Graph Synopses, Sketches, and Streams: A Survey

Sudipto Guha
University of Pennsylvania

Andrew McGregor
University of Massachusetts
Massive Graphs

- **Classic Big Graphs:**
  Call graph (5x10^8 nodes), web graph (5x10^{10} nodes), IP graph (2^{32} nodes), social networks (10^9 nodes), ...

**Challenge:** Can’t use conventional algorithms on graphs this large. Sometimes can’t even store graph in memory! Graphs may be dynamic and/or distributed.
Massive Graphs

- **Classic Big Graphs:**
  Call graph (5x10^8 nodes), web graph (5x10^{10} nodes), IP graph (2^{32} nodes), social networks (10^9 nodes), ...

  **Challenge:** Can’t use conventional algorithms on graphs this large. Sometimes can’t even store graph in memory!
  Graphs may be dynamic and/or distributed.

- **Use Abstraction of Structure:**
  Graphs are a natural way to encode structural information where we have data about both basic entities and their relationships. Examples include graphical networks, citation networks, protein interaction and metabolic networks, ...
Focus of Tutorial
Focus of Tutorial

*Question 1:* What are appropriate *synopsis data structures* for massive graphs? How do we trade-off space and accuracy?
Focus of Tutorial

**Question 1:** What are appropriate synopsis data structures for massive graphs? How do we trade-off space and accuracy?

**Question 2:** How can we construct these synopses efficiently? In particular, what is the input is streaming or distributed?
Focus of Tutorial

**Question 1:** What are appropriate synopsis data structures for massive graphs? How do we trade-off space and accuracy?

**Question 2:** How can we construct these synopses efficiently? In particular, what is the input is streaming or distributed?

- Tutorial focuses on the algorithmic and theoretical issues. Consider arbitrary graphs rather than being domain specific.

  **This Talk:** Definitions & Basic Building Blocks

  **Next Talk:** Applications & Extensions
Mark and Erica are now friends.
Mark and Erica are no longer friends.
Eduardo and Mark are now friends.
Tyler and Cameron are friends with Mark.
Sean and Mark are now friends.
Eduardo and Mark are no longer friends.
Tyler and Cameron are no longer friends with Mark.
Lawyers are now friends with everyone.
Data Streams

- **Input:** Observe stream of edges on n nodes added/deleted.
Input: Observe stream of edges on n nodes added/deleted.

Example: Using $\tilde{O}(n)$ space, maintain connected components.
Data Streams

- **Input**: Observe stream of edges on $n$ nodes added/deleted.
- **Example**: Using $\tilde{O}(n)$ space, maintain connected components.
- **Other Results**: Dense subgraphs, matchings, distances, clustering, partitioning and cuts, diameter, random walks, ...

Distributed Processing

Input: $G=(V,E)$
Distributed Processing

Input: $G = (V, E)$

$G_1 = (V, E_1)$  $G_2 = (V, E_2)$  $G_3 = (V, E_3)$  $G_4 = (V, E_4)$
Distributed Processing

Input: $G=(V,E)$

$G_1=(V,E_1)$

$G_2=(V,E_2)$

$G_3=(V,E_3)$

$G_4=(V,E_4)$

$f(G_1)$

$f(G_2)$

$f(G_3)$

$f(G_4)$
Distributed Processing

Input: $G=(V,E)$

$G_1=(V,E_1)$
$G_2=(V,E_2)$
$G_3=(V,E_3)$
$G_4=(V,E_4)$

$f(G_1)$
$f(G_2)$
$f(G_3)$
$f(G_4)$

Output: $f(G)$ given $f(G_1), \ldots, f(G_4)$
I. Spanners

II. Sparsifiers

III. Sketches
I. Spanners

Synopsis for Distance Estimation
“Greedy” Stream Algorithm
Extensions
Spanners & Distances
Spanners & Distances

- **Measure:** The distance $d_G(u,v)$ between two nodes $u$, $v$ is the length of the shortest path between the nodes.

Original Graph $G$
Spanners & Distances

- **Measure:** The distance $d_G(u,v)$ between two nodes $u, v$ is the length of the shortest path between the nodes.

- **Synopsis:** A subgraph $H$ of $G$ is a $k$-spanner if

  $$d_G(u,v) \leq d_H(u,v) \leq k \cdot d_G(u,v)$$

  for all node pairs.
Spanners & Distances

• **Measure:** The distance $d_G(u,v)$ between two nodes $u$, $v$ is the length of the shortest path between the nodes.

• **Synopsis:** A subgraph $H$ of $G$ is a $k$-spanner if

$$d_G(u,v) \leq d_H(u,v) \leq k \cdot d_G(u,v)$$

for all node pairs.
Spanners & Distances

- **Measure**: The distance $d_G(u,v)$ between two nodes $u, v$ is the length of the shortest path between the nodes.

- **Synopsis**: A subgraph $H$ of $G$ is a $k$-spanner if

  $d_G(u,v) \leq d_H(u,v) \leq k \cdot d_G(u,v)$ for all node pairs.

- **Thm**: Streaming construction using $O(n^{1+2/(k+1)})$ space.
Spanner: Algorithm
Spanner: Algorithm
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u, v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
**Algorithm**: Add new edge \((u, v)\) to \(H\) if \(d_H(u, v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
\textbf{Algorithm:} Add new edge \((u, v)\) to \(H\) if \(d_H(u, v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Spanner: Algorithm

Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).
Algorithm: Add new edge \((u,v)\) to \(H\) if \(d_{H}(u,v) > 3\).

Lemma: All distances preserved up to a factor 3.
**Spanner: Algorithm**

**Algorithm:** Add new edge \((u,v)\) to \(H\) if \(d_H(u,v) > 3\).

**Lemma:** All distances preserved up to a factor 3.

**Lemma:** \(O(n^{3/2})\) edges stored since shortest cycle among stored edges has length at least 5.
Spanners: Analysis
Spanners: Analysis

If H has m edges, average degree is $d=\frac{2m}{n}$. 
If $H$ has $m$ edges, average degree is $d = \frac{2m}{n}$.

Claim: $H$ contains a non-empty subgraph $H'$ with minimum degree at least $d' = \frac{d}{2}$
If $H$ has $m$ edges, average degree is $d = \frac{2m}{n}$.

Claim: $H$ contains a non-empty subgraph $H'$ with minimum degree at least $d' = \frac{d}{2}$

Proof: Remove all nodes with degree $< d'$. Can only remove $< nd' = nd/2 = m$ edges so $H'$ non-empty.
**Spanners: Analysis**

- If $H$ has $m$ edges, average degree is $d = 2m/n$.

**Claim:** $H$ contains a non-empty subgraph $H'$ with minimum degree at least $d' = d/2$.

**Proof:** Remove all nodes with degree $< d'$. Can only remove $< nd' = nd/2 = m$ edges so $H'$ non-empty.

Consider node in $H'$:
Claim: $H$ contains a non-empty subgraph $H'$ with minimum degree at least $d' = d/2$

Proof: Remove all nodes with degree < $d'$. Can only remove < $nd' = nd/2 = m$ edges so $H'$ non-empty.

Consider node in $H'$:

![Diagram of a node connected to three other nodes with degree $d'$]
If $H$ has $m$ edges, average degree is $d = 2m/n$.

**Claim:** $H$ contains a non-empty subgraph $H'$ with minimum degree at least $d' = d/2$

**Proof:** Remove all nodes with degree $< d'$. Can only remove $< nd' = nd/2 = m$ edges so $H'$ non-empty.

Consider node in $H'$:

If length of all cycles is $\geq 5$, the node has at least $d'(d'-1) < n$ distinct neighbors of neighbors.
If \( H \) has \( m \) edges, average degree is \( d = \frac{2m}{n} \).

**Claim:** \( H \) contains a non-empty subgraph \( H' \) with minimum degree at least \( d' = \frac{d}{2} \).

**Proof:** Remove all nodes with degree < \( d' \). Can only remove < \( nd' = \frac{nd}{2} = m \) edges so \( H' \) non-empty.

Consider node in \( H' \):

If length of all cycles is \( \geq 5 \), the node has at least \( d'(d'-1) \) < \( n \) distinct neighbors of neighbors.
Spanners Summary

- **Thm:** There’s a $O(n^{1+1/t})$-space stream algorithm returns a $(2t-1)$-spanner. [Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Extension:** Can process weighted graphs by rounding weights and constructing spanners for each weight class.
II. Sparsifiers

Synopsis for Cut Estimation
Merge-Reduce Stream Algorithm
Extensions
Sparsifiers & Cuts
Sparsifiers & Cuts

- **Measure:** Given a cut \((L,R)\), the size of a cut \(c_G(L,R)\) is the weight of all edges crossing the cut.
Sparsifiers & Cuts

- **Measure:** Given a cut $(L,R)$, the size of a cut $c_G(L,R)$ is the weight of all edges crossing the cut.

- **Synopsis:** A subgraph $H$ of $G$ is a $(1+\varepsilon)$ *sparsifier* if

$$c_G(L,R) \leq c_H(L,R) \leq (1+\varepsilon) c_G(L,R)$$

for all cuts.
Sparsifiers & Cuts

- **Measure:** Given a cut \((L,R)\), the size of a cut \(c_G(L,R)\) is the weight of all edges crossing the cut.

- **Synopsis:** A subgraph \(H\) of \(G\) is a \((1+\varepsilon)\) sparsifier if

\[
c_G(L,R) \leq c_H(L,R) \leq (1+\varepsilon) c_G(L,R) \quad \text{for all cuts.}
\]
Sparsifiers & Cuts

**Measure:** Given a cut $(L,R)$, the size of a cut $c_G(L,R)$ is the weight of all edges crossing the cut.

**Synopsis:** A subgraph $H$ of $G$ is a $(1+\varepsilon)$ sparsifier if

$$c_G(L,R) \leq c_H(L,R) \leq (1+\varepsilon) c_G(L,R)$$

for all cuts.

**Thm (Benzur-Karger):** For any graph $G$ there exists a $(1+\varepsilon)$ sparsifier with only $O(\varepsilon^{-2} n)$ edges.
Sparsifiers & Cuts

- **Measure:** Given a cut \((L, R)\), the size of a cut \(c_G(L, R)\) is the weight of all edges crossing the cut.

- **Synopsis:** A subgraph \(H\) of \(G\) is a \((1+\varepsilon)\) sparsifier if
  
  \[c_G(L, R) \leq c_H(L, R) \leq (1+\varepsilon) c_G(L, R)\]
  
  for all cuts.

- **Thm (Benzur-Karger):** For any graph \(G\) there exists a \((1+\varepsilon)\) sparsifier with only \(O(\varepsilon^{-2} n)\) edges.

- **Thm:** Streaming construction in \(O(\varepsilon^{-2} n \log^3 n)\) space.

*Graphs courtesy of Nick Harvey*
Sparsifier: Algorithm
Sparsifier: Algorithm

Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.
Sparsifier: Algorithm

Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.
Sparsifier: Algorithm

Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2} n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.
Main Idea: Segment stream as $E_1, E_2, \ldots$ each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.
Main Idea: Segment stream as $E_1, E_2, \ldots$ each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.
Sparsifier: Algorithm

Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\epsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\epsilon / \log n)$ yields a $(1+\epsilon)$ sparsifier.
**Sparsifier: Algorithm**

**Main Idea:** Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

**Lemma:** $H_{TOP}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

**Lemma:** Can find $H_{TOP}$ with $O(\gamma^{-2} n \log n)$ memory.
Sparsifier: Algorithm

- **Main Idea:** Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2} n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

- **Lemma:** $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

- **Lemma:** Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
**Main Idea:** Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2} n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

**Lemma:** $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

**Lemma:** Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
**Sparsifier: Algorithm**

- **Main Idea:** Segment stream as $E_1, E_2, ...$ each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

  - **Lemma:** $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.
  - **Lemma:** Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
Main Idea: Segment stream as $E_1, E_2, \ldots$ each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon/\log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{\text{TOP}}$ with $O(\gamma^{-2}n \log n)$ memory.
**Sparsifier: Algorithm**

**Main Idea:** Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

**Lemma:** $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

**Lemma:** Can find $H_{\text{TOP}}$ with $O(\gamma^{-2}n \log n)$ memory.
Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2} n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d = O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
Sparsifier: Algorithm

Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2} n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
Main Idea: Segment stream as $E_1, E_2, \ldots$ each of size $O(\varepsilon^{-2} n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{TOP}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{TOP}$ with $O(\gamma^{-2} n \log n)$ memory.
Sparsifier: Algorithm

Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon/\log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon/\log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{TOP}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{TOP}$ with $O(\gamma^{-2}n \log n)$ memory.
**Sparsifier: Algorithm**

- **Main Idea:** Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

- **Lemma:** $H_{TOP}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon/\log n)$ yields a $(1+\varepsilon)$ sparsifier.

- **Lemma:** Can find $H_{TOP}$ with $O(\gamma^{-2} n \log n)$ memory.
Sparsifier: Algorithm

Main Idea: Segment stream as $E_1, E_2, \ldots$ each of size $O(\varepsilon^{-2} n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{TOP}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{TOP}$ with $O(\gamma^{-2} n \log n)$ memory.
**Sparsifier: Algorithm**

- **Main Idea:** Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^2 n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

- **Lemma:** $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d = O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

- **Lemma:** Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
Main Idea: Segment stream as $E_1, E_2, \ldots$ each of size $O(\varepsilon^{-2}n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon/\log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{\text{TOP}}$ with $O(\gamma^{-2}n \log n)$ memory.
Sparsifier: Algorithm

Main Idea: Segment stream as $E_1$, $E_2$, ... each of size $O(\varepsilon^{-2} n)$. Let $H_1$ be $(1+\gamma)$ sparsifier of $E_1 \cup E_2$ etc.

Lemma: $H_{\text{TOP}}$ is a $(1+\gamma)^d$ sparsifier for $d=O(\log n)$. Setting $\gamma = O(\varepsilon / \log n)$ yields a $(1+\varepsilon)$ sparsifier.

Lemma: Can find $H_{\text{TOP}}$ with $O(\gamma^{-2} n \log n)$ memory.
Sparsifier Summary

• **Thm:** A \((1+\varepsilon)\) sparsifier of a graph can be constructed in \(O(\varepsilon^{-2} n \text{ polylog } n)\) space.

  [Ahn, Guha 09], [Goel, Kapralov, Khanna 10], [Sidiropoulos 10]

• Generalizes to spectral sparsification which preserves properties relating to random walks.

  [Kelner, Levin 11]
III. Sketches

Family of Linear Synopses
Distributed & Supports Deletions
Two Connectivity Examples
Linear Sketches
Linear Sketches

\[
\begin{bmatrix}
  v
\end{bmatrix}
\]
Linear Sketches

- **Random linear projection**: $M : \mathbb{R}^n \rightarrow \mathbb{R}^k$ (where $k \ll n$) that preserves properties of any $v \in \mathbb{R}^n$ with high probability.
• **Random linear projection:** $M : \mathbb{R}^n \rightarrow \mathbb{R}^k$ (where $k \ll n$) that preserves properties of any $v \in \mathbb{R}^n$ with high probability.

\[
\begin{bmatrix}
M \\
\end{bmatrix}
\begin{bmatrix}
v \\
\end{bmatrix}
\]
Linear Sketches

- *Random linear projection:* $M: \mathbb{R}^n \rightarrow \mathbb{R}^k$ (where $k \ll n$) that preserves properties of any $v \in \mathbb{R}^n$ with high probability.

$$
\begin{bmatrix}
M
\end{bmatrix}
\begin{bmatrix}
v
\end{bmatrix}
= 
\begin{bmatrix}
Mv
\end{bmatrix}
$$
• **Random linear projection:** $M: \mathbb{R}^n \rightarrow \mathbb{R}^k$ (where $k \ll n$) that preserves properties of any $v \in \mathbb{R}^n$ with high probability.

$$
\begin{bmatrix}
M \\
v
\end{bmatrix} =
\begin{bmatrix}
Mv
\end{bmatrix} \rightarrow \text{answer}
$$
Random linear projection: $M : \mathbb{R}^n \rightarrow \mathbb{R}^k$ (where $k \ll n$) that preserves properties of any $v \in \mathbb{R}^n$ with high probability.

\[
\begin{bmatrix}
M \\
\end{bmatrix}
\begin{bmatrix}
v \\
\end{bmatrix} = \begin{bmatrix}
Mv \\
\end{bmatrix} \rightarrow \text{answer}
\]

Many results for numerical statistics and basic geometric properties... extensive theory with connections to hashing, compressed sensing, dimensionality reduction, metric embeddings... widely applicable since embarrassingly parallelizable and suitable for stream processing.
Linear Sketches

• **Random linear projection:** $M: \mathbb{R}^n \rightarrow \mathbb{R}^k$ (where $k \ll n$) that preserves properties of any $v \in \mathbb{R}^n$ with high probability.

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

• **Many results** for numerical statistics and basic geometric properties... *extensive theory* with connections to hashing, compressed sensing, dimensionality reduction, metric embeddings... *widely applicable* since embarrassingly parallelizable and suitable for stream processing.

? **Question:** What about analyzing massive graphs via sketches?
Distributed Data
Distributed Data

• **Input:** Each player knows neighborhood $\Gamma(v)$ for a node $v$
**Distributed Data**

- **Input**: Each player knows neighborhood $\Gamma(v)$ for a node $v$
- **Goal**: Simultaneously, each player sends $O(\text{polylog } n)$ bits to a central player who then determines if the graph is connected.
This can’t be possible?! 

- Suppose there’s a *bridge* \((u,v)\) in the graph, i.e., a friendship that is essential to ensuring the graph is connected.
• Suppose there’s a bridge \((u,v)\) in the graph, i.e., a friendship that is essential to ensuring the graph is connected.

• \textbf{Dubious Claim:} At least one player needs to send \(\Omega(n)\) bits.
• Suppose there’s a bridge \((u,v)\) in the graph, i.e., a friendship that is essential to ensuring the graph is connected.

• **Dubious Claim:** At least one player needs to send \(\Omega(n)\) bits.
  a) Central player needs to know about the special friendship.
• Suppose there’s a bridge \((u,v)\) in the graph, i.e., a friendship that is essential to ensuring the graph is connected.

• **Dubious Claim:** At least one player needs to send \(\Omega(n)\) bits.
  a) Central player needs to know about the special friendship.
  b) Participant doesn’t know which friendships are special.
• Suppose there’s a bridge \((u,v)\) in the graph, i.e., a friendship that is essential to ensuring the graph is connected.

• **Dubious Claim:** At least one player needs to send \(\Omega(n)\) bits.
  a) Central player needs to know about the special friendship.
  b) Participant doesn’t know which friendships are special.
  c) Participants may have \(\Omega(n)\) friends.
How to do it...
How to do it...

- Players send carefully-designed sketches of address books.
How to do it...

- Players send carefully-designed sketches of address books.
- **Main Idea:** Exploit homomorphic properties of linear sketches and emulate a classical algorithm in *sketch space.*
How to do it...

- Players send carefully-designed sketches of address books.
- **Main Idea:** Exploit homomorphic properties of linear sketches and emulate a classical algorithm in *sketch space*.
How to do it...

- Players send carefully-designed sketches of address books.
- **Main Idea:** Exploit homomorphic properties of linear sketches and emulate a classical algorithm in *sketch space*.
How to do it...

- Players send carefully-designed sketches of address books.
- **Main Idea:** Exploit homomorphic properties of linear sketches and emulate a classical algorithm in *sketch space*.
How to do it...

- Players send carefully-designed sketches of address books.
- **Main Idea:** Exploit homomorphic properties of linear sketches and emulate a classical algorithm in *sketch space*. 
Two Examples

**First Theorem:** Testing Connectivity

a) *Dynamic Graph Stream:* $O(n \text{ polylog } n)$ space.

b) *Distributed Setting:* $O(\text{polylog } n)$ length messages.

**Second Theorem:** Checking every cut has size $\geq k$

a) *Dynamic Graph Stream:* $O(n k \text{ polylog } n)$ space.

b) *Distributed Setting:* $O(k \text{ polylog } n)$ length.
Ingredient 1: Basic Algorithm
Ingredient 1: **Basic Algorithm**

- Algorithm (Spanning Forest):
Ingredient 1: **Basic Algorithm**

Algorithm (Spanning Forest):
1. For each node: pick incident edge
Ingredient 1: Basic Algorithm

Algorithm (Spanning Forest):

1. For each node: pick incident edge
Ingredient 1: **Basic Algorithm**

Algorithm (Spanning Forest):

1. For each node: pick incident edge
Ingredient 1: **Basic Algorithm**

Algorithm (Spanning Forest):

1. For each node: pick incident edge
Ingredient 1: Basic Algorithm

Algorithm (Spanning Forest):

1. For each node: pick incident edge
2. For each connected comp: pick incident edge
Ingredient 1: **Basic Algorithm**

Algorithm (Spanning Forest):

1. For each node: pick incident edge
2. For each connected comp: pick incident edge
**Ingredient 1: Basic Algorithm**

**Algorithm (Spanning Forest):**

1. For each node: pick incident edge
2. For each connected comp: pick incident edge
Ingredient 1: **Basic Algorithm**

- 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.
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: After $O(\log n)$ rounds 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 $j<i$.

\[
a_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\]
\[
a_2 = \begin{pmatrix} -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\]
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 $j<i$.

$a_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

$a_2 = \begin{pmatrix} -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
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 $j<i$.

Ingredient 2: Sketching Neighborhoods

$$a_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$a_2 = \begin{pmatrix} -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$a_1 + a_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$
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 \( j<i \).

\[
\begin{align*}
a_1 & = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \{1,2\} & \{1,3\} & \{1,4\} & \{1,5\} & \{2,3\} & \{2,4\} & \{2,5\} & \{3,4\} & \{3,5\} & \{4,5\} \end{pmatrix} \\
a_2 & = \begin{pmatrix} -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \{1,2\} & \{1,3\} & \{1,4\} & \{1,5\} & \{2,3\} & \{2,4\} & \{2,5\} & \{3,4\} & \{3,5\} & \{4,5\} \end{pmatrix} \\
a_1 + a_2 & = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \{1,2\} & \{1,3\} & \{1,4\} & \{1,5\} & \{2,3\} & \{2,4\} & \{2,5\} & \{3,4\} & \{3,5\} & \{4,5\} \end{pmatrix}
\end{align*}
\]

**Lemma:** For any subset of nodes \( S \subset V \),

\[
support \left( \sum_{i \in S} a_i \right) = E(S, V \setminus S)
\]
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 $j<i$.

\[
\begin{align*}
a_1 &= \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
a_2 &= \begin{pmatrix} -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\
a_1 + a_2 &= \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{align*}
\]

Lemma: For any subset of nodes $S \subset V$,

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

Lemma: $\exists$ random $M: \mathbb{R}^N \rightarrow \mathbb{R}^k$ with $k=O(polylog N)$ such that for any $a \in \mathbb{R}^N$, with high probability

\[
Ma \rightarrow e \in support(a)
\]
Recipe: Sketch & Compute on Sketches
Recipe: **Sketch & Compute on Sketches**

- **Sketch:** Each player sends $\text{Maj}$
Recipe: Sketch & Compute on Sketches

Sketch: Each player sends Maj

Central Player Runs Algorithm in Sketch Space:
Recipe: Sketch & Compute on Sketches

- **Sketch**: Each player sends $\text{Maj}$
- **Central Player Runs Algorithm in Sketch Space**: Use $\text{Maj}$ to get incident edge on each node $j$
Recipe: Sketch & Compute on Sketches

- **Sketch:** Each player sends $\text{Maj}$
- **Central Player Runs Algorithm in Sketch Space:**
  - Use $\text{Maj}$ to get incident edge on each node $j$
  - For $i=2$ to $\log n$:
    - To get incident edge on component $S \subseteq V$ use:
Recipe: Sketch & Compute on Sketches

Sketch: Each player sends Maj

Central Player Runs Algorithm in Sketch Space:

- Use Maj to get incident edge on each node j
- For i=2 to log n:
  - To get incident edge on component S⊂V use:

\[ \sum_{j \in S} Maj = M(\sum_{j \in S} a_j) \]
Recipe: Sketch & Compute on Sketches

- **Sketch:** Each player sends $\text{Maj}$
- **Central Player Runs Algorithm in Sketch Space:**
  - Use $\text{Maj}$ to get incident edge on each node $j$
  - For $i=2$ to $\log n$:
    - To get incident edge on component $S \subset V$ use:

$$\sum_{j \in S} \text{Maj} = M\left(\sum_{j \in S} a_j\right) \quad \rightarrow \quad e \in \text{support}\left(\sum_{j \in S} a_j\right) = E(S, V \setminus S)$$
Recipe: Sketch & Compute on Sketches

- **Sketch:** Each player sends $\text{Maj}$
- **Central Player Runs Algorithm in Sketch Space:**
  - Use $\text{Maj}$ to get incident edge on each node $j$
  - For $i=2$ to $\log n$:
    - To get incident edge on component $S \subset V$ use:
      $$\sum_{j \in S} \text{Ma}_j = M(\sum_{j \in S} a_j) \quad \rightarrow \quad e \in \text{support}(\sum_{j \in S} a_j) = E(S, V \setminus S)$$

**Detail:** Actually each player sends $\log n$ indept sketches $\text{M}_1a_j, \text{M}_2a_j, \ldots$ and central player uses $\text{M}_ia_j$ when emulating $i^{th}$ iteration of the algorithm.
Two Examples

**First Theorem:** Testing Connectivity
a) *Dynamic Graph Stream:* $O(n \text{ polylog } n)$ space.
b) *Distributed Setting:* $O(\text{polylog } n)$ length messages.

**Second Theorem:** Checking every cut has size $\geq k$
(a) *Dynamic Graph Stream:* $O(n^k \text{ polylog } n)$ space.
(b) *Distributed Setting:* $O(k \text{ polylog } n)$ length.
Two Examples

**First Theorem:** Testing Connectivity

a) **Dynamic Graph Stream:** $O(n \text{ polylog } n)$ space.

b) **Distributed Setting:** $O(\text{polylog } n)$ length messages.

**Second Theorem:** Checking every cut has size $\geq k$

a) **Dynamic Graph Stream:** $O(n k \text{ polylog } n)$ space.

b) **Distributed Setting:** $O(k \text{ polylog } n)$ length.
Ingredient 1: Basic Algorithm
Ingredient 1: Basic Algorithm

Algorithm (k-Connectivity):
Ingredient 1: **Basic Algorithm**

**Algorithm (k-Connectivity):**

1. Let $F_1$ be spanning forest of $G(V,E)$
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 $G(V,E-F_1-\ldots-F_{i-1})$

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
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent connectivity sketches $\{M_1G, M_2G, \ldots, M_kG\}$. 
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent connectivity sketches $\{M_1 G, M_2 G, \ldots M_k G\}$.

**Run Algorithm in Sketch Space:**

- Use $M_1 G$ to find a spanning forest $F_1$ of $G$
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent connectivity sketches $\{M_1G, M_2G, \ldots M_kG\}$.

**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 $F_2$. 

Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent connectivity sketches $\{M_1G, M_2G, \ldots, M_kG\}$.

**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 $F_3$
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent connectivity sketches \{\(M_1G, M_2G, \ldots, M_kG\)\}.

**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.
Sketches Summary

- **Graph Sketches:** Linear projections that preserve structural graph properties. Results *parallelizable, streamable, and support deletions.*
- **Talk Results:** Projecting $O(n)$-dimensional neighborhoods to $O(\text{polylog } n)$ dimensions while preserving connectivity and cuts.
- **Other Results:** Spanners, Bipartiteness, MST, Triangles, Matching, ...
And over to Part II...

Sağ olun!