0368.416701 Sublinear Time Algorithms December 15, 2014

Lecture 8

Lecturer: Ronitt Rubinfeld *Scribe: Nir Aviv , Gal Bahar, Dor Minzer*

1 Lecture Overview

1. Testing properties of dense graphs - Bipartiteness

2 Property testing in dense graphs

When we discussed property testing in sparse graphs, the graph representation was an adjacency list. For dense graphs, the property testing algorithms will use an adjacency matrix representation.

Definition 1 (Adjacency matrix) *Given a graph G=(V,E), The adjacency matrix of G is the matrix A such that*

$$A_{i,j} = \begin{cases} 1 & \textit{if } (v_i, v_j) \in E \\ 0 & \textit{otherwise} \end{cases}$$

A property testing algorithm must pass with high probability for a graph that has the property, and must fail with high probability for a graph that is ϵ-far from having the property.

Definition 2 (ϵ-far from having a property) *A graph G is ϵ-far from having property P if it takes more than ϵn^2 changes to entries in A_G in order to make G a member of P*

Properties are closed under nodes relabeling, meaning that relabeling nodes doesn't affect whether or not a graph has a certain property.

We can see that any graph is ϵ-close to being connected, since it takes a maximum of $n - 1$ changes to A_G in order to connect a graph, and we know that $n - 1 < \epsilon n^2$ regardless of ϵ.

3 Testing for bipartiteness

We will discuss testing of bipartiteness in desnse graphs.

3.1 Property definition

Definition 3 (Bipartiteness) *A graph G=(V,E) is bipartite if (Both these definitions are equivalent)*

1. *It's nodes can be colored red or blue so that no edge is monochromatic (has both it's nodes colored the same color)*
2. *V can be partitioned into (V_1, V_2) s.t. $\nexists e = (u, v) \in E$ and both $u, v \in V_1$ or both $u, v \in V_2$*

Definition 4 (Violating edge) *For a graph G=(V,E) and some partitioning (V_1, V_2) of V, a violating edge is an edge e=(u,v) such that $u \in V_1$ and $v \in V_2$*

If a given graph G is ϵ-far from bipartite:

1. it takes more than ϵn^2 changes to make it bipartite
2. for every partitioning (V_1, V_2) of V, there are more than ϵn^2 violating edges.

Remark

1. For $\epsilon = 1/n$, testing for bipartiteness in known to be in NP-HARD
2. For sparse graphs, testing bipartiteness to ϵdn is known to have a lower bound on query complexity of $\Omega(\sqrt{n})$
3. For dense graphs, we will show a bipartite tester with query complexity that depends only on ϵ

3.2 Proposed algorithm

First we will present a correct algorithm, which we will not prove:

Algorithm 1: Proposed algorithm

```
Pick a sample of nodes of size Θ(1/ε² log(1/ε))
Consider the induced graph of only the samples
if the induced graph is bipartite then
    Pass
else
    Fail
```

This algorithm is a correct tester for bipartiteness in dense graphs. It has query complexity of $\Theta(\frac{1}{\epsilon^2} log(\frac{1}{\epsilon}))$ and it's runtime is $poly(\frac{1}{\epsilon})$, since testing bipartiteness can be done with a simple BFS.

3.3 Another attempt - Algorithm 0

In order to check if a graph G is ϵ-far from bipartite we can use the following algorithm:

Algorithm 2: Tester for graphs that are ϵ-far from bipartite

```
Pick some partitioning V1, V2 of V
Take a sample of edges of size m = Θ(1/ε log(1/δ))
if found a violating edge according to the partitioning then
    Fail
else
    Pass
```

Since G has more than ϵn^2 violating edges, we will hit one with probability $\geq 1 - (1 - \epsilon)^m \geq 1 - \delta$. This algorithm works only for graphs that are ϵ-far from bipartite. Graphs which are bipartite might have only one partitioning for which they don't have any violating edges. We will change the above algorithm so that it would check every partitioning of V:

Algorithm 3: Another attempt at a bipartite tester

```
Take a sample of edges of size m = Θ(1/ε log(1/δ))
for every partition V1, V2 of V do
    count the number of violating edges in the sample according to the partition (Let's call this number
    Violating_{V1,V2})
if for all partitioning, Violating_{V1,V2} > 0 then
    Fail
else
    Pass
```

This algorithm will pass every bipartite graph, since there is some partitioning for which there are no violating edges. If the algorithm gets a graph G that is ϵ-far from bipartite:

- For each partitioning V_1, V_2 of V, $Pr[Violating_{V_1,V_2} > 0] \geq (1 - \delta)$ (We saw this in the previous algorithm)

- The probability the algorithm output fails is $Pr[\forall V_1, V_2(Violating_{V_1,V_2} > 0)] \geq (1 - 2^n\delta)$ (using union bound)

We need this probability to be high so we must pick $\delta < \frac{1}{2^n}$. This algorithm's sample complexity is $m = \Theta(\frac{1}{\epsilon}n)$ and it's runtime is $O(2^n)$.

An Improved Algorithm

We had two main problems with the previous algorithm. The first one is the running time. We had to go over every one of the 2^n partitions of V which is not feasible. The second one was the query complexity. Although $\Theta(n/\epsilon)$ is sub-linear in the input size, one can do better.

Definition 5 (partition oracle) *Let $U \subseteq V(G)$, and let (U_1, U_2) be a partition of U. We define the induced partition by (U_1, U_2) on V and denote by (W_1^{U1U2}, W_2^{U1U2}) the following partition:*

$$W_2^{U1U2} = \{v \in V \mid \exists x \in U_1 \text{ such that } (x, v) \in E\}$$
$$W_1^{U1U2} = V \setminus (V_1 \cup V_2 \cup W_2^{U1U2})$$

Algorithm 4: Algorithm1

```
Pick U of size Θ(1/ϵ log(1/ϵ)) and U′ of size Θ(1/ϵ² log(1/ϵ)) nodes.
Denote U′ = {u1, v1, u2, v2, ...}. Define a pair off of those vertices by P = {(u1, v1), (u2, v2), ...}.
Query the adjacency matrix A on every pair (u, v) ∈ P.
foreach (U1, U2) partition of U do
    Let W = (W1^{U1U2}, W2^{U1U2}) be the induced partition.
    Count m(U1, U2) the number of edges in P that violate W.
    if m(U1,U2)/|P| ≥ 3/4 ϵ then
        Pass
Fail
```

Analysis

If G Is Far From Bipartite

Suppose G is ϵ-far from bipartite. Since U is of size $\Theta(\frac{1}{\epsilon}\log\left(\frac{1}{\epsilon}\right))$, there are $\frac{1}{\delta} = 2^{\frac{1}{\epsilon}\log\left(\frac{1}{\epsilon}\right)}$ partitions of U. Fix some partitioning U_1,U_2, and look at the partitioning W they induce on V. Since G is ϵ-far from bipartite, there are at least ϵn^2 edges which violating W. Let X be a random variable that is the number of edges in P which violate W. We can think of U' as being chosen after W was fixed. Therefore by linearity of expectation over the edges in P

$$E(X) \geq |P| \frac{\epsilon n^2}{n^2} = |P| \epsilon$$

And since elements in P are independent, using chernoff bounds we conclude that

$$Pr(\frac{X}{|P|} < \frac{3}{4}\epsilon) \leq \frac{1}{8}\delta$$

So by the union bound the probability that one of the partitions of U causes the tester to pass can be upper bounded:

$$Pr(\exists(U_1, U_2) \text{ such that } \frac{X(U_1, U_2)}{|P|} < \frac{3}{4}\epsilon) \leq \sum_{(U_1,U_2)} Pr(\frac{X(U_1, U_2)}{|P|} < \frac{3}{4}\epsilon) \leq \frac{1}{8}\delta \cdot \frac{1}{\delta} = \frac{1}{8}$$

So the probability we will fail given an ϵ-far from bipartite graph is at least $\frac{7}{8}$.

If G Is Bipartite

In this case we know there exists some partition $V = (Y_1, Y_2)$ of the vertex set such that there are no violating edges. The problem is for a certain U, even if we look at the induced partition of B on U, our partition oracle does not promise us we will get B back once we use it(mainly because of the case that if a vertex v does not have any neighbours in both U_1 and U_2, we put it arbitrarily in W_1^{U1U2}).

However, we will show this induced partition is not that far from B. For U, define

$$U_1 = Y_1 \cap U$$

$$U_2 = Y_2 \cap U$$

And look at the partitioning $W = (W_1^{U1U2}, W_2^{U1U2})$ it induces on V. Notice that only vertices which do not have a neighbour in U can contribute violating edges (for example, if $v_1 \in Y_1$ then it will be in W_1^{U1U2} for sure). We divide those vertices without a neighbour in U into two groups

$$A = \left\{ v \mid deg(v) < \frac{\epsilon}{4} n \right\}$$

$$B = \left\{ v \mid deg(v) \geq \frac{\epsilon}{4} n \right\}$$

Every edge that violates W is either violating for (Y_1, Y_2) or touches a vertex in A or in B. So

$$|\{W \text{ violating edges}\}| \leq |\{(Y_1, Y_2) \text{ violating edges}\}| + |\{\text{edges touching } A\}| + |\{\text{edges touching } B\}|$$

The first summand is clearly 0. The second and the third summands are bounded by $\frac{\epsilon}{4} n\, |A|$, $n\, |B|$ respectively. So we conclude that

$$|\{W \text{ violating edges}\}| \leq \frac{\epsilon}{4} n\, |A| + n\, |B|$$

We will show that with high probability over the choice of U it holds that $|B| \leq \frac{1}{4}\epsilon n$ which will result that with high probability

$$|\{W \text{ violating edges}\}| \leq \frac{\epsilon}{4} n\, |A| + n\, |B| \leq \frac{\epsilon}{4} n^2 + n\frac{1}{4}\epsilon n = \frac{\epsilon}{2} n^2$$

For the next lemma, let us call a vertex with degree at least $\frac{\epsilon}{4} n$ a high degree vertex.

Lemma 6 *The probability there are at most $\frac{\epsilon}{4} n$ high degree nodes in V with no neighbour in U is at least $\frac{7}{8}$(Over the choice of U). That is*

$$Pr(|B| \leq \frac{\epsilon}{4} n) \geq \frac{7}{8}$$

Proof Define indicator random variables σ_v for each $v \in V$ such that $\sigma_v = 1$ if and only if v is a high degree graph and has no neighbours in v.

Let p be the probability the ith node in U is not a neighbour of v(note it does not depend on i). Because v is of high degree, $p \leq 1 - \frac{\epsilon}{4}$. The vertices in U are independent so

$$E(\sigma_v) = Pr(\sigma_v = 1) = p^{|U|} \leq (1 - \frac{\epsilon}{4})^{|U|} \leq (1 - \frac{\epsilon}{4})^{\frac{4}{\epsilon} \log \frac{\epsilon}{32}} \leq \frac{\epsilon}{32}$$

Note that $|B| = \sum_{v \in V} \sigma_v$, so by linearity of expectation and the previous computation

$$E(|B|) \leq \frac{\epsilon}{32} n$$

And the end is due to Markov inequality

$$Pr(|B| \geq \frac{\epsilon}{4}n) \leq \frac{E(|B|)}{(n\epsilon/4)} \leq \frac{1}{8}$$

■

Remark One way to prove the correctness of the "dream" algorithm (the proposed algorithm) is to argue that it follows from the correctness of the algorithm we have just analyzed.