Analyzing Graph Connectivity *via* Random Linear Projections

Kook Jin Ahn
Sudipto Guha
Andrew McGregor

Classic Sketches

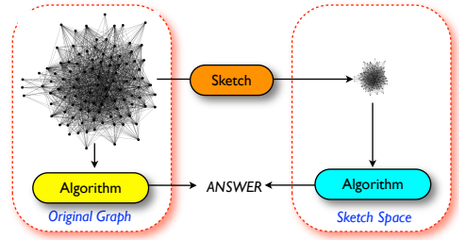
Random linear projections such that projection yields estimate of properties of original vector.

$$\begin{bmatrix} & M & \end{bmatrix} \begin{bmatrix} \\ v \\ \\ \end{bmatrix} = \begin{bmatrix} Mv \end{bmatrix} \longrightarrow \text{answer}$$

e.g., norms, heavy hitters, support size, quantiles.

Can linear sketches capture combinatorial graph properties?

Exploit Homomorphic Properties



Dynamic Graph Stream Problem

Input: Stream of edges are inserted and deleted in a graph on n nodes.

Goal: Determine connectivity properties using O(n polylog n) memory.

Mark and Erica are now friends.
Like · Add Friend

Mark and Erica are no longer friends.
Like · Add Friend

Eduardo and Mark are now friends.
Like · Add Friend

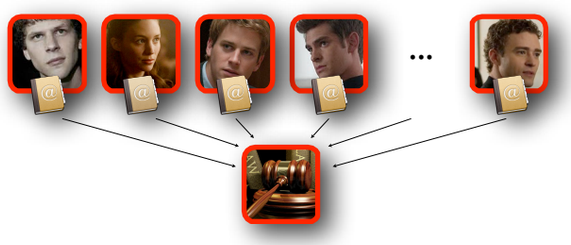
⋮

Lawyers are now friends with everyone.
Like · Add Friend

Communication Problem

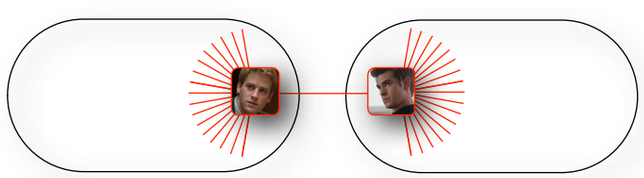
Input: Each player knows neighborhood Γ(v) of a node v in graph G.

Goal: Simultaneously, each player sends O(polylog n) bits to a central player who estimates connectivity properties.



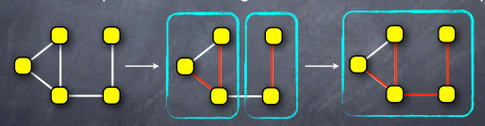
Why's It Hard?

Consider a bridge (u,v). Messages need to convey existence of (u,v). But players with u and v can't distinguish (u,v) from other Ω(n) friendships. It *appears* that one of the players need to send Ω(n) bits.



Ingredient 1: Basic Connectivity Algorithm

- Basic Algorithm (Spanning Forest):
  1. For each node: pick incident edge
  2. For each connected comp: pick incident edge
  3. Repeat until no edges between connected comp.

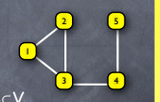


- Lemma: Takes O(log n) steps and selected edges include spanning forest.

Ingredient 2: Sketching Neighborhoods

- For node i, let a_i be vector indexed by node pairs. Non-zero entries: $a_i[i,j]=1$ if $j>i$ and $a_i[i,j]=-1$ if $j<i$.

$$a_1 = \begin{pmatrix} \{1,2\} & \{1,3\} & \{1,4\} & \{1,5\} & \{2,3\} & \{2,4\} & \{2,5\} & \{3,4\} & \{3,5\} & \{4,5\} \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$
$$a_2 = \begin{pmatrix} & & & & & & & & & \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



- Lemma: For any subset of nodes S⊂V,

$$\text{support}\left(\sum_{i\in S} a_i\right) = E(S, V \setminus S)$$

- Lemma: ∃ random M: $\mathbb{R}^N \to \mathbb{R}^k$ with $k=O(\log^2 N)$ such that for any $a\in\mathbb{R}^N$, with probability 0.9

$$Ma \longrightarrow e \in \text{support}(a)$$

Recipe: Sketch & Compute on Sketches

- Sketch: Each player sends Ma_j
- Central Player Runs Algorithm in Sketch Space:
  - Use Ma_j to get incident edge on each node j
  - For i=2 to t:
    - To get incident edge on component S⊂V use:

$$\sum_{j\in S} Ma_j = M\left(\sum_{j\in S} a_j\right) \longrightarrow e \in \text{support}\left(\sum_{j\in S} a_j\right) = E(S, V \setminus S)$$

I. Connectivity. Can check connectivity of dynamic graph stream in O(n polylog n) space.

Ingredient 1: Basic Algorithm

- Algorithm (k-Connectivity):
  1. Let F_1 be spanning forest of G(V,E)
  2. For i=2 to k:
     2.1. Let F_i be spanning forest of $G(V,E-F_1-\ldots-F_{i-1})$
- Lemma: $G(V,F_1+\ldots+F_k)$ is k-connected iff G(V,E) is.

Ingredient 2: Connectivity Sketches

- Sketch: Simultaneously construct k independent sketches $\{M_1G, M_2G, \ldots M_kG\}$ for connectivity.
- Run Algorithm in Sketch Space:
  - Use M_1G to find a spanning forest F_1 of G
  - Use $M_2G-M_2F_1=M_2(G-F_1)$ to find F_2
  - Use $M_3G-M_3F_1-M_3F_2=M_3(G-F_1-F_2)$ to find F_3
  - etc.

II. k-Connectivity. Can check k-connectivity in O(n k polylog n) space.

Ingredient 1: Sparsification

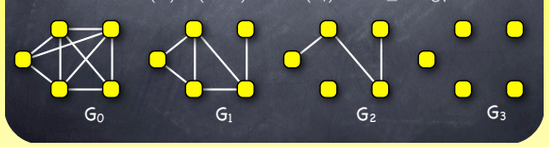
- Lemma (Karger): Sample each edge of G with prob.

$$p \geq p^* = 6\lambda^{-1}\epsilon^{-2}\log n \quad \text{where } \lambda = \text{value of min-cut}$$

Weight by 1/p. Cuts are preserved to 1+ε factor.

- Corollary: Define G_i by sampling edges with probability 2^{-i} and applying weights 2^i. Then

$$\text{Min-Cut}(G) = (1\pm\epsilon)\text{Min-Cut}(G_i) \quad \text{for } i \leq -\log p^*$$



Ingredient 2: k-Connectivity

- k-connectivity: Given G_i returns subgraph H_i that contains min(k,all) edges across each cut.
- Lemma: For $k=O(\epsilon^{-2}\log n)$, with good probability

$$\text{Min-Cut}(G_i) = \text{Min-Cut}(H_i) < 2^i k \quad \text{for } i \geq -\log p^*$$

- Putting it together: Construct H_i for all G_i. Return Min-Cut(H_i) for smallest i with Min-Cut(H_i) < 2^i k.

III. Min-Cut. Can (1+ε)-approx min-cut in O(n ϵ^{-2} polylog n) space.

Other Results

Sparsification: A cut-sparsifier is a weighted subgraph in which all cuts are preserved up to a factor 1+ε. Can construct in O(n ϵ^{-2} polylog n) space.

Spanners: A (2t-1)-spanner is a subgraph which preserves distances up to a factor 2t-1. Multi-pass construction in O($n^{1+1/t}$ polylog n) space.

Subgraphs: Algorithm for estimating the frequency of small (induced) subgraphs of a dynamic graph.

Conclusions

There exist sketches projecting $O(n^2)$-dimensional graphs to O(n) dimensions such that structural properties can be inferred from the projection. Applications to streams and distributed computing.

References

Ahn, Guha, McGregor. *Graph Sketching: Sparsification, Spanners, and Subgraphs*, PODS 2012.

Ahn, Guha, McGregor. *Analyzing Graph Structure via Linear Measurements*, SODA 2012.