Graph Sketching, Matchings

Lecture 23

Nov 17, 2022

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 $M_r 1_i$ for vector 1_i (which has 1 in i'th coordinate and 0 in all other entries) can be computed efficiently from r.
- When x is updated to $x + \alpha 1_i$ we update sketch by $\alpha M_r 1_i$.
- 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).

Implications

Sketching gives a streaming algorithm that uses $O(n\text{polylog}(n))$ space and can with high probability output the connected components in the strict turnstile setting

Similar ideas can be used to compute cut sparsifiers in dynamic streams

Also implies a data structure with $O(n\text{polylog}(n))$ space and $O(\text{polylog}(n))$ time per edge update that gives randomized guarantees on connectivity maintanence. Others have built on this in various applications to *offline* algorithms

Original idea to Ahn, Guha, MacGregor.

Part II

Matchings

Matchings

Definition

A *matching* $M \subseteq E$ in a graph $G = (V, E)$ is a set of edges that do not intersect (share vertices).

Definition

A *matching* $M \subseteq E$ in a graph $G = (V, E)$ is a perfect matching if all vertices are matched.

Matchings

Definition

A *matching* $M \subseteq E$ in a graph $G = (V, E)$ is a set of edges that do not intersect (share vertices).

Definition

A *matching* $M \subseteq E$ in a graph $G = (V, E)$ is a perfect matching if all vertices are matched.

- Given a graph G does it have a perfect matching?
- Find a maximum cardinality matching.
- Find a maximum weight matching.
- Find a minimum cost perfect matching.
- Count number of (perfect) matchings.

Matching theory: extensive, fundamental in theory and practice, beautiful, $\cdots$

Algorithms

- Given a graph G does it have a perfect matching?
- Find a maximum cardinality matching.
- Find a maximum weight matching.
- Find a minimum cost perfect matching.
- Count number of (perfect) matchings.

All of the above solvable in polynomial time.

- Bipartite graphs: via flow techniques
- Non-bipartite/general graphs: more advanced techniques
- Classical topics in combinatorial optimization

Semi-streaming setting

Edges $e_1, e_2, \ldots, e_m$ come in some (adversarial) order

Questions:

- With $\tilde{O}(n)$ memory approximate maximum cardinality matching
- With $\tilde{O}(n)$ memory approximate maximum weight matching
- Multiple passes
- Estimate size of maximum cardinality matching
- $\cdots$

Substantial literature on upper and lower bounds

Maximum cardinality

Definition

A matching M is maximal if for all $e \in E \setminus M$, $M + e$ is not a matching.

Lemma

If M is maximal then $|M| \geq |M^*|/2$ for any matching M^*. Hence, a maximal matching is a $1/2$-approximation.

Maximal matching in streams

```
M = ∅
While (stream is not empty) do
    e is next edge in stream
    If (M + e) is a matching
        M ← M + e
EndWhile
Output M
```

Maximum-weight matching

Offline algorithm: greedy after sorting.

```
Sort edges such that w(e_1) ≥ w(e_2) ≥ ... ≥ w(e_m)
M = ∅
For (i = 1 to m) do
    If (M + e_i) is a matching
        M ← M + e_i
EndWhile
Output M
```

Maximum-weight matching

Offline algorithm: greedy after sorting.

```
Sort edges such that w(e_1) ≥ w(e_2) ≥ ... ≥ w(e_m)
M = ∅
For (i = 1 to m) do
    If (M + e_i) is a matching
        M ← M + e_i
EndWhile
Output M
```

Claim: $w(M) \geq w(M^*)/2$.

Maximum-weight matching

Offline algorithm: greedy after sorting.

```
Sort edges such that w(e_1) ≥ w(e_2) ≥ ... ≥ w(e_m)
M = ∅
For (i = 1 to m) do
    If (M + e_i) is a matching
        M ← M + e_i
EndWhile
Output M
```

Claim: $w(M) \geq w(M^*)/2$.

Streaming setting? Cannot sort!

Maximum-weight matching

```
M = ∅
For (i = 1 to m) do
    C = {e' ∈ M | e' ∩ e_i ≠ ∅}
    If (w(e_i) > w(C)) then
        M ← M − C + e_i
EndWhile
Output M
```

Maximum-weight matching

$M = \emptyset$
For ($i = 1$ to m) do
    $C = \{e' \in M \mid e' \cap e_i \neq \emptyset\}$
    If ($w(e_i) > w(C)$) then
        $M \leftarrow M - C + e_i$
EndWhile
Output M

Can be arbitrarily bad compared to optimum weight.

Maximum-weight matching

```
M = ∅
For (i = 1 to m) do
    C = {e' ∈ M | e' ∩ e_i ≠ ∅}
    If (w(e_i) > (1 + γ)w(C)) then
        M ← M − C + e_i
EndWhile
Output M
```

Maximum-weight matching

```
M = ∅
For (i = 1 to m) do
     C = {e' ∈ M | e' ∩ e_i ≠ ∅}
     If (w(e_i) > (1 + γ)w(C)) then
          M ← M − C + e_i
EndWhile
Output M
```

Theorem

$w(M) \geq f(\gamma) w(M^*)$.

Analysis

Consider edge $e \in M$ at end of algorithm. Let T_e set of edges in G that were "killed" by e.

Analysis

Consider edge $e \in M$ at end of algorithm. Let T_e set of edges in G that were "killed" by e.

Claim: $w(T_e) \leq w(e)/\gamma$.

Analysis

Consider edge $e \in M$ at end of algorithm. Let T_e set of edges in G that were "killed" by e.

Claim: $w(T_e) \le w(e)/\gamma$.

$e = C_0$ killed C_1 which killed $C_2 \ldots$ killed C_h

$w(C_i) \ge (1+\gamma)w(C_{i+1})$ for $i \ge 0$ and adding up

$w(e) + w(T_e) \ge (1+\gamma)w(T_e)$

Analysis

Claim: $w(M^*) \leq (1+\gamma)\sum_{e \in M}(w(T_e) + 2w(e))$.

Analysis

Claim: $w(M^*) \leq (1+\gamma)\sum_{e \in M}(w(T_e) + 2w(e))$.

Fix any $f \in M^*$

- If $f \in M$ at some point then $f \in T_e$ for some $e \in M$, or $f \in M$. Charge f to itself.
- Else, when f considered it was not added to M. Let C_f conflicting edges at that time. $w(f) \leq (1+\gamma)w(C_f)$.
  - If $|C_f| = 1$ charge f to single edge $e \in C_f$.
  - If $|C_f| = 2$ charge f in proportion to weights of edges in C_f.
  - If f charges e' and e' gets killed by e'', transfer charge of f from e' to e''.

Analysis

Claim: $w(M^*) \leq (1+\gamma)\sum_{e\in M}(w(T_e) + 2w(e))$.

Fix any $f \in M^*$

- If $f \in M$ at some point then $f \in T_e$ for some $e \in M$, or $f \in M$. Charge f to itself.
- Else, when f considered it was not added to M. Let C_f conflicting edges at that time. $w(f) \leq (1+\gamma)w(C_f)$.
  - If $|C_f| = 1$ charge f to single edge $e \in C_f$.
  - If $|C_f| = 2$ charge f in proportion to weights of edges in C_f.
  - If f charges e' and e' gets killed by e'', transfer charge of f from e' to e''.
- If $e \in M$ can be charged twice hence total is $2(1+\gamma)w(e)$

Analysis

Claim: $w(M^*) \leq (1+\gamma)\sum_{e \in M}(w(T_e) + 2w(e))$.

Fix any $f \in M^*$

- If $f \in M$ at some point then $f \in T_e$ for some $e \in M$, or $f \in M$. Charge f to itself.
- Else, when f considered it was not added to M. Let C_f conflicting edges at that time. $w(f) \leq (1+\gamma)w(C_f)$.
  - If $|C_f| = 1$ charge f to single edge $e \in C_f$.
  - If $|C_f| = 2$ charge f in proportion to weights of edges in C_f.
  - If f charges e' and e' gets killed by e'', transfer charge of f from e' to e''.
- If $e \in M$ can be charged twice hence total is $2(1+\gamma)w(e)$
- If $e' \in T_e$ then only one edge of M^* leaves charge on e'. Why?

Analysis

Claim: $w(T_e) \leq w(e)/\gamma$.

Claim: $w(M^*) \leq (1+\gamma)\sum_{e \in M}(w(T_e) + 2w(e))$.

Setting $\gamma = 1$ we obtain $w(M^*) \leq 6w(M)$.

Another algorithm/approach for weighted matching

We describe another algorithm for weighted matching that uses the unweighted matching algorithm as a black box

We make some assumptions that can be gotten rid of with more care

- Smallest edge weights is at least 1, that is, $\min_e w(e) \geq 1$.
- Largest weight edges is polynomially bounded in n, that is, $\max_e w(e) \leq n^c$

Another algorithm/approach for weighted matching

We will describe the algorithm as an offline algorithm first.

Algorithm($G = (V, E)$)
    Assume edge weights are in $[1, W]$
    For $i = 1$ to $k = O(\frac{1}{\epsilon} \log W)$ do
        $E_i = \{e \mid w(e) \geq (1+\epsilon)^i\}$
        Let M_i be a maximal matching in $G_i = (V, E_i)$

    $M = \emptyset$
    For $i = k$ down to 1 do
        For each ($e \in M_i$) do
            If $M + e$ is a matching then $M \leftarrow M + e$
    Output M

Another algorithm/approach for weighted matching

We will describe the algorithm as an offline algorithm first.

```
Algorithm(G = (V, E))
    Assume edge weights are in [1, W]
    For i = 1 to k = O((1/ε) log W) do
        E_i = {e | w(e) ≥ (1 + ε)^i}
        Let M_i be a maximal matching in G_i = (V, E_i)

    M = ∅
    For i = k down to 1 do
        For each (e ∈ M_i) do
            If M + e is a matching then M ← M + e
    Output M
```

Exercise: Show that algorithm above can be implemented in streaming setting with space $O(n\frac{\log W}{\epsilon})$.

Analysis

Theorem

Algorithm outputs a matching M such that $w(M) \geq \frac{1}{4(1+\epsilon)} OPT$.

Analysis

Theorem

Algorithm outputs a matching M such that $w(M) \geq \frac{1}{4(1+\epsilon)} OPT$.

Assume weights are power of $(1+\epsilon)^i$ with a loss of $(1+\epsilon)$ factor.

Let M^* be an optimum matching and let $M_i^* = M^* \cap E_i$.

Claim $|M_i| \geq |M_i^*|/2$ since M_i is a maximal matching in E_i

Analysis continued

Let $C_i = M \cap E_i$ be the set of edges in the output from E_i

Claim: $|C_i| \geq |M_i|/2 \geq |M_i^*|/4$
since C_i is a maximal matching in M_i

Exercise: The preceding claim yields the theorem.

Other results

There is a clever and simple $(\frac{1}{2} - \epsilon)$-approximation [Paz-Schwartzman'17]

Many other results on matchings in streaming: multipass, random arrival order, lower bounds, ...