CS 59000 CTT Current Topics in Theoretical CS Aug 28, 2012

Lecture 3

Lecturer: Elena Grigorescu *Scribe: Purnima Kapoor*

1 Intro

In the previous lectures we introduced various sublinear models of computation and we talked about some basic probabilistic tools for analyzing the performance of randomized algorithms. We then showed two applications of Chernoff's bound. We saw how to quickly estimate the fraction of 1's in a binary string, and then we saw how to reduce the error of a randomized algorithm that makes a yes/no decision, say from 49% to 1%. More generally, if such an algorithm outputs incorrect answers with probability $1/3$ then by running it $\log \frac{1}{\delta}$ times and outputting the majority answer, the error probability can be reduced to $\delta > 0$.

In this lecture we will revisit the Property testing model, which will be the focus of a few future lectures as well. We will then show a property testing algorithm for the problem of testing connectivity in sparse graphs.

2 The Property Testing model

The Property testing model is a randomized model of computation that relaxes the usual decision model. While in the classical decision model an algorithm distinguishes between inputs having a property and inputs not having the property, in the Property Testing model an algorithm is required to only distinguish between inputs having the property and inputs that are very far from having the property. That is, for the remaining inputs (inputs that are close to having the property), the algorithm can output an arbitrary answer.

What do we gain from this relaxation? It turns out that many hard decision problems (say for problems that are NP complete, eg. 3-colorability) one can obtain very efficient (sublinear time, or even constant time) property testing algorithms. Sublinear time PT algorithms are also interesting even for problems for which their classical decision version can be solved in polynomial time. In practice, such algorithms are useful when dealing with massive data sets (say, internet data, social data, biological data) and when dealing with data that changes at a rapid rate. In fact we're running property testing algorithms in our daily lives; for example, if you would be offered a few sample chocolate bytes from a chocolate store it could help you quickly decide whether you'd want to enter the store (if their brand seems great) or stay away from it (if their brand seems very bad).

We'll next start formalizing the model.

We will denote by a *property* $\mathcal{P}$ the collection of objects that share a certain property. For example,

1. $\mathcal{P} = \{$graphs that can be colored with 3-colors$\}$
2. $\mathcal{P} = \{$bipartite graphs$\}$
3. $\mathcal{P} = \{$linear functions$f : Z_2^n \to Z_2\}$
4. $\mathcal{P} = \{$uniform distribution over domain $D.\}$

5. $\mathcal{P} = \{f : D \to \{0,1\}, f(x) = 1\}$.

Input access A PT algorithm for a property $\mathcal{P}$ is given oracle/black-box access to its input. If the input is represented as a function $f : D \to R$, then the algorithm A tosses some random coins based on which it picks a query $i \in D$ and receives back $f(i)$ from the oracle for f. Based on the answers to a few queries, A should output a yes/no verdict, that represents its decision that distinguishes between the input f having the property $\mathcal{P}$ or being far from having it. If the input is represented as a graph $G = G(V, E)$ with vertex set V and edge set E, then a query is a pair $(v, u) \in V \times V$ and the answer to this query is 1 if there is an edge between u and v in E, and 0 otherwise.

Notions of distance What does it mean to be far/close to a property? The most common notion of distance over discrete domains is the Hamming distance. The relative Hamming distance between functions $f, g : D \to R$ is $\delta_H(f, g) = \frac{|\{x:f(x)\neq g(x)\}|}{|D|}$. The relative distance between f and a property $\mathcal{P}$ is $\delta_H(f, \mathcal{P}) = \min_{g\in\mathcal{P}} \delta_H(f, g)$.

For two graphs G_1, G_2 on n vertices, a typical notion of distance is given by the number of edges that need to be added/removed to/from G_1 in order to obtain G_2. The relative distance between two graphs is computed with respect to an upper bound on the number of edges in the graphs, and it depends on the graph representation model (ie. adjacency matrix model (for dense graphs) or adjacency list model (for sparse graphs)).

We say that f is ϵ-far from $\mathcal{P}$ if $\delta_H(f, \mathcal{P}) > \epsilon$. Otherwise f is ϵ-close to $\mathcal{P}$. A graph on n vertices in the dense graph model is ϵ-far from a graph property $\mathcal{P}$ if it needs to be modified in $\epsilon\binom{n}{2}$ many edges in order to obtain a graph in $\mathcal{P}$. A graph on n vertices and of bounded degree d (in the sparse graph model) is ϵ-far from a property $\mathcal{P}$ if it needs to be modified in ϵnd many edges in order to obtain a graph from $\mathcal{P}$.

We can now formalize the PT model.

Definition 1 *A property $P_n \subset \{h : D \to R\}$ (where D and R are the finite domain and range, and $|D| = n$) is called* k-locally testable *if there exists a randomized algorithm A such that*

1. *on input $\epsilon > 0$ and $f : D \to R$ that A can access via queries*
2. *A makes $k = k(\epsilon, n)$ queries $q_1, q_2, \ldots, q_k$ to f*
3. *based on the answers it receives (namely the values $f(q_1), f(q_2), \ldots, f(q_k)$) A will output as follows:*
   (a) *(Correctness:) If $f \in P_n$, then A says yes with probability $\frac{2}{3}$;*
   (b) *(Soundness:) If f is ϵ-far from P_n then A says no with probability $\frac{2}{3}$.*

The main parameter of the algorithm is its query complexity $k = k(\epsilon, n)$, which is often a measure of the running time of the algorithm. The goal is to obtain algorithms of query complexity $k << n$. We will see a few PT algorithms of query complexity independent of n, namely $k = k(\epsilon)$.

A PT algorithm is single-sided if it always accepts inputs in $\mathcal{P}$. Otherwise it is 2-sided.

In the first lecture we saw a very simple example of a single-sided PT algorithm running in time $\frac{2}{\epsilon}$ that was a tester for the property $\mathcal{P} = \{f : [n] \to \{0,1\}, f(i) = 1\}$. As a quick reminder,

the way the algorithm operated was by picking a random sample of $\frac{2}{\epsilon}$ many elements $i \in D$ and rejecting if for any such i, $f(i) = 0$. If so, then it would have found a witness to f not satisfying the property and rejected the input. Otherwise, if all the queries returned the value 1 then the algorithm accepted its input f.

We emphasize here that usually, when building a one-sided error PT algorithm, we look for small violating patterns in the input (i.e. witnesses to not having the property). If such patterns are found we reject. A typical test A usually looks as follows: on input ϵ, f, repeat s times a basic subroutine A' and only accept if A' has never rejected. For example, in the test above the basic subroutine was: pick a random $i \in [n]$ and reject if $f(i) = 0$. What should s be? The value of s can be computed from analyzing the probability of finding a witness in the basic test. We saw in the analysis of the test above from the 1st lecture that if the basic test catches a witness with probability p, then repeating it $s = 2/p$ times increases the probability of catching a witness to $2/3$. We will use this observation again and again in the analysis of future property testing algorithms.

In the remainder of the lecture we show the first example of a non-trivial PT algorithm, namely for the problem of testing connectivity of graphs.

3 Testing graph connectivity

Earlier in the lecture we mentioned about the fact that the distance between graphs depends on the representation model. If G and H are dense graphs (i.e. graphs with αn^2 edges where $\alpha > 0$, then

$$\delta(G, H) = \frac{1}{\binom{n}{2}} \cdot (\#\text{ edges that need to be modified in order to get from } G \text{ to } H).$$

On the other hand, if G and H are sparse graphs, i.e. in the adjacency list model where M is a bound on the number of edges in the graphs, then

$$\delta(G, H) = \frac{1}{M} \cdot (\#\text{ edges that need to be modified in order to get from } G \text{ to } H).$$

Let $\mathcal{P} = \{\text{graphs that are connected}\}$. Suppose that we work in the adjacency matrix model. What would be a PT algorithm for testing $\mathcal{P}$? First, let us understand what being far from $\mathcal{P}$ means. By definition a graph that is ϵ-far from $\mathcal{P}$ should need to be modified in $\epsilon\binom{n}{2}$ places in order to become connected. But notice that one can easily make any n-vertex graph into a connected graph by only adding $n - 1$ edges, i.e. assymptotically much fewer than $\epsilon\binom{n}{2}$. So any graph in the dense graph model is ϵ close to being connected, and hence the trivial test that always accepts is a test for connectivity.

Things become more interesting though if we focus on graphs where the number of edges is linear in n, for example in the bounded degree graph model where a bound nd on the number of edges is known.

In what follows we show a variant of a test for connectivity that can be modified to work in the bounded degree model. We show how to test the property $\mathcal{P}$ defined above (i.e. the set of connected graphs) under the following notion of distance.

Definition 2 *Graph G is ϵ-far from $\mathcal{P}$ if we need to add ϵnd edges to G in order to obtain a graph in $\mathcal{P}$(i.e. a connected graph).*

The test and its analysis exemplify the basic ideas that are also used in testing connectivity in the bounded degree list model, where in order to maintain the degree bound one is allowed to both add and delete edges from the graph. For a treatment of this model we refer to [1].

Theorem 3 *$\mathcal{P}$ is testable with one-sided error by a test that makes $O(\frac{1}{(\epsilon d)^3})$ queries.*

Notice that the query complexity is a constant, independent of the size of the input graph. Also, we are looking for a one sided error test, namely a test that will always accept connected graphs. And we want to reject graphs that are far from being connected. What do graphs that are far from being connected look like?

The next lemma tells us about the structure of such graphs.

Lemma 4 *If G is ϵ-far from being connected then G has at least $\epsilon dn+1$ connected components.*

Proof Suppose G has fewer than $\epsilon dn + 1$ connected components. Then one can connect all of them by adding $\epsilon dn - 1$ edges, which contradicts the fact that G is ϵ-far from $\mathcal{P}$.

■

Lemma 5 *If G is ϵ-far from being connected then it has $> \frac{\epsilon}{2}dn$ connected components of size $< \frac{2}{\epsilon d}$.*

Proof We show that G has $\leq \frac{\epsilon}{2}dn$ components of size $\geq \frac{2}{\epsilon d}$. Indeed, otherwise G would have $> \frac{\epsilon}{2}dn$ components of size $\geq \frac{2}{\epsilon d}$ and so it would have more than n vertices, a contradiction. By Lemma 4 it follows that G had $\geq \epsilon dn + 1 - \frac{\epsilon}{2}dn = \frac{\epsilon}{2}dn + 1$ connected components of size $< \frac{2}{\epsilon d}$.

■

So we have shown that graphs that are ϵ-far from being connected have many small connected components (i.e. of size $< \frac{2}{\epsilon d}$.) This structural property that describes the local structure of graphs that are far from being connected should help us design a test for connectivity. The small connected components will be witnesses to the fact that the input is not connected.

We are now ready to describe our test for connectivity.

Algorithm 1 Property testing algorithm for connectivity

Input: $\epsilon > 0$, G on n nodes

Pick $s = \frac{4}{\epsilon d}$ vertices $v_1, v_2, \ldots, v_s$

For each v_i run a Breath First Search (BFS) until we reached $\frac{2}{\epsilon d}$ nodes, or we discovered a connected component of size $< \frac{2}{\epsilon d}$. In the latter case output **reject**

If no small connected component was found at the end of the previous step then output **accept.**

Proof [of Theorem 3]

Query analysis The number of queries that Algorithm 1 makes is the same as the number of edges it sees. There are at most $(\frac{2}{\epsilon d})^2$ edges queried in each BFS search, so the query

complexity of the test is $\frac{4}{\epsilon d} \cdot \frac{4}{(\epsilon d)^2} = O(1/(\epsilon d)^3)$ queries (so a number independent of the size of the graph G).

Let us now verify that Algorithm 1 is a one-sided error PT algorithm for $\mathcal{P}$.

Correctness: Suppose $G \in \mathcal{P}$. Then clearly G is always accepted.

Soundness: Suppose G is ϵ-far from $\mathcal{P}$. Then by Lemma 5 it must have at least $> \frac{\epsilon}{2}dn$ connected components of size $< \frac{2}{\epsilon d}$ (call such a component *small*). Since each such component has at least one vertex it follows that if we uniformly picked a random vertex $v \in V$

$$p = \Pr[v \in \text{ small component}] \geq \frac{\epsilon}{2n}dn = \frac{\epsilon}{2}d.$$

Therefore,

$$\begin{aligned}\Pr[\text{test accepts}] &= \Pr[\text{none of the} s \text{ vertices picked landed in a small component}] \\ &< (1-p)^{2/p} \leq e^{-2} < 1/3.\end{aligned}$$

Observe again that the 'basic test' for connectivity is to pick a uniformly random vertex, run BFS and see if it belongs to a small connected component in which case reject. As we have seen above this test catches a small connected component (witness) with probability p and therefore, in order to improve the test's chance of catching a witness we repeated the basic test $s = 2/p$ many times.

■

We remark that a tighter analysis leads to a test of query complexity $O(\frac{1}{\epsilon^2}\log^2(\frac{1}{\epsilon d}))$ for $\mathcal{P}$ (in the unbounded degree setting) and of query complexity $O(\frac{1}{\epsilon}\log^2(\frac{1}{\epsilon d}))$ for in the bounded degree setting where d is a bound on the degrees of vertices in G (See [1]).

References

[1] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.