Graph Streaming and Sketching

Lecture 19

Nov 5, 2020

Graphs

- $G = (V, E)$ is an undirected graph
- $n = |V|$ and $m = |E|$
- Edges $e_1, e_2, \ldots, e_m$ seen as a stream, n known

Graphs

- $G = (V, E)$ is an undirected graph
- $n = |V|$ and $m = |E|$
- Edges $e_1, e_2, \ldots, e_m$ seen as a stream, n known

Questions:

- What graph problems can be solve with small space?
- Can we handle edge deletions?

Semi-streaming Model

Lower bounds show that we require $\Omega(n)$ memory

Assume we have $\Theta(n \text{polylog}(n))$ memory. About polylog per vertex of the graph

Can solve several interesting problems. Essentially reduce dense graphs to sparse graphs.

Connectivity

- Is G connected? Output a spanning tree if it is.
- Output an MST of G in the weighted case.
- Is G k-edge connected?

Basic Connectivity

- Maintain spanning forest: need only $O(n)$ edges
- When edge $e_i = (u, v)$ arrives. If u and v are in different components add e_i to spanning forest. Otherwise discard e_i.

MST

- Maintain spanning forest: need only $O(n)$ edges
- When edge $e_i = (u, v)$ arrives. If u and v are in different components add e_i to spanning forest.
- What if u and v are in same connected component?

MST

- Maintain spanning forest: need only $O(n)$ edges
- When edge $e_i = (u, v)$ arrives. If u and v are in different components add e_i to spanning forest.
- What if u and v are in same connected component? Check cycle formed by adding e_i and discard heaviest edge in cycle.

MST

- Maintain spanning forest: need only $O(n)$ edges
- When edge $e_i = (u, v)$ arrives. If u and v are in different components add e_i to spanning forest.
- What if u and v are in same connected component? Check cycle formed by adding e_i and discard heaviest edge in cycle.

Exercise: Prove that algorithm outputs an MST if G is connected.

MST

- Maintain spanning forest: need only $O(n)$ edges
- When edge $e_i = (u, v)$ arrives. If u and v are in different components add e_i to spanning forest.
- What if u and v are in same connected component? Check cycle formed by adding e_i and discard heaviest edge in cycle.

Exercise: Prove that algorithm outputs an MST if G is connected.

Note: we did not focus on time to process each edge in stream. Can use data structures to implement in $O(\log n)$ time per operation.

k-edge-connectivity

Definition

A graph $G = (V, E)$ is k-edge-connected if deleting any $k - 1$ edges still leaves a connected graph.

k-edge-connectivity

Definition

A graph $G = (V, E)$ is k-edge-connected if deleting any $k - 1$ edges still leaves a connected graph.

Definition

Given a graph $G = (V, E)$ and $S \subset V$, $\delta(S)$ is the set of edges with exactly one end point in S.

k-edge-connectivity

Definition

A graph $G = (V, E)$ is k-edge-connected if deleting any $k - 1$ edges still leaves a connected graph.

Definition

Given a graph $G = (V, E)$ and $S \subset V$, $\delta(S)$ is the set of edges with exactly one end point in S.

Lemma

A graph G is k-edge connected iff $|\delta(S)| \geq k$ for all $S \subset V$.

Sparse certificates for k-edge connectivity

Observation: If G is k-edge-connected than $m \geq kn/2$. Why?

Sparse certificates for k-edge connectivity

Observation: If G is k-edge-connected than $m \geq kn/2$. Why?

Question: Suppose G is *edge-minimal* k-edge-connected graph on n nodes. What is an upper bound on the number of edges?

Sparse certificates for k-edge connectivity

Observation: If G is k-edge-connected than $m \geq kn/2$. Why?

Question: Suppose G is *edge-minimal* k-edge-connected graph on n nodes. What is an upper bound on the number of edges?

Theorem

An edge-minimal k-edge-connected graph on n nodes has at most $k(n-1)$ edges.

Sparse certificates for k-edge connectivity

Observation: If G is k-edge-connected than $m \geq kn/2$. Why?

Question: Suppose G is *edge-minimal* k-edge-connected graph on n nodes. What is an upper bound on the number of edges?

Theorem

An edge-minimal k-edge-connected graph on n nodes has at most $k(n-1)$ edges.

Theorem

Given a graph G finding the smallest 2-edge-connected subgraph is NP-Hard.

Sparse certificates for k-edge connectivity

Theorem

An edge-minimal k-edge-connected graph on n nodes has at most $k(n-1)$ edges.

Constructive proof via algorithm.

For $i = 1$ to k do
    Let F_i be a spanning forest in $(V, E \setminus \cup_{j=1}^{i-1} F_j)$

Output $H = (V, F_1 \cup F_2 \ldots \cup F_k)$

Sparse certificates for k-edge connectivity

Theorem

An edge-minimal k-edge-connected graph on n nodes has at most $k(n-1)$ edges.

Constructive proof via algorithm.

```
For i = 1 to k do
    Let F_i be a spanning forest in (V, E \ ∪_{j=1}^{i-1} F_j)

Output H = (V, F_1 ∪ F_2 ... ∪ F_k)
```

Easy to see that H as at most $k(n-1)$ edges.

Lemma

H is k-edge-connected if G is.

Streaming setting

```
For $i = 1$ to $k$ do
    Let $F_i$ be a spanning forest in $(V, E \setminus \cup_{j=1}^{i-1} F_j)$

Output $H = (V, F_1 \cup F_2 \ldots \cup F_k)$
```

Algorithm can be implemented in streaming setting. How?

k-node-connectivity

Definition

A graph $G = (V, E)$ is k-node-connected (or k-vertex-connected) if deleting any $k - 1$ nodes leaves a connected graph.

k-node-connectivity

Definition

A graph $G = (V, E)$ is k-node-connected (or k-vertex-connected) if deleting any $k - 1$ nodes leaves a connected graph.

Theorem

An edge-minimal k-edge-connected graph on n nodes has at most kn edges.

Above theorem is much more tricky than for the edge case.

See [Zelke] for references and streaming algorithm.

Part I

Graph sketching for connectivity

Graph sketching

We saw previously that *linear* sketching on vectors x allows for several powerful applications including ability to handle *deletions*

Graph streaming with deletions: each token in stream is of the form (e, Δ) where e is an edge and $\Delta \in \{-1, 1\}$.

Want to maintain a sketch/data structure of size $O(n \text{polylog}(n))$ such that one can answer basic questions. **Example:** connectivity queries.

Linear sketching recap

- Vector $x \in \mathbb{R}^n$ that is updated one coordinate at a time.
- Pick a sketch matrix $M_r \in \mathbb{R}^{k \times n}$ and maintain sketch $M_r x$ of dimension k
- The sketch matrix M_r depends on a random string r and is *implicitly* defined and not explicitly stored. Assumption is that $M_r \mathbf{1}_i$ for vector $\mathbf{1}_i$ (which has $\mathbf{1}$ in i'th coordinate and $\mathbf{0}$ in all other entries) can be computed efficiently from r.
- When x is updated to $x + \alpha \mathbf{1}_i$ we update sketch by $\alpha M_r \mathbf{1}_i$.
- Do postprocessing of $M_r x$

ℓ_0 sampling in turnstile model

$\|x\|_0$ is number of non-zero coordinates (distinct elements)

ℓ_0-sampling: output a non-zero coordinate of x near uniformly. Can be done with $O(\log^2 n)$-sized sketch

Note: allow positive and negative entries in x

Sketching for graphs

Consider vector $f \in \mathbb{R}^{\binom{n}{2}}$ where $f_i \in \{0, 1\}$ indicating whether edge i in the complete graph on n nodes is in the graph or not.

Example:

Sketching f is not adequate for most graph applications. We need information about edges incident to each vertex.

For node v let $f_v \in \mathbb{R}^{\binom{n}{2}}$ be a vector that only considers edges incident to v in the complete graph. Essentially the row of v in the adjacency matrix.

Sketching for graphs

Consider vector $f \in \mathbb{R}^{\binom{n}{2}}$ where $f_i \in \{0, 1\}$ indicating whether edge i in the complete graph on n nodes is in the graph or not.

Example:

Sketching f is not adequate for most graph applications. We need information about edges incident to each vertex.

For node v let $f_v \in \mathbb{R}^{\binom{n}{2}}$ be a vector that only considers edges incident to v in the complete graph. Essentially the row of v in the adjacency matrix. Why use $\binom{n}{2}$ dimensions?

Sketching for graphs

Consider vector $f \in \mathbb{R}^{\binom{n}{2}}$ where $f_i \in \{0, 1\}$ indicating whether edge i in the complete graph on n nodes is in the graph or not.

Example:

Sketching f is not adequate for most graph applications. We need information about edges incident to each vertex.

For node v let $f_v \in \mathbb{R}^{\binom{n}{2}}$ be a vector that only considers edges incident to v in the complete graph. Essentially the row of v in the adjacency matrix. Why use $\binom{n}{2}$ dimensions? To be able to use linear operations over different nodes.

We sketch each f_v using same sketch matrix M and this takes $O(n\text{polylog}(n))$ space.

Sketching for graphs: connectivity

For connectivity the following specific representation is useful.

Assume wlog that $V = [n]$

Define vector $a^{(i)}$ for node i of dimension $\binom{n}{2}$ as follows:

- $a^{(i)}(\{k, j\}) = 0$ if $i \neq k$ and $i \neq j$ (edge is not incident to i)
- $a^{(i)}(\{k, j\}) = 1$ if $i = k$ and $i < j$ (edge is incident to i and neighbor has higher index)
- $a^{(i)}(\{k, j\}) = -1$ if $i = j$ and $k < i$ (edge is incident to i and neighbor has higher index)

Sketching for graphs: connectivity

For connectivity the following specific representation is useful.

Assume wlog that $V = [n]$

Define vector $a^{(i)}$ for node i of dimension $\binom{n}{2}$ as follows:

- $a^{(i)}(\{k,j\}) = 0$ if $i \neq k$ and $i \neq j$ (edge is not incident to i)
- $a^{(i)}(\{k,j\}) = 1$ if $i = k$ and $i < j$ (edge is incident to i and neighbor has higher index)
- $a^{(i)}(\{k,j\}) = -1$ if $i = j$ and $k < i$ (edge is incident to i and neighbor has higher index)

Lemma

Suppose $S \subset [n]$ then $\sum_{i \in S} a^{(i)}$ is the representation for the node obtained by contracting S into a single node.

Example

Connectivity using sketching

Setting: stream of edge updates (e_i, Δ_i) where e_i specifies the end points and $\Delta_i \in \{-1, 1\}$ (insert or delete). Strict turnstile.

Want to know if G is connected at end of stream and find a spanning tree

Want to use $O(n \log^c n)$ space for some small c

Offline algorithm

Consider following “parallel” algorithm for spanning tree computation similar to Bourouvka’s algorithm for MST

- Start with each vertex in separate connected component
- In each round each connected component picks a single edge leaving it.
- All chosen edges added and connected components updated (equivalently shrink the connected components into a single node)
- Repeat until graph has a single connected component (or equivalently we have only one node)

Offline algorithm

Consider following "parallel" algorithm for spanning tree computation similar to Bourouvka's algorithm for MST

- Start with each vertex in separate connected component
- In each round each connected component picks a single edge leaving it.
- All chosen edges added and connected components updated (equivalently shrink the connected components into a single node)
- Repeat until graph has a single connected component (or equivalently we have only one node)

Algorithm terminates in $O(\log n)$ iterations.

Emulation via sketching

Focus on implementing the first iteration of the offline algorithm.

- Pick a sketching matrix M and keep sketches of $Ma^{(i)}$ for each $i \in [n]$ while edges are seen in the stream. Note: each edge $e = (i, j)$ updates $a^{(i)}$ and $a^{(j)}$.
- After seeing all edges use ℓ_0 sampling from the sketch to pick a non-zero coordinate from $a^{(i)}$ which corresponds to an edge incident to node i.

Sketch size is $O(n \log^c n)$ to enable correctness of ℓ_0 sampling with high probability.

Emulation via sketching

Focus on implementing the first iteration of the offline algorithm.

- Pick a sketching matrix M and keep sketches of $Ma^{(i)}$ for each $i \in [n]$ while edges are seen in the stream. Note: each edge $e = (i, j)$ updates $a^{(i)}$ and $a^{(j)}$.
- After seeing all edges use ℓ_0 sampling from the sketch to pick a non-zero coordinate from $a^{(i)}$ which corresponds to an edge incident to node i.

Sketch size is $O(n \log^c n)$ to enable correctness of ℓ_0 sampling with high probability.

We need to recurse after picking edges in first iteration and contract to create new contracted graph.

Emulation via sketching

Focus on implementing the first iteration of the offline algorithm.

- Pick a sketching matrix M and keep sketches of $Ma^{(i)}$ for each $i \in [n]$ while edges are seen in the stream. Note: each edge $e = (i, j)$ updates $a^{(i)}$ and $a^{(j)}$.
- After seeing all edges use ℓ_0 sampling from the sketch to pick a non-zero coordinate from $a^{(i)}$ which corresponds to an edge incident to node i.

Sketch size is $O(n \log^c n)$ to enable correctness of ℓ_0 sampling with high probability.

We need to recurse after picking edges in first iteration and contract to create new contracted graph. But contracted graph depends on sketch and we cannot make another pass!

Emulation via sketching

Focus on implementing the first iteration of the offline algorithm.

- Pick a sketching matrix M and keep sketches of $Ma^{(i)}$ for each $i \in [n]$ while edges are seen in the stream. Note: each edge $e = (i, j)$ updates $a^{(i)}$ and $a^{(j)}$.
- After seeing all edges use ℓ_0 sampling from the sketch to pick a non-zero coordinate from $a^{(i)}$ which corresponds to an edge incident to node i.

Sketch size is $O(n \log^c n)$ to enable correctness of ℓ_0 sampling with high probability.

We need to recurse after picking edges in first iteration and contract to create new contracted graph. But contracted graph depends on sketch and we cannot make another pass! Linearity to the rescue!

Emulation via sketching

Implementing two iterations of the offline algorithm

- Pick independent sketching matrices M_1 and M_2 and keep sketches for $M_1 a^{(i)}$ and $M_2 a^{(i)}$ for each i as before
- Let H be contracted graph obtained by using M_1 for first iteration
- Suppose S is a connected component that gets contracted to a node v. By lemma we have sketch for nodes in graph H! $M_2 a^{(v)} = \sum_{i \in S} M_2 a^{(i)}$.

Emulation via sketching

Implementing two iterations of the offline algorithm

- Pick independent sketching matrices M_1 and M_2 and keep sketches for $M_1 a^{(i)}$ and $M_2 a^{(i)}$ for each i as before
- Let H be contracted graph obtained by using M_1 for first iteration
- Suppose S is a connected component that gets contracted to a node v. By lemma we have sketch for nodes in graph H! $M_2 a^{(v)} = \sum_{i \in S} M_2 a^{(i)}$.

Question: Why do we need M_2? Can we not use M_1 itself?

Emulation via sketching

Implementing the offline algorithm

- Pick independent sketching matrices $M_1, M_2, \ldots, M_t$ where $t = O(\log n)$ and keep sketches for $M_j a^{(i)}$ for each node i and for each $1 \leq j \leq t$. Total space is $O(n \log^c n)$ since $t = O(\log n)$
- Use M_j, via linearity, for the contracted graph in iteration j to create graph for next iteration.

Emulation via sketching

Implementing the offline algorithm

- Pick independent sketching matrices $M_1, M_2, \ldots, M_t$ where $t = O(\log n)$ and keep sketches for $M_j a^{(i)}$ for each node i and for each $1 \leq j \leq t$. Total space is $O(n \log^c n)$ since $t = O(\log n)$
- Use M_j, via linearity, for the contracted graph in iteration j to create graph for next iteration.

Correctness requires that each iteration has high probability. Use union bound over iterations (since sketches are independent) and in each iteration use union bound over all vertices (using high probability of ℓ_0 sampling).