ECS 189A Sublinear Algorithms for Big Data **Fall 2024**

Lecture 6: Bounded degree graphs, biclique testing in dense graphs

Lecturer: Jasper Lee *Scribe: Josh Petrack*

This will be the last class on property testing in graphs. Next class will start on the PCP theorem, and will be complexity theoretic in flavor. Today we'll do mostly more bounded degree sparse graph property testing, with a little bit on dense graph property testing.

Most results will largely follow the same formula as those we've shown so far: we will introduce some property we want to test, we will describe some structural feature that can be used to differentiate graphs that are ϵ-far from the property from graphs in our property—which we show via the "contrapositive" that "any graph without the structural property must be ϵ-close to the property"—and we will give an algorithm that tries to find places in the graph with such structural feature (so as to reject the graph as being far from the property). The algorithm will reject if and only if it finds such a place. Notably, such algorithms will always have one-sided error: they will always accept members of the property, because members of the property won't have any features for us to find and reject.

Before showing results following the above recipe, however, we will first look at a problem where the most efficient algorithms require two-sided error, where sometimes the algorithm can reject a graph with the given property.

1 Detecting cycle-free graphs

Given a graph, in the bounded-degree sparse model (where vertices have degree at most d), we want to test if it is cycle-free. Note that another name for a cycle-free graph is a forest (a collection of trees).

Fact 6.1. *A graph G with n vertices, m edges and k connected components is cycle-free if and only if $m = n - k$.*

This can be seen because starting with a graph with no edges ($n = k, m = 0$), adding any edge that does not create a cycle will increase m by one while decreasing k by one. If G has one or more cycles, then we should have $m > n - k$, because this would require adding an edge that does not bridge two connected components. The amount by which m exceeds $n - k$ gives a measurement of how far the graph is from cycle-free.

We want to turn this into a property testing algorithm, with parameters denoted by "??" which we will figure out:

Algorithm 6.2 Cycle-freeness tester (incomplete)

1. Get an estimate $\hat{k}$ for the number of connected components with some additive error ?? using some number ?? of queries.
2. Get an estimate $\hat{m}$ for the number of edges with some additive error ?? using some number ?? of queries.
3. Check that $\hat{m} \leq n - \hat{k} +$?? for some error term ??.

To use the property testing formulation we need some structural result. What does it mean to be ϵ-far from cycle-free?

Proposition 6.3 (Cycle-freeness structural property)**.** *Given a graph G with n vertices, k connected components and m edges, G is ϵ-far from cycle-free if and only if $m > n - k + \frac{\epsilon dn}{2}$.*

Note that we are using the same bounded degree graph model: the distance between two graphs is the Hamming distance between their adjacency lists, represented as a $d \times n$ matrix that may have null entries.

Proof. As with most structural properties, we show the contrapositive: that if $m \leq n - k + \frac{\epsilon dn}{2}$, then we can make ϵdn changes to the adjacency list to make the graph cycle free. All we need to do is remove $\frac{\epsilon dn}{2}$ edges, and to remove an edge, we need to remove two entries from the adjacency list (to remove the edge uv, we have to remove v from u's list and u from v's list). □

Knowing this, we know that the additive error values in algorithm 6.2 must be $O(\epsilon dn)$ with some small constant, and that will be good enough. We already know how to estimate the number of connected components from last lecture; to estimate the number of edges, we can effectively sample random elements of the $d \times n$ adjacency list-matrix and treat these samples as Bernoulli coins, where we just need to estimate the bias.

We can now combine what we know to a full description of Algorithm 6.2:

Algorithm 6.4 Cycle-freeness tester

1. Get an estimate $\hat{k}$ for the number of connected component swith additive error $\frac{\epsilon dn}{100}$ using $O\left(\frac{1}{d^2\epsilon^3}\right)$ queries.
2. Get an estimate $\hat{m}$ for the number of edges with additive error $\frac{\epsilon dn}{100}$ using $O\left(\frac{1}{\epsilon^2}\right)$ queries.
3. Check that $\hat{m} \leq n - \hat{k} + \frac{\epsilon dn}{50}$.

At a high level, we took a structural property of cycle-free graphs (Fact 6.1 and Proposition 6.3) and then leveraged that structural property - in this case, a gap in edge count - to make an algorithm.

This is notably the first property testing algorithm algorithm we've seen where we might have *2-sided error*: because we're doing estimation for connected component count and edge count that may have some probability of error, we might reject cycle-free graphs. Note that if we want 1-sided error, there is a known $\Omega(\sqrt{n})$ lower bound for query complexity.

2 Generalizing cycle-free graph detection

We will now test more general subgraph-freeness, in the same bounded-degree sparse model.

Definition 6.5 (Subgraph-freeness)**.** let H be a fixed connected graph over $\ell \leq n$ vertices. Then a graph G on n vertices is H-free if no subgraph of G is isomorphic to H. Note that this subgraph H may be referred to as a graph motif in some literature.

We need some way to tell how complex the subgraph we're trying to avoid is, so that we can measure the difficulty of the problem.

Definition 6.6 (Radius and center of H)**.** The radius $\mathrm{rd}(H)$ is defined as the minimal r such that $\exists v \in H, \forall u \in H, d(u,v) \leq r$. In other words, pick a *center* v, and the radius relative to that center is the maximal distance from it to another vertex. The radius of H is the minimal relative radius among all possible choices of center.

Algorithm 6.7 Subgraph-freeness testing

Repeat $O(\frac{1}{\epsilon})$ times:

1. Pick a random vertex $u \in G$
2. Run BFS from u for distance equal to $\mathrm{rd}(H)$
3. Check if the resulting subgraph is H-free.

Note that time complexity may be an issue here, because of the final step: the subgraph isomorphism problem is NP-complete. But the size of the subgraph H is considered a constant, and we primarily care about scaling with the size n of the main graph.

Proposition 6.8. *Algorithm 6.7 accepts all H-free graphs. (This is obvious because we will never hit the rejection condition).*

The algorithm repeats a process $O(\frac{1}{\epsilon})$ times, and must reject graphs that are ϵ-far with probability $\frac{2}{3}$. So, we need each individual iteration of the algorithm to succeed with probability $\Omega(\epsilon)$. If it does, then we can then pick the big-O constant large enough to get a $\frac{2}{3}$ overall success probability. We now show that each iteration has this success probability.

Definition 6.9 (Detecting vertex)**.** A vertex u is a *detecting vertex* if it is *a* center of a copy of H in G. In other words, if running BFS from u for distance $\mathrm{rd}(H)$ will yield a graph containing a copy of H.

Proposition 6.10 (Subgraph-freeness structural property)**.** *If G is ϵ-far from H-free, then there are at least $\frac{\epsilon n}{2}$ detecting vertices.*

Proof. This is a structural property, so we prove it by contrapositive. If G has fewer than $\frac{\epsilon n}{2}$ detecting vertices, then we can remove edges around these vertices to isolate them. We may need to remove $\frac{\epsilon dn}{2}$ total edges incident to the detecting vertices, because each detecting vertex may have up to d incident edges to remove. The resulting graph must be H-free. □

Because at least an $O(\epsilon)$ fraction of the vertices are detecting vertices, each individual trial in Algorithm 6.7 has the requisite probability of success, proving the algorithm's correctness.

3 Bipartiteness testing

Reminder: a graph is bipartite if we can partition the vertices into subsets U_1 and $U_2 = V \setminus U_1$, such that no edges go between vertices within one of the two subsets.

Fact 6.11 (Bipartiteness structural property)**.** *A graph G is bipartite if and only if G has no odd-length cycles.*

We won't use BFS to test for this property: instead, we will use random walks.

Definition 6.12 (Random walk)**.** A *random walk* of length ℓ starting at a vertex u is a path $u \to v_1 \to v_2 \to \ldots \to v_\ell$ such that $v_{i+1} \leftarrow \text{unif}(\mathcal{N}(v_i))$. In other words, draw each step uniformly from the neighbors of the previous step.

Why random walks? In an ϵ-far graph, there should be lots of odd-length cycles (we don't prove this, but one can imagine proving it by contraposition: if there are few odd-length cycles, then we can modify the graph by removing an edge from each of those cycles). But these cycles may be long. If we use BFS, we might need to look $O(n)$ distance from the starting vertex to even discover an odd-length cycle, because for all we know, most or all of the odd-length cycles we're looking for might be that large. Random walks allow us to explore farther away from our starting vertex without worrying about finding exponentially many vertices to account for.

Repeat $O(\frac{1}{\epsilon})$ times:

1. Uniformly pick a vertex u.
2. (a) Perform $m = \sqrt{n}\,\text{poly}(\frac{\log n}{\epsilon})$ many random walks of length $\ell = \text{poly}(\frac{\log n}{\epsilon})$.
   (b) Let R_0 be the set of vertices reached from u in an even number of steps, and R_1 be the set of vertices reached from u in an odd number of steps
   (c) Check if $R_0 \cap R_1 \neq \emptyset$. If so, we have an odd-length cycle, so return false.

It turns out that the time complexity of this algorithm is $O(\sqrt{n}\,\text{poly}(\frac{\log n}{\epsilon})\log d$. The $\log d$ is for binary search to find a vertex's degree, so that the algorithm can do random sampling of that vertex's neighbors.

Proposition 6.13. *Like most of the algorithms we've seen, Section 3 is 1-sided: bipartite graphs will never be rejected.*

Theorem 6.14 (Soundness)**.** *Section 3 rejects all graphs that are ϵ-far from being bipartite with probability $\geq \frac{2}{3}$.*

This theorem is hard to prove. The result can be found at http://www.eng.tau.ac.il/~danar/Public-pdf/bip.pdf.

4 Dense graph property testing

We now turn to a new model. Despite the name, the graph we're representing doesn't actually need to be dense; this just means that we're using the graph representation that is most sensible for representing dense graphs. Specifically, we will use a symmetric matrix, where the (i, j) entry represents if there is an edge between vertices i and j. One query is of the form $Query(u, v) = 1_{\{(u,v)\in E\}}$. This is different from the previous model, where the result of a query might be null if we query in a vertex's edge list beyond that vertex's degree.

For distance, we use the obvious: $d(G_1, G_2) = \frac{1}{n^2}$ times the number of entries where these two matrices differ. Here, two graphs are ϵ-far if they differ by $\frac{\epsilon n^2}{2}$ edges, compared to $\frac{\epsilon dn}{2}$ edges in the bounded degree model. This simple fact has dramatic consequences on

what problems are easy, hard, or even worth thinking about. In particular, in the dense model, the promise of ϵ-farness is stronger: there will be even more edges that the two graphs must differ by.

Definition 6.15. A *Hamiltonian cycle* of a graph is a cycle that includes every vertex exactly once.

Property testing question: test whether a graph G contains a Hamiltonian cycle versus being ϵ-far from having one, under the dense representation. We assume that ϵ is a tiny constant here, something like 10^{-5}, that does not decrease as n increases.

Algorithm 6.16 Hamiltonian cycle detection

Accept.

Why does this work? Because ϵ-far means you have to add $O(\epsilon n^2)$ edges, and you only need to add $O(n)$ edges to get a Hamiltonian cycle, we get:

Proposition 6.17. *There is no graph that is ϵ-far from having a Hamiltonian cycle as long as $\epsilon = \Omega(\frac{1}{n})$.*

So under this assumption that ϵ is held constant, we can always accept because there are no legal inputs where we should reject.

5 Biclique testing in the dense graph model

Definition 6.18. A graph $G = (V, E)$ is a *biclique* if there is a partition of the vertices $V = V_1 \sqcup V_2$ such that $E \cong V_1 \times V_2$. In other words, you can partition the vertices into two subsets, and the set of edges in the graph is exactly all possible edges from one set to the other. Essentially, a biclique is a maximally-connected bipartite graph.

Proposition 6.19 (Biclique structural property)**.** *Suppose G is ϵ-far from being a biclique. Then for every possible partition $V = V_1 \sqcup V_2$, there exist some $Omega(\epsilon n^2)$ vertex pairs that "violate" bicliqueness for that partition: they either have an edge but lie in the same part, or lack an edge despite being in opposite parts.*

In practice, dense graph testing algorithms tend to be simple. In this case:

Algorithm 6.20 Biclique tester (Dense graph)

Repeat $O(\frac{1}{\epsilon})$ many times:

1. Pick a random vertex u
2. Pick a random pair of vertices v, w
3. Check whether (u, v, w) is a biclique: either they have no edges, or they are a line graph.

Proposition 6.21. *Algorithm 6.20 accepts all bicliques. (The algorithm is 1-sided).*

Proposition 6.22. *A single iteration of Algorithm 6.20 rejects graphs that are ϵ-far from being a biclique with probability $\Omega(\epsilon)$.*

Proof. Choosing the vertex u induces a bipartition, $(\mathcal{N}(u), V \setminus \mathcal{N}(u))$ (the neighbors of u versus everything else). Once we fix u and fix (v, w), we know based on u's connectivity to v and w whether or not the edge (v, w) should exist, because 3-vertex bicliques always have an even number of edges. Because we know from Proposition 6.19 that there are many edges that "violate" any possible partition, there must be many choices of neighbors of u that witness this violation, giving us the $\Omega(\epsilon)$ bound on successfully finding a violation. $\square$