Graph Streaming and Sketching

Lecture 19
April 2, 2019
Part I

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, ···
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
- \(\ldots\)

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

\[ M = \emptyset \]

While (stream is not empty) do
  \( e \) is next edge in stream
  If \((M + e)\) is a matching
    \[ M \leftarrow M + e \]
  EndWhile
Output \( M \)
Maximum-weight matching

Offline algorithm: greedy after sorting.

Sort edges such that $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_m)$

$M = \emptyset$

For ($i = 1$ to $m$) do

- If $(M + e_i)$ is a matching

  $M \leftarrow M + e_i$

EndWhile

Output $M$

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

Streaming setting? Cannot sort!
Maximum-weight matching

Offline algorithm: greedy after sorting.

Sort edges such that $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_m)$

$M = \emptyset$

For ($i = 1$ to $m$) do

If ($M + e_i$) is a matching

$M \leftarrow M + e_i$

EndWhile

Output $M$

Claim: $w(M) \geq w(M^*)/2$. 
Maximum-weight matching

Offline algorithm: greedy after sorting.

Sort edges such that \( w(e_1) \geq w(e_2) \geq \ldots \geq w(e_m) \)

\[ M = \emptyset \]

For \( i = 1 \) to \( m \) do

If \((M + e_i)\) is a matching

\[ M \leftarrow M + e_i \]

EndWhile

Output \( M \)

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

Streaming setting? Cannot sort!
Maximum-weight matching

\( M = \emptyset \)

For \( (i = 1 \text{ to } m) \) 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 \( M \)
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

\[ M = \emptyset \]

For \( i = 1 \) to \( m \) do

\[ C = \{ e' \in M \mid e' \cap e_i \neq \emptyset \} \]

If \( w(e_i) > (1 + \gamma)w(C) \) then

\[ M \leftarrow M - C + e_i \]

EndWhile

Output \( M \)
Maximum-weight matching

\[ M = \emptyset \]

For \((i = 1 \text{ to } m)\) do

\[ C = \{e' \in M \mid e' \cap e_i \neq \emptyset\} \]

If \((w(e_i) > (1 + \gamma)w(C))\) then

\[ M \leftarrow M - C + e_i \]

EndWhile

Output \(M\)

Theorem

\[ w(M) \geq f(\gamma)w(M^*) . \]
Consider edge $e \in M$ at end of algorithm. Let $T_e$ set of edges in $G$ that were “killed” by $e$. 

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

$e = C_0 \text{ killed } C_1 \text{ killed } C_2 \ldots \text{ killed } C_h$

$w(C_i) \geq (1 + \gamma)w(C_{i+1})$ for $i \geq 0$ and adding up $w(e) + w(T_e) \geq (1 + \gamma)w(T_e)$. 

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

Claim: $w(T_e) \leq w(e)/\gamma$. 
Consider edge $e \in M$ at end of algorithm. Let $T_e$ set of edges in $G$ that were “killed” by $e$.

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

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

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

$w(e) + w(T_e) \geq (1 + \gamma)w(T_e)$
Claim: \( w(M^*) \leq (1 + \gamma) \sum_{e \in M} (w(T_e) + 2w(e)) \).
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.
- 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.
- 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 \( 2(1 + \gamma)w(e) \)
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.
- 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?
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 \( w(M^*) \leq 6w(M). \)
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 \( w(M^*) \leq 6w(M) \).

A clever and simple \((\frac{1}{2} - \epsilon)\)-approximation [Paz-Schwartzman’17]
Stores more than a matching and then postprocesses.

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

Cut Sparsifiers
Graph Sparsification

\( G = (V, E) \) input graph and could be dense
- \( n \) is reasonable to store
- \( n^2 \) may be unreasonable to store
- edges are sometimes implicit and may be generated on the fly

**Sparsification:** Given \( G = (V, E) \) create a *sparse* graph \( H = (V, F) \) such that \( H \) mimics \( G \) for some property of interest
Graph Sparsification

\( G = (V, E) \) input graph and could be dense
- \( n \) is reasonable to store
- \( n^2 \) may be unreasonable to store
- edges are sometimes implicit and may be generated on the fly

**Sparsification**: Given \( G = (V, E) \) create a *sparse* graph \( H = (V, F) \) such that \( H \) mimics \( G \) for some property of interest
- Connectivity
- Distances (spanners and variants)
- Cuts (cut sparsifiers)
- ...
**Cut Sparsifier**

**Definition**

Given an edge weighted graph \( G = (V, E) \) with \( w : E \to \mathbb{R}_+ \) an edge weighted graph \( H = (V, F) \) with \( w' : F \to \mathbb{R}_+ \) is an \( \epsilon \)-approximate cut sparsifier if for all \( S \subset V \),

\[
(1 - \epsilon)w(\delta_G(S)) \leq w'(\delta_H(S)) \leq (1 + \epsilon)w(\delta_G(S)).
\]
Cut Sparsifier

Definition

Given an edge weighted graph $G = (V, E)$ with $w : E \rightarrow \mathbb{R}_+$ an edge weighted graph $H = (V, F)$ with $w' : F \rightarrow \mathbb{R}_+$ is an $\epsilon$-approximate cut sparsifier if for all $S \subseteq V$, 

$$(1 - \epsilon)w(\delta_G(S)) \leq w'(\delta_H(S)) \leq (1 + \epsilon)w(\delta_G(S)).$$

Very important concept and many powerful applications in graph algorithms and beyond.
Definition

Given an edge weighted graph $G = (V, E)$ with $w : E \rightarrow \mathbb{R}_+$ an edge weighted graph $H = (V, F)$ with $w' : F \rightarrow \mathbb{R}_+$ is an $\epsilon$-approximate cut sparsifier if for all $S \subset V$, 

$$(1 - \epsilon)w(\delta_G(S)) \leq w'(\delta_H(S)) \leq (1 + \epsilon)w(\delta_G(S)).$$
Fundamental results

**Theorem (Benczur-Karger’00)**

*Given a graph $G = (V, E)$ on $m$ edges and $n$ nodes and any $\epsilon > 0$, one can construct in randomized $O(m \log^3 n)$ time a cut-sparsifier with $O\left(\frac{1}{\epsilon^2} n \log n\right)$ edges.*

**Theorem (Batson-Spielman-Srivastava’08)**

*Given a graph $G = (V, E)$ on $m$ edges and $n$ nodes and any $\epsilon > 0$, one can construct in deterministic polynomial time a cut-sparsifier with $O\left(\frac{1}{\epsilon^2} n\right)$ edges.*
Fundamental results

**Theorem (Benczur-Karger’00)**

Given a graph $G = (V, E)$ on $m$ edges and $n$ nodes and any $\epsilon > 0$, one can construct in randomized $O(m \log^3 n)$ time a cut-sparsifier with $O(\frac{1}{\epsilon^2} n \log n)$ edges.

**Theorem (Batson-Spielman-Srivastava’08)**

Given a graph $G = (V, E)$ on $m$ edges and $n$ nodes and any $\epsilon > 0$, one can construct in deterministic polynomial time a cut-sparsifier with $O(\frac{1}{\epsilon^2} n)$ edges.

What is a cut-sparsifier of a complete graph $K_n$?
Fundamental results

Theorem (Benczur-Karger’00)

Given a graph $G = (V, E)$ on $m$ edges and $n$ nodes and any $\epsilon > 0$, one can construct in randomized $O(m \log^3 n)$ time a cut-sparsifier with $O\left(\frac{1}{\epsilon^2} n \log n\right)$ edges.

Theorem (Batson-Spielman-Srivastava’08)

Given a graph $G = (V, E)$ on $m$ edges and $n$ nodes and any $\epsilon > 0$, one can construct in deterministic polynomial time a cut-sparsifier with $O\left(\frac{1}{\epsilon^2} n\right)$ edges.

What is a cut-sparsifier of a complete graph $K_n$? An expander graph!
Cut sparsifiers in streaming

**Question:** Can we create a cut-sparsifier on the fly in roughly $O(n \text{polylog}(n))$ space as edges come by?

Can use cut-sparsifier algorithms as a black box.
Merge and Reduce

Observation (Merge): If $H_1 = (V, F_1)$ is a $\alpha$-approximate sparsifier for $G_1 = (V, E_1)$ and $H_2 = (V, F_2)$ is a $\alpha$-approximate cut-sparsifier for $G_2 = (V, E_2)$ then $H_1 \cup H_2 = (V, F_1 \cup F_2)$ is a $\alpha$-approximate cut-sparsifier for $G_1 \cup G_2 = (V, E_1 \cup E_2)$. 
Observation (Merge): If $H_1 = (V, F_1)$ is a $\alpha$-approximate sparsifier for $G_1 = (V, E_1)$ and $H_2 = (V, F_2)$ is a $\alpha$-approximate cut-sparsifier for $G_2 = (V, E_2)$ then $H_1 \cup H_2 = (V, F_1 \cup F_2)$ is a $\alpha$-approximate cut-sparsifier for $G_1 \cup G_2 = (V, E_1 \cup E_2)$.

Observation (Reduce): If $H = (V, F)$ is a $\alpha$-approximate sparsifier for $G = (V, E_1)$ and $H' = (V, F')$ is a $\beta$-approximate cut-sparsifier for $H$ then $H'$ is a $(\alpha \beta)$-approximate cut-sparsifier for $G$. 
Cut sparsifiers in streaming

**Question:** Can we create a cut-sparsifier on the fly in roughly $O(n \text{polylog}(n))$ space as edges come by?

Can use cut-sparsifier algorithms as a black box.

Merge and Reduce via a binary tree approach over the $m$ edges in the stream. Seen this approach twice already: range queries in CountMin sketch and quantile summaries.
Cut sparsifiers in streaming

- Split stream of $m$ edges into $k$ graphs of $m/k$ edges each. Let $G_1, G_2, \ldots, G_k$ be the $k$ graphs. Assume for simplicity that $k$ is a power of 2.
- Imagine a binary tree with $G_1, \ldots, G_k$ as leaves.
- Build a sparsifier bottom up. At each internal node merge the sparsifiers and reduce with approximation $\alpha$.
Cut sparsifiers in streaming

- Split stream of $m$ edges into $k$ graphs of $m/k$ edges each. Let $G_1, G_2, \ldots, G_k$ be the $k$ graphs. Assume for simplicity that $k$ is a power of 2.
- Imagine a binary tree with $G_1, \ldots, G_k$ as leaves.
- Build a sparsifier bottom up. At each internal node merge the sparsifiers and reduce with approximation $\alpha$.

Questions:
- What is $\alpha$ to ensure that final sparsifier is $\epsilon$-approximate?
- How much space needed in streaming setting?
Cut sparsifiers in streaming

- What is $\alpha$ to ensure that final sparsifier is $\epsilon$-approximate?
- How much space needed in streaming setting?

Depth of tree is $\leq \log(m/n) \leq \log n$. Due to reduce operations final approximation is $(1 + \alpha)^d$. Hence $(1 + \alpha)^d \leq (1 + \epsilon)$ implies $\alpha \simeq \epsilon/(ed) \simeq \epsilon/(e \log n)$
Cut sparsifiers in streaming

- What is $\alpha$ to ensure that final sparsifier is $\epsilon$-approximate?
- How much space needed in streaming setting?

Depth of tree is $\leq \log(m/n) \leq \log n$. Due to reduce operations final approximation is $(1 + \alpha)^d$. Hence $(1 + \alpha)^d \leq (1 + \epsilon)$ implies $\alpha \approx \epsilon/(ed) \approx \epsilon/(e \log n)$

Memory analysis: Sparsifier size with $\alpha = \epsilon/\log n$ is $O(n \log^2 n/\epsilon^2)$ (if one uses BSS sparsifier, otherwise another log factor for Benczur-Karger sparsifier).
Need another $\log n$ factor to store sparsifiers at $\log n$ levels for streaming.