Sketching Graphs

Kook Jin Ahn  University of Pennsylvania
Sudipto Guha  University of Pennsylvania
Andrew McGregor  University of Massachusetts
Linear Sketches
Linear Sketches

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

\[
\begin{pmatrix}
\vdots \\
v
\end{pmatrix}
\]
Linear Sketches

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

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

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

$$
\begin{pmatrix}
M \\
\vdots \\
0
\end{pmatrix}
\begin{pmatrix}
v \\
\vdots
\end{pmatrix}
= 
\begin{pmatrix}
Mv \\
\vdots
\end{pmatrix}$$
Linear Sketches

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

$$
\begin{pmatrix}
M
\end{pmatrix}
\begin{pmatrix}
v
\end{pmatrix}
= \begin{pmatrix}
Mv
\end{pmatrix} \rightarrow \text{answer}
$$
Linear Sketches

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

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

- **Many Results**: Estimating norms, entropy, support size, quantiles, heavy hitters, fitting histograms and polynomials, ...
Linear Sketches

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

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

- **Many Results**: Estimating norms, entropy, support size, quantiles, heavy hitters, fitting histograms and polynomials, ...

- **Rich Theory**: Related to compressed sensing and sparse recovery, dimensionality reduction and metric embeddings, ...
Sketching Graphs?
Sketching Graphs?

Question: Are there sketches for structured objects like graphs?
Sketching Graphs?

? **Question:** Are there sketches for structured objects like graphs?

\[
\begin{pmatrix}
A_G
\end{pmatrix}
\]
Sketching Graphs?

? **Question:** Are there sketches for structured objects like graphs?

\[
\begin{pmatrix}
M
\end{pmatrix}
\begin{pmatrix}
A_G
\end{pmatrix}
\]
Sketching Graphs?

? **Question**: Are there sketches for structured objects like graphs?

\[
\begin{pmatrix}
M \\
A_G
\end{pmatrix}
\begin{pmatrix}
\end{pmatrix}
= \\
\begin{pmatrix}
MA_G
\end{pmatrix}
\]
Question: Are there sketches for structured objects like graphs?

\[
\begin{pmatrix}
M \\
A_G
\end{pmatrix}
\begin{pmatrix}
\end{pmatrix}
= \begin{pmatrix}
MA_G
\end{pmatrix} \rightarrow \text{answer}
\]
Sketching Graphs?

? Question: Are there sketches for structured objects like graphs?

\[
\begin{pmatrix}
M \\
A_G
\end{pmatrix}
\begin{pmatrix}
\end{pmatrix}
= \begin{pmatrix}
MA_G
\end{pmatrix} \rightarrow \text{answer}
\]

• Example: Project \(O(n^2)\)-dimensional adjacency matrix \(A_G\) to \(\tilde{O}(n)\) dimensions and still determine if graph is bipartite?
Sketching Graphs?

? **Question:** Are there sketches for structured objects like graphs?

\[
\begin{pmatrix}
M \\
A_G
\end{pmatrix}
\begin{pmatrix}
\cdot \\
\cdot
\end{pmatrix}
= \begin{pmatrix}
\cdot \\
MA_G
\end{pmatrix} \rightarrow \text{answer}
\]

• **Example:** Project $O(n^2)$-dimensional adjacency matrix $A_G$ to $\tilde{O}(n)$ dimensions and still determine if graph is bipartite?

! No cheating! Assume $M$ is finite precision etc.
Why? Graph Streams
Why? Graph Streams

- In **semi-streaming**, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]
Why? Graph Streams

• In semi-streaming, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.
  
  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!
Why? Graph Streams

- In semi-streaming, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs:** Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example:** Connectivity is easy if edges are only inserted...
Why? Graph Streams

• In **semi-streaming**, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

• **Example**: Connectivity is easy if edges are only inserted...

```
1  2  3
4  5  6
7  8  9
```
Why? Graph Streams

• In **semi-streaming**, want to process graph defined by edges $e_1, \ldots, e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

• **Example**: Connectivity is easy if edges are only inserted...

![Diagram of a graph with nodes 1 to 9 and edges connecting them](image)
Why? Graph Streams

- In semi-streaming, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.
  
  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs:** Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example:** Connectivity is easy if edges are only inserted...

```
   1 -- 2 -- 3
   |     |     |
  4----5----6
   |     |     |
   7 -- 8 -- 9
```
Why? Graph Streams

- In **semi-streaming**, want to process graph defined by edges $e_1, \ldots, e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example**: Connectivity is easy if edges are only inserted...

![Graph diagram](image)
Why? Graph Streams

- In **semi-streaming**, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example**: Connectivity is easy if edges are only inserted...

![Graph Diagram](image-url)
Why? Graph Streams

• **In semi-streaming**, want to process graph defined by edges $e_1, \ldots, e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs:** Work on graph streams doesn't support edge deletions! Work on dynamic graphs stores entire graph!

• **Example:** Connectivity is easy if edges are only inserted...

![Graph Diagram](image-url)
Why? Graph Streams

• In **semi-streaming**, want to process graph defined by edges \( e_1, \ldots, e_m \) with \( \tilde{O}(n) \) memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

• **Example**: Connectivity is easy if edges are only inserted...

![Graph Diagram](image)
Why? Graph Streams

- In **semi-streaming**, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.
  
  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example**: Connectivity is easy if edges are only inserted...

![Graph Diagram]
Why? Graph Streams

- In **semi-streaming**, want to process graph defined by edges $e_1, \ldots, e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.
  
  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example**: Connectivity is easy if edges are only inserted...

![Graph Diagram]

- Vertices: 1, 2, 3, 4, 5, 6, 7, 8, 9
- Edges:
  - 1 to 2
  - 2 to 3
  - 4 to 5
  - 5 to 6
  - 7 to 8
  - 8 to 9
Why? Graph Streams

- In semi-streaming, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.
  
  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs:** Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example:** Connectivity is easy if edges are only inserted...

```
1 -- 2 -- 3
|
4 -- 5 -- 6
|
7 -- 8 -- 9
```
Why? Graph Streams

• In semi-streaming, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

• **Example**: Connectivity is easy if edges are only inserted...
Why? Graph Streams

- In **semi-streaming**, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example**: Connectivity is easy if edges are only inserted...

![Graph Diagram]
Why? Graph Streams

• In semi-streaming, want to process graph defined by edges \( e_1, ..., e_m \) with \( \tilde{O}(n) \) memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

• **Example**: Connectivity is easy if edges are only inserted...

![Graph Diagram]

[1] 2 3
[4] 5 6
[7] 8 9
Why? Graph Streams

• In semi-streaming, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

[Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

• **Example**: Connectivity is easy if edges are only inserted...

![Graph Diagram](image)
Why? Graph Streams

• In **semi-streaming**, want to process graph defined by edges $e_1, \ldots, e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.
  
  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

• **Example**: Connectivity is easy if edges are only inserted...

[Diagram of a graph with vertices 1, 2, 3, 4, 5, 6, 7, 8, 9 and edges connecting them.]

Why? Graph Streams

- In semi-streaming, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.
  
  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example**: Connectivity is easy if edges are only inserted...

![Graph Diagram]

1  2  3

4  5  6

7  8  9
Why? Graph Streams

- In semi-streaming, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.
  
  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

- **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

- **Example**: Connectivity is easy if edges are only inserted...
Why? Graph Streams

• In **semi-streaming**, want to process graph defined by edges $e_1, ..., e_m$ with $\tilde{O}(n)$ memory and reading sequence in order.

  [Muthukrishnan 05; Feigenbaum, Kannan, McGregor, Suri, Zhang 05]

• **Dynamic Graphs**: Work on graph streams doesn’t support edge deletions! Work on dynamic graphs stores entire graph!

• **Example**: Connectivity is easy if edges are only inserted...

• **Sketches**: To delete $e$ from $G$: update $MA_G \rightarrow MA_G - MA_e = MA_{G-e}$
Why? Distributed Processing

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

$G_1 = (V, E_1)$

$G_2 = (V, E_2)$

$G_3 = (V, E_3)$

$G_4 = (V, E_4)$

Why? Distributed Processing
Why? Distributed Processing

Input: $G = (V, E)$

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

$M G_1$  $M G_2$  $M G_3$  $M G_4$
Why? Distributed Processing

Input: $G = (V, E)$

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

Output: $MG = MG_1 + MG_2 + MG_3 + MG_4$
a) Connectivity

b) Applications
a) Connectivity

b) Applications
Connectivity
Connectivity

• *Thm:* Can check connectivity with $O(n \log^3 n)$-size sketch.
Connectivity

• *Thm:* Can check connectivity with $O(n \log^3 n)$-size sketch.
• *Main Idea:* a) Sketch! b) Run Algorithm in Sketch Space
Connectivity

- **Thm:** Can check connectivity with $O(n \log^3 n)$-size sketch.
- **Main Idea:** a) Sketch! b) Run Algorithm in Sketch Space
Connectivity

- **Thm:** Can check connectivity with $O(n \log^3 n)$-size sketch.
- **Main Idea:** 
  a) Sketch! 
  b) Run Algorithm in Sketch Space
Connectivity

- **Thm:** Can check connectivity with $O(n \log^3 n)$-size sketch.
- **Main Idea:**
  1. Sketch!
  2. Run Algorithm in Sketch Space
Connectivity

- **Thm:** Can check connectivity with $O(n \log^3 n)$-size sketch.

- **Main Idea:**
  a) Sketch!
  b) Run Algorithm in Sketch Space
Connectivity

- **Thm:** Can check connectivity with $O(n \log^3 n)$-size sketch.
- **Main Idea:**
  1. Sketch!
  2. Run Algorithm in Sketch Space
- **Catch:** Sketch must be homomorphic for algorithm operations.
Ingredient 1: Basic Connectivity Algorithm
Ingredient 1: **Basic Connectivity Algorithm**

Algorithm (Spanning Forest):
Ingredient 1: Basic Connectivity Algorithm

Algorithm (Spanning Forest):

1. For each node, select an incident edge
Ingredient 1: Basic Connectivity Algorithm

**Algorithm (Spanning Forest):**

1. For each node, select an incident edge
2. Contract selected edges. Repeat until no edges.
Algorithm (Spanning Forest):

1. For each node, select an incident edge
2. Contract selected edges. Repeat until no edges.

Ingredient 1: Basic Connectivity Algorithm
Ingredient 1: **Basic Connectivity Algorithm**

**Algorithm (Spanning Forest):**

1. For each node, select an incident edge
2. Contract selected edges. Repeat until no edges.
Ingredient 1: **Basic Connectivity Algorithm**

Algorithm (Spanning Forest):
1. For each node, select an incident edge
2. Contract selected edges. Repeat until no edges.
Ingredient 1: **Basic Connectivity Algorithm**

**Algorithm (Spanning Forest):**

1. For each node, select an incident edge
2. Contract selected edges. Repeat until no edges.
Ingredient 1: Basic Connectivity Algorithm

**Algorithm (Spanning Forest):**

1. For each node, select an incident edge
2. Contract selected edges. Repeat until no edges.
Ingredient 1: **Basic Connectivity Algorithm**

**Algorithm (Spanning Forest):**

1. For each node, select an incident edge
2. Contract selected edges. Repeat until no edges.

**Lemma:** Takes $O(\log n)$ steps and selected edges include spanning forest.
Ingredient 2: Graph Representation
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: Graph Representation**
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 $j<i$.

Example:
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 \( j<i \).

**Example:**

```
Ingredient 2: Graph Representation

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 \( j<i \).

**Example:**
```

![Graph Representation Example]

```
```
Ingredient 2: Graph Representation

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 $j<i$.

Example:

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

Example:

$\mathbf{a}_1 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

$\mathbf{a}_2 = \begin{pmatrix} -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$
Ingredient 2: Graph Representation

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 $j<i$.

Example:

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

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

$$\text{support} \left( \sum_{i \in S} a_i \right) = E(S, V \setminus S)$$
Ingredient 3: $l_0$-Sampling
Ingredient 3: $l_0$-Sampling

Lemma: Exists random $C \in \mathbb{R}^{d \times m}$ with $d = O(\log^2 m)$ such that for any $a \in \mathbb{R}^m$

$$Ca \rightarrow e \in \text{support}(a)$$

with probability $9/10.$

[Cormode, Muthukrishnan, Rozenbaum 05; Jowhari, Saglam, Tardos 11]
Recipe: Sketch & Compute on Sketches
Recipe: Sketch & Compute on Sketches

- **Sketch**: Apply log n sketches $C_i$ to each $a_j$
- **Run Algorithm in Sketch Space**:
Recipe: Sketch & Compute on Sketches

- **Sketch:** Apply log n sketches $C_i$ to each $a_j$
- **Run Algorithm in Sketch Space:**
  - Use $C_1a_j$ to get incident edge on each node $j$
Recipe: Sketch & Compute on Sketches

- **Sketch**: Apply log n sketches $C_i$ to each $a_j$
- **Run Algorithm in Sketch Space**:
  - Use $C_1a_j$ to get incident edge on each node $j$
  - For $i=2$ to $t$:
Recipe: Sketch & Compute on Sketches

- **Sketch:** Apply log n sketches $C_i$ to each $a_j$
- Run Algorithm in Sketch Space:
  - Use $C_1a_j$ to get incident edge on each node $j$
  - For $i=2$ to $t$:
    - To get incident edge on supernode $S \subset V$ use:

$$\sum_{j \in S} C_ia_j = C_i \left( \sum_{j \in S} a_j \right)$$
Recipe: Sketch & Compute on Sketches

- **Sketch:** Apply log n sketches $C_i$ to each $a_j$
- **Run Algorithm in Sketch Space:**
  - Use $C_1a_j$ to get incident edge on each node $j$
  - For $i=2$ to $t$:
    - To get incident edge on supernode $S \subset V$ use:

\[
\sum_{j \in S} C_i a_j = C_i \left( \sum_{j \in S} a_j \right) \rightarrow e \in \text{support} \left( \sum_{j \in S} a_j \right) = E(S, V \setminus S)
\]
Connectivity

- **Thm:** Can check connectivity with $O(n \log^3 n)$-size sketch.
- **Main Idea:** a) Sketch! b) Run Algorithm in Sketch-Space

**Catch:** Sketch must be homomorphic for algorithm operations.
a) Connectivity

b) Applications
a) Connectivity

b) Applications
Bipartiteness
Bipartiteness

- **Idea:** Given G, define graph G’ where a node v becomes $v_1$ and $v_2$ and edge $(u,v)$ becomes $(u_1,v_2)$ and $(u_2,v_1)$. 
**Bipartiteness**

- **Idea:** Given G, define graph G’ where a node v becomes $v_1$ and $v_2$ and edge $(u,v)$ becomes $(u_1,v_2)$ and $(u_2,v_1)$.
Bipartiteness

- **Idea:** Given $G$, define graph $G'$ where a node $v$ becomes $v_1$ and $v_2$ and edge $(u,v)$ becomes $(u_1,v_2)$ and $(u_2,v_1)$.

- **Lemma:** Number of connected components doubles iff $G$ is bipartite. Can sketch $G'$ implicitly.
Bipartiteness

- **Idea:** Given G, define graph G’ where a node v becomes $v_1$ and $v_2$ and edge $(u,v)$ becomes $(u_1,v_2)$ and $(u_2,v_1)$.

- **Lemma:** Number of connected components doubles iff G is bipartite. Can sketch G’ implicitly.
Bipartiteness

- **Idea:** Given $G$, define graph $G'$ where a node $v$ becomes $v_1$ and $v_2$ and edge $(u,v)$ becomes $(u_1,v_2)$ and $(u_2,v_1)$.

- **Lemma:** Number of connected components doubles iff $G$ is bipartite. Can sketch $G'$ implicitly.

- **Thm:** Can check bipartiteness with $O(n \log^3 n)$-size sketch.
Minimum Spanning Tree
Minimum Spanning Tree

• **Idea:** If $n_i$ is the number of connected components if we ignore edges with weight greater than $(1+\varepsilon)^i$, then:

$$w(\text{MST}) \leq \sum_{i} \varepsilon (1 + \varepsilon)^i n_i \leq (1 + \varepsilon) w(\text{MST})$$
Minimum Spanning Tree

- **Idea:** If \( n_i \) is the number of connected components if we ignore edges with weight greater than \((1+\varepsilon)^i\), then:

\[
w(MST) \leq \sum_{i} \epsilon(1 + \epsilon)^i n_i \leq (1 + \epsilon)w(MST)
\]

- **Thm:** Can \((1+\varepsilon)\) approximate MST in one-pass dynamic semi-streaming model.
Minimum Spanning Tree

• **Idea:** If $n_i$ is the number of connected components if we ignore edges with weight greater than $(1+\varepsilon)^i$, then:

$$w(MST) \leq \sum_{i} \varepsilon(1+\varepsilon)^i n_i \leq (1+\varepsilon)w(MST)$$

• **Thm:** Can $(1+\varepsilon)$ approximate MST in one-pass dynamic semi-streaming model.

• **Thm:** Can find exact MST in dynamic semi-streaming model using $O(\log n/\log \log n)$ passes.
k-Connectivity
k-Connectivity

- A graph is \textit{k-connected} if every cut has size $\geq k$. 
k-Connectivity

- A graph is **k-connected** if every cut has size $\geq k$.
- **Thm:** Can check k-connectivity with $O(nk\log^3 n)$-size sketch.
**k-Connectivity**

- A graph is **k-connected** if every cut has size $\geq k$.
- **Thm:** Can check k-connectivity with $O(nk\log^3 n)$-size sketch.
- **Extension:** There exists a $O(\varepsilon^{-2} n\log^5 n)$-size sketch with which we can approximate all cuts up to a factor $(1+\varepsilon)$.

![Original Graph](image1)

![Sparsifier Graph](image2)
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 $G(V,E)$
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 $G(V,E-F_1-...-F_{i-1})$
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 $G(V,E-F_1-...-F_{i-1})$

Lemma: $G(V,F_1+...+F_k)$ is $k$-connected iff $G(V,E)$ is.
Ingredient 2: Connectivity Sketches
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent sketches $\{M_1A_G, M_2A_G, \ldots, M_kA_G\}$ for connectivity.
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent sketches \{M$_1$A$_G$, M$_2$A$_G$, ... M$_k$A$_G$\} for connectivity.

**Run Algorithm in Sketch Space:**

- Use M$_1$A$_G$, to find a spanning forest $F_1$ of G
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent sketches $\{M_1A_G, M_2A_G, ... M_kA_G\}$ for connectivity.

**Run Algorithm in Sketch Space:**
- Use $M_1A_G$ to find a spanning forest $F_1$ of $G$
- Use $M_2A_G - M_2A_{F_1} = M^2(A_G - A_{F_1}) = M^2(A_{G-F_1})$ to find $F_2$
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent sketches $\{M_1A_G, M_2A_G, \ldots M_kA_G\}$ for connectivity.

**Run Algorithm in Sketch Space:**

- Use $M_1A_G$, to find a spanning forest $F_1$ of $G$
- Use $M_2A_G - M_2A_{F1} = M^2(A_G - A_{F1}) = M^2(A_{G-F1})$ to find $F_2$
- Use $M_3A_G - M_3A_{F1} - M_3A_{F2} = M^3(A_G - A_{F1-F2})$ to find $F_3$
Ingredient 2: Connectivity Sketches

**Sketch:** Simultaneously construct $k$ independent sketches $\{M_1A_G, M_2A_G, \ldots, M_kA_G\}$ for connectivity.

**Run Algorithm in Sketch Space:**

- Use $M_1A_G$ to find a spanning forest $F_1$ of $G$.
- Use $M_2A_G - M_2A_{F_1} = M_2(A_G - A_{F_1}) = M_2(A_{G-F_1})$ to find $F_2$.
- Use $M_3A_G - M_3A_{F_1} - M_3A_{F_2} = M_3(A_G - A_{F_1-F_2})$ to find $F_3$.
- etc.
A graph is **k-connected** if every cut has size $\geq k$.

**Thm:** Can check $k$-connectivity with $O(nk\log^3 n)$-size sketch.

**Extension:** There exists a $O(\varepsilon^{-2}n\log^4 n)$-size sketch with which we can approximate all cuts up to a factor $(1+\varepsilon)$.

**Original Graph**

**Sparsifier Graph**
Summary

• **Graph Sketches:** Initiates the study of linear projections that preserve structural properties of graphs. Application to dynamic-graph streams and are embarrassingly parallelizable.

• **Properties:** Connectivity, sparsifiers, spanners, bipartite, minimum spanning trees, small cliques, matchings, ...