- A is not all 0s

with probability at least $2/3$.[1]

[1] Meaning that, in the first case, the testing algorithm should return "All 0s" with probability at least 2/3, and in the second case, the algorithm should return "Not all 0s" with probability at least 2/3.

The trivial algorithm is to deterministically check the entire list using exactly n queries. A randomised variant would be to randomly sample $O(n)$ spots and check. For a list with m many 1s, the probability of failing to catch a 1 is

$$\left(1 - \frac{m}{n}\right)^{O(n)} \leq \left(1 - \frac{1}{n}\right)^{O(n)} \leq e^{-O(1)} \leq \frac{1}{3}$$

Here, we used the inequality $1 + x \leq e^x$. Note also that this randomised algorithm has a *1-sided* error, because it never fails for the all 0s list.

Both of these (trivial) algorithms take linearly many queries and hence linear time, and unfortunately it is necessary for this simple problem. That is, there cannot be any sublinear algorithm. In homework 1, you will show the corresponding query lower bounds.

General lesson/heuristic: If you can't beat the game, change the game. What if we change the problem formulation and consider the following (a bit extreme) special case?

Problem 1.2 (Toy Problem 2) Given a 0/1 list A of length n, test whether

- A is all 0s, versus
- A is at least half 1s

with probability at least 2/3. We require no guarantees on the algorithm if A contains fewer than half 1s.

In this special case, all we need is 2 random queries! However, this is not a very practical problem. Can we generalise this second problem formulation just a bit, to make it more useful?

Problem 1.3 (Toy Problem 3) Given a 0/1 list A of length n, test whether

- A is all 0s, versus
- A has at least an ϵ-fraction of 1s

with probability at least 2/3. We require no guarantees on the algorithm if A contains fewer than an ϵ-fraction of 1s.

In this case, the number of random queries required is $O(1/\epsilon)$, yielding a (1-sided) failure probability upper bounded by

$$(1 - \epsilon)^{O(1/\epsilon)} \leq e^{-O(1)} \leq \frac{1}{3}$$

The query complexity grows as ϵ tends to 0, but it is completely independent of n and therefore a (very!) sublinear algorithm.

This toy example illustrates the key idea in a property testing problem: the existence of an ϵ-gap. Here, ϵ is a parameter independent of n, and it is often useful to think about it as a "constant". The constant gap is what allows the possibility of a sublinear algorithm – the first problem formulation had a gap of $1/n$ that is sub-constant, which we cannot leverage.

Note however that the existence of a (constant) gap means that we are solving an *approximate* problem, in this case, an approximate decision problem. It is up to you, the user/algorithm designer, to decide whether the approximation is acceptable in your use case or not.

3.1 General Property Testing Formulation

Suppose we have a set $\mathcal{C}$ of objects (e.g. lists, graphs), equipped with a metric (distance measure) $d : \mathcal{C} \times \mathcal{C} \to [0, 1]$.

Definition 1.4 (Property) A property $\mathcal{P} \subseteq \mathcal{C}$ is a set of objects. For example,

- $\mathcal{P}_n = \{\text{The length } n \text{ list of 0s}\}$
- $\mathcal{P}_n =$ The set of n vertex bipartite graphs

Given a property $\mathcal{P}$, we want to formulate a corresponding property testing problem, using $\mathcal{P}$ and the metric d. To do so, we formulate the notion of an object's *farness* from having property $\mathcal{P}$.

Definition 1.5 (ϵ-farness) Given an object $O \in \mathcal{C}$, its distance $d(O, \mathcal{P})$ from property $\mathcal{P}$ is its minimum distance to any object having the property. That is, $d(O, \mathcal{P}) = \min_{O' \in \mathcal{P}} d(O, O')$. We say that O is ϵ-far from $\mathcal{P}$ if $d(O, \mathcal{P}) \geq \epsilon$, or equivalently, for all objects $O' \in \mathcal{P}$ satisfying the property, $d(O, O') \geq \epsilon$.

Now we can define the generic property testing problem.

Problem 1.6 (Testing for Property $\mathcal{P}$) Given an object $O \in \mathcal{C}$, test whether

- $O \in \mathcal{P}$, versus
- O is ϵ-far from $\mathcal{P}$.

with probability at least $2/3$. We require no guarantees on the algorithm if O has distance less than ϵ from $\mathcal{P}$.

The goal is to minimise the query and time complexities of our algorithm.

Remark 1.7 In this problem formulation, we fix the ϵ-gap and ask what is the minimum query complexity. However, in practice, we might have a fixed budget and want to run a test with the smallest ϵ-gap. To do so, we frequently would need an algorithm whose execution does not explicitly depend on ϵ, for example our testing algorithm for testing whether a list is all 0s.

4 Point Set Diameter

We move on to a different and more serious problem of finding the *diameter* of a point set, to illustrate a different point about sublinear algorithmic problems.

Definition 1.8 ((Pseudo-)metric on a Set S) A (pseudo-)metric on the set S is a function $d : S \times S \to \mathbb{R}_{\geq 0}$ such that

- For all $x, y \in S$, $d(x, y) = d(y, x)$ (Symmetry)
- For all $x, y, z \in S$, $d(x, z) \leq d(x, y) + d(y, z)$ (Triangle Inequality)

Definition 1.9 (Diameter) The *diameter* of a point set S equipped with metric d is the maximum distance between any pair of points in S, that is $\max_{x,y \in S} d(x, y)$.

Problem 1.10 (2-Approximation for Point Set Diameter) Given a point set S of size m, find a 2-approximation of its diameter, that is, the answer should be within a factor of 2 of the true diameter. The algorithm should minimise the number of queries/evaluations to the function d.

Before we think about what a sublinear algorithm for the problem could be, let's determine the (easy) linear time algorithm as a benchmark.

Algorithm 1 Linear Time Algorithm for Point Set Diameter

Examine all $\binom{m}{2} = \Theta(m^2)$ pairs of points and return maximum distance.

An important point to be careful about is that "linear" is (typically) in terms of the size of the input, namely the structure we are querying. In this case, the input we are considering is the metric d (and not the point set S). Since the function d is really an $m \times m$ matrix, the "linear" we benchmark against is $O(m^2)$.

Now for the sublinear algorithm, made possible by the approximation (of a factor of 2).

Algorithm 2 2-Approximation Algorithm for Point Set Diameter

1. Pick an arbitrary $x \in S$.
2. Return $\max_{y \in S} d(x, y)$.

Proposition 1.11 *The answer returned by Algorithm 2 is at least half the diameter of S.*

Proof. Let i, j be a pair of points farthest apart in S, then

$$d(i, j) \leq d(i, x) + d(x, j) \leq d(x, y) + d(x, y)$$

□