Solving Graph Problems in the Streaming Model

Sofya Vorotnikova

University of Massachusetts Amherst
Graph algorithms have a variety of applications across many areas of computer science and other fields.

**Problem**: the graph in question can be
- too large to be stored in main memory
- distributed across many machines
- changing over time

**Streaming** to the rescue!
Streaming Model: Insert-Only

Edges of the graph arrive one-by-one in a sequence.

In an **insert-only** stream we only insert edges into the graph.
Streaming Model: Insert-Only

Edges of the graph arrive one-by-one in a sequence.

In an **insert-only** stream we only insert edges into the graph.

(1, 4)
Streaming Model: Insert-Only

Edges of the graph arrive one-by-one in a sequence.

In an **insert-only** stream we only insert edges into the graph.

(1, 4), (2, 3)
Streaming Model: Insert-Only

Edges of the graph arrive one-by-one in a sequence.

In an **insert-only** stream we only insert edges into the graph.

(1, 4), (2, 3), (1, 2)
Streaming Model: Insert-Only

Edges of the graph arrive one-by-one in a sequence.

In an **insert-only** stream we only insert edges into the graph.

(1, 4), (2, 3), (1, 2), (1, 3)
Streaming Model: Insert-Only

Edges of the graph arrive one-by-one in a sequence.

In an **insert-only** stream we only insert edges into the graph.

\[(1, 4), (2, 3), (1, 2), (1, 3), (3, 4)\]
Streaming Model: Insert-Only

Edges of the graph arrive one-by-one in a sequence.

In an **insert-only** stream we only insert edges into the graph.

\[(1, 4), (2, 3), (1, 2), (1, 3), (3, 4)\]
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an insert-delete or dynamic stream we can both insert and delete edges.
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an insert-delete or dynamic stream we can both insert and delete edges.

\[ + (1, 4) \]
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an insert-delete or dynamic stream we can both insert and delete edges.

\[+(1, 4), +(2, 3)\]
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an **insert-delete** or **dynamic** stream we can both insert and delete edges.

\[+(1, 4), +(2, 3), +(2, 4)\]
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an **insert-delete** or **dynamic** stream we can both insert and delete edges.

\[ + (1, 4), + (2, 3), + (2, 4), + (1, 2) \]
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an insert-delete or dynamic stream we can both insert and delete edges.

\[+ (1, 4), + (2, 3), + (2, 4), + (1, 2), - (2, 4)\]
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an insert-delete or dynamic stream we can both insert and delete edges.

$+(1, 4), +(2, 3), +(2, 4), +(1, 2), -(2, 4), +(1, 3)$
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an insert-delete or dynamic stream we can both insert and delete edges.

$+(1, 4), + (2, 3), + (2, 4), + (1, 2), - (2, 4), + (1, 3), + (3, 4)$
Streaming Model: Dynamic

Edges of the graph arrive one-by-one in a sequence.

In an insert-delete or dynamic stream we can both insert and delete edges.

\[+(1, 4), +(2, 3), +(2, 4), +(1, 2), -(2, 4), +(1, 3), +(3, 4)\]
Streaming Model: Adjacency List

Edges of the graph arrive one-by-one in a sequence.

Usually, we assume arbitrary order of edges. However, in adjacency list model edges incident to the same vertex arrive together.
Streaming Model: Adjacency List

Edges of the graph arrive one-by-one in a sequence.

Usually, we assume arbitrary order of edges. However, in **adjacency list model** edges incident to the same vertex arrive together.

2 : (2, 1), (2, 3)
Streaming Model: Adjacency List

Edges of the graph arrive one-by-one in a sequence.

Usually, we assume arbitrary order of edges. However, in adjacency list model edges incident to the same vertex arrive together.

2 : (2, 1), (2, 3)
4 : (4, 1), (4, 3)
Streaming Model: Adjacency List

Edges of the graph arrive one-by-one in a sequence.

Usually, we assume arbitrary order of edges. However, in adjacency list model edges incident to the same vertex arrive together.

2 : (2, 1), (2, 3)
4 : (4, 1), (4, 3)
1 : (1, 2), (1, 3), (1, 4)
Streaming Model: Adjacency List

Edges of the graph arrive one-by-one in a sequence.

Usually, we assume arbitrary order of edges. However, in **adjacency list model** edges incident to the same vertex arrive together.

```
2 : (2, 1), (2, 3)
4 : (4, 1), (4, 3)
1 : (1, 2), (1, 3), (1, 4)
3 : (3, 1), (3, 2), (3, 4)
```
Streaming Model: Adjacency List

Edges of the graph arrive one-by-one in a sequence.

Usually, we assume arbitrary order of edges. However, in adjacency list model edges incident to the same vertex arrive together.

2 : (2, 1), (2, 3)
4 : (4, 1), (4, 3)
1 : (1, 2), (1, 3), (1, 4)
3 : (3, 1), (3, 2), (3, 4)
Objective: minimize the amount of information we store throughout the stream that is still sufficient to (approximately) solve the problem.

- Densest subgraph
- Vertex cover
  - Solution size is small
- Maximum matching
  - Solution size is small
  - Input graph is sparse
- Counting triangles
Densest Subgraph

**Density** of a graph is defined as $\frac{\text{#edges}}{\text{#vertices}}$. Given an input graph, want to find its subgraph with maximum density.

Used for community detection in social networks and identifying link spam on the web, in addition to applications on financial and biological data.

If we have unlimited access to the entire graph, this problem can be solved in polynomial time.

We show[1] that for an arbitrary order stream (insertion-only or dynamic) we can find a $(1 + \epsilon)$-approximation while storing $\tilde{O}(n)$ edges.
Densest Subgraph: Main Idea

Sample $\tilde{O}(n)$ edges uniformly at random.

- If a subgraph has small density, after sampling its density remains relatively small
- If a subgraph has high density, after sampling its density remains relatively high
Densest Subgraph: Main Idea

Sample $\tilde{O}(n)$ edges uniformly at random.

- If a subgraph has small density, after sampling its density remains relatively small
- If a subgraph has high density, after sampling its density remains relatively high
Vertex cover of a graph is a set of vertices s.t. every edge has at least one endpoint in the set.

Matching is a set of edges that don’t share endpoints.

We want to find a vertex cover of minimum size and a matching of maximum size.

The problems are closely related:

\[
\text{match}(G) \leq \text{vc}(G) \leq 2 \text{match}(G)
\]

We show[2] that if the size of min vertex cover/max matching is at most \( k \), then we can find it exactly using \( \tilde{O}(k^2) \) space in insert-only or dynamic arbitrary order stream.
Vertex Cover and Matching of Small Size

Uniform sampling doesn’t work in this case.
Vertex Cover and Matching of Small Size

Uniform sampling doesn’t work in this case.
Vertex Cover and Matching of Small Size

Uniform sampling doesn’t work in this case.
Uniform sampling doesn’t work in this case.

**Problem:** Oversampling from dense parts of the graph. This was good for densest subgraph, but not here.
Vertex Cover and Matching of Small Size: Color Sampling
Vertex Cover and Matching of Small Size: Color Sampling

- Color vertices with $b$ colors.
Vertex Cover and Matching of Small Size: Color Sampling

- Color vertices with \( b \) colors.
- For every color pair, sample one edge uniformly at random.
Vertex Cover and Matching of Small Size: Color Sampling

This avoids oversampling from high degree vertices.
Vertex Cover and Matching of Small Size: Color Sampling

This avoids oversampling from high degree vertices.
Vertex Cover and Matching of Small Size: Color Sampling

This avoids oversampling from high degree vertices.
Assume $\text{match}(G) \leq \text{vc}(G) \leq k$

$H$ = set of **heavy** vertices with degree at least $10k$

$L$ = set of **light** edges s.t. both endpoints are not heavy

$\text{vc}(G) = |H| + \text{vc}(L)$

$\text{match}(G) = |H| + \text{match}(L)$
Vertex Cover and Matching of Small Size

Let $G'$ be the graph obtained by sampling using $\Theta(k)$ colors.

With constant probability:

- If an edge is light, then it is sampled
- If a vertex is heavy, its degree in $G'$ is at least $5k$, which is high enough to recognize that it is heavy

By repeating the sampling $\Theta(\log n)$ times we sample all light edges and detect all heavy vertices with high probability.

Thus, preserving the size of vertex cover and matching.
In the previous result we were finding the matching itself. However, if there are no size restrictions, maximum matching can be as large as $n/2$.

By approximating the size of the matching without finding the matching itself, we can use smaller space.
We concentrate on the class of graphs of arboricity $\alpha$.

**Arboricity** is the minimum number of forests into which the edges of the graph can be partitioned.

*Property:* In a graph with arboricity $\alpha$, every induced subgraph on $r$ vertices has at most $\alpha r$ edges.

Planar graphs have arboricity at most 3.

We show[3] an $(\alpha + 2)$-approximation using space

- $\tilde{O}(\alpha n^{2/3})$ in insert-only arbitrary order streams
- $\tilde{O}(\alpha n^{4/5})$ in dynamic arbitrary order streams
Approximating Size of Maximum Matching:
Local Fractional Matching

\[ x_{uv} = \min \left( \frac{1}{\deg(u)}, \frac{1}{\deg(v)}, \frac{1}{\alpha + 1} \right) \]
Approximating Size of Maximum Matching:
Local Fractional Matching

$$x_{uv} = \min \left( \frac{1}{\text{deg}(u)}, \frac{1}{\text{deg}(v)}, \frac{1}{\alpha + 1} \right)$$
Approximating Size of Maximum Matching:
Local Fractional Matching

\[ x_{uv} = \min \left( \frac{1}{\deg(u)}, \frac{1}{\deg(v)}, \frac{1}{\alpha + 1} \right) \]
Approximating Size of Maximum Matching: Local Fractional Matching

\[ x_{uv} = \min \left( \frac{1}{\deg(u)}, \frac{1}{\deg(v)}, \frac{1}{\alpha + 1} \right) \]

- \( x \) is a fractional matching
- \( x_e \) is only a function of edges that share an endpoint with \( e \)
- \((\alpha + 1) \sum_e x_e\) is an \((\alpha + 2)\)-approx of \( \text{match}(G) \)
Sample a set of vertices $S$ uniformly at random with probability $p$. Let $E_S$ be edges with both endpoints in $S$. In parallel:

- Compute (approximate) matching if it is small
- If the matching is large, use $\frac{1}{p^2} \sum_{e \in E_S} x_e$ as estimate

We balance the size of set $S$ and space required to find (approximate) small matching to get space:

- $\tilde{O}(\alpha n^{2/3})$ in insert-only streams
- $\tilde{O}(\alpha n^{4/5})$ in dynamic streams
Triangle Counting

**Problem:** count/approximate the number of triangles in a graph.

**Hard in arbitrary order streaming!**

THE TOWER OF DOOM
Triangle Counting in Adjacency List Streams

Counting triangles in **adjacency list streams is easier**, since we see every triangle twice.
Counting triangles in adjacency list streams is easier, since we see every triangle twice.
Triangle Counting in Adjacency List Streams

Counting triangles in **adjacency list streams is easier**, since we see every triangle twice.
Triangle Counting: Results

We parametrize the algorithm in terms of $T$, the number of triangles. This convention is widely adopted in the literature. The problem can then be viewed as distinguishing between graphs with at most $t$ and at least $(1 + \epsilon)t$ triangles.

In the insert-only adjacency list stream we obtain[4] space

- $\tilde{O}(\frac{m}{\sqrt{T}})$ in one pass
- $\tilde{O}(\frac{m^{3/2}}{T})$ in two passes

Note that

$$\frac{m}{\sqrt{T}} \leq \frac{m^{3/2}}{T} \text{ when } T \leq m \text{ (small number of triangles)}$$

$$\frac{m}{\sqrt{T}} \geq \frac{m^{3/2}}{T} \text{ when } T \geq m \text{ (large number of triangles)}$$
Triangle Counting: One Pass Algorithm

Assign each triangle to one of the edges. A triangle $xyz$ where the vertices appear in order $x$, then $y$, then $z$ in the stream, is assigned to edge $xz$. Call the number of triangles assigned to $xz$, $R_{xz}$.

$R_{1,4} = 2$
Triangle Counting: One Pass Algorithm

Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Large tower
Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

When the first vertex of the tower arrives, sample edges uniformly.
Triangle Counting: One Pass Algorithm

Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Large tower

When the first vertex of the tower arrives, sample edges uniformly.
Triangle Counting: One Pass Algorithm

Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Large tower

For middle vertices, do nothing.
Triangle Counting: One Pass Algorithm

Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Large tower

For middle vertices, do nothing.
Triangle Counting: One Pass Algorithm

**Main idea:** estimate large $R_{xz}$ separately from small $R_{xz}$.

**Large tower**

For middle vertices, do nothing.
Triangle Counting: One Pass Algorithm

Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Large tower

For middle vertices, do nothing.
Triangle Counting: One Pass Algorithm

**Main idea:** estimate large $R_{xz}$ separately from small $R_{xz}$.

**Large tower**

For the last vertex, count the number of triangles formed.
Triangle Counting: One Pass Algorithm

Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Large tower

For the last vertex, count the number of triangles formed. Since the tower is large, that gives an accurate estimate.
Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.
Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Small tower

When the first vertex of the tower arrives, sample edges uniformly (only care about the base of the tower).
Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Small tower

When the first vertex of the tower arrives, sample edges uniformly (only care about the base of the tower).
Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Small tower

If the base was sampled: for middle vertices, increment the counter. $R_{xz} = 1$
Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Small tower

If the base was sampled: for middle vertices, increment the counter. $R_{xz} = 2$
**Main idea:** estimate large $R_{xz}$ separately from small $R_{xz}$.

**Small tower**

For the last vertex, do nothing.

$R_{xz} = 2$
Triangle Counting: One Pass Algorithm

Main idea: estimate large $R_{xz}$ separately from small $R_{xz}$.

Small towers

Depend on whether the base was sampled.
Since these are small, the estimate is still accurate.
Again, we assign each triangle to one of the edges. A triangle $xyz$ with vertex degrees $\deg(x) \leq \deg(y) \leq \deg(z)$, is assigned to edge $xy$. Call the number of triangles assigned to $xy$, $R_{xy}$. 
Triangle Counting: Two Pass Algorithm

Again, we assign each triangle to one of the edges. A triangle $xyz$ with vertex degrees $\deg(x) \leq \deg(y) \leq \deg(z)$, is assigned to edge $xy$. Call the number of triangles assigned to $xy$, $R_{xy}$. 
Again, we assign each triangle to one of the edges. A triangle $xyz$ with vertex degrees $\deg(x) \leq \deg(y) \leq \deg(z)$, is assigned to edge $xy$. Call the number of triangles assigned to $xy$, $R_{xy}$. 
Again, we assign each triangle to one of the edges. A triangle \( xyz \) with vertex degrees \( \deg(x) \leq \deg(y) \leq \deg(z) \), is assigned to edge \( xy \). Call the number of triangles assigned to \( xy \), \( R_{xy} \).
Triangle Counting: Two Pass Algorithm

**Main idea:** All $R_{xy}$ are now relatively small. $R_{xy} \leq \sqrt{2m}$ for all xy.

**First pass**

- sample each edge uniformly at random
- store the sampled edges and degrees of their endpoints
Triangle Counting: Two Pass Algorithm

Main idea: All $R_{xy}$ are now relatively small. $R_{xy} \leq \sqrt{2m}$ for all $xy$.

Second pass

- Set counters for all sampled edges to 0
Triangle Counting: Two Pass Algorithm

**Main idea:** All $R_{xy}$ are now relatively small. $R_{xy} \leq \sqrt{2m}$ for all $xy$.

**Second pass**

- $v$ arrives: check all sampled edges $ab$
- $vab$ is $\Delta$, $\deg(a) \leq \deg(v)$, $\deg(b) \leq \deg(v)$: increment $R_{ab}$
Triangle Counting: Two Pass Algorithm

Main idea: All $R_{xy}$ are now relatively small. $R_{xy} \leq \sqrt{2m}$ for all $xy$.

Second pass

- $v$ arrives: check all sampled edges $ab$
- $vab$ is $\Delta$, $\deg(a) \leq \deg(v)$, $\deg(b) \leq \deg(v)$: increment $R_{ab}$
Triangle Counting: Two Pass Algorithm

Main idea: All $R_{xy}$ are now relatively small. $R_{xy} \leq \sqrt{2m}$ for all $xy$.

- $v$ arrives: check all sampled edges $ab$
- $vab$ is $\Delta$, $\deg(a) \leq \deg(v)$, $\deg(b) \leq \deg(v)$: increment $R_{ab}$
Triangle Counting: Two Pass Algorithm

Main idea: All $R_{xy}$ are now relatively small. $R_{xy} \leq \sqrt{2m}$ for all $xy$.

- Since all towers are small, the estimate is accurate.
Conclusion: Things To Do With Streams

**Sampling!**
- Sample edges uniformly.
- Sample edges non-uniformly. Example: color sampling.
- Sample vertices, then use the induced subgraph.

**Other things:**
- Compute degrees of vertices or other quantities depending on degrees.
- Using stream ordering as part of the algorithm.
Thank you for your attention!

Questions?
1. Densest Subgraph in Dynamic Graph Streams. A. McGregor, D. Tench, S. Vorotnikova, and H. Vu. MFCS 2015

