Streaming Model of Computation

A streaming algorithm processes a data stream $S$:

- Input is presented as a sequence of items and can be examined in only a few passes (typically just one).
- The algorithm has limited memory and cannot store the whole input sequence.
- The algorithm can spend limited processing time per item.
- In some problems we are satisfied with an approximate answer.
- Approximation algorithms can be based on sketches (summaries) of the data stream in memory.
Streaming Graph Algorithms

In many applications we deal with massive graphs. E.g. (vertices – edges):

- Web-pages – hyperlinks
- Neurons – synapses
- IP addresses – network flows
- People – friendships

Processing such graphs with a classic graph algorithm may be infeasible!

But it may be possible to use an algorithm developed for the data stream model.
Streaming Graph Algorithms

In many applications we deal with massive graphs. E.g. (vertices – edges):

• Web-pages – hyperlinks
• Neurons – synapses
• IP addresses – network flows
• People – friendships

Processing such graphs with a classic graph algorithm may be infeasible!

But it may be possible to use an algorithm developed for the data stream model.

Presentation based on:
A. McGregor “Graph Stream Algorithms: A Survey” [ACM SIGMOD Record 2014]
The Internet

Graph from Albert-László Barabási’s SIGIR09 keynote
Streaming Graph Algorithms

Data stream model

- The input is given by a stream of data. E.g., the stream could be the graph edges.
- The algorithm can use a limited amount of memory to process the stream.
- The input stream must be processed in the order it arrives.

Related goals:

- Real-time systems.
- I/O efficiency.
- Trade-off size and accuracy.
Streaming Graph Algorithms

Data stream model

How much memory should our model allow in order to be able to process a graph with $n$ vertices?

• Most problems are intractable if space is $< n$.

• We will work in the **semi-streaming model** that allows $O(n \log^k n)$ memory, for some constant $k$.

• Some algorithms will be randomized. We will say that an event $E$ occurs **with high probability** if $\Pr[E] \geq 1 - 1/n$. 
Streaming Graph Algorithms

Graph connectivity

Data stream $S$: Edges of a graph $G = (V, E)$ with $n = |V|$

E.g., data stream $S = (1,2), (2,3), (1,3), (3,4)$
Graph connectivity

Data stream $S$: Edges of a graph $G = (V, E)$ with $n = |V|$

The goal is to test if $G$ is connected, i.e., for any two vertices there is a path that connects them.
Streaming Graph Algorithms

Graph connectivity

Data stream $S$: Edges of a graph $G = (V, E)$ with $n = |V|$

The goal is to test if $G$ is connected, i.e., for any two vertices there is a path that connects them.

Simple algorithm: Maintain a set of edges $H$. When we read the next edge $(u, v)$ from the stream, we add it to $H$ if there is currently no path between $u$ and $v$. 
Streaming Graph Algorithms

Spanners

\( a \)-spanner \( H \) of a graph \( G = (V, E) \): subgraph of \( G \) such that for all pairs \( u, v \in V \),

\[
d_G(u, v) \leq d_H(u, v) \leq a \cdot d_G(u, v)
\]

\( d_G(u, v) \) = length of the shortest path between \( u \) and \( v \) in \( G \)

\( d_H(u, v) \) = length of the shortest path between \( u \) and \( v \) in \( H \)
**Streaming Graph Algorithms**

**Spanners**

An $a$-spanner $H$ of a graph $G = (V, E)$: subgraph of $G$ such that for all pairs $u, v \in V$,

\[ d_G(u, v) \leq d_H(u, v) \leq a \cdot d_G(u, v) \]

$d_G(u, v) = \text{length of the shortest path between } u \text{ and } v \text{ in } G$

$d_H(u, v) = \text{length of the shortest path between } u \text{ and } v \text{ in } H$
Streaming Graph Algorithms

Spanners

Construction of an \( a \)-spanner \( H \): add next edge \((u, v)\) if it does not create a short cycle in \( H \)

Greedy Spanner Algorithm

1. \( H \leftarrow \emptyset \)
2. \textbf{for} each edge \((u, v) \in S\) \textbf{do}
3. \textbf{if} \( d_H(u, v) > a \) \textbf{then} add \((u, v)\) to \( H \)
4. \textbf{return} \( H \)

• Does this work?
• What is the size (\#edges) of the spanner?
Streaming Graph Algorithms

Spanners

Proof that the Greedy Spanner Algorithm works:

For any edge \((x, y)\) of \(G\) we have \(d_H(x, y) \leq a\).

Consider a path \(P = (v_0, v_1, \ldots, v_{k-1}, v_k)\) in \(G\)

\[
\begin{align*}
    d_H(v_0, v_1) &\leq a \\
    d_H(v_{k-1}, v_k) &\leq a
\end{align*}
\]

Length of \(P\) in \(G = k = d_G(v_0, v_1) + d_G(v_1, v_2) + \cdots + d_G(v_{k-1}, v_k)\)
Streaming Graph Algorithms

Spanners

Proof that the Greedy Spanner Algorithm works:

For any edge \((x, y)\) of \(G\) we have \(d_H(x, y) \leq a\).

Consider a path \(P = (v_0, v_1, \ldots, v_{k-1}, v_k)\) in \(G\)

\[
\begin{array}{c}
P \\
\quad v_0 \rightarrow v_1 \rightarrow v_{k-1} \rightarrow v_k
\end{array}
\]

\[
d_H(v_0, v_1) \leq a \\
d_H(v_{k-1}, v_k) \leq a
\]

Length of \(P\) in \(G\) = \(k = d_G(v_0, v_1) + d_G(v_1, v_2) + \cdots + d_G(v_{k-1}, v_k)\)

Length in \(H\) \(\leq d_H(v_0, v_1) + d_H(v_1, v_2) + \cdots + d_H(v_{k-1}, v_k)\)

\[
\leq a \cdot d_G(v_0, v_1) + a \cdot d_G(v_1, v_2) + \cdots + a \cdot d_G(v_{k-1}, v_k) = a \cdot k
\]
Spanners

How many edges are inserted into $H$?

- Let $a = 2t - 1$, for some integer $t$.
- Then $H$ does not contain cycles of length $< 2t$.
- By a known result in Graph Theory, any such graph has at most $O(n^{1+1/t})$ edges.
Streaming Graph Algorithms

Minimum Spanning Tree

Data stream $S$: Edges of a weighted graph $G = (V, E, w)$ with $n = |V|$

Construction: if next edge $(u, v)$ creates a cycle $C$ in $H$, delete from $H$ the maximum weight edge of $C$.

Greedy MST Algorithm

1. $H \leftarrow \emptyset$
2. for each edge $e = (u, v) \in S$ do
3. if $e$ creates a cycle $C$ in $H$ then
4. find the maximum weight edge $f \in C$
5. add $e$ to $H$
6. delete $f$ from $H$
7. return $H$
Streaming Graph Algorithms

Graph Sparsification

Given a graph \( G = (V, E) \) we want to construct a **weighted subgraph** \( H = (V, E_H, w) \) of \( G \) that estimates various (connectivity) properties of \( G \)

E.g.:

- Cut sparsification [Benczur-Karger]

- Spectral sparsification [Spielman-Teng]
Streaming Graph Algorithms

Sample Application

1TB

Sparsify

10MB

Output

Approx Output

Picture from https://simons.berkeley.edu/sites/default/files/docs/1768/slidessrivastava1.pdf
Cuts in Graphs

Weighted graph $G = (V, E, w)$. Edge weights $w : E \to R$

$A$-cut: partition of $V$ into two sets $A$ and $V \setminus A$

$\delta_G(A) =$ set of edges in $G$ crossing the $A$-cut. 
$\delta_G(A) = \{(u, v) \in E : u \in A, v \in V \setminus A\}$

Size of $A$-cut in $G$: 
$\lambda_A(G) = \sum_{e \in \delta_G(A)} w(e)$
Streaming Graph Algorithms

Cuts in Graphs

Weighted graph $G = (V, E, w)$. Edge weights $w : E \rightarrow R$

$A$-cut: partition of $V$ into two sets $A$ and $V \setminus A$

$\delta_G(A) = \text{set of edges in } G \text{ crossing the } A\text{-cut. } \delta_G(A) = \{(u, v) \in E : u \in A, v \in V \setminus A\}$

Size of $A$-cut in $G$: $\lambda_A(G) = \sum_{e \in \delta_G(A)} w(e)$
Streaming Graph Algorithms

Cuts in Graphs

Weighted graph \( G = (V, E, w) \). Edge weights \( w : E \rightarrow R \)

\( A \)-cut: partition of \( V \) into two sets \( A \) and \( V \setminus A \)

\( \delta_G(A) \) = set of edges in \( G \) crossing the \( A \)-cut. \( \delta_G(A) = \{(u, v) \in E : u \in A, v \in V \setminus A\} \)

Size of \( A \)-cut in \( G \): \( \lambda_A(G) = \sum_{e \in \delta_G(A)} w(e) \)

\[
\lambda_A(G) = w(a,f) + w(b,f) + w(c,f) + w(c,e) + w(c,d) = 17 + 10 + 9 + 7 + 20 = 63
\]
Cuts in Graphs

Weighted graph $G = (V, E, w)$. Edge weights $w : E \to R$

$A$-cut: partition of $V$ into two sets $A$ and $V \setminus A$

$\delta_G(A) = \text{set of edges in } G \text{ crossing the } A\text{-cut. } \delta_G(A) = \{(u, v) \in E : u \in A, v \in V \setminus A\}$

Size of $A$-cut in $G$: $\lambda_A(G) = \sum_{e \in \delta_G(A)} w(e)$

$\lambda_A(G) = w(b, c) + w(f, c) + w(f, e)$

$= 12 + 9 + 14 = 35$
Cut Sparsification

Given a graph $G = (V, E)$ we want to construct a **weighted subgraph** $H = (V, E_H, w)$ of $G$ that estimates the size of each cut of $G$

$(1 + \varepsilon)$ cut sparsification

$$(1 - \varepsilon) \cdot \lambda_A(G) \leq \lambda_A(H) \leq (1 + \varepsilon) \cdot \lambda_A(G)$$

for all vertex subsets $A \subset V$
Graph Laplacian

Weighted graph $G = (V, E, w)$. Edge weights $w : E \to R$

Laplacian of $G$: $n \times n$ real matrix $L_G$, $n = |V|$

$$L_G(i, j) = \begin{cases} \sum_{(i,k) \in E} w(i, k), & i = j \\ -w(i, j), & i \neq j \end{cases}$$

where $w(i, j) = 0$ if $(i, j) \notin E$
Streaming Graph Algorithms

Graph Laplacian

Weighted graph $G = (V, E, w)$. Edge weights $w : E \to R$

Laplacian of $G$: $n \times n$ real matrix $L_G$, $n = |V|

$$
L_G(i, j) = \begin{cases} 
\sum_{(i,k) \in E} w(i, k), & i = j \\
-w(i, j), & i \neq j 
\end{cases}
$$

where $w(i, j) = 0$ if $(i, j) \notin E$

$$
L_G = \begin{pmatrix}
14 & -3 & -9 & -2 \\
-3 & 4 & -1 & 0 \\
-9 & -1 & 18 & -8 \\
-2 & 0 & -8 & 10
\end{pmatrix}
$$
Streaming Graph Algorithms

Graph Laplacian

Weighted graph $G = (V, E, w)$. Edge weights $w : E \to \mathbb{R}$

Laplacian of $G$: $n \times n$ real matrix $L_G$, $n = |V|$ 

$$L_G(i, j) = \begin{cases} \sum_{(i,k) \in E} w(i,k), & i = j \\ -w(i,j), & i \neq j \end{cases} \quad \text{where } w(i,j) = 0 \text{ if } (i,j) \notin E$$

Let $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ be a real vector in $\mathbb{R}^n$. Recall that $x^T = (x_1 \ \cdots \ \ x_n)$
Streaming Graph Algorithms

Graph Laplacian

Weighted graph \( G = (V, E, w) \). Edge weights \( w : E \rightarrow \mathbb{R} \)

Laplacian of \( G \): \( n \times n \) real matrix \( L_G \), \( n = |V| \)

\[
L_G(i, j) = \begin{cases} 
\sum_{(i,k)\in E} w(i,k), & i = j \\
-w(i,j), & i \neq j 
\end{cases}
\]

where \( w(i,j) = 0 \) if \((i,j) \notin E\)

Let \( x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \) be a real vector in \( \mathbb{R}^n \). Recall that \( x^T = (x_1 \cdots x_n) \)

Then

\[
x^T L_G x = \sum_{(i,j)\in E} w(i,j)(x_i - x_j)^2
\]
Streaming Graph Algorithms

Spectral Sparsification

Graph $G = (V, E)$

A weighted subgraph $H = (V, E_H, w)$ of $G$ is a $(1 + \varepsilon)$ spectral sparsifier of $G$ if

$$(1 - \varepsilon) \cdot x^T L_G x \leq x^T L_H x \leq (1 + \varepsilon) \cdot x^T L_G x$$

for all real vectors $x \in \mathbb{R}^n$
Streaming Graph Algorithms

Spectral Sparsification

Graph $G = (V, E)$

A weighted subgraph $H = (V, E_H, w)$ of $G$ is a $(1 + \varepsilon)$ spectral sparsifier of $G$ if

$$(1 - \varepsilon) \cdot x^T L_G x \leq x^T L_H x \leq (1 + \varepsilon) \cdot x^T L_G x$$

for all real vectors $x \in \mathbb{R}^n$

A spectral sparsifier of $G$ can approximate:

- Size of all cuts
- Eigenvalues
- Effective resistances (in the corresponding electrical network)
- Properties of random walks
Streaming Graph Algorithms

Spectral Sparsification

Graph $G = (V, E)$

A *weighted subgraph* $H = (V, E_H, w)$ of $G$ is a $(1 + \varepsilon)$ spectral sparsifier of $G$ if

$$(1 - \varepsilon) \cdot x^T L_G x \leq x^T L_H x \leq (1 + \varepsilon) \cdot x^T L_G x$$

for all real vectors $x \in \mathbb{R}^n$

**Theorem** [Spielman and Teng] A $(1 + \varepsilon)$ spectral sparsifier with $O(n \log n / \varepsilon^2)$ edges can be constructed in $O(m \text{polylog}(n)/\varepsilon^2)$, where $n$ is the number of vertices and $m$ is the number of edges of the input graph.
Streaming Graph Algorithms

Spectral Sparsification

Graph \( G = (V, E) \)

A weighted subgraph \( H = (V, E_H, w) \) of \( G \) is a \((1 + \varepsilon)\) spectral sparsifier of \( G \) if

\[
(1 - \varepsilon) \cdot x^T L_G x \leq x^T L_H x \leq (1 + \varepsilon) \cdot x^T L_G x
\]

for all real vectors \( x \in \mathbb{R}^n \)

**Theorem** [Spielman and Teng] A \((1 + \varepsilon)\) spectral sparsifier with \( O(n \log n / \varepsilon^2) \) edges can be constructed in \( O(m \text{ polylog}(n) / \varepsilon^2) \), where \( n \) is the number of vertices and \( m \) is the number of edges of the input graph.

**Theorem** [Batson, Spielman and Srivastava] A graph with \( n \) vertices has a \((1 + \varepsilon)\) spectral sparsifier with \( O(n / \varepsilon^2) \) edges.
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

• Use as a black box any existing algorithm ALG that returns a $(1 + \gamma)$ spectral sparsifier.

• ALG returns a spectral sparsifier with $size(\gamma) = O(n/\gamma^2)$ number of edges.
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

• Use as a black box any existing algorithm ALG that returns a \((1 + \gamma)\) spectral sparsifier.

• ALG returns a spectral sparsifier with \(\text{size}(\gamma) = O(n/\gamma^2)\) number of edges.

We use the following properties of spectral sparsification

• **Mergeable:** Suppose \(H_1\) and \(H_2\) are \(\beta\) spectral sparsifiers of two graphs \(G_1\) and \(G_2\) on the same set of vertices. Then \(H_1 \cup H_2\) is a \(\beta\) spectral sparsifier of \(G_1 \cup G_2\).

• **Composable:** If \(H_3\) is a \(\beta\) spectral sparsifier for \(H_2\) and \(H_2\) is a \(\delta\) spectral sparsifier for \(H_1\) then \(H_3\) is a \(\beta\delta\) spectral sparsifier for \(H_1\).
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

Let $G = (V, E)$ be the input graph with $n = |V|$ and $m = |E|$

Data stream $S$ = the $m$ edges of $G$

Set $t = m/\text{size}(\gamma)$. For simplicity assume that $t$ is a power of 2

We divide $S$ into $t$ segments of $\text{size}(\gamma)$ edges

$G_i^0 = \text{graph that consists of the edges in the } i\text{-th segment}$

<table>
<thead>
<tr>
<th></th>
<th>$\text{size}(\gamma)$</th>
<th>$\text{size}(\gamma)$</th>
<th>...</th>
<th>$\text{size}(\gamma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$G_1^0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$G_2^0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$G_t^0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

Set $t = \frac{m}{\text{size}(\gamma)}$. For simplicity assume that $t$ is a power of 2 ($t = 2^k$, $k = \lg t$)

We divide $S$ into $t$ segments of $\text{size}(\gamma)$ edges

$G_i^0 = \text{graph that consists of the edges in the } i\text{-th segment}$

For $i = 1, 2, \ldots, \lg t$ and $j = 1, 2, \ldots, t/2^i$ define $G_i^j = G_{2i-1}^{j-1} \cup G_{2i}^{j-1}$

E.g., for $t = 4$
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

Set $t = m/\text{size}(\gamma)$. For simplicity assume that $t$ is a power of 2 ($t = 2^k$, $k = \lg t$)

We divide $S$ into $t$ segments of $\text{size}(\gamma)$ edges

$G_i^0 =$ graph that consists of the edges in the $i$-th segment

For $i = 1, 2, \ldots, \lfloor \lg t \rfloor$ and $j = 1, 2, \ldots, t/2^i$ define $G_i^j = G_{2i-1}^{j-1} \cup G_{2i}^{j-1}$

For each $G_i^j$ define a weighted subgraph $H_i^j$:

- $H_i^0 = G_i^0$
- $H_i^j = \text{ALG}(H_{2i-1}^{j-1} \cup H_{2i}^{j-1})$, $j > 0$
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

Set $t = m/\text{size}(\gamma)$. For simplicity assume that $t$ is a power of 2 ($t = 2^k, \ k = \lg t$)

We divide $S$ into $t$ segments of $\text{size}(\gamma)$ edges

$G_i^0 = \text{graph that consists of the edges in the } i\text{-th segment}$

For $i = 1, 2, ..., \lg t$ and $j = 1, 2, ..., t/2^i$ define $G_i^j = G_{2i-1}^{j-1} \cup G_{2i}^{j-1}$

For each $G_i^j$ define a weighted subgraph $H_i^j$:

- $H_i^0 = G_i^0$
- $H_i^j = \text{ALG}(H_{2i-1}^{j-1} \cup H_{2i}^{j-1}), \ j > 0$

By the mergeable and composable properties $H_1^{\lg t}$ is a $(1 + \gamma)^{\lg t}$ sparsifier of $G$
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

By the mergeable and composable properties $H_{1}^{\lg t}$ is a $(1 + \gamma)^{\lg t}$ sparsifier of $G$

Set $\gamma = \frac{\varepsilon}{(2 \lg t)} \Rightarrow (1 + \gamma)^{\lg t} \sim (1 + \varepsilon)$

Then $H_{1}^{\lg t}$ is a $(1 + \varepsilon)$ sparsifier of $G$
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

By the mergeable and composable properties $H_{1}^{\lg t}$ is a $(1 + \gamma)^{\lg t}$ sparsifier of $G$

Set $\gamma = \varepsilon / (2 \lg t) \Rightarrow (1 + \gamma)^{\lg t} \sim (1 + \varepsilon)$

Then $H_{1}^{\lg t}$ is a $(1 + \varepsilon)$ sparsifier of $G$

Space required

Delete $H_{2i-1}^{j-1}$ and $H_{2i}^{j-1}$ as soon as $H_{i}^{j}$ is computed

For each $j$ we need to store $H_{i}^{j}$ only for two values of $i$
Streaming Graph Algorithms

Spectral Sparsification – Construction in the semi-streaming model

By the mergeable and composable properties $H_1^{\lg t}$ is a $(1 + \gamma)^{\lg t}$ sparsifier of $G$

Set $\gamma = \varepsilon/(2 \lg t) \Rightarrow (1 + \gamma)^{\lg t} \sim (1 + \varepsilon)$

Then $H_1^{\lg t}$ is a $(1 + \varepsilon)$ sparsifier of $G$

Space required

\[ H_i^j = \text{ALG}(H_{2i-1}^{j-1} \cup H_{2i}^{j-1}) \]

Delete $H_{2i-1}^{j-1}$ and $H_{2i}^{j-1}$ as soon as $H_i^j$ is computed

For each $j$ we need to store $H_i^j$

only for two values of $i$

So at any given time we need to store $\leq 2 \cdot \text{size} (\gamma) \cdot \lg t = \mathcal{O}(n \lg^3 n/\varepsilon^2)$
Streaming Graph Algorithms

Matchings

Graph $G = (V, E)$

Matching: Subset of edges $M \subseteq E$ such that each vertex is adjacent to at most one edge in $M$

Goal: Find a maximum cardinality matching $M^*$
Streaming Graph Algorithms

Matchings

Graph $G = (V, E)$

Matching: Subset of edges $M \subseteq E$ such that each vertex is adjacent to at most one edge in $M$

Goal: Find a maximum cardinality matching $M^*$
Streaming Graph Algorithms

Matchings

Graph $G = (V, E)$

Matching: Subset of edges $M \subseteq E$ such that each vertex is adjacent to at most one edge in $M$

Goal: Find a maximum cardinality matching $M^*$

Greedy Matching Algorithm

1. $M \leftarrow \emptyset$
2. for each edge $e \in S$ do
3.   if $M \cup \{e\}$ is a matching then add $e$ to $M$
4. return $M$
Matchings

Graph $G = (V, E)$

Matching: Subset of edges $M \subseteq E$ such that each vertex is adjacent to at most one edge in $M$

Goal: Find a maximum cardinality matching $M^*$

The Greedy Matching Algorithm computes a matching $M$ with cardinality $|M| \geq |M^*|/2$
Streaming Graph Algorithms

Matchings

Graph $G = (V, E)$

Matching: Subset of edges $M \subseteq E$ such that each vertex is adjacent to at most one edge in $M$

Goal: Find a maximum cardinality matching $M^*$

The Greedy Matching Algorithm computes a matching $M$ with cardinality $|M| \geq |M^*|/2$

Consider an edge $(u, v) \in M^*$

If $(u, v) \not\in M$ then $M$ must contain at least one edge $e$ adjacent to $u$ or to $v$

$e$ is adjacent to at most 2 edges of $M^*$
Streaming Graph Algorithms

Weighted Matchings

Weighted graph $G = (V, E, w)$. Edge weights $w : E \rightarrow \mathbb{R}^+ \ (w(e) > 0, \forall e \in E)$

Goal: Find a maximum weight matching $M^*$

As before, we process the edges of the stream $S$ as they arrive and try to augment the current matching $M$
**Streaming Graph Algorithms**

**Weighted Matchings**

Weighted graph $G = (V, E, w)$. Edge weights $w : E \rightarrow \mathbb{R}^+ \ (w(e) > 0, \forall e \in E)$

Goal: Find a maximum weight matching $M^*$

As before, we process the edges of the stream $S$ as they arrive and try to augment the current matching $M$

Let $e$ be the next edge read from $S$. Let $C$ be the edges of $M$ that are in conflict with $e$ : and edge in $C$ and $e$ are adjacent to a common vertex.
Streaming Graph Algorithms

Weighted Matchings

Weighted graph $G = (V, E, w)$. Edge weights $w : E \to \mathbb{R}^+$ ($w(e) > 0$, $\forall e \in E$)

Goal: Find a maximum weight matching $M^*$

As before, we process the edges of the stream $S$ as they arrive and try to augment the current matching $M$

Let $e$ be the next edge read from $S$. Let $C$ be the edges of $M$ that are in conflict with $e$ : and edge in $C$ and $e$ are adjacent to a common vertex.

$C$ has at most two edges. Let $w(C)$ be the total weight of the edges in $C$.

If $w(e) > w(C)$ then we increase the weight of $M$ by including $e$ and deleting the edges of $C$. 
Streaming Graph Algorithms

Weighted Matchings

Let $e$ be the next edge read from $S$. Let $C$ be the edges of $M$ that are in conflict with $e$: and edge in $C$ and $e$ are adjacent to a common vertex.

$$w(C) = \text{total weight of the edges in } C.$$ 

If $w(e) > w(C)$ then we increase the weight of $M$ by including $e$ and deleting the edges of $C$.

Greedy Weighted Matching Algorithm

1. $M \leftarrow \emptyset$
2. for each edge $e \in S$ do
3. let $C$ be the set of edges that are in conflict with $e$
4. if $w(e) > w(C)$ then add $e$ to $M$ and delete $C$ from $M$
5. return $M$
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i+1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{ \} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n - 1) \]

Edge \( e_i = (i, i + 1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(1,2)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i + 1) \) has weight \( w(e_i) = 1 + i\epsilon \), for a small \( \epsilon > 0 \)

\[ M = \{(1,2)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n - 1) \]

Edge \( e_i = (i, i + 1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(2,3)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i+1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(2,3)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i + 1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(3,4)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i+1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(3,4)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i+1) \) has weight \( w(e_i) = 1 + i\epsilon \), for a small \( \epsilon > 0 \)

\[ M = \{(3,4)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n,n-1) \]

Edge \( e_i = (i, i+1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(4,5)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i + 1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(4,5)\} \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i+1) \) has weight \( w(e_i) = 1 + i \varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(5,6)\} \quad w(M) = 1 + 5 \varepsilon \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i+1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

\[ M = \{(5,6)\} \]
\[ w(M) = 1 + 5\varepsilon \]

\[ M^* = \{(1,2), (3,4), (5,6)\} \]
\[ w(M^*) = 3 + 9\varepsilon \]
Streaming Graph Algorithms

Weighted Matchings

Consider the following scenario

\[ S = (1,2), (2,3), (3,4), \ldots, (n, n-1) \]

Edge \( e_i = (i, i + 1) \) has weight \( w(e_i) = 1 + i\varepsilon \), for a small \( \varepsilon > 0 \)

The computed matching \( M \) has weight \( w(M) = 1 + (n - 1)\varepsilon \)

The optimal matching \( M^* \) has weight \( w(M^*) = \sum_i (1 + (2i - 1)\varepsilon) > (n - 1)/2 \)

Hence, the approximation ratio is \( \frac{w(M^*)}{w(M)} > \frac{(n - 1)/2}{1 + (n - 1)\varepsilon} \sim \frac{n}{2} \)
Streaming Graph Algorithms

Weighted Matchings

The problem is that the trailing edges of $S$ that were once inserted into $M$ but removed later may have much larger total weight than the edges added later.
Streaming Graph Algorithms

**Weighted Matchings**

Modified algorithm

We include $e$ in $M$ if $w(e) > \beta w(C)$ for some constant $\beta = (1 + \gamma) > 1$.

**Greedy Weighted Matching Algorithm**

1. $M \leftarrow \emptyset$
2. for each edge $e \in S$ do
3. let $C$ be the set of edges that are in conflict with $e$
4. if $w(e) > (1 + \gamma) \cdot w(C)$ then add $e$ to $M$ and delete $C$ from $M$
5. return $M$
The problem is that the trailing edges of $S$ that were once inserted into $M$ but removed later may have much larger total weight than the edges added later.

We include $e$ in $M$ if $w(e) > \beta w(C)$ for some constant $\beta = (1 + \gamma) > 1$.

For an edge $e$ define

- $C_0 = \{e\}$
- $C_i = $ edges removed when an edge in $C_{i-1}$ was added to $M$
- $T_e = C_1 \cup C_2 \cup \ldots$

Then $w(T_e) \leq w(e)/\gamma$
It can be shown that

\[ w(M^*) \leq (1 + \gamma) \cdot \sum_{e \in M} (w(T_e) + 2w(e)) \]

By applying a careful charging scheme we get \( \frac{w(M^*)}{w(M)} < 5.828 \).
We can get a $(2 + \varepsilon)$-approximation with $O(\varepsilon^{-3})$ passes over $S$, where $\gamma = O(\varepsilon)$.
Streaming Graph Algorithms

Graph Sketches

Random linear projection $M : \mathbb{R}^n \rightarrow \mathbb{R}^k$, where $k \ll n$

For any vector $\mathbf{v} \in \mathbb{R}^n$, the projection $M\mathbf{v} \in \mathbb{R}^k$ preserves properties of $\mathbf{v}$ with high probability

Many applications: estimating entropy, heavy hitters, estimating norms, fitting polynomials,…

Rich theory: dimensionality reduction, sparse recovery, metric embeddings,…
Streaming Graph Algorithms

Graph Sketches

Can we use this approach for graphs?

That is, can we project the adjacency matrix $A_G$ of a graph $G$ to a smaller matrix $MA_G$, so that we can use $MA_G$ to compute properties of $G$?

- For a graph $G$ with $n$ vertices, $A_G$ has $O(n^2)$ dimensions.
- To work in the semi-streaming model we want $MA_G$ to have $O(n \text{ polylog}(n))$ dimensions.
Streaming Graph Algorithms

Graph Sketches

Picture from https://people.cs.umass.edu/~mcgregor/711S12/lec-2-2.pdf
Streaming Graph Algorithms

Graph Sketches

Dynamic graph stream $S = \langle a_1, a_2, ... \rangle$ where $a_i = (e_i, \Delta_i)$

$e_i = \text{an edge of the graph}$

$\Delta_i = \begin{cases} 
+1, & \text{if } e_i \text{ is inserted} \\
-1, & \text{if } e_i \text{ is deleted}
\end{cases}$

Multiplicity of edge $e$ : $f_e = \sum_{i: e_i = e} \Delta_i$

For simplicity we will assume that $f_e \in \{0, 1\}$, for all edges $e$. 
Streaming Graph Algorithms

Graph Sketches

Vector of edge multiplicities $f \in \{0,1\}^{\binom{n}{2}}$

Each entry of $f$ is a multiplicity $f_e$ of a (potential) edge $e$ of $G$ (a simple graph with $n$ vertices has up to $\binom{n}{2}$ edges).

$$f = \begin{pmatrix} f_{e_{12}} \\ f_{e_{13}} \\ f_{e_{23}} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Graph:
- Vertices: 1, 2, 3
- Edges: $e_{12}, e_{13}, e_{23}$
Streaming Graph Algorithms

Graph Sketches

Vector of edge multiplicities \( f \in \{0,1\}^{n\choose 2} \)

Each entry of \( f \) is a multiplicity \( f_e \) of a (potential) edge \( e \) of \( G \) (a simple graph with \( n \) vertices has up to \( n^2 \) edges).

Index vector of edge \( e : i^e \in \{0,1\}^{n\choose 2} \). The only nonzero entry of \( i^e \) is the one that corresponds to edge \( e \).

\[
\begin{pmatrix}
i_{e_{12}} \\
i_{e_{13}} \\
i_{e_{23}}
\end{pmatrix}
= \begin{pmatrix}0 \\ 0 \\ 1\end{pmatrix}
\]
Vector of edge multiplicities $f \in \{0,1\}^{\binom{n}{2}}$

Each entry of $f$ is a multiplicity $f_e$ of a (potential) edge $e$ of $G$ (a simple graph with $n$ vertices has up to $\binom{n}{2}$ edges).

Index vector of edge $e$ : $i^e \in \{0,1\}^{\binom{n}{2}}$. The only nonzero entry of $i^e$ is the one that corresponds to edge $e$.

Sketch of $f$ : $A(f) \in \mathbb{R}^d$, $d =$dimensionality of the sketch

When we read the next item $(e, \Delta)$ from the stream, we can update the sketch as follows:

$$A(f) = A(f) + \Delta \cdot A(i^e)$$
Streaming Graph Algorithms

Homomorphic Sketches

Vector of edge multiplicities $f \in \{0,1\}^{\binom{n}{2}}$

Each entry of $f$ is a multiplicity $f_e$ of a (potential) edge $e$ of $G$ (a simple graph with $n$ vertices has up to $\binom{n}{2}$ edges).

For a vertex $v$ let $f^v \in \{0,1\}^{n-1}$ be the restriction of $f$ to the coordinates that involve $v$ (i.e., the $n - 1$ edges that can be adjacent to $v$ in $G$)

$$f = \begin{pmatrix} f_{e_{12}} \\ f_{e_{13}} \\ f_{e_{23}} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

$$f^1 = \begin{pmatrix} f_{e_{12}} \\ f_{e_{13}} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
Vector of edge multiplicities $f \in \{0,1\}^{\binom{n}{2}}$

Each entry of $f$ is a multiplicity $f_e$ of a (potential) edge $e$ of $G$ (a simple graph with $n$ vertices has up to $\binom{n}{2}$ edges).

For a vertex $v$ let $f^v \in \{0,1\}^{n-1}$ be the restriction of $f$ to the coordinates that involve $v$ (i.e., the $n - 1$ edges that can be adjacent to $v$ in $G$)

The sketches of $f$ are formed by concatenation ($\circ$) of the sketches of each $f^v$

$$A(f) = A_1(f^{v_1}) \circ A_2(f^{v_2}) \circ \cdots \circ A_n(f^{v_n})$$

Homomorphic sketches: For each operation on $G$ there is a corresponding operation on the sketches
Streaming Graph Algorithms

Connectivity via Sketches

We wish to maintain a spanning forest of a graph $G = (V, E)$
Streaming Graph Algorithms

Connectivity via Sketches

We wish to maintain a spanning forest of a graph \( G = (V, E) \)

Let’s begin with a simple (non-sketch) algorithm

Connectivity Algorithm

1. repeat
2. for each vertex \( v \) of the current graph do
3. select an edge incident to \( v \)
4. contract all selected edges
5. until the current graph has no edges
Streaming Graph Algorithms

Connectivity via Sketches

We wish to maintain a spanning forest of a graph $G = (V, E)$
Streaming Graph Algorithms

Connectivity via Sketches

We wish to maintain a spanning forest of a graph $G = (V, E)$
Streaming Graph Algorithms

Connectivity via Sketches

We wish to maintain a spanning forest of a graph $G = (V, E)$

![Graph Diagram]
Streaming Graph Algorithms

Connectivity via Sketches

We wish to maintain a spanning forest of a graph $G = (V, E)$
Streaming Graph Algorithms

Connectivity via Sketches

We wish to maintain a spanning forest of a graph $G = (V, E)$.
Streaming Graph Algorithms

Connectivity via Sketches

We wish to maintain a spanning forest of a graph $G = (V, E)$

Let’s begin with a simple (non-sketch) algorithm

Connectivity Algorithm

1. repeat
2. for each vertex $v$ of the current graph do
3. select an edge incident to $v$
4. contract all selected edges
5. until the current graph has no edges

Finds the connected components of $G$, and a spanning forest, in $O(\log n)$ rounds
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

2. Apply $\ell_0$-sampling via linear sketches
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

2. Apply $\ell_0$-sampling via linear sketches

$\ell_0$-sampling

Let $K = \text{polylog}(N)$. There is a distribution over matrices $M \in \mathbb{R}^{K \times N}$ such that for any $x \in \mathbb{R}^N$, a random non-zero element of $x$ can be reconstructed from $Mx$ with high probability.
To design an algorithm that uses sketches we have to:

1. **Define an appropriate graph representation**

For each vertex $v_i$ we define a vector $a^i \in \{-1,0,1\}^n$

with entries

$$a^i_{(j,k)} = \begin{cases} 
+1, & \text{if } i = j < k \text{ and } (v_j, v_k) \in E \\
-1, & \text{if } j < k = i \text{ and } (v_j, v_k) \in E \\
0, & \text{otherwise} 
\end{cases}$$
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

Vector of vertex $i$: $\mathbf{a}^i = (a^i_{(1,2)} \ a^i_{(1,3)} \ a^i_{(1,4)} \ a^i_{(2,3)} \ a^i_{(2,4)} \ a^i_{(3,4)})^T$
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

Vector of vertex $i$: \[
\mathbf{a}^i = (\mathbf{a}_{(1,2)}^i, \mathbf{a}_{(1,3)}^i, \mathbf{a}_{(1,4)}^i, \mathbf{a}_{(2,3)}^i, \mathbf{a}_{(2,4)}^i, \mathbf{a}_{(3,4)}^i)^T
\]

\[
\begin{align*}
\mathbf{a}^1 &= (1 \ 1 \ 0 \ 0 \ 0 \ 0)^T \\
\mathbf{a}^2 &= (-1 \ 0 \ 0 \ 1 \ 0 \ 0)^T \\
\mathbf{a}^3 &= (0 \ -1 \ 0 \ -1 \ 0 \ 1)^T \\
\mathbf{a}^4 &= (0 \ 0 \ 0 \ 0 \ 0 \ -1)^T
\end{align*}
\]
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

Vector of vertex $i$:

$$a^i = (a^i_{1,2}, a^i_{1,3}, a^i_{1,4}, a^i_{2,3}, a^i_{2,4}, a^i_{3,4})^T$$

$$a^1 = (1 \ 1 \ 0 \ 0 \ 0 \ 0)^T$$

$$a^2 = (-1 \ 0 \ 0 \ 1 \ 0 \ 0)^T$$

$$a^3 = (0 \ -1 \ 0 \ -1 \ 0 \ 1)^T$$

$$a^4 = (0 \ 0 \ 0 \ 0 \ 0 \ -1)^T$$

$$a^1 + a^2 = (0 \ 1 \ 0 \ 1 \ 0 \ 0)^T$$
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

![Diagram of a graph with vertices 1, 2, 3, 4 and edges (1, 3), (3, 4), (1, 2), (2, 3)]

Vector of vertex $i$: $\mathbf{a}^i = (a^i_{(1,2)}, a^i_{(1,3)}, a^i_{(1,4)}, a^i_{(2,3)}, a^i_{(2,4)}, a^i_{(3,4)})^T$

$\mathbf{a}^1 = (1 \ 1 \ 0 \ 0 \ 0 \ 0)^T$

$\mathbf{a}^2 = (-1 \ 0 \ 0 \ 1 \ 0 \ 0)^T$

$\mathbf{a}^3 = (0 \ -1 \ 0 \ -1 \ 0 \ 1)^T$

$\mathbf{a}^4 = (0 \ 0 \ 0 \ 0 \ 0 \ -1)^T$

$\mathbf{a}^1 + \mathbf{a}^2 = (0 \ 1 \ 0 \ 1 \ 0 \ 0)^T$
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

For each vertex $v_i$ we define a vector $\mathbf{a}^i \in \{-1,0,1\}^n$ with entries

$$
\mathbf{a}_{(j,k)}^i = \begin{cases} 
+1, & \text{if } i = j < k \text{ and } (v_j, v_k) \in E \\
-1, & \text{if } j < k = i \text{ and } (v_j, v_k) \in E \\
0, & \text{otherwise}
\end{cases}
$$

For any subset of vertices $U \subseteq V$, let

$$
\mathbf{a}(U) = \sum_{v_i \in U} \mathbf{a}^i
$$
To design an algorithm that uses sketches we have to:

1. **Define an appropriate graph representation**

For each vertex \( v_i \) we define a vector \( a^i \in \{-1,0,1\}^n \)

with entries

\[
a^i_{(j,k)} = \begin{cases} 
+1, & \text{if } i = j < k \text{ and } (v_j, v_k) \in E \\
-1, & \text{if } j < k = i \text{ and } (v_j, v_k) \in E \\
0, & \text{otherwise}
\end{cases}
\]

For any subset of vertices \( U \subseteq V \), let \( a(U) = \sum_{v_i \in U} a^i \)

The non-zero entries of \( a(U) \) correspond to \( \delta_G(U) = \) the set of edges of \( G \) that cross the cut \( (U,V \setminus U) \)
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

For any subset of vertices $U \subseteq V$, let $a(U) = \sum_{v_i \in U} a^i$

The non-zero entries of $a(U)$ correspond to $\delta_G(U) = \text{the set of edges of } G \text{ that cross the cut } (U, V \setminus U)$
Streaming Graph Algorithms

Connectivity via Sketches

To design an algorithm that uses sketches we have to:

1. Define an appropriate graph representation

For any subset of vertices $U \subseteq V$, let $a(U) = \sum_{v_i \in U} a^i$

The non-zero entries of $a(U)$ correspond to $\delta_G(U) = \text{the set of edges of } G \text{ that cross the cut } (U, V \setminus U)$

Thus $\sum_{v_i \in U} M a^i = M (\sum_{v_i \in U} a^i)$ gives a random edge in $\delta_G(U)$
Streaming Graph Algorithms

Connectivity via Sketches

Connectivity via Sketches Algorithm I: Compute the Sketches in a Single Pass

1. Choose $t = \Theta(\log n)$
2. \textbf{for} $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, t$ \textbf{do}
3. Construct the random projection $M_j a^i$
4. \textbf{for} $i = 1, 2, \ldots, n$ \textbf{do}
5. Compute $A_i (f^{v_i}) = (M_1 a^i) \circ (M_2 a^i) \circ \cdots \circ (M_t a^i)$
## Streaming Graph Algorithms

### Connectivity via Sketches

<table>
<thead>
<tr>
<th>Connectivity via Sketches Algorithm I: Compute the Sketches in a Single Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Choose $t = O(\log n)$</td>
</tr>
<tr>
<td>2. for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, t$ do</td>
</tr>
<tr>
<td>3. Construct the random projection $M_j a^i$</td>
</tr>
<tr>
<td>4. for $i = 1, 2, \ldots, n$ do</td>
</tr>
<tr>
<td>5. Compute $A_i (f^v) = (M_1 a^i) \circ (M_2 a^i) \circ \cdots \circ (M_t a^i)$</td>
</tr>
</tbody>
</table>

- Each sketch $A_i$ has dimension $O(\text{polylog} n)$
- Since there are $n$ sketches, the required space is $O(n \text{ polylog} n)$
Connectivity via Sketches

Connectivity via Sketches Algorithm II: Emulate Connectivity Algorithm

1. Let $\hat{V} = V$ be the initial set of super-vertices
2. for $i = 1, 2, ..., t$ do
3. for each super-vertex $U \in \hat{V}$ do
4. use $\sum_{v_i \in U} Ma^i$ to sample an edge between $U$ and another super-vertex $W$
5. collapse $U$ and $W$ to form a new super-vertex
Streaming Graph Algorithms

Connectivity via Sketches

Connectivity via Sketches Algorithm II: Emulate Connectivity Algorithm

1. Let $\hat{V} = V$ be the initial set of super-vertices
2. for $i = 1, 2, \ldots, t$ do
   3. for each super-vertex $U \in \hat{V}$ do
      4. use $\sum_{v_i \in U} \alpha^i$ to sample an edge between $U$ and another super-vertex $W$
      5. collapse $U$ and $W$ to form a new super-vertex

The update time (to process the next edge in $S$) is $O(\text{polylog} n)$
Concluding remarks

• Many graph algorithms in the data stream model are known for basic problems. E.g., estimating connectivity, approximating distances, finding approximate matchings, counting subgraphs,…

• But limited work on directed graphs!

• Space constraints: semi-stream model not suited for sparse graphs ($m = O(n \text{ polylog} n)$)
Streaming Architectures

[Diagram of Spark Streaming]

Records processed in batches with short tasks. Each batch is a RDD (partitioned dataset).

Picture from https://databricks.com/blog/2015/07/30/diving-into-spark-streamings-execution-model.html
Streaming Architectures

Google Cloud Platform

https://cloud.google.com/solutions/architecture/streamprocessing