# Streaming Algorithms

Meng-Tsung Tsai

05/29/2018

## Reminder

| Tuesday | Friday |
|---|---|
| | **Today** Appl. of $L_0$-samplers. |
| **Jun 5** No class. OH (EC 336) | **Jun 8** Proposal. **10:10 - 12:40** |
| **Jun 12** Supporting Materials. | **Jun 15** Supporting Materials. |
| **Jun 19** No class. OH (EC336) | **Jun 22** Presentation. **10:10-12:40** |

## References

- "Analyzing graph structure via linear measurements," Ahn, Guha, and McGregor.
- "Tight Approximations of Degeneracy in Large Graphs," Farach-Colton and Tsai.
- "Densest Subgraph in Dynamic Graph Streams," McGregor et al.

## Spanning Trees

Input: a sequence of edge insertions and deletions.

Output: a spanning tree of final graph G.

Goal: use O(n polylog n) space.

In hw2, we know how to reconstruct G. Is it hard to grab a spanning tree from G?

# Spanning Trees

Input: a sequence of edge insertions and deletions.

Output: a spanning tree of final graph G.

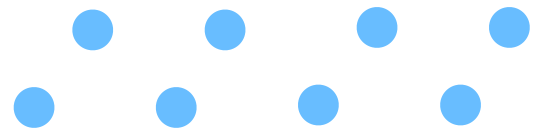
Goal: use O(n polylog n) space.

Is it possible to sample a set of edges from G so that the set of edges has a spanning tree of G?

# Spanning Trees

Idea: construct a spanning tree as Prim's algorithm on unweighted graphs.

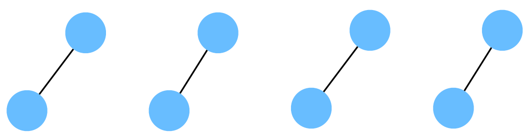
**phase 0**



# Spanning Trees

Idea: construct a spanning tree as Prim's algorithm on unweighted graphs.

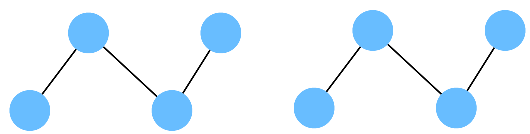
**phase 1**



# Spanning Trees

Idea: construct a spanning tree as Prim's algorithm on unweighted graphs.

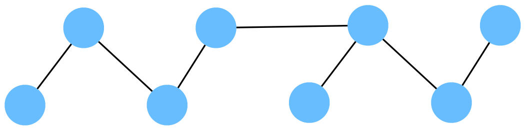
**phase 2**



## Spanning Trees

Idea: construct a spanning tree as Prim's algorithm on unweighted graphs.

**phase 3**

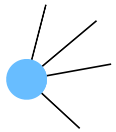


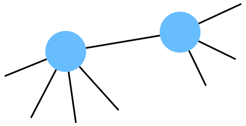
## Spanning Trees

Tool: We need a building block that can return an edge from the set of edges has exactly one end incident to each connected component.

**By $L_0$-sampler, one can easily sample an edge adjacent to each node.**

**But we don't know which nodes will form a connected component after phase 1. Can we sample an edge adjacent to each component?**





## Spanning Trees

Tool: We need a building block that can return an edge from the set of edges has exactly one end incident to each connected component.

For each node x, define a vector $E_x$ of length C(n, 2). The (i, j)-th entry is setted as 1 if i = x, -1 if j = x, or otherwise 0.

If an $L_0$-sampler allows some coordinates to have negative values and it samples a coordinate with a non-zeor value. Then, phase 1 works well. (Note that our $L_0$-sampler does not satisfy the requirement.)

## Spanning Trees

Tool: We need a building block that can return an edge from the set of edges has exactly one end incident to each connected component.

For each node x, define a vector $E_x$ of length C(n, 2). The (i, j)-th entry is setted as 1 if i = x, -1 if j = x, or otherwise 0.

In phase 2, if two nodes x, y are connected, then we add $E_x$ and $E_y$. Observe that $E_x+E_y$ will cancel out the interconnected edges and leave those edges that has exactly one end incident to the component {x, y}. (Note that this requires $L_0$-sampler to be additive, and our $L_0$-sampler satisfy the requirement.)

## Spanning Trees

Tool: We need a building block that can return an edge from the set of edges has exactly one end incident to each connected component.

For each node x, define a vector $E_x$ of length C(n, 2). The (i, j)-th entry is setted as 1 if i = x, -1 if j = x, or otherwise 0.

In the following phases, for each component S, we add the vectors associated with all nodes x in S. Then, we can obtain an edge to connect another component. There are log n phases.

We need **n** $L_0$-samplers for each phase. Each $L_0$-sampler need O(poly log n) space. Consequently, spanning trees can be sampled using O(n poly log n) bits, as desired.

## Spanning Trees

Consequence: the streaming algorithms for k-EC, k-VC, ... in the insertion model can implemented in the dynamic model as well.

## Graph Degeneracy

Input: a sequence of edge insertions and deletions.

Output: a node-ordering of G so that every node has ≤ k latter neighbors in the ordering and k is minimized.

Goal: use O(n polylog n) space.

The RAM algorithm is
to iteratively remove the min-degree node.

## Graph Degeneracy

Idea: Given G, we construct a subgraph H by sampling each edge in G with probability p. One can relate the degree of node x in G to that in H, i.e.

$$E[\deg_H(x)] = p*\deg_G(x).$$

Furthermore, $\deg_H(x)$ is highly concentrated to the expectation.

Guess that a min-degree node in H is likely to be a roughly min-degree node in G.

Theorem. The degenerate ordering in H yields an $(1+\varepsilon)$-approximation of that in G.

# Densest Subgraph

Input: a sequence of edge insertions and deletions.

Output: a subgraph induced by a node set U in G so that $|E_G|/|U|$ is the largest.

Goal: use O(n polylog n) space.

# Densest Subgraph

Idea: Given G, we construct a subgraph H by sampling each edge in G with probability p. One can relate the density of a subgraph induced by node set U in G to that in H, i.e.

$$E[\text{density}_H(H_U)] = p*\text{density}_G(G_U).$$

Furthermore, $\text{density}_H(H_U)$ is highly concentrated to the expectation.

There are some missing technical details. See Ref. 3.

Theorem. The max desnity subgraph in H yields an $(1+\varepsilon)$-approximation of that in G.