Sublinear Algorithms for Big Data

Qin Zhang
Part 2: Sublinear in Communication
Sublinear in communication

- **The model**
  
  \[ x_1 = 010011 \quad x_2 = 111011 \quad x_3 = 111111 \quad x_k = 100011 \]

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

  Goal: minimize total bits of communication

- **Applications**

  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
Algorithm for $L_0$ sampling

Goal: sample an element from the support of $a \in \mathbb{R}^n$
Algorithm for $L_0$ sampling

**Goal**: sample an element from the support of $a \in \mathbb{R}^n$

**Algorithm** (output can be thought as $Ma$ for a fixed matrix $M$)

- Maintain $\tilde{F}_0$, an $(1 \pm 0.1)$-approximation to $F_0$.
- Hash items using $h_j : [n] \rightarrow [0, 2^j - 1]$ for $j \in [\log n]$.
- For each $j$, maintain:
  - $D_j = (1 \pm 0.1) |\{ t \mid h_j(t) = 0\}|$
  - $S_j = \sum_{t, h_j(t) = 0} (f_t \cdot a_t)$
  - $C_j = \sum_{t, h_j(t) = 0} f_t$
Algorithm for $L_0$ sampling

**Goal**: sample an element from the support of $a \in \mathbb{R}^n$

**Algorithm** (output can be thought as $Ma$ for a fixed matrix $M$)

- Maintain $\tilde{F}_0$, an $(1 \pm 0.1)$-approximation to $F_0$.
- Hash items using $h_j : [n] \to [0, 2^j - 1]$ for $j \in [\log n]$.
- For each $j$, maintain:
  - $D_j = (1 \pm 0.1) |\{t \mid h_j(t) = 0\}|$
  - $S_j = \sum_{t,h_j(t)=0} (f_t \cdot a_t)$
  - $C_j = \sum_{t,h_j(t)=0} f_t$

**Lemma**

At level $j = 2 + \lceil \log \tilde{F}_0 \rceil$, there is a *unique* element in the stream that maps to 0 with constant probability.
Algorithm for $L_0$ sampling

**Goal:** sample an element from the support of $a \in \mathbb{R}^n$

**Algorithm** (output can be thought as $Ma$ for a fixed matrix $M$)

- Maintain $\tilde{F}_0$, an $(1 \pm 0.1)$-approximation to $F_0$.
- Hash items using $h_j : [n] \to [0, 2^j - 1]$ for $j \in [\log n]$.
- For each $j$, maintain:
  - $D_j = (1 \pm 0.1) \left| \{ t \mid h_j(t) = 0 \} \right|$  
  - $S_j = \sum_{t, h_j(t) = 0} (f_t \cdot a_t)$  
  - $C_j = \sum_{t, h_j(t) = 0} f_t$

**Lemma**

At level $j = 2 + \lceil \log \tilde{F}_0 \rceil$, there is a *unique* element in the stream that maps to 0 with constant probability.

Uniqueness is verified if $D_j = 1 \pm 0.1$. If unique, then $S_j / C_j$ gives identity of the element and $C_j$ is the count.

**Analysis:** (on the board)
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
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 $j<i$.

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

$$a_2 = \begin{pmatrix} \{1,2\} & \{1,3\} & \{1,4\} & \{1,5\} & \{2,3\} & \{2,4\} & \{2,5\} & \{3,4\} & \{3,5\} & \{4,5\} \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$
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 $j<i$.

\[
\begin{align*}
    a_1 &= (1 1 0 0 0 0 0 0 0 0) \\
    a_2 &= (-1 0 0 0 1 0 0 0 0 0)
\end{align*}
\]
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 $j<i$.

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

$$a_1 = (1 1 0 0 0 0 0 0 0 0 0)$$

$$a_2 = (-1 0 0 0 1 0 0 0 0 0 0)$$

$$a_1 + a_2 = (0 1 0 0 1 0 0 0 0 0 0)$$

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

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

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

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

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

**Sketch:** Each player sends $M_{aj}$
Recipe: Sketch & Compute on Sketches

- **Sketch**: Each player sends $\text{Maj}$
- **Central Player Runs Algorithm in Sketch Space**:
Recipe: Sketch & Compute on Sketches

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

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

\[
\sum_{j \in S} M_a_j = M(\sum_{j \in S} a_j)
\]
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:

\[
\sum_{j \in S} M_{aj} = M(\sum_{j \in S} a_j) \quad \longrightarrow \quad e \in \text{support}(\sum_{j \in S} a_j) = E(S, V \setminus S)
\]
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 $S \subset V$ use:

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

**Detail:** Actually each player sends log $n$ indept sketches $M_1a_j, M_2a_j, \ldots$ and central player uses $M_ia_j$ when emulating $i^{th}$ 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):
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 $G(V,E-F_1-\ldots-F_{i-1})$
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 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 $F_2$
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  

**Theorem:** $(1+\epsilon)$-approximating minimum cut  

a) *Dynamic Graph Stream:* $O(\epsilon^{-2} n \text{ polylog } n)$ space.  

b) *Simultaneous Messages:* $O(\epsilon^{-2} \text{ polylog } n)$ length.
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 $p^* = 6\epsilon^{-2} \log n/\text{Min-Cut}(G)$
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 $p^* = 6\epsilon^{-2} \log n / \text{Min-Cut}(G)$
**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 $p^* = 6\epsilon^{-2} \log n / \text{Min-Cut}(G)$
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 $p^* = 6\epsilon^{-2}\log n/\text{Min-Cut}(G)$
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 $p^* = 6\epsilon^{-2} \log n / \text{Min-Cut}(G)$

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

Ingredient 1: Subsampling
Ingredient 2: \textit{k-Connectivity}
Ingredient 2: \textit{k-Connectivity}

\textit{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 \( k = 12\epsilon^{-2} \log n \), 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
  \[
  \text{Min-Cut}(G_i) = \text{Min-Cut}(H_i) \quad \text{if} \quad \text{Min-Cut}(G_i) < k
  \]

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

- **Putting it together:** Construct \( H_i \) for all \( i \). Return
  \[
  2^i \text{Min-Cut}(H_i) \quad \text{for smallest} \quad i \quad \text{with} \quad \text{Min-Cut}(H_i) < k.
  \]
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 $j = \min \{ i : \text{Min-Cut}(H_i) < k \}$
Algorithm for Sparsification

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 $2^j$. 