Advanced Topics in Theoretical Computer Science

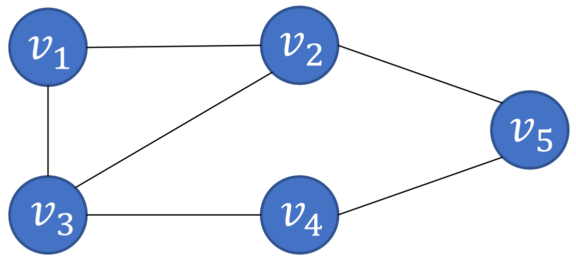
University of Bristol

Lecture 9

Graph Streams: Connectivity and Bipartiteness

Streaming Algorithms for Graph Problems

Input graph $G = (V, E), n = |V|, m = |E|$

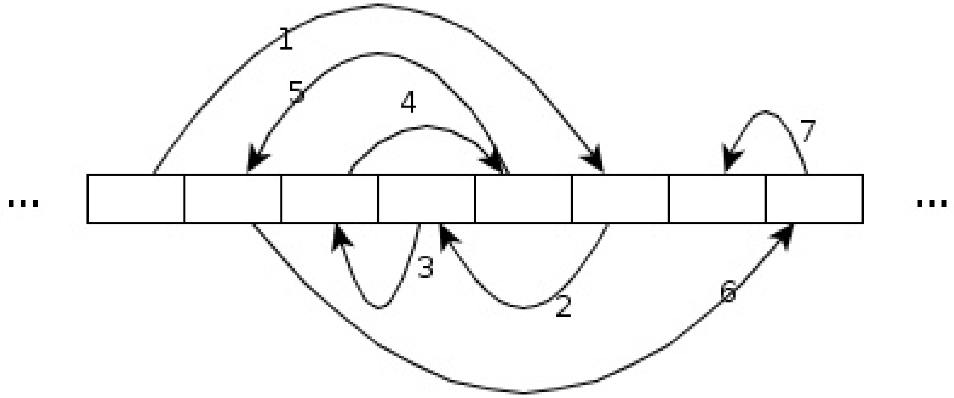


How to process G in a streaming fashion?

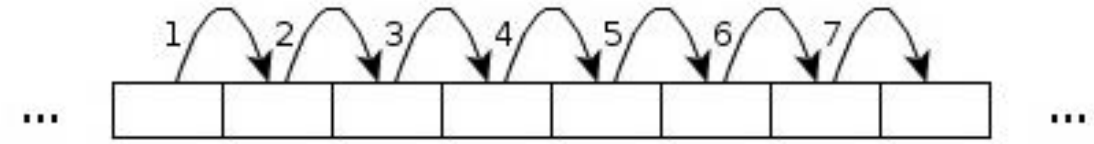
1. Streaming (or linear or sequential) access
2. Sublinear space

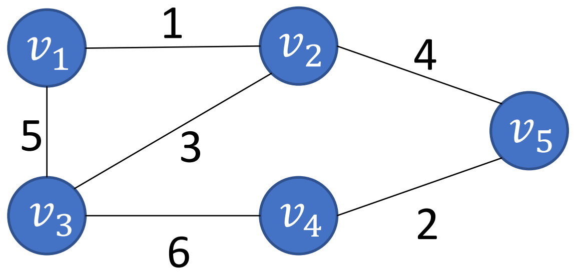
1. Streaming Access: Edge-arrival Model

Random Access



Streaming Access





Edge-arrival Model: (Insertion-only Model)

- Sequence of the edges of the input graph
- No assumption on the order of the edges, e.g.,

$$S = v_1v_2, v_4v_5, v_2v_3, v_2v_5, v_1v_3, v_3v_4.$$

2. Sublinear Space

Stream length: m edges

How large can $\boldsymbol{m}$ be in terms of $\boldsymbol{n}$?

Lemma. A (simple) graph on n vertices has $O(n^2)$ edges.

Proof.

- Every (simple) graph is a subgraph of the complete graph, i.e., the graph that contains all potential edges
- The complete graph on n vertices has $\binom{n}{2} = \frac{n\,(n-1)}{2} = \theta(n^2)$ edges.

$\square$

2. Sublinear Space: Semi-streaming Algorithms

Space Considerations:

- Space $o(m)$ is sublinear space
- We will however focus on space in terms of n
- Space $o(n^2)$ is therefore sublinear for very dense graphs (and non-trivial)

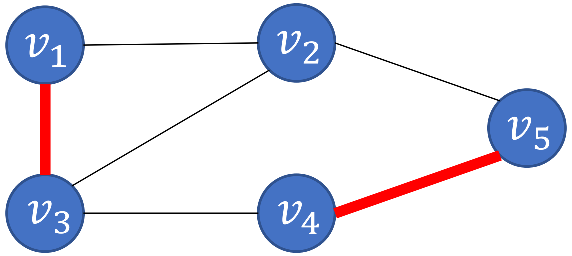
Semi-streaming Algorithms: (Feigenbaum et al. 2004)

- Streaming algorithms for graph problems with space $O(n \text{ poly} \log n) = O(n \log^c n)$, for some constant c, are called "semi-streaming" algorithms
- Allows storing a poly-logarithmic number of edges per vertex (on average)
- Sublinear for graphs with $m = \Omega(n^{1+\varepsilon})$, for any $\varepsilon > 0$

Why Semi-streaming?

Why space $O(n \text{ poly} \log n)$? (e.g., why not $O(\sqrt{n})$?)

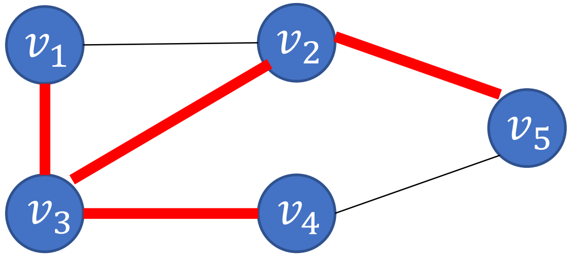
1. Output size of graph problems



Maximum Matching

Largest subset of vertex-disjoint edges

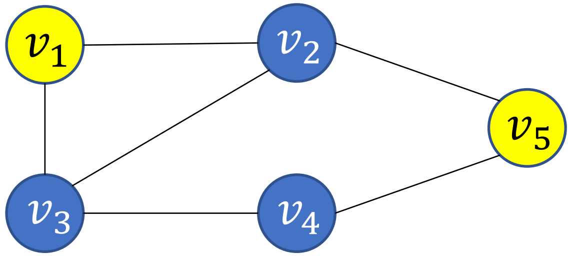
Size: at most n / 2



Spanning Tree

Subtree that spans all vertices of the graph

Size: $n - 1$



Maximum Independent Set

Largest subset of non-adjacent vertices

Size: at most n

Why Semi-streaming? (2)

Why space $O(n \ \text{poly} \log n)$?

2. Many problems provably cannot be solved with less space! (see lower bounds lectures)

- **Connectivity:** Is the graph connected?
- **Bipartiteness:** Is the graph bipartite?
- **Cycle Freeness:** Does the graph contain a cycle?

Boolean output!

Theorem (Sun, Woodruff 2015): Every 1-pass streaming algorithm for **Connectivity, Bipartiteness, or Cycle Freeness** requires space $\Omega(n \log n)$.

Practical Considerations

1. Big graphs exist and are important

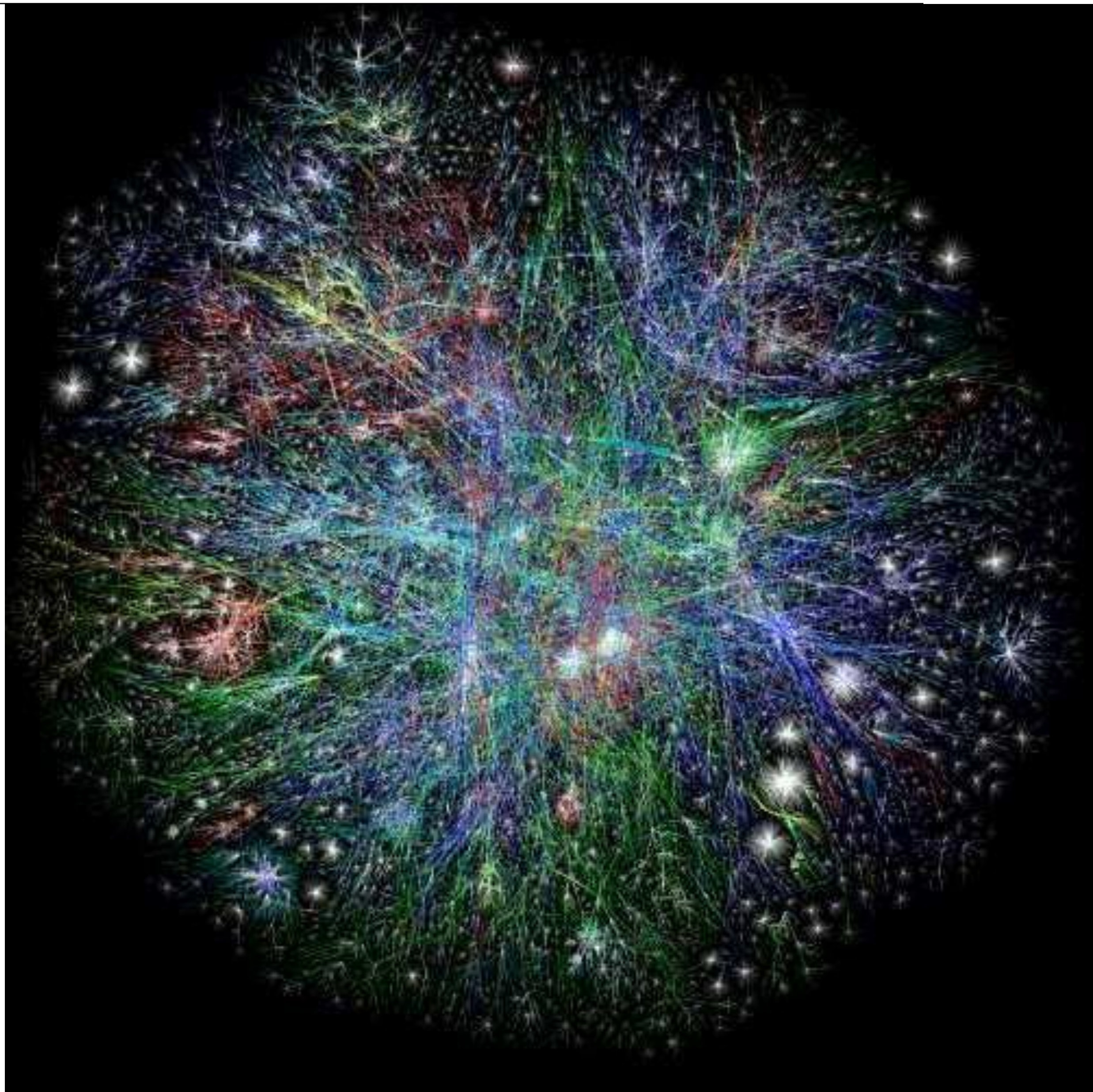
- Social Network graphs

  E.g. Facebook: 2.6 billion active users
  $\rightarrow$ graph on 2.6 billion vertices…

- Web graph
- Graph databases
- Brain models

2. Big graphs and Streaming?

- Memory considerations
- Facebook: stream of new friendships forms edge stream
- Twitter updates

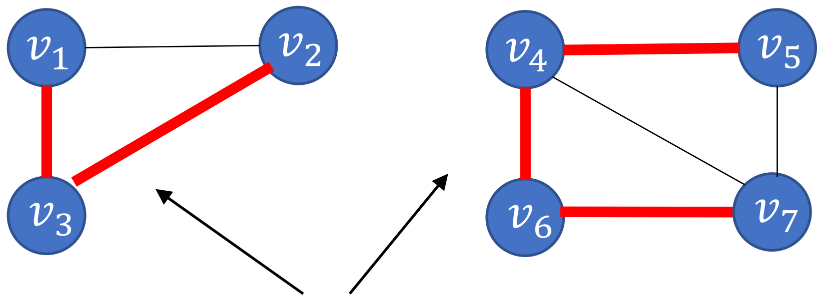


First Graph Streaming Algorithm: Connectivity

Goal: Semi-streaming algorithm for Connectivity in edge-arrival model

- *Semi-streaming:* space $O(n \text{ poly} \log n)$
- *Connectivity*: output $\begin{cases} 0, \text{if } G \text{ is not connected} \\ 1, \text{if } G \text{ is connected} \end{cases}$
- *Edge-arrival model*: Edges arrive in arbitrary order

Spanning Forest



Two Connected Components

Idea:

- Maintain a *spanning forest*:

  $G = (V, E)$ connected: $F \subseteq E$ is a ***spanning tree*** if F (or (V, F)) is a tree that covers every vertex $v \in V$

  $G = (V, E)$ disconnected: $F \subseteq E$ is a ***spanning forest*** if F is the disjoint union of spanning trees of the connected components of G

- If spanning forest becomes a tree then graph is connected.

Can we maintain a spanning forest in semi-streaming space?

First Graph Streaming Algorithm: Connectivity

Maintaining a Spanning Forest in Semi-streaming Space:

1. $F \leftarrow \emptyset$
2. While(stream not yet empty)
   a) Let e be next edge in stream
   b) **if** ($F \cup \{e\}$) does not contain a cycle **then**
   $F \leftarrow F \cup \{e\}$
3. **return** 1 if F is a tree (e.g. $|F| = n - 1$) and 0 otherwise

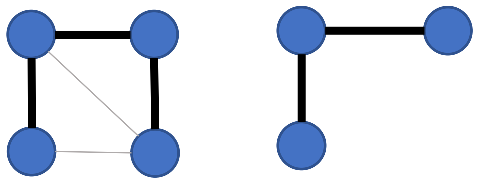
Analysis:

- Let E_i be the set consisting of the first i edges, let $G_i = (V, E_i)$
- Denote by F_i variable F after iteration i
- By induction: F_i is a spanning forest in $G_i \Rightarrow |F_i| \leq n - 1$, for every $i \Rightarrow F_m$ is spanning forest in $G_m = G$.
- Store at most $n - 1$ edges, which yields space $O(n \log n)$.

First Graph Streaming Algorithm: Connectivity

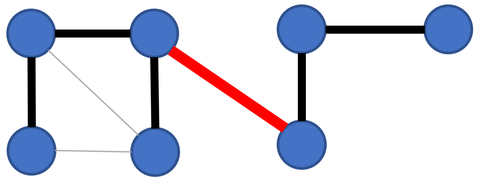
Induction:

- **Hypothesis:** F_i is spanning forest in G_i
- **Induction Start:** $F_0 = \emptyset$ is spanning forest in G_0
- **To show:** F_{i+1} is spanning forest in G_{i+1}



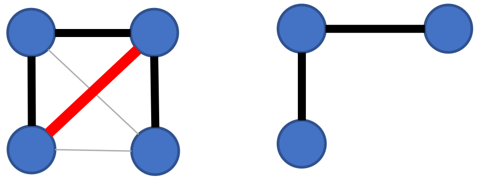
Case 1: $F_i \cup \{e\}$ does not contain a cycle

- $F_{i+1} = F_i \cup \{e\}$ is clearly a forest as it remains acyclic
- e merges two components in G, and e connects the two spanning trees of the two components in F_i, F_{i+1} is thus a spanning forest



Case 2: $F_i \cup \{e\}$ contain a cycle

- $F_{i+1} = F_i \Rightarrow F_{i+1}$ is a forest
- Connected components do not change, F_{i+1} is thus a spanning forest

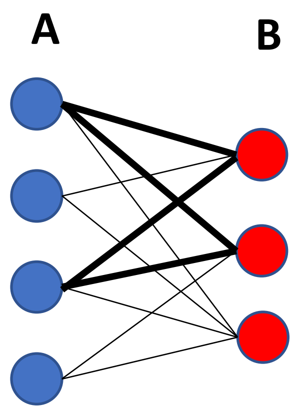


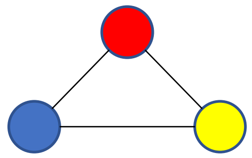
Testing Bipartiteness

Goal: Semi-streaming algorithm for Testing Bipartiteness

Definition: A graph $G = (V, E)$ is *bipartite* if (the three items are equivalent)

1. $V = A \dot{\cup} B$ (V is the disjoint union of A and B) and all edges have one endpoint in A and one in B; (we usually write $G = (A, B, E)$)
2. G admits a 2-coloring, i.e., an assignment $c: V \rightarrow \{0, 1\}$ of (at most) two colors to V such that no edge is *monochromatic*, i.e., both endpoints have the same color
3. G does not contain any odd-length cycles .





Testing Bipartiteness: Algorithm

Semi-streaming Algorithm for Bipartiteness Testing

1. $F \leftarrow \emptyset$
2. While(stream not yet empty)
   a) Let e be next edge in stream
   b) **if** ($F \cup \{e\}$) does not contain a cycle **then**
   $F \leftarrow F \cup \{e\}$
   else if ($F \cup \{e\}$) contains an odd-length cycle **then**
   return "*not bipartite*"
3. **return "***bipartite*"

Analysis:

- Space $O(n \log n)$ as in Connectivity algorithm
- Why is the algorithm correct?

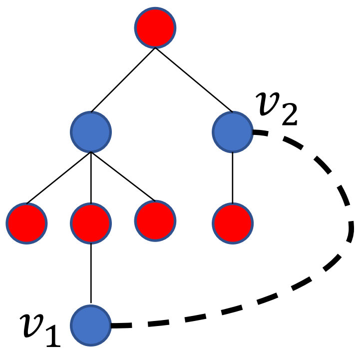
Testing Bipartiteness: Algorithm

Correctness:

1st case: Algorithm reports “not bipartite”
Only happens when odd cycle detected. Algorithm therefore correct.

2nd case: Algorithm reports “bipartite”

- Consider spanning forest (V, F) when algorithm terminates
- Define 2-coloring $c: V \rightarrow \{0,1\}$ that colors the forest (V, F)
- **Claim:** c is also a valid coloring of input graph $G = (V, E)$



- Suppose it is not. Then, $\exists\, v_1 v_2 \in E$ such that $c(v_1) = c(v_2)$. Observe that nodes with the same color are at even distance in (V, F). Let $P \subseteq F$ be the edges of the path from v_1 to v_2 in (V, F). Then $P \cup \{v_1, v_2\}$ forms an odd cycle, a contradiction to the fact that the algorithm did not enter the first case.

□

Summary and References

Summary:

- We introduced the semi-streaming model (i.e., space $O(n \operatorname{poly} \log n)$)
- We can maintain a spanning forest in semi-streaming space
- This allows us to decide Connectivity and Bipartiteness

References:

- Xiaoming Sun, David P. Woodruff: “Tight Bounds for Graph Problems in Insertion Streams”. APPROX-RANDOM 2015: 435-448
- Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, Jian Zhang: “On Graph Problems in a Semi-streaming Model.” ICALP 2004: 531-543