Sublinear Algorithms for Big Data

Qin Zhang
Part 2: Sublinear in Communication
Sublinear in communication

The model

\[ x_1 = 010011 \]
\[ x_2 = 111011 \]
\[ x_3 = 111111 \]
\[ x_k = 100011 \]

They want to jointly compute \( f(x_1, x_2, \ldots, x_k) \)

Goal: minimize total bits of communication

Applications

e etc.
A natural approach

- **The model**

  \[ x_1 = 010011 \]
  \[ x_2 = 111011 \]
  \[ x_3 = 111111 \]
  \[ x_k = 100011 \]

  They want to jointly compute \( f(x_1, x_2, \ldots, x_k) \)
  
  Goal: minimize total bits of communication

- **The natural approach**

  Each \( S_i \) computes a sketch of its input \( sk(S_i) \) and send it to \( C \), and then \( C \) computes \( f(x_1, \ldots, x_k) \) based on \( sk(S_1), \ldots, sk(S_k) \)

  The slides from next page are borrowed from Andrew McGregor
I. Connectivity

II. $k$-Connectivity

III. Min-Cut
Theorem: Testing Connectivity

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

**Lemma:** After $O(\log n)$ rounds selected edges include spanning forest.
Ingredient 2: Sketching Neighborhoods
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 $j<i$.

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

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

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

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

$\begin{align*}
\mathbf{a}_1 &= (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\
\mathbf{a}_2 &= (-1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \\
\mathbf{a}_1 + \mathbf{a}_2 &= (0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0)
\end{align*}$
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 $S \subset V$, \[ \text{support} \left( \sum_{i \in S} a_i \right) = E(S, V \setminus S) \]
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 $j<i$.

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

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

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

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

\[
Ma \longrightarrow e \in \text{support}(a)
\]
Recipe: Sketch & Compute on Sketches
Recipe: Sketch & Compute on Sketches

Sketch: Each player sends $\text{Ma}_j$
Recipe: Sketch & Compute on Sketches

- Sketch: Each player sends $M_{aj}$
- Central Player Runs Algorithm in Sketch Space:
Recipe: Sketch & Compute on Sketches

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

Sketch: Each player sends $M_{aj}$

Central Player Runs Algorithm in Sketch Space:

- Use $M_{aj}$ to get incident edge on each node $j$
- For $i=2$ to $\log n$:
  - To get incident edge on component $S \subset V$ use:
Recipe: Sketch & Compute on Sketches

- **Sketch:** Each player sends $M_{a_j}$
- **Central Player Runs Algorithm in Sketch Space:**
  - Use $M_{a_j}$ to get incident edge on each node $j$
  - For $i=2$ to $\log n$:
    - To get incident edge on component $S \subset V$ use:
      \[
      \sum_{j \in S} M_{a_j} = M\left(\sum_{j \in S} a_j\right)
      \]
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 $\log n$:
  - To get incident edge on component $S \subseteq V$ use:

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

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

**Detail:** Actually each player sends $\log n$ indept sketches $M_{1\text{aj}}, M_{2\text{aj}}, \ldots$ and central player uses $M_{i\text{aj}}$ when emulating $i^{\text{th}}$ iteration of the algorithm.
I. Connectivity

II. $k$-Connectivity

III. Min-Cut
Theorem: Checking every cut has size $\geq k$

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

b) Simultaneous Messages: $O(k \text{ polylog } n)$ 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 $G(V,E)$
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 $G(V,E-F_1-...-F_{i-1})$
Ingredient 1: Basic Algorithm

Algorithm (k-Connectivity):

1. Let $F_1$ be a spanning forest of $G(V,E)$
2. For $i = 2$ to $k$:
   2.1. Let $F_i$ be a 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 $\{M_1G, M_2G, \ldots, M_kG\}$.

**Run Algorithm in Sketch Space:**

- Use $M_1G$ 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.
I. Connectivity

II. k-Connectivity

III. Min-Cut
I. Connectivity

II. \textit{k}-Connectivity

III. Min-Cut

\textbf{Theorem}: \((1 + \epsilon)\)-approximating minimum cut

- a) \textit{Dynamic Graph Stream}: \(O(\epsilon^{-2} n \ \text{polylog } n)\) space.
- b) \textit{Simultaneous Messages}: \(O(\epsilon^{-2} \ \text{polylog } n)\) length.
Ingredient 1: Subsampling
Ingredient 1: Subsampling

Lemma (Karger): Define subgraph $G_i$ by sampling edges w/p $2^{-i}$. Then

$$\text{Min-Cut}(G) = (1 \pm \epsilon) \cdot 2^i \cdot \text{Min-Cut}(G_i) \quad \text{if} \ i < -\log p^*$$
Ingredient 1: Subsampling

Lemma (Karger): Define subgraph $G_i$ by sampling edges w/p $2^{-i}$. Then

$$\text{Min-Cut}(G) = (1 \pm \epsilon) \cdot 2^i \cdot \text{Min-Cut}(G_i) \quad \text{if } i < -\log p^*$$

where $p^* = 6\epsilon^{-2} \log n/\text{Min-Cut}(G)$
Ingredient 1: Subsampling

Lemma (Karger): Define subgraph $G_i$ by sampling edges w/p $2^{-i}$. Then

$$\text{Min-Cut}(G) = (1 \pm \epsilon) \cdot 2^i \cdot \text{Min-Cut}(G_i) \quad \text{if } i < - \log p^*$$

where $p^* = 6\epsilon^{-2} \log \frac{n}{\text{Min-Cut}(G)}$
Lemma (Karger): Define subgraph $G_i$ by sampling edges w/p $2^{-i}$. Then

$\text{Min-Cut}(G) = (1 \pm \epsilon) \cdot 2^i \cdot \text{Min-Cut}(G_i)$ if $i < -\log p^*$

where $p^* = 6\epsilon^{-2} \log n/\text{Min-Cut}(G)$
Lemma (Karger): Define subgraph $G_i$ by sampling edges w/p $2^{-i}$. Then

$$\text{Min-Cut}(G) = (1 \pm \epsilon) \cdot 2^i \cdot \text{Min-Cut}(G_i) \quad \text{if} \quad i < -\log p^*$$

where $p^* = 6\epsilon^{-2} \log n / \text{Min-Cut}(G)$


Lemma (Karger): Define subgraph $G_i$ by sampling edges w/p $2^{-i}$. Then

$$\text{Min-Cut}(G) = (1 \pm \epsilon) \cdot 2^i \cdot \text{Min-Cut}(G_i) \quad \text{if } i < -\log p^*$$

where $p^* = 6\epsilon^{-2} \log n / \text{Min-Cut}(G)$
Ingredient 1: Subsampling

Lemma (Karger): Define subgraph $G_i$ by sampling edges w/p $2^{-i}$. Then

$$\text{Min-Cut}(G) = (1 \pm \epsilon) \cdot 2^i \cdot \text{Min-Cut}(G_i) \quad \text{if} \quad i < - \log p^*$$

where $p^* = 6\epsilon^{-2} \log n / \text{Min-Cut}(G)$

G = G_0

G_1

G_2

G_3

Suffices to find $\text{Min-Cut}(G_i)$ for some $i < - \log p^*$. 
Ingredient 2: k-Connectivity
Ingredient 2: $k$-Connectivity

$k$-Connectivity: Given $G_i$ returns subgraph $H_i$ with

$$\text{Min-Cut}(G_i) = \text{Min-Cut}(H_i) \quad \text{if} \quad \text{Min-Cut}(G_i) < k$$
**Ingredient 2: \( k \)-Connectivity**

\( k \)-Connectivity: Given \( G_i \) returns subgraph \( H_i \) with

\[
\text{Min-Cut}(G_i) = \text{Min-Cut}(H_i) \quad \text{if} \quad \text{Min-Cut}(G_i) < k
\]

**Lemma:** For \( k = 12\varepsilon^{-2} \log n \), with high probability

\[
\text{Min-Cut}(G_i) < k \quad \text{for} \quad i = -\log p^*
\]
**Ingredient 2: k-Connectivity**

- **k-Connectivity:** Given $G_i$ returns subgraph $H_i$ with
  \[ \text{Min-Cut}(G_i) = \text{Min-Cut}(H_i) \quad \text{if} \quad \text{Min-Cut}(G_i) < k \]

- **Lemma:** For $k = 12\epsilon^{-2} \log n$, with high probability
  \[ \text{Min-Cut}(G_i) < k \quad \text{for} \quad i = -\log p^* \]

  since expectation of $\text{Min-Cut}(G_i)$ is $< 6\epsilon^{-2} \log n$. 
**Ingredient 2: \( k \)-Connectivity**

- **\( k \)-Connectivity:** Given \( G_i \) returns subgraph \( H_i \) with
  \[
  \text{Min-Cut}(G_i) = \text{Min-Cut}(H_i) \quad \text{if } \text{Min-Cut}(G_i) < k
  \]

- **Lemma:** For \( k = 12\varepsilon^{-2} \log n \), with high probability
  \[
  \text{Min-Cut}(G_i) < k \quad \text{for } i = -\log p^*
  \]
  since expectation of \( \text{Min-Cut}(G_i) \) is \(< 6\varepsilon^{-2} \log n \).

- **Putting it together:** Construct \( H_i \) for all \( i \). Return \( 2^i \text{Min-Cut}(H_i) \) for smallest \( i \) with \( \text{Min-Cut}(H_i) < k \).
Algorithm for Min-Cut

1. For $i = \{1, \ldots, 2 \log n\}$, let $h_i \rightarrow \{0, 1\}$ be a uniform hash function.

2. For $i = \{1, \ldots, 2 \log n\}$,
   (a) Let $G_i$ be the subgraph of $G$ containing edges $e$ such that $\prod_{j \leq i} h_j(e) = 1$.
   (b) Let $H_i \leftarrow k$-Connected($G_i$) for $k = O(\epsilon^{-2} \log n)$.

3. Return $2^i \cdot \text{Min-Cut}(H_j)$, where $j = \min\{i : \text{Min-Cut}(H_i) < k\}$
Example: Checking Bipartiteness
Example: Checking Bipartiteness

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

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

**Idea:** Given $G$, define $G'$ 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:** $G$ is bipartite iff number of connected components doubles. Can sketch $G'$ implicitly.
Example: Checking Bipartiteness

**Idea:** Given $G$, define $G'$ 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:** $G$ is bipartite iff number of connected components doubles. Can sketch $G'$ implicitly.
Example: Checking Bipartiteness

Idea: Given $G$, define $G'$ 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: $G$ is bipartite iff number of connected components doubles. Can sketch $G'$ implicitly.

Thm: $\tilde{O}(n)$-dimensional sketch for bipartiteness.
Example: Minimum Spanning Tree
Example: Minimum Spanning Tree

**Idea:** Let $n_i$ be number of connected components if we ignore edges with weight $\geq (1 + \epsilon)^i$, then:

$$w(MST) \leq \sum_{i} \epsilon (1 + \epsilon)^i n_i \leq (1 + \epsilon) w(MST)$$
Example: Minimum Spanning Tree

**Idea:** Let $n_i$ be the number of connected components if we ignore edges with weight $\geq (1 + \epsilon)^i$, then:

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

**Thm:** Can $(1 + \epsilon)$ approximate MST in one-pass dynamic semi-streaming model.
Algorithm for Sparsification

\( \lambda_e(G) \): size of the minimum cut for each edge \( e = (u, v) \) in \( G \)

1. For \( i = \{1, \ldots, 2 \log n\} \), let \( h_i \rightarrow \{0, 1\} \) be a uniform hash function.

2. For \( i = \{1, \ldots, 2 \log n\} \),
   (a) Let \( G_i \) be the subgraph of \( G \) containing edges \( e \) such that \( \prod_{j \leq i} h_j(e) = 1 \).
   (b) Let \( H_i \leftarrow k\text{-Connected}(G_i) \) for \( k = O(\epsilon^{-2} \log^2 n) \).

3. For each edge \( e = (u, v) \), find \( j = \min\{i : \lambda_e(H_i) < k\} \). If \( e \in H_j \), add \( e \) to the sparsifier with weight \( 2^j \).

**Azuma’s inequality** A sequence of random variables \( X_1, X_2, \ldots \) is called a martingale is for all \( i \geq 1 \), \( E[X_{i+1}|X_i] = X_i \). If \( |X_{i+1} - X_i| \leq c_i \) almost surely for all \( i \), then

\[
\Pr[|X_n - X_1| \geq t] < 2e^{-\frac{t^2}{2 \sum_{i=1}^{n-1} c_i^2}}.
\]