6.1220 LECTURE 17

BRENDON JIANG

Contents

1. Sublinear Algorithms

Sometimes when we deal with very large scale of data, it is impossible to access all of it. The entire set of data might be too big to fit inside the computer's memory. Or, perhaps, after we access some data, it changes quickly, rendering our answers obsolete.

Definition 1.1

In the world graph, each node is a person and each edge represents people that know each other. Is the graph connected?

The question of connectedness for everyone on earth is too big and changes too fast for linear time algorithms. Thus, we need to design some algorithm that can perform the task without viewing most of the data. This constraint means that we can't answer statements that involve "for all" or "exactly" type statements. Examples include:

- Exactly how many individuals on earth are left-handed?
- Are all individuals connected?

However, we can perhaps answer approximately to some of the questions (e.g. approximately how many individuals on earth are left-handed?).

In this lecture, we consider two types of approximation.

- "Classical" approximation for optimization problems, where the output is a number close to optimal value.
- Property testing for decision problems: output is correct for some input close to given input.

Date: November 2024

1.1. Approximation Problems

Definition 1.2: Diameter of a Point Set

The input of the problem consists of pairwise distance of m points given by a $m \times m$ matrix D such that D_{ij} is the distance from i to j, and the triangle inequality is satisfie ($D_{ij} \leq D_{ik} + D_{kj}$). The goal here is teo estimate the diameter of the set of points, which is

$$d^* = \max_{i,j} D_{ij}$$

We will produce a 2-approximation to this problem.

Solution 1.3: 2-approximation solution

We can pick an arbitrary vertex k. We iterate over all other vertices l, and output the maximum D_{kl}.

Proof

The algorithm we have runs in $O(m)$ time. Suppose i, j are such that $D_{ij} = d^*$. Then by the triangle inequality, $D_{ij} \leq D_{ik} + D_{kj} \leq D_{kl} + D_{kl} = 2D_{kl}$.

1.2. Decision Problems

Our main goal with decision problems property testing is to quickly distinguish objects that have specific property from those that are far from having the property. This could be useful in a variety of cases:

- As a natural question
- Just as good as an algorithm when data is constantly changing
- Fast sanity check to rule out bad inputs or to decide when expensive processing is worthwhile.

Definition 1.4: Sortedness of a Sequence

Given a list $y_1, y_2, \ldots y_n$. Is the list sorted?

To answer the full decision problem, we must look at each element, which requires $\Omega(n)$. However, we can quickly test if the list is sorted vs far from being sorted. Here, we define ”quickness” as the number of queries we make, where each query asks for the value of element at position i. We will shoot for $O(\log n)$ queries.

What do we mean by ”far” from being sorted?

Definition 1.5: ϵ-close

A list of size n is ϵ-close to sorted if we can delete at most $\epsilon \cdot n$ values to make it sorted. Otherwise, it is ϵ-far.

We want for our algorithm to

- Pass sorted lists
- Fail lists that are ϵ-far.
- Don't care if list is neither sorted or $\epsilon - far$.
- Probability of making a correct call is $\frac{3}{4}$.
- Can test in $O\left(\frac{1}{\epsilon \log n}\right)$

Solution 1.6: An attempt

We can pick a random i and test whether $y_i \leq y_{i+1}$

This algorithm does not work, because there are test cases that break the algorithm such as $y = \{1, 2, \ldots, n/2, 1, 2, \ldots, n/2\}$. The probability of picking the point where we have $y_i > y_{i+1}$ is $\frac{1}{n}$.

Solution 1.7: A second attempt

We can pick random i, j and test whether $y_i \leq y_j$

This algorithm also does not work, by considering $y = \{4, 3, 2, 1, 8, 7, 6, 5, \ldots, 4k, 4k-1, 4k-2, 4k-3\}$. The largest monotone increasing sequence here is $n/4$, but we must pick i, j in the same 4-block to see the problem.

Solution 1.8: Final algorithm

We assume for simplicity that the list is distinct (append a decimal point .i to x_i). Repeat the following steps for $\left\lceil \frac{2}{\epsilon} \right\rceil$ times. Pick some random i and query the value of y_i. We can then pretend to do binary search for y_i on the list using log queries. If the binary search finds any inconsistency or the search doesn't end up at location i, the algorithm fails. If the algorithm doesn't fail anywhere, algorithm passes.

Proof

The run time of this algorithm is $O(\epsilon^{-1} \log n)$. But why does this algorithm work?
If the list is sorted, the algorithm definitely will pass. If the list is not sorted however, it is not clear what will happen.

Define index i to be good if the binary search for y_i is successful. Our algorithm essentially takes $\frac{2}{\epsilon}$ indices i at random and pass if they are all good. Note that the good elements form an increasing subsequence, because they all lie on the same binary tree formed by binary searches, which obeys in-order traversal. This means if the the number of good elements is $\geq (1-\epsilon)n$, then the list is ϵ-close to being sorted. The contrapositive states if the list is ϵ-far from sorted, then the number of good elements is less than $(1-\epsilon) \cdot n$ or greater than $\epsilon \cdot n$. Thus, the probability that no repitation fails is equivalent to $(1-\epsilon)^(2/\epsilon) \leq \frac{1}{e^2} < \frac{1}{4}$. In conclusion, if the list is ϵ - far from sorted, then the probability that the test fails is $\geq \frac{3}{4}$. If the list is sorted, then the test accepts it with probability 1.

Now we look at the connecitivity problem again.

Definition 1.9: $\epsilon-$close

G is $\epsilon-$close to connected if we can add $< \epsilon dn$ edges and transform it to connected.