# Sublinear Algorithms

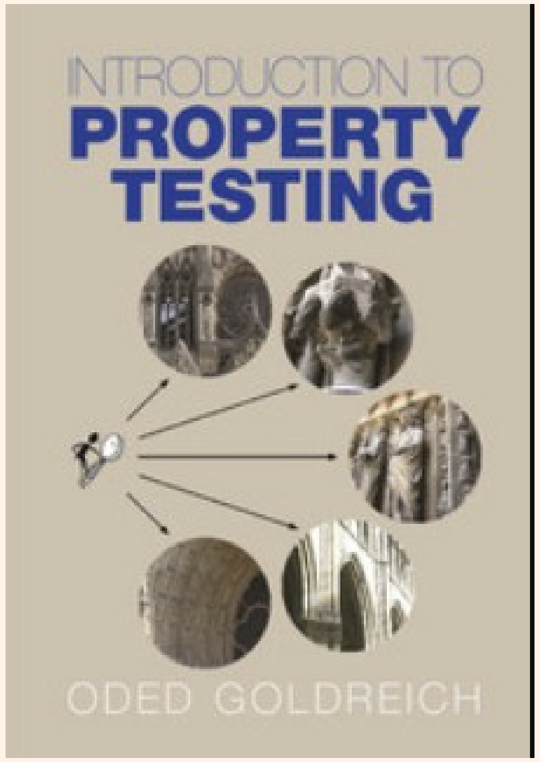
## Lecture 08: Property Testing II

Previous lecture

*Linearity and Monotonicity Testing*

This lecture

Testing graph properties



# Property Testing on Graphs

**Goal**: Infer a property of a graph without reading all of the *input*

(Many) Models of Computation

-**Dense** Graph Model

Oracle acess of the following form: are *u,v* are connected?

-**Sparse** Graph Model

Upper bound on the maximum degree and oracle acess of the following form: For *u,i* which is the *i*-th neighbor of u?

Directed and undirected graphs

# What *far* means?

Decide whether a graph has some property Π
Or
Asone needs to remove/add at least $\varepsilon m$ edges to make it have Π

Decide whether a graph has some property Π
Or
As one needs to remove/add at least $\varepsilon n^2$ edges to make it have Π

Structural and algorithmic part

A graph which is far from having Π should look somehow special...

Simple algorithms, all the complexity is pushed to the analysis

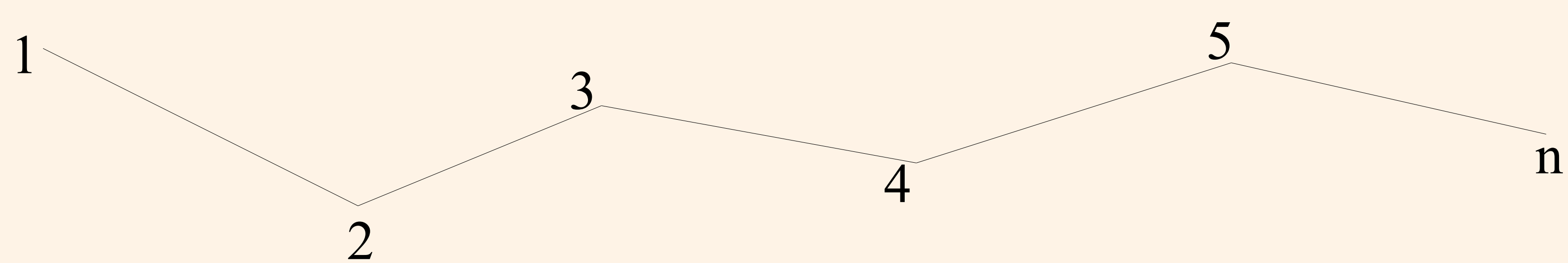
# Testing Connectivity

Given aν **undirected** graph G in the sparse graph model,
Decide whether it is connected or
one needs to add at least **εm** edges to make it connected

**Oracle access** to a graph in the following way:
For vertices *u,v*, are *u* and *v* connected?

A first approach:
Pick a number of edges, and test
whether the induced graph is connected

Fails miserably on the path

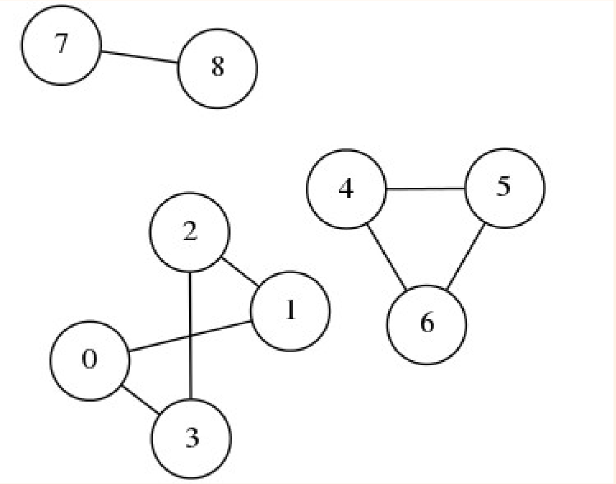


# Testing Connectivity

Algorithm: Pick $m/(\varepsilon n)$ vertices at random,
and perform *breadth-first search* from each vertex
halting after $2n/(\varepsilon m)$ found vertices.

If detected disconnectivity, report FAR

This algorithm accepts *connected* graphs.
What about far from connected graphs?




Structural Claim I: Any graph which is $\varepsilon$-far from being connected
has at least $\varepsilon m+1$ connected components.

Structural Claim II: Any graph which is $\varepsilon$-far from being connected
has at least $\varepsilon m/2$ components with at most $2n/(\varepsilon m)$ vertices.

Proof: *Average size* of a component is *at most* $n/(\varepsilon m + 1)$.
*There exist at most* $\varepsilon m/2$ *components of size*
*at least* $2n/\varepsilon m$ , otherwise
$(\varepsilon m/2 + 1) * (2n/\varepsilon m) > n$

# Putting everything together

Probability of falling in a *small* connected component: $\Omega(\varepsilon m / n)$

If that happens, we are happy.

After $O(n/\varepsilon m)$ steps a vertex in a small connected component is sampled

*(geometric distribution)*

If a coin is heads with probability $\delta$,
Then after $\Theta(\delta)$ flips we get a heads
With constant probability.

Running Time

$O(n/\varepsilon m) * O((n/\varepsilon m)^2) = O((n/\varepsilon m)^3) = O(1/\varepsilon^3)$

# Testing Acyclicity on Directed Graphs

Given oracle access to a **directed** graph G,
determine whether G is **acyclic** or
One needs to remove at least $\varepsilon n^2$ **edges** to make it acyclic

Oracle acess: For vertices *u,v*, is there an edge from *u* to *v*?

Algorithm:
Pick **s random** vertices
Test whether there is a cycle in the **induced** graph.

**Structural Claim I**:If G **ε-far** from being acyclic, then there exists a **subset W of the vertices,**

Such that $|W| \geq \sqrt{\frac{\epsilon}{2}} n$ and the induced graph has **minimum degree** $\frac{\epsilon}{2} n$

**Structural Claim II**: For every subset W of vertices with minimum degree at least $\zeta|W|$,
select $O\log(1/\zeta)/\zeta$ vertices in **W** uniformly at random:
then with probability 9/10 the induced graph has a **cycle** .

# Testing Acyclicity on Directed Graphs

Algorithm:
Pick s random vertices
Test whether there is a cycle in the induced graph.

Structural Claim I: If G $\varepsilon$-far from being acyclic, then there exists a subset W of the vertices,

Such that $|W| \geq \sqrt{\frac{\epsilon}{2}}n$ and the induced graph has minimum degree $\frac{\epsilon}{2}n$

Structural Claim II: For every subset W of vertices, with minimum degree at least $\zeta|W|$
Selecting $O(\log(1/\zeta)/\zeta)$ vertices in W uniformly at random suffices
for the induced graph to have a cycle.

Proof of correctness: $\alpha = |W|/n$. Expected number of sampled vertices in W is $\alpha * s$.

Concentrated around its expected value if $\alpha * s = \Omega(1)$ [be careful when applying Chernoff bound]

We need $\alpha * s/2 = \Omega(\log(1/\zeta)/\zeta)$
(the sampled vertices in W are uniform in it),

for $\zeta = \varepsilon/(2\alpha)$ to use the Claim II

So $s = \text{poly}(1/\varepsilon)$ suffices.

# Testing Acyclicity on Directed Graphs

Structural Claim II: For every subset W of vertices, with minimum degree at least $\zeta|W|$
Selecting $O(\log(1/\zeta)/\zeta)$ vertices in W uniformly at random suffices
for the induced graph to have a cycle.

Proof: Let $r = \Theta(\log(1/\zeta)/\zeta)$ and $v_1, v_2, .., v_r$ the sampled vertices.

If for every $v_i$ there exists a $v_j$ such that edge $(v_i, v_j)$ exists, then no $v_i$ is a sink,
thus cycle exists.

$$\Pr\{v_i \text{ has outdegree } 0 \text{ in } W\} \approx (1-\zeta)^r$$

$$(1-\zeta)^{\Theta(\zeta \log(1/\zeta))} = \frac{1}{\text{poly}(1/\zeta)}$$

Can afford a union-bound over all sampled vertices.

# Sketch of proof of Claim 1

**Structural Claim I**: If G **ε-far** from being acyclic, then there exists a **subset W of the vertices,**

Such that $|W| \geq \sqrt{\frac{\epsilon}{2}}n$ and the induced graph has **minimum degree** $\frac{\epsilon}{2}n$

Assume this is not the case. We will show that G is **ε-close** to acyclic.

We will define an **ordering** of the vertices (left to right).
Initially all **undefined**.
As long as at least *threshold* vertices undefined, pick the one with **lowest degree** and add it to the ordering.
Otherwise, order vertices arbitrarily.

Deleting all edges pointing to the right results in an acyclic graph.

# Recap

Testing connectivity and acyclicity

Testing on graphs requires
A structural characterization of far graphs

Different models, lots of avenue to explore

In principle, directed case harder than undirected case

Next 3 lectures:
Applications of sublinear algorithms to traditional algorithms

*Thank you*