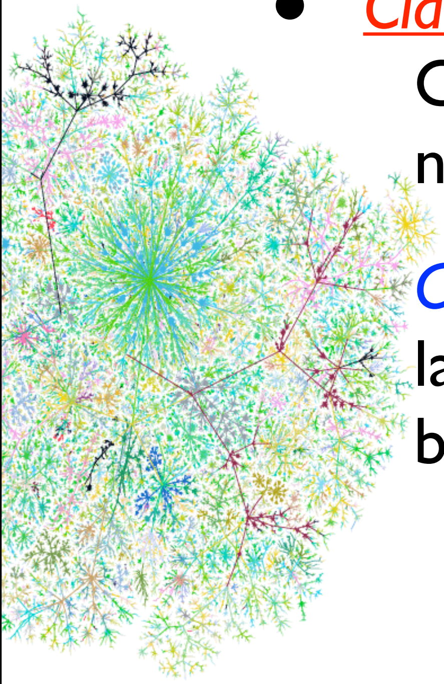
Sampling, Sketching, Streaming, Small-Space Optimization: Algorithmic Approaches *for* Analyzing Large Graphs

Sudipto Guha
Amazon

Andrew McGregor
University of Massachusetts, Amherst



- *Classic Big Graphs*

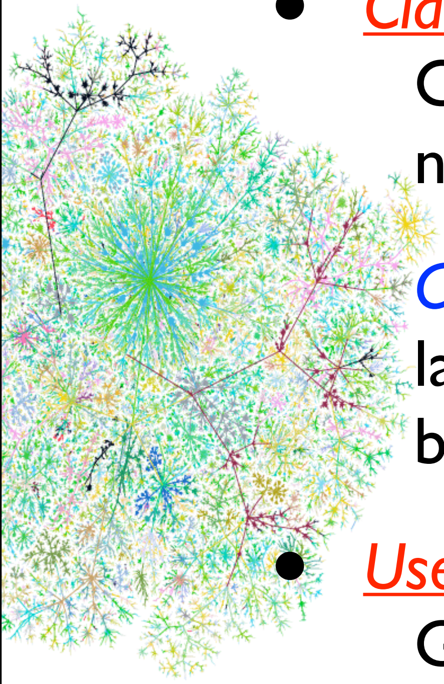
  Call graph, web graph, IP graph, social networks, citation networks, protein interaction and metabolic networks....

  Challenge: Can’t use conventional algorithms on graphs this large. Often can’t even store graph in memory. Graphs may be changing over time and data may be distributed.

- *Classic Big Graphs*

  Call graph, web graph, IP graph, social networks, citation networks, protein interaction and metabolic networks....

  Challenge: Can't use conventional algorithms on graphs this large. Often can't even store graph in memory. Graphs may be changing over time and data may be distributed.



- *Use Abstraction of Structure*

  Gives a natural way to encode structural information when there's data about both *basic entities* and their *relationships*.

- ***Classic Big Graphs***

  Call graph, web graph, IP graph, social networks, citation networks, protein interaction and metabolic networks....

  Challenge: Can't use conventional algorithms on graphs this large. Often can't even store graph in memory. Graphs may be changing over time and data may be distributed.

- ***Use Abstraction of Structure***

  Gives a natural way to encode structural information when there's data about both *basic entities* and their *relationships*.

- Want streaming, parallel, distributed algorithms…

- ***Tutorial Goals and Caveats***

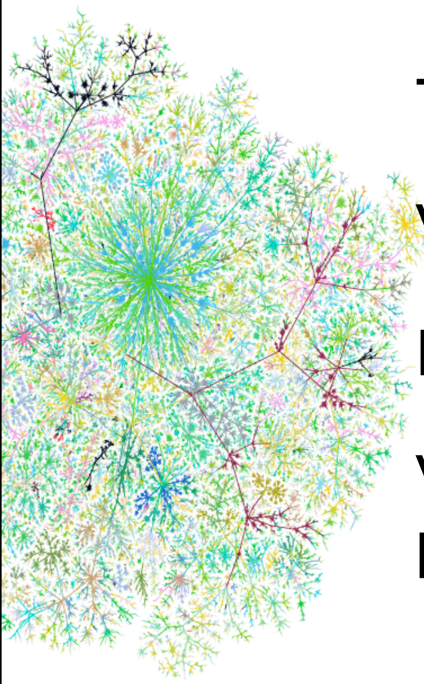
Present some new algorithmic primitives for large graphs.

Techniques are widely applicable; we'll be platform agnostic.

Won't be comprehensive; will cherry pick illustrative results.

Focus on arbitrary graphs rather than specific applications.

Won't focus on proofs but will give basic outline when it helps convey why certain approaches are effective.



- ***Tutorial Goals and Caveats***

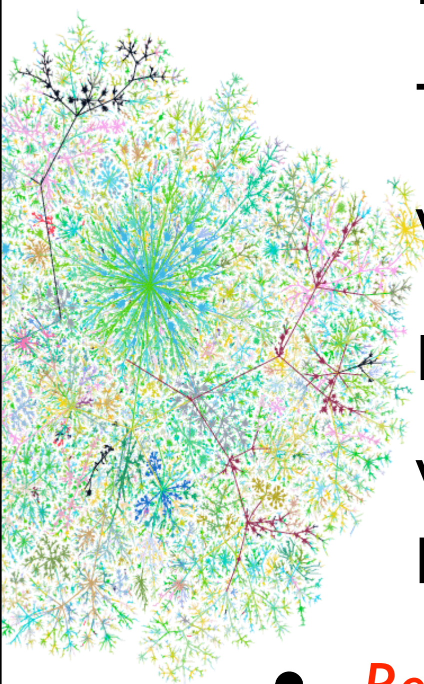
  Present some new algorithmic primitives for large graphs.

  Techniques are widely applicable; we'll be platform agnostic.

  Won't be comprehensive; will cherry pick illustrative results.

  Focus on arbitrary graphs rather than specific applications.

  Won't focus on proofs but will give basic outline when it helps convey why certain approaches are effective.



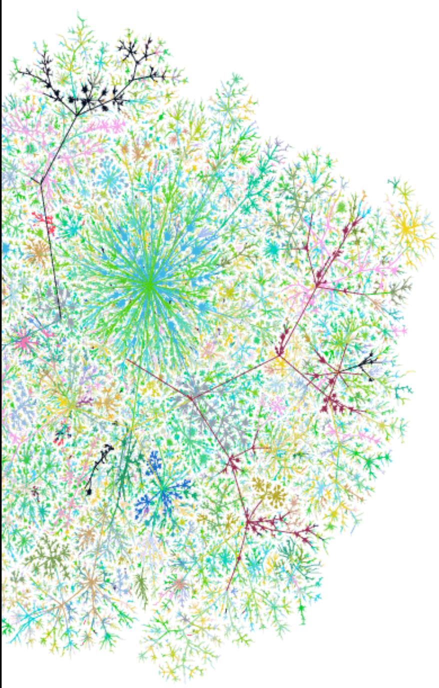
- ***Resources***

Survey: SIGMOD Record
http://people.cs.umass.edu/~mcgregor/papers/graphsurvey.pdf

Tutorial: Slides and Bibliography
http://people.cs.umass.edu/~mcgregor/graphs

Lectures: Ten Lectures on Graph Streams
https://people.cs.umass.edu/~mcgregor/courses/CS711S18/

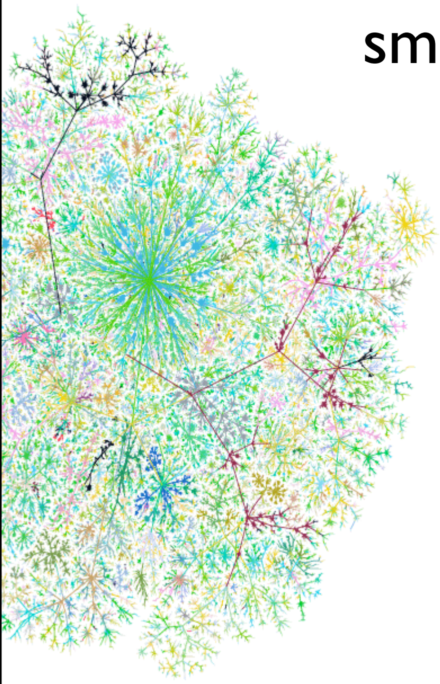
Overview



Overview

- *Part 1: Sampling* Sampling for finding densest subgraphs, small matchings, triangles, spectral properties.

  "Different sampling techniques for different problems"



Overview

- *Part I: Sampling* Sampling for finding densest subgraphs, small matchings, triangles, spectral properties.

  "Different sampling techniques for different problems"

- *Part II: Sketching* Dimensionality reduction for graph data. Examples include connectivity and sparsification.

  "Homomorphic compression: sketch first and then run algorithms on the sketched data"

Overview

- *Part I: Sampling* Sampling for finding densest subgraphs, small matchings, triangles, spectral properties.

  "Different sampling techniques for different problems"

- *Part II: Sketching* Dimensionality reduction for graph data. Examples include connectivity and sparsification.

  "Homomorphic compression: sketch first and then run algorithms on the sketched data"

- *Part III: Streaming* What can you compute in limited memory with only a few passes over the edges.

  "A little inspiration yields a lot less iteration"

Overview

- *Part I: Sampling* Sampling for finding densest subgraphs, small matchings, triangles, spectral properties.

  "Different sampling techniques for different problems"

- *Part II: Sketching* Dimensionality reduction for graph data. Examples include connectivity and sparsification.

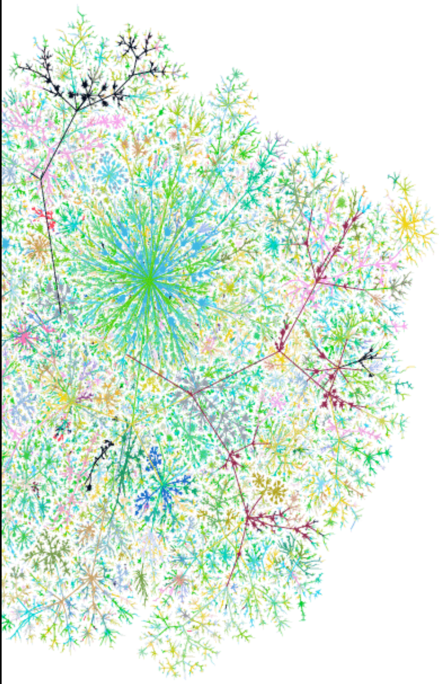
  "Homomorphic compression: sketch first and then run algorithms on the sketched data"

- *Part III: Streaming* What can you compute in limited memory with only a few passes over the edges.

  "A little inspiration yields a lot less iteration"

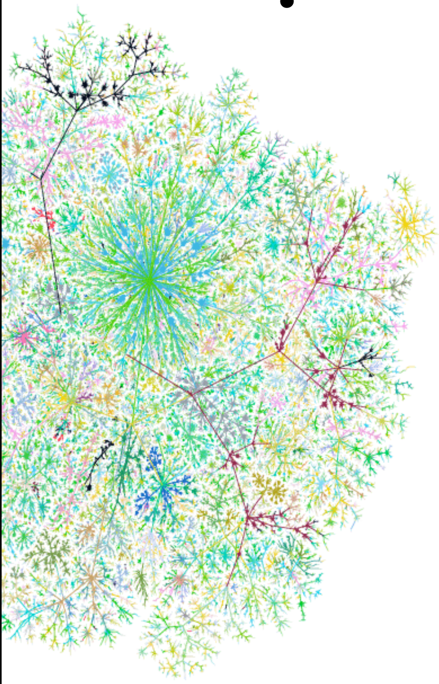
- *Part IV: Small-Space Optimization* Combining sparsification and multiplicative weights for fast, small-space optimization. Examples include large matching and correlation clustering.

Recurring Theme



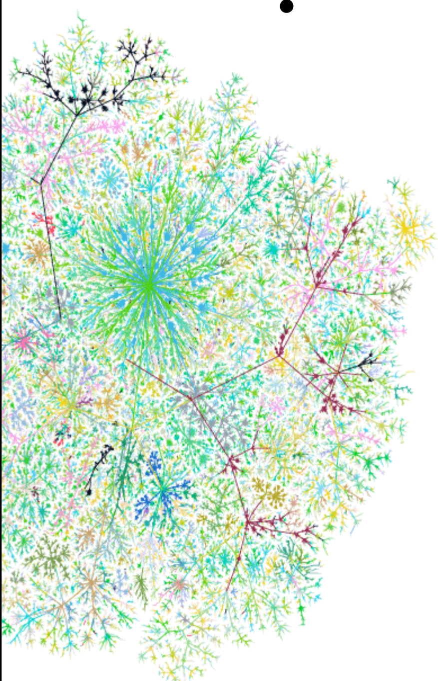
Recurring Theme

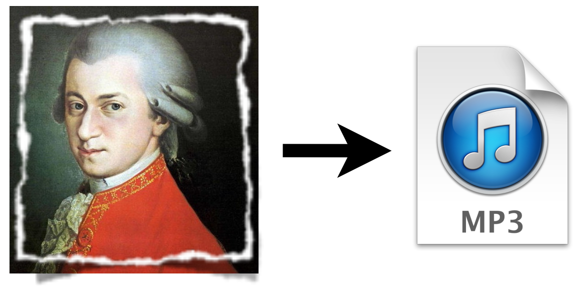
- What's appropriate notion of lossy compression for graphs?



Recurring Theme

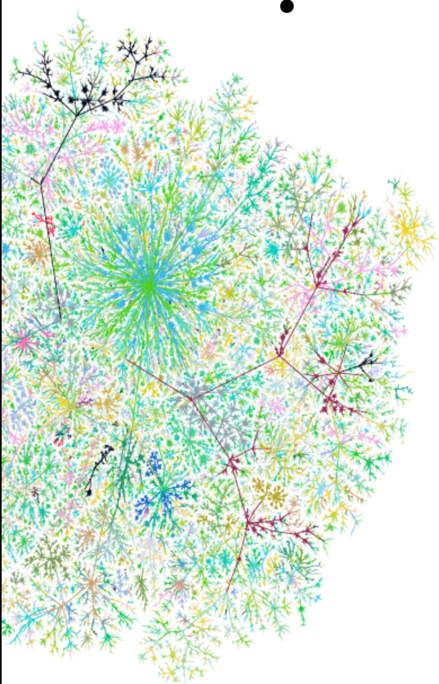
? What's appropriate notion of lossy compression for graphs?

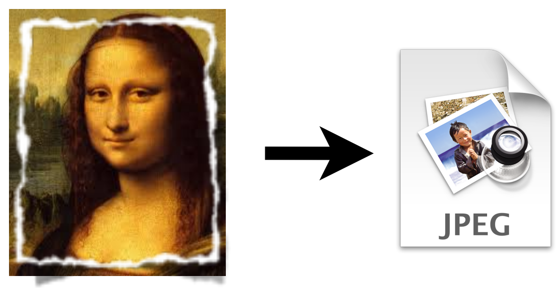




Recurring Theme

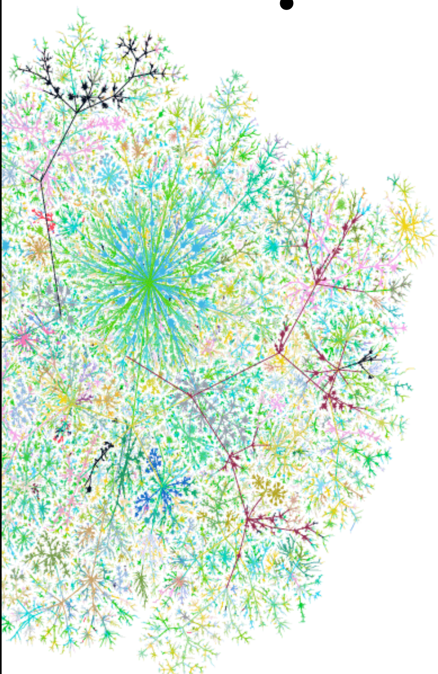
- What's appropriate notion of lossy compression for graphs?

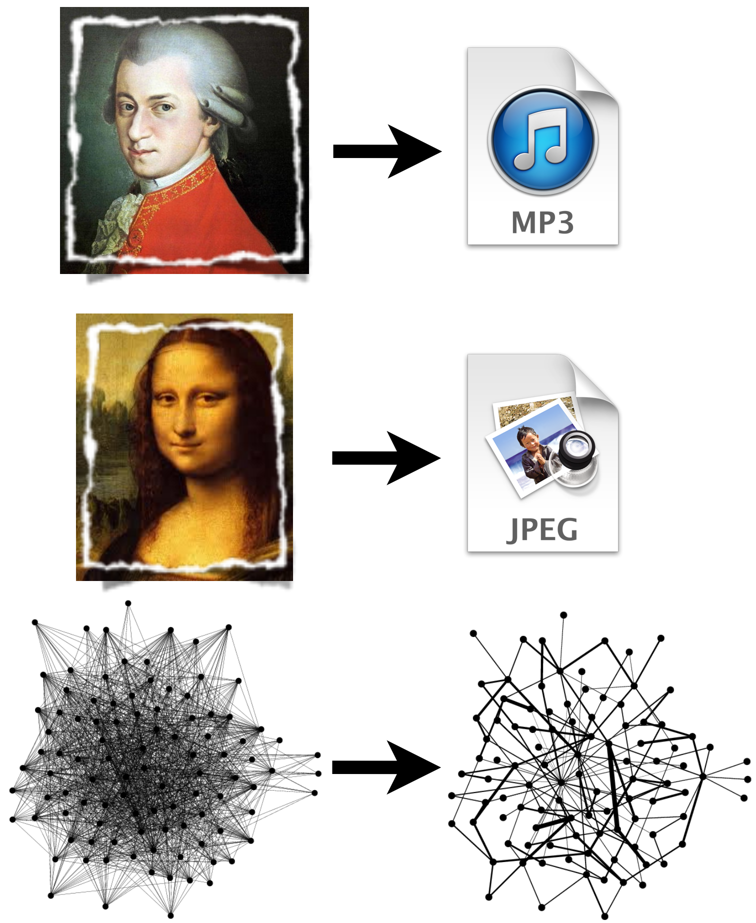




Recurring Theme

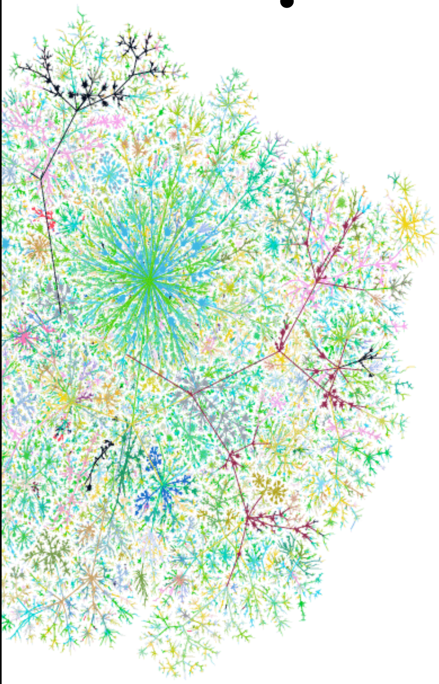
? What's appropriate notion of lossy compression for graphs?

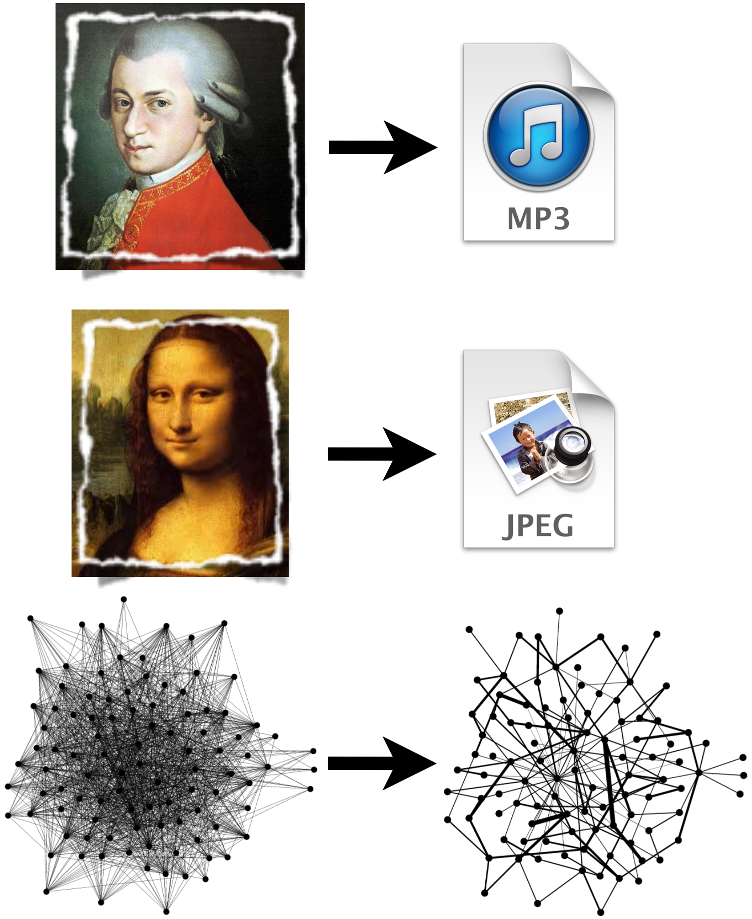




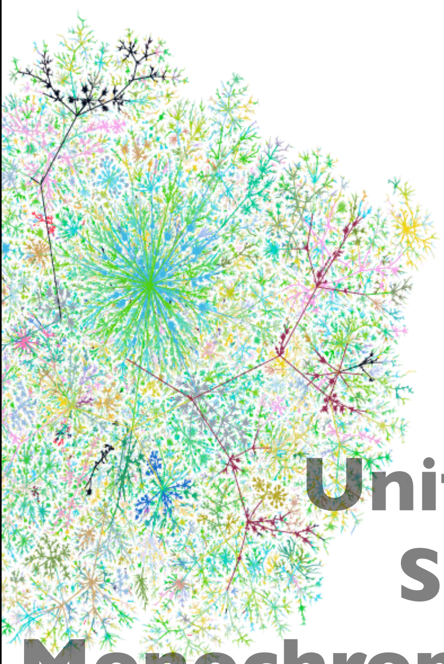
Recurring Theme

? What's appropriate notion of lossy compression for graphs?





- If compression is easy, we get faster and more-space efficient algorithms by using existing algorithms on compressed graphs.



Part I

Sampling

Uniform Sampling + Densest Subgraph
Snape Sampling + Matching
Monochromatic Sampling + Clustering Coefficient
Edge-Weighted Sampling + Cuts and Sparsification

Part I

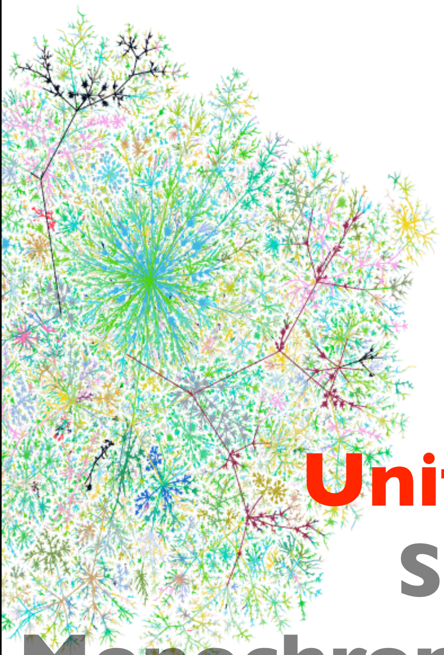
Sampling

Uniform Sampling + Densest Subgraph

Snape Sampling + Matching

Monochromatic Sampling + Clustering Coefficient

Edge-Weighted Sampling + Cuts and Sparsification



- Given a graph G, the *density* of a set of nodes $S \subset V$ is:

$$D_S = \frac{\#\text{ of edges with both endpoints in } S}{\#\text{ of nodes in } S}$$

- Given a graph G, the *density* of a set of nodes $S \subset V$ is:

$$D_S = \frac{\#\text{ of edges with both endpoints in } S}{\#\text{ of nodes in } S}$$

- *Problem* Estimating $D^* = \max_S D_S$ is a basic graph problem with numerous applications. Studied in a variety of models.

See tutorial *Gionis, Tsourakakis* [KDD 15]

- Given a graph G, the *density* of a set of nodes $S \subset V$ is:

$$D_S = \frac{\#\text{ of edges with both endpoints in } S}{\#\text{ of nodes in } S}$$

- *Problem* Estimating $D^* = \max_S D_S$ is a basic graph problem with numerous applications. Studied in a variety of models.

See tutorial *Gionis, Tsourakakis* [KDD 15]

- *Thm* Sample of $\tilde{O}(\varepsilon^{-2} n)$ edges uniformly and find the densest subgraph in sampled graph. Gives a $(1+\varepsilon)$-approx whp.

McGregor et al. [MFCS 15], *Esfandiari et al.* [SPAA 16]

Mitzenmacher et al. [KDD 15]

Why Uniform Sampling Works...

Why Uniform Sampling Works…

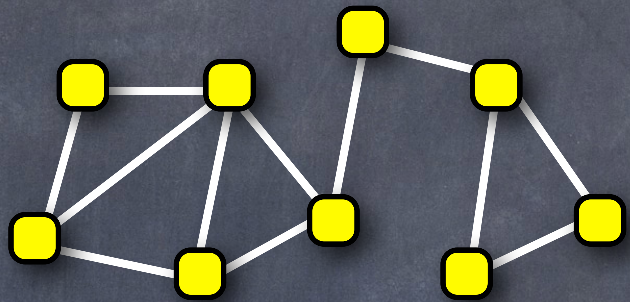
- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.

Why Uniform Sampling Works…

- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2}n/m$.
- Let D'_S be density of S in sampled graph.

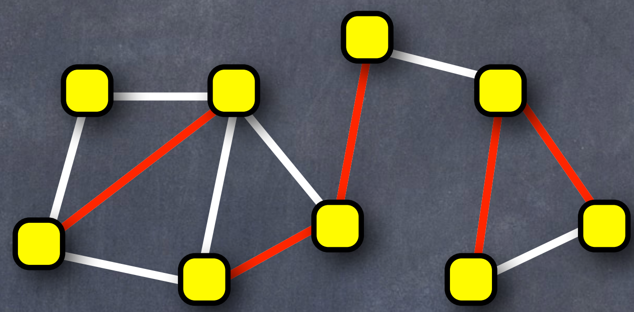
Why Uniform Sampling Works...

- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.



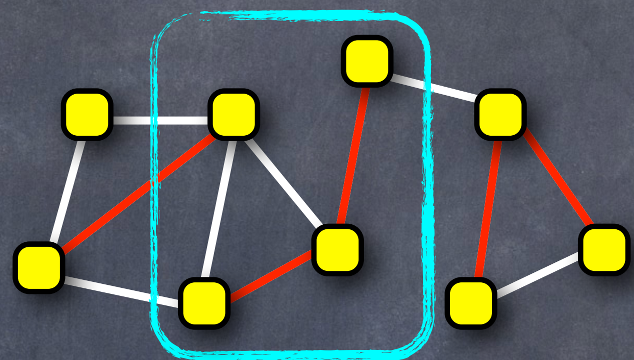
Why Uniform Sampling Works…

- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.



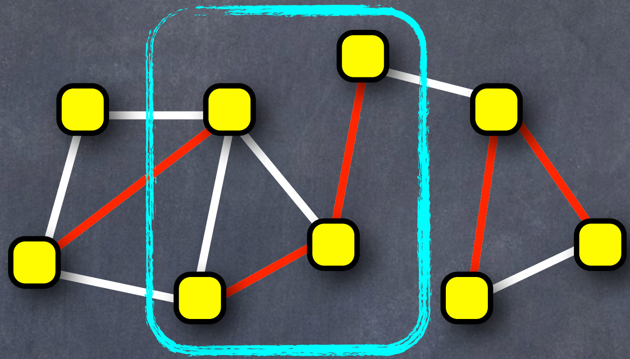
Why Uniform Sampling Works…

- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.



Why Uniform Sampling Works...

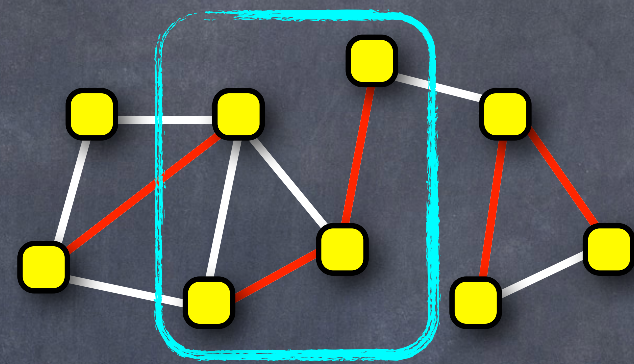
- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.



$D_S = 1.0$
$D'_S = 0.5$

Why Uniform Sampling Works...

- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.

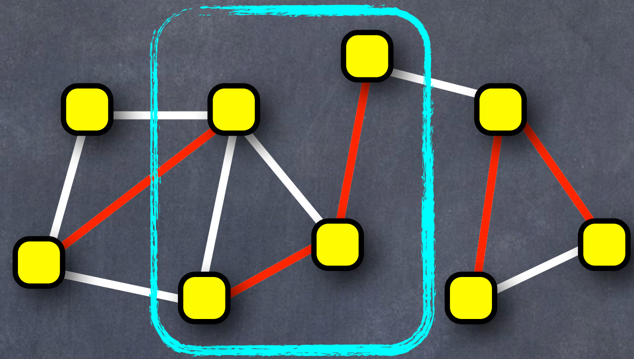


$D_S = 1.0$
$D'_S = 0.5$

- Chernoff: For each S, $D_S = D'_S/p \pm \varepsilon D^*$ w/p $1-n^{-3|S|}$

Why Uniform Sampling Works...

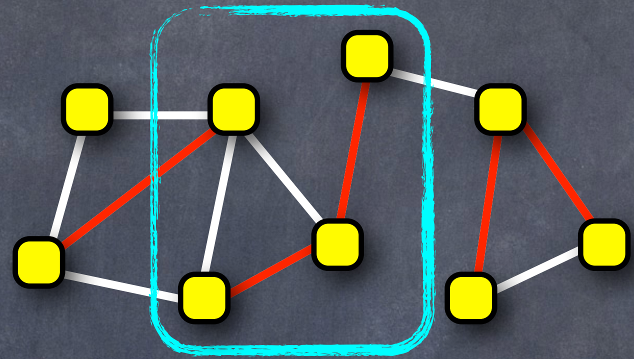
- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.



- Chernoff: For each S, $D_S = D'_S/p \pm \varepsilon D^*$ w/p $1-n^{-3|S|}$
- Union Bound: Bound applies for all S w/p $1-n^{-1}$

Why Uniform Sampling Works...

- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.

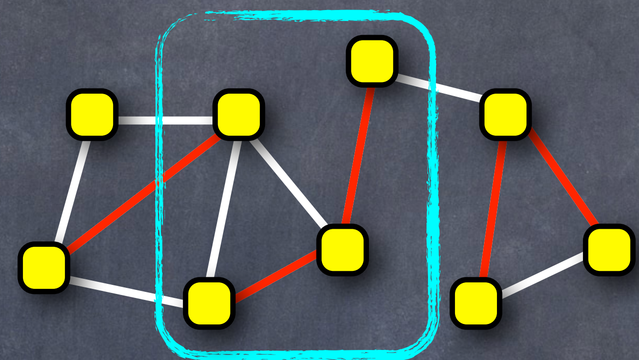


$D_S = 1.0$
$D'_S = 0.5$

- Chernoff: For each S, $D_S = D'_S/p \pm \varepsilon D^*$ w/p $1-n^{-3|S|}$
- Union Bound: Bound applies for all S w/p $1-n^{-1}$
  - There are $\leq n^k$ subsets of k nodes. So bound fails for some subset of size k w/p $\leq n^k n^{-3k} = n^{-2k}$

Why Uniform Sampling Works…

- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.

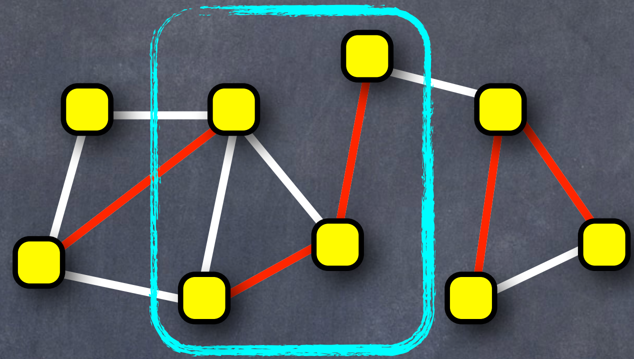


$D_S = 1.0$
$D'_S = 0.5$

- Chernoff: For each S, $D_S = D'_S/p \pm \varepsilon D^*$ w/p $1-n^{-3|S|}$
- Union Bound: Bound applies for all S w/p $1-n^{-1}$
  - There are $\leq n^k$ subsets of k nodes. So bound fails for some subset of size k w/p $\leq n^k n^{-3k} = n^{-2k}$
  - Bound fails for some subset w/p $\leq n^{-2}+n^{-4}+\ldots+n^{-2n} \leq n^{-1}$

Why Uniform Sampling Works…

- We're essentially sampling each edge w/p $p \approx \varepsilon^{-2} n/m$.
- Let D'_S be density of S in sampled graph.

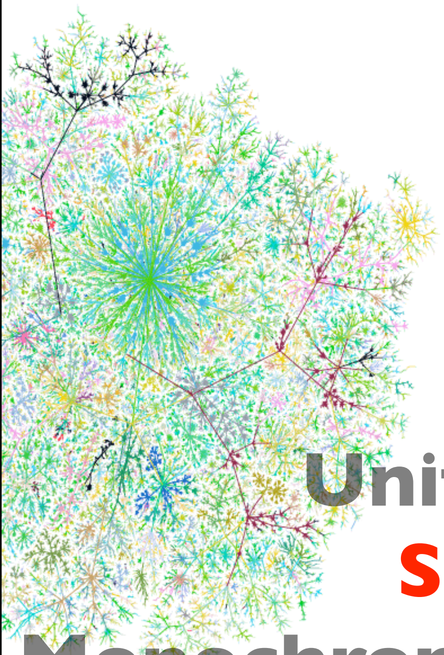


$D_S = 1.0$
$D'_S = 0.5$

- Chernoff: For each S, $D_S = D'_S/p \pm \varepsilon D^*$ w/p $1-n^{-3|S|}$
- Union Bound: Bound applies for all S w/p $1-n^{-1}$
  - There are $\leq n^k$ subsets of k nodes. So bound fails for some subset of size k w/p $\leq n^k n^{-3k} = n^{-2k}$
  - Bound fails for some subset w/p $\leq n^{-2}+n^{-4}+\ldots+n^{-2n} \leq n^{-1}$
- So max density of sampled graph gives $1+\varepsilon$ approx.

Part I

Sampling



Uniform Sampling + Densest Subgraph

Snape Sampling + Matching

Monochromatic Sampling + Clustering Coefficient

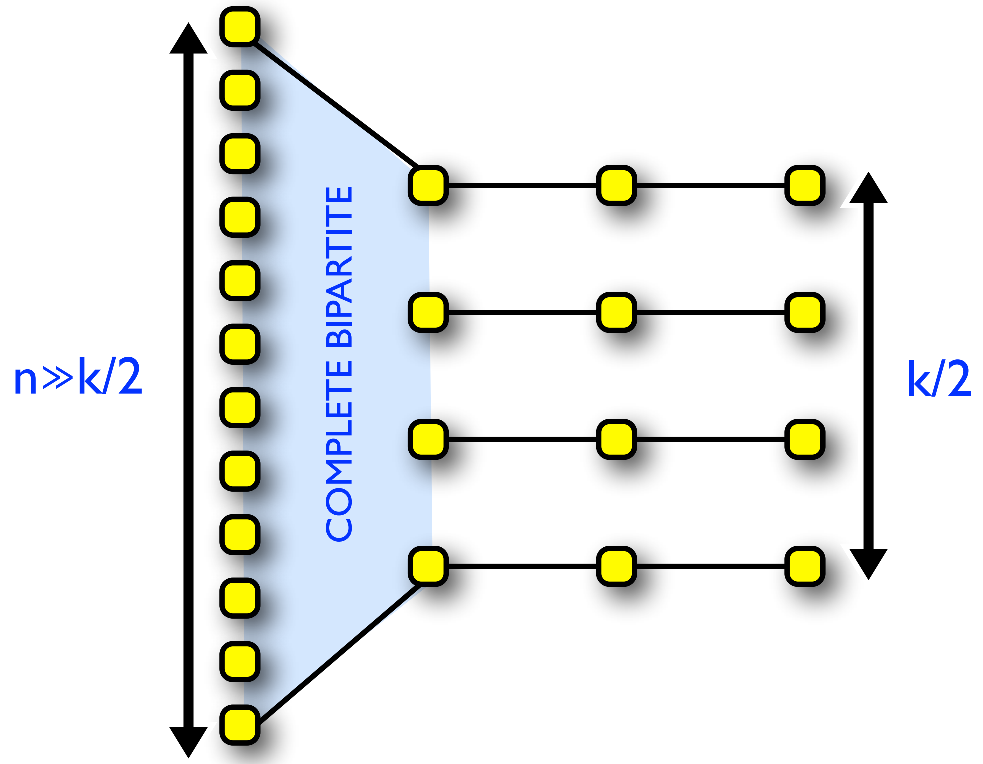
Edge-Weighted Sampling + Cuts and Sparsification

- *Matching Problem* Find large set of edges such that no two edges share an endpoint.

- *Matching Problem* Find large set of edges such that no two edges share an endpoint.
- How many “samples” are needed to find a matching of size k?

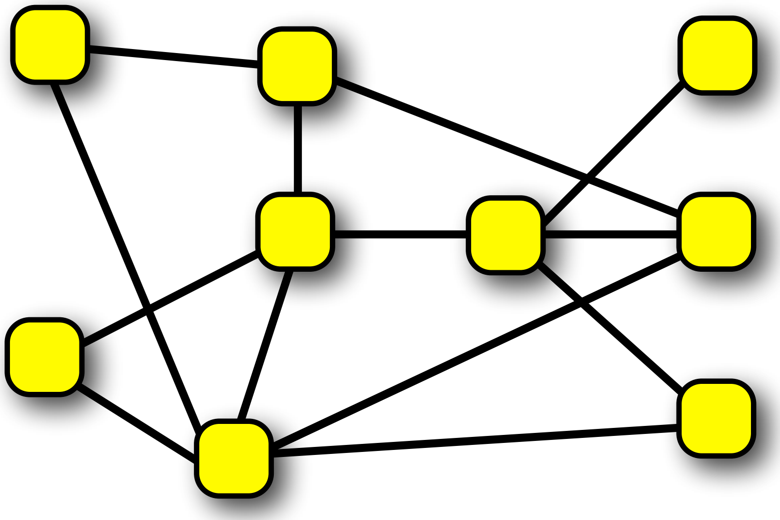
- *Matching Problem* Find large set of edges such that no two edges share an endpoint.
- How many "samples" are needed to find a matching of size k?
- Sampling uniformly can be very inefficient…

- *Matching Problem* Find large set of edges such that no two edges share an endpoint.
- How many "samples" are needed to find a matching of size k?
- Sampling uniformly can be very inefficient…

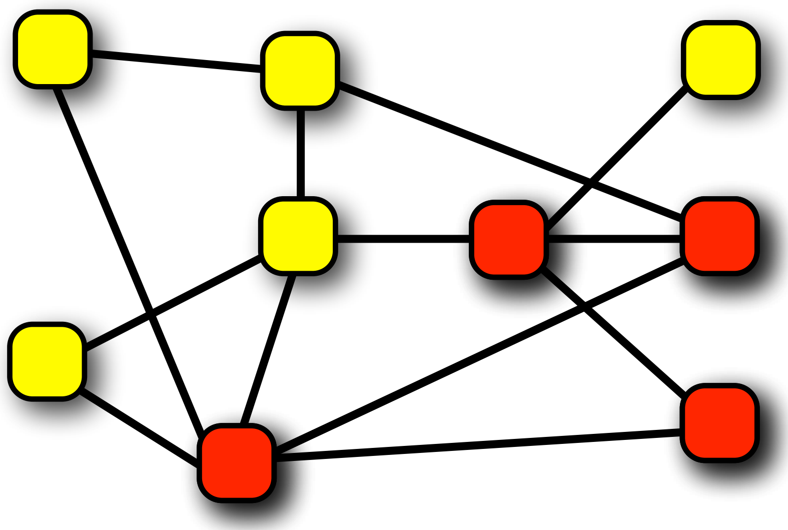


- *SNAPE "Sample Nodes And Pick Edge" Sampling:*

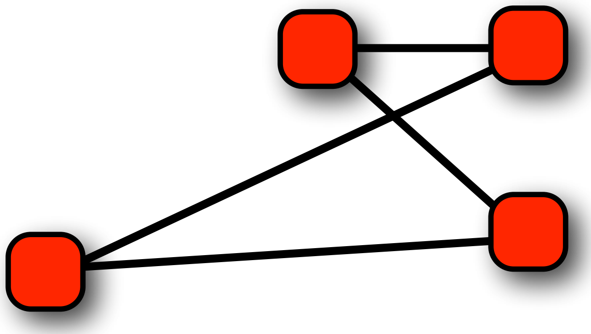
- *SNAPE "Sample Nodes And Pick Edge" Sampling:*
  - Sample each node with probability 1/k and delete rest



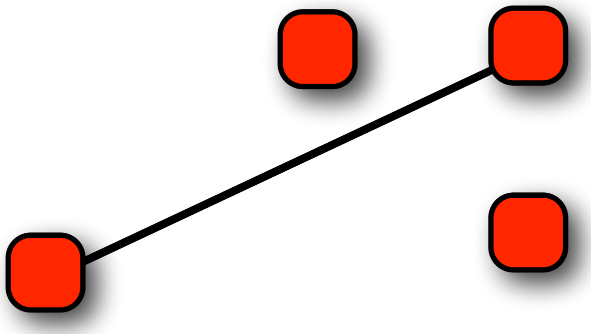
- *SNAPE "Sample Nodes And Pick Edge" Sampling:*
    - Sample each node with probability 1/k and delete rest



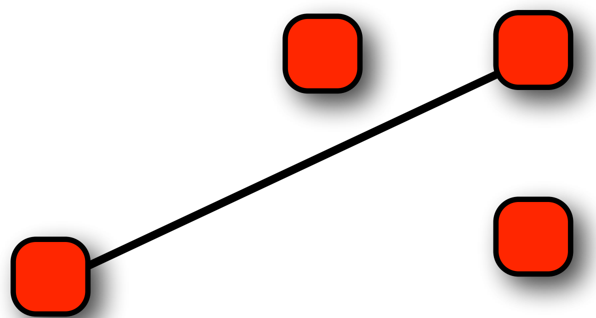
- *SNAPE "Sample Nodes And Pick Edge" Sampling:*
  - Sample each node with probability 1/k and delete rest



- *SNAPE "Sample Nodes And Pick Edge" Sampling:*
  - Sample each node with probability 1/k and delete rest
  - Pick a random edge amongst those that remain.



- *SNAPE "Sample Nodes And Pick Edge" Sampling:*
  - Sample each node with probability 1/k and delete rest
  - Pick a random edge amongst those that remain.



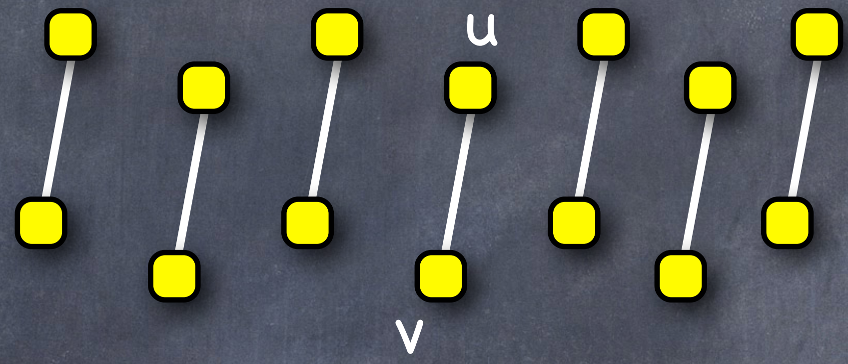
- *Theorem* If G has max matching size k, then $O(k^2 \log k)$ SNAPE samples will include a max matching from G.

Chitnis et al. [SODA 16], *Bury, Schwiegelshohn* [ESA 15]

Why SNAPE Sampling Works…

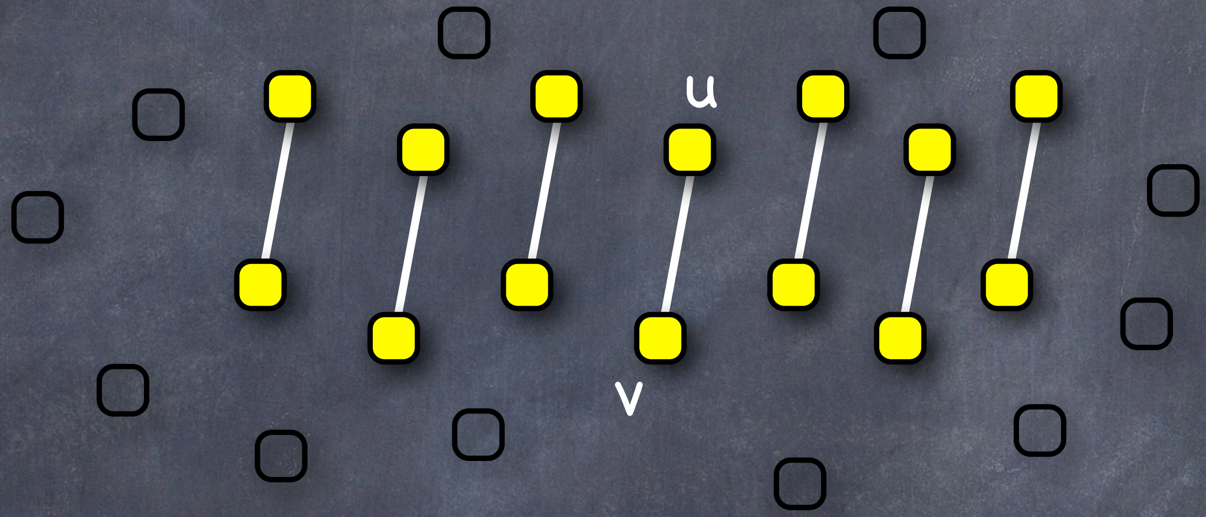
Why SNAPE Sampling Works...

- Consider a maximum matching M of size k and focus on arbitrary edge {u,v} in this matching.



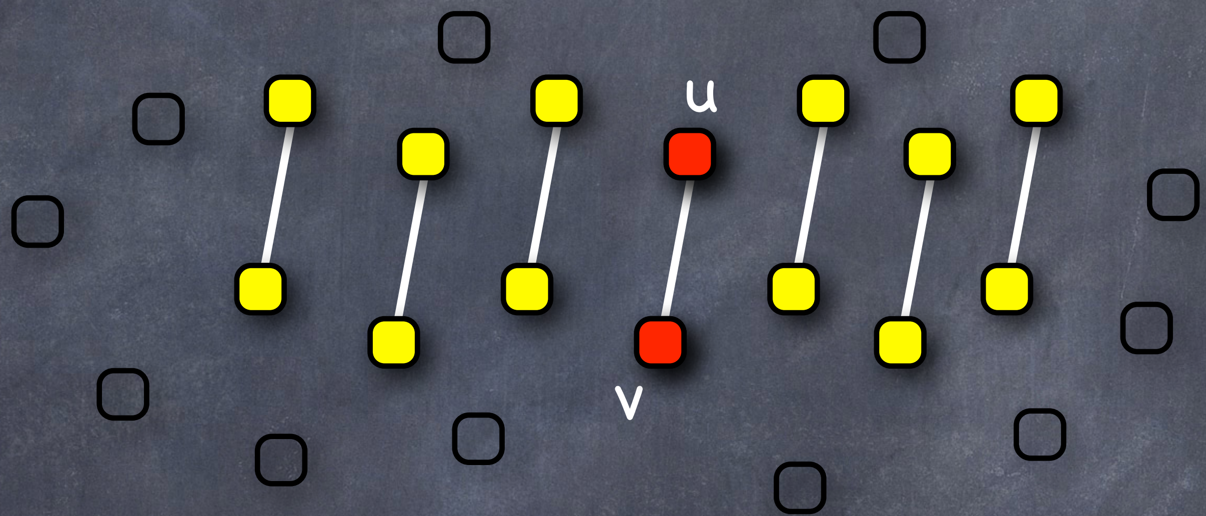
Why SNAPE Sampling Works…

- Consider a maximum matching M of size k and focus on arbitrary edge {u,v} in this matching.



Why SNAPE Sampling Works...

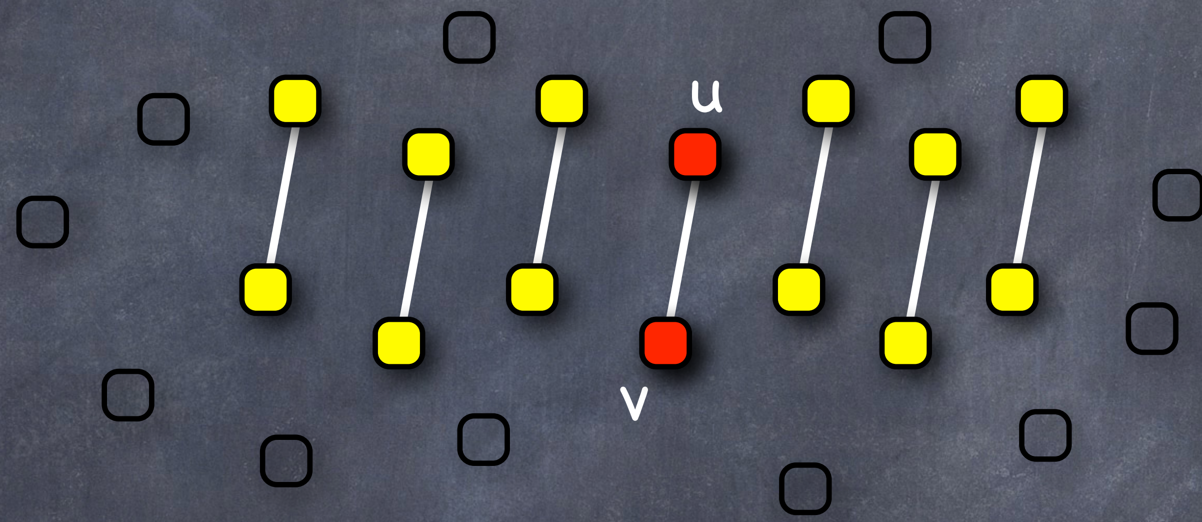
- Consider a maximum matching M of size k and focus on arbitrary edge {u,v} in this matching.



- W/p $\Omega(k^{-2})$ u and v only endpoints of M sampled.

Why SNAPE Sampling Works...

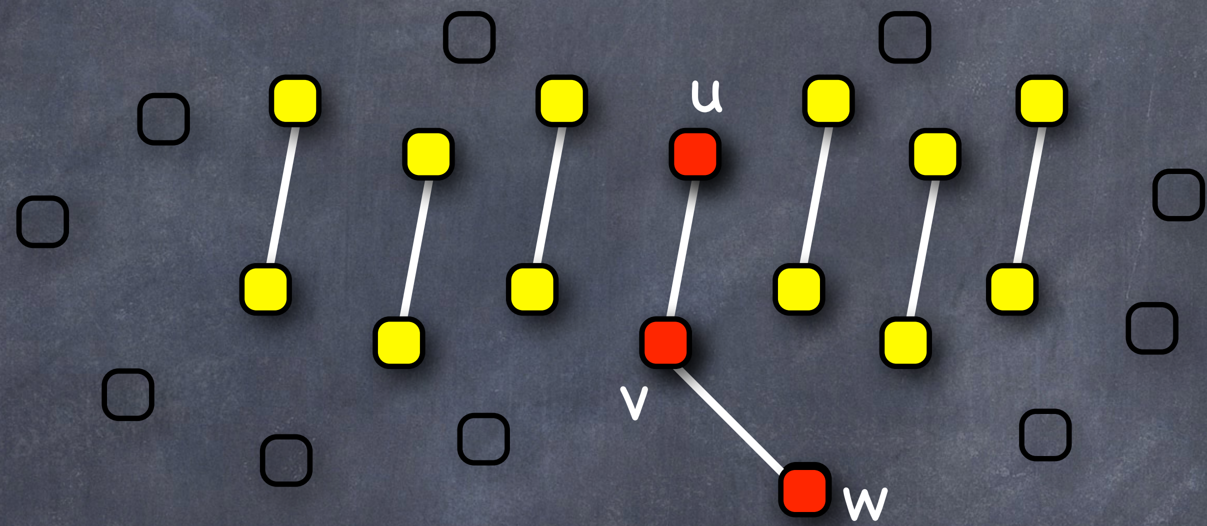
- Consider a maximum matching M of size k and focus on arbitrary edge {u,v} in this matching.



- W/p $\Omega(k^{-2})$ u and v only endpoints of M sampled.
- Hence, when we pick one of the remaining edges it's either {u,v} or another edge that's equally useful.

Why SNAPE Sampling Works...

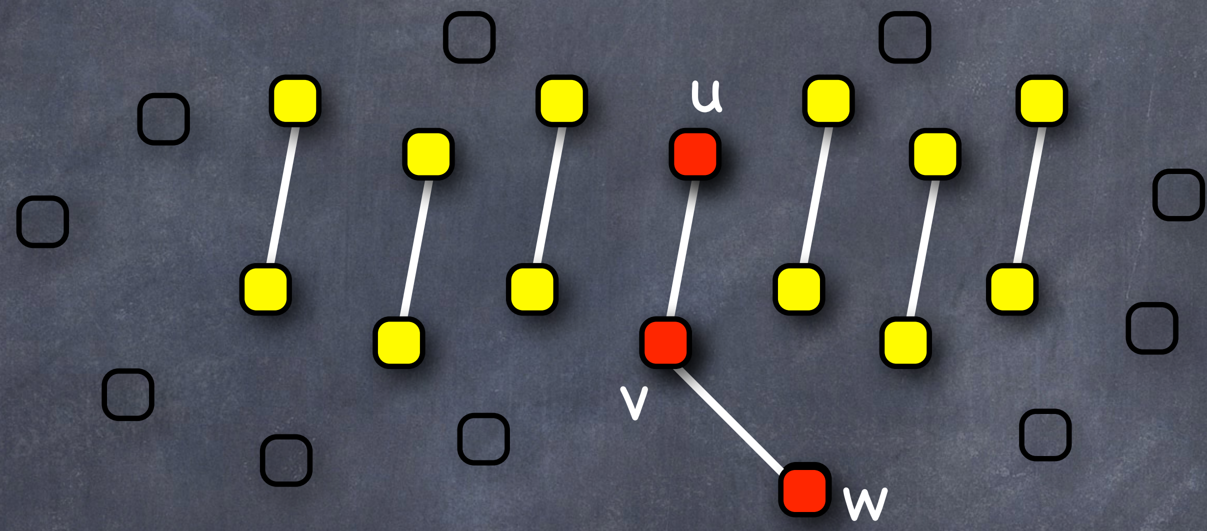
- Consider a maximum matching M of size k and focus on arbitrary edge {u,v} in this matching.



- W/p $\Omega(k^{-2})$ u and v only endpoints of M sampled.
- Hence, when we pick one of the remaining edges it's either {u,v} or another edge that's equally useful.

Why SNAPE Sampling Works...

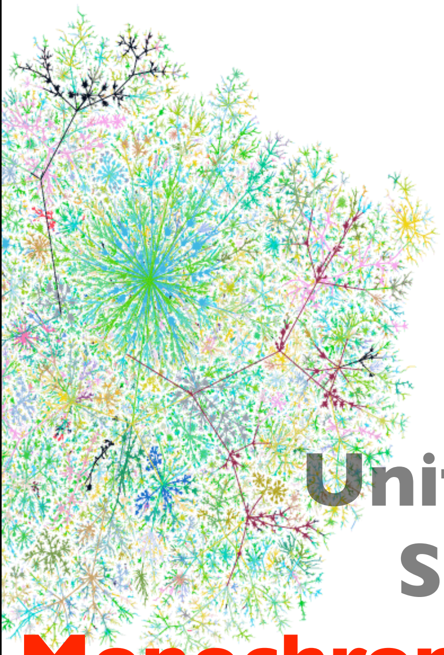
- Consider a maximum matching M of size k and focus on arbitrary edge {u,v} in this matching.



- W/p $\Omega(k^{-2})$ u and v only endpoints of M sampled.
- Hence, when we pick one of the remaining edges it's either {u,v} or another edge that's equally useful.
- Take $O(k^2 \log k)$ samples; apply analysis to all edges.

Part I

Sampling



Uniform Sampling + Densest Subgraph

Snape Sampling + Matching

Monochromatic Sampling + Clustering Coefficient

Edge-Weighted Sampling + Cuts and Sparsification

- Given a graph G, the *global clustering coefficient* is

$$\kappa = \frac{3 \times \text{number of triangles}}{\text{number of length 2 paths}}$$

A measure of how much nodes tend to cluster together.

- Given a graph G, the *global clustering coefficient* is

$$\kappa = \frac{3 \times \text{number of triangles}}{\text{number of length 2 paths}}$$

  A measure of how much nodes tend to cluster together.

- *Monochromatic Sampling* Randomly color each node from a set of colors. Store all edges with monochromatic endpoints. If length-2 path {u,v}, {v,w} is stored, {u,w} also stored if it exists.

- Given a graph G, the *global clustering coefficient* is

$$\kappa = \frac{3 \times \text{number of triangles}}{\text{number of length 2 paths}}$$

  A measure of how much nodes tend to cluster together.

- *Monochromatic Sampling* Randomly color each node from a set of colors. Store all edges with monochromatic endpoints. If length-2 path {u,v}, {v,w} is stored, {u,w} also stored if it exists.

- *Thm* Can additively estimate κ from $\tilde{O}(\sqrt{n})$ samples.

  Pagh, Tsourakakis [IPL 12], *Jha, Seshadhri, Pinar* [KDD 15]

- Given a graph G, the *global clustering coefficient* is

$$\kappa = \frac{3 \times \text{number of triangles}}{\text{number of length 2 paths}}$$

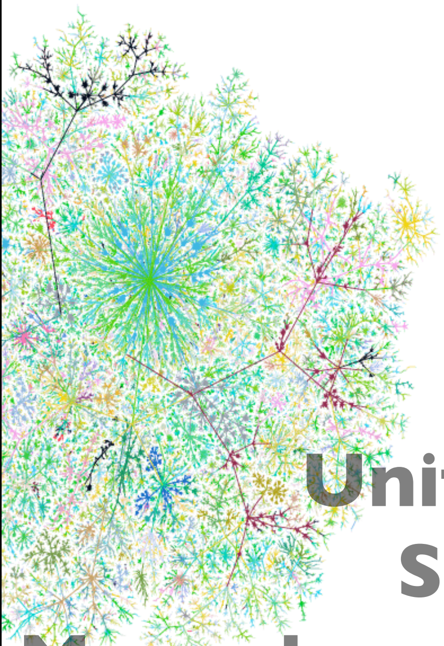
  A measure of how much nodes tend to cluster together.

- *Monochromatic Sampling* Randomly color each node from a set of colors. Store all edges with monochromatic endpoints. If length-2 path {u,v}, {v,w} is stored, {u,w} also stored if it exists.

- *Thm* Can additively estimate κ from $\tilde{O}(\sqrt{n})$ samples.

  Pagh, Tsourakakis [IPL 12], *Jha, Seshadhri, Pinar* [KDD 15]

- *Proof Idea* Compute expectation and variance of number of triangles amongst sampled edges and apply Chebyshev bound.



Part I

Sampling

Uniform Sampling + Densest Subgraph

Snape Sampling + Matching

Monochromatic Sampling + Clustering Coefficient

Edge-Weighted Sampling + Cuts and Sparsification

- *Defn* A *sparsifier* of graph G is a weighted subgraph H with:

- *Defn* A *sparsifier* of graph G is a weighted subgraph H with:

  ∀ cuts: "size of cut in G" = $(1\pm\varepsilon)$ "size of cut in H"

- *Defn* A *sparsifier* of graph G is a weighted subgraph H with:

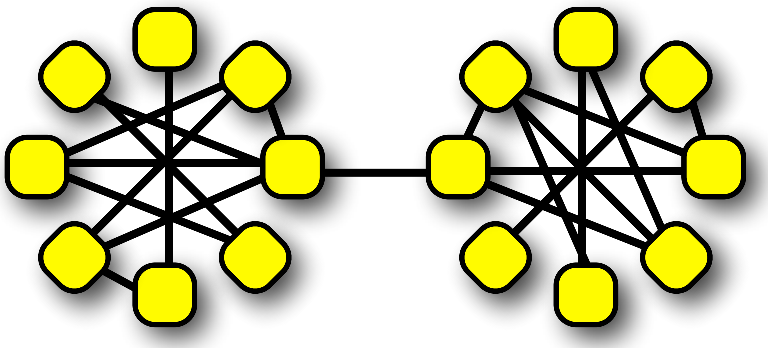
  ∀ cuts: "size of cut in G" = $(1\pm\varepsilon)$ "size of cut in H"

- *Basic Approach* Sample each edge uv with probability p_{uv} and reweight by $1/p_{uv}$. Probabilities depend on edge properties…

- *Defn* A *sparsifier* of graph G is a weighted subgraph H with:

  $\forall$ cuts: "size of cut in G" = $(1\pm\varepsilon)$ "size of cut in H"

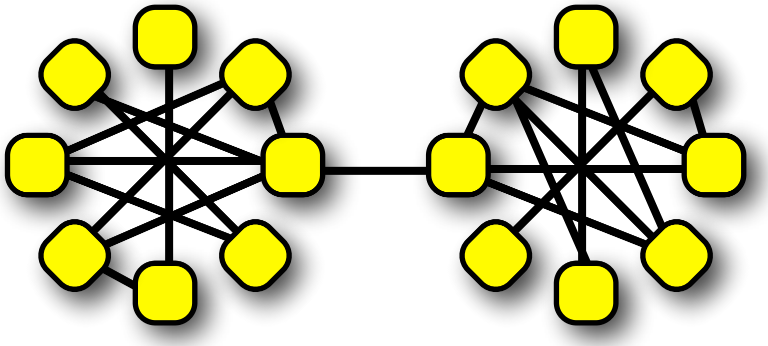
- *Basic Approach* Sample each edge uv with probability p_{uv} and reweight by $1/p_{uv}$. Probabilities depend on edge properties…



- *Defn* A *sparsifier* of graph G is a weighted subgraph H with:

  $\forall$ cuts: "size of cut in G" = $(1\pm\varepsilon)$ "size of cut in H"

- *Basic Approach* Sample each edge uv with probability p_{uv} and reweight by $1/p_{uv}$. Probabilities depend on edge properties…

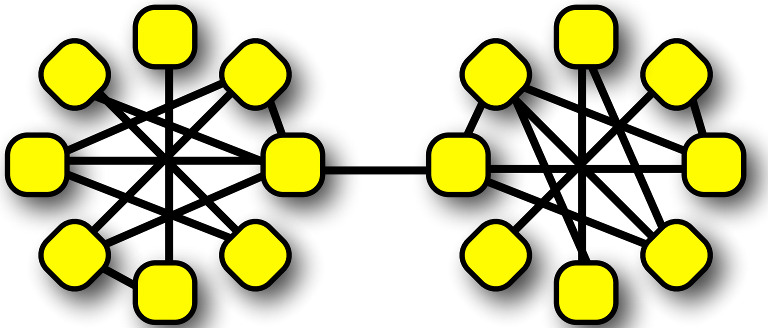


- *Thm* If $p_{uv}\approx\varepsilon^{-2}/\lambda_{uv}$ or $p_{uv}\approx\varepsilon^{-2}r_{uv}$ then result is sparsifier with $\tilde{O}(\varepsilon^{-2}\, n)$ edges. *Fung et al.* [STOC 11], *Spielman, Srivastava* [STOC 08]

- *Defn* A *sparsifier* of graph G is a weighted subgraph H with:

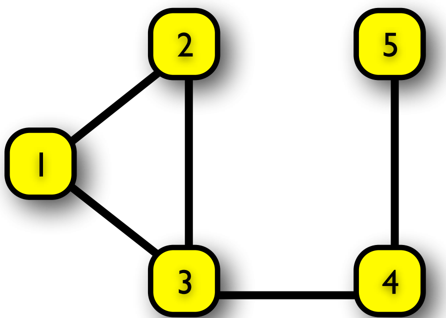
  $\forall$ cuts: "size of cut in G" = $(1\pm\varepsilon)$ "size of cut in H"

- *Basic Approach* Sample each edge uv with probability p_{uv} and reweight by $1/p_{uv}$. Probabilities depend on edge properties…



- *Thm* If $p_{uv} \approx \varepsilon^{-2}/\lambda_{uv}$ or $p_{uv} \approx \varepsilon^{-2} r_{uv}$ then result is sparsifier with $\tilde{O}(\varepsilon^{-2}\, n)$ edges. *Fung et al.* [STOC 11], *Spielman, Srivastava* [STOC 08]

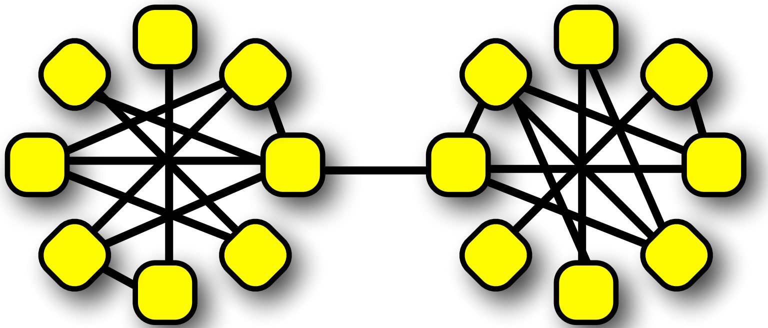
λ_{uv} is the min number of edges whose removal disconnects u and v



- *Defn* A *sparsifier* of graph G is a weighted subgraph H with:

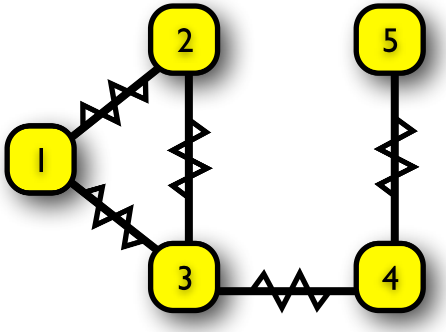
  ∀ cuts: "size of cut in G" = $(1\pm\varepsilon)$ "size of cut in H"

- *Basic Approach* Sample each edge uv with probability p_{uv} and reweight by $1/p_{uv}$. Probabilities depend on edge properties…



- *Thm* If $p_{uv} \approx \varepsilon^{-2}/\lambda_{uv}$ or $p_{uv} \approx \varepsilon^{-2} r_{uv}$ then result is sparsifier with $\tilde{O}(\varepsilon^{-2}\, n)$ edges. *Fung et al.* [STOC 11], *Spielman, Srivastava* [STOC 08]

λ_{uv} is the min number of edges whose removal disconnects u and v

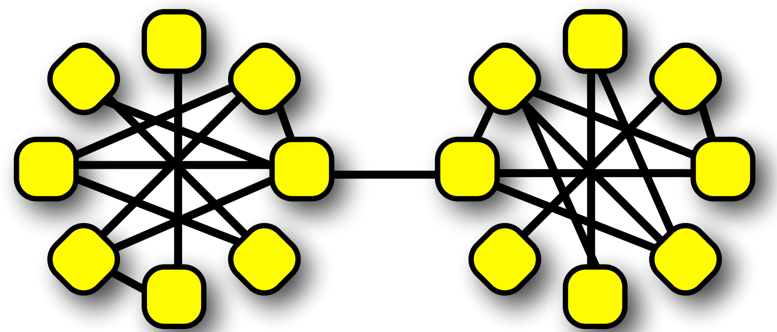


r_{uv} is potential difference when unit of flow injected at u and extracted at v

- *Defn* A *sparsifier* of graph G is a weighted subgraph H with:

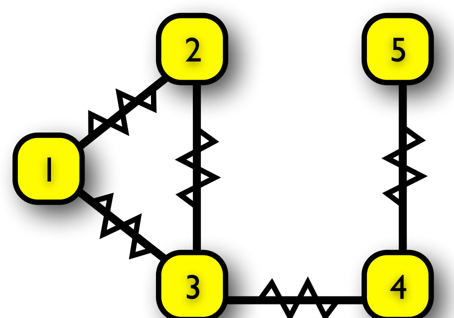
  ∀ cuts: "size of cut in G" = $(1\pm\varepsilon)$ "size of cut in H"

- *Basic Approach* Sample each edge uv with probability p_{uv} and reweight by $1/p_{uv}$. Probabilities depend on edge properties…



- *Thm* If $p_{uv}\approx\varepsilon^{-2}/\lambda_{uv}$ or $p_{uv}\approx\varepsilon^{-2}r_{uv}$ then result is sparsifier with $\tilde{O}(\varepsilon^{-2}\, n)$ edges. *Fung et al.* [STOC 11], *Spielman, Srivastava* [STOC 08]

λ_{uv} is the min number of edges whose removal disconnects u and v



r_{uv} is potential difference when unit of flow injected at u and extracted at v

- *Simpler Thm* If min-cut is $\gg \varepsilon^{-2}\log n$ then $p_e=1/2$ works.

Proof Idea of Simpler Theorem ...

Proof Idea of Simpler Theorem …

- Lemma (Chernoff) Let k′ be the number of edges that were sampled across some cut of size k. Then

$$\Pr[k' = (1 \pm \varepsilon)k/2] < \exp(-\varepsilon^2 k/6)$$

Proof Idea of Simpler Theorem ...

- Lemma (Chernoff) Let k′ be the number of edges that were sampled across some cut of size k. Then

$$\Pr[k' = (1 \pm \varepsilon)k/2] < \exp(-\varepsilon^2 k/6)$$

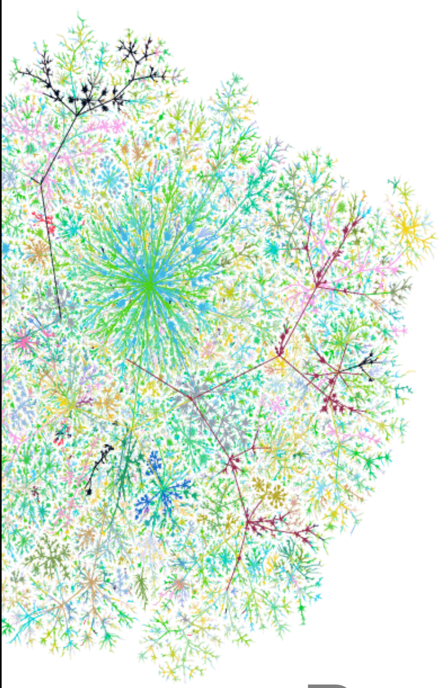
- Lemma (Karger) The number of cuts with k edges is < $\exp(2k \log n / \lambda)$ where λ is size of min-cut.

Proof Idea of Simpler Theorem ...

- Lemma (Chernoff) Let k′ be the number of edges that were sampled across some cut of size k. Then

$$\Pr[k' = (1 \pm \varepsilon)k/2] < \exp(-\varepsilon^2 k/6)$$

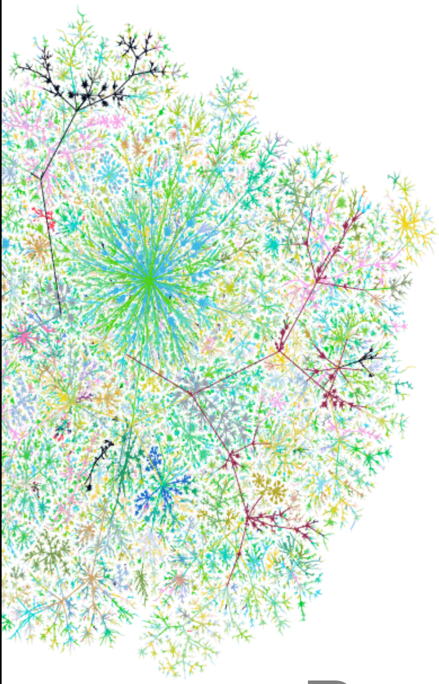
- Lemma (Karger) The number of cuts with k edges is < $\exp(2k \log n / \lambda)$ where λ is size of min-cut.
- Result then follows by substituting bound for λ and applying union bound over all cuts.



Part II

Sketching

What is sketching?
Surprising connectivity example
Revisiting graph cuts and sparsification



Part II

Sketching

What is sketching?

Surprising connectivity example

Revisiting graph cuts and sparsification

$$\begin{bmatrix} \\ v \\ \\ \end{bmatrix}$$

- *Random linear projection* M: $\mathbb{R}^N \rightarrow \mathbb{R}^D$ where $D \ll N$ that preserves properties of any $v \in \mathbb{R}^N$ with high probability.

$$\begin{bmatrix} \\ v \\ \\ \end{bmatrix}$$

- *Random linear projection* M: $\mathbb{R}^N \rightarrow \mathbb{R}^D$ where $D \ll N$ that preserves properties of any $v \in \mathbb{R}^N$ with high probability.

$$\begin{bmatrix} & & M & & \end{bmatrix} \begin{bmatrix} \\ v \\ \\ \end{bmatrix}$$

- *Random linear projection* M: $\mathbb{R}^N \rightarrow \mathbb{R}^D$ where $D \ll N$ that preserves properties of any $v \in \mathbb{R}^N$ with high probability.

$$\begin{bmatrix} & & M & & \end{bmatrix} \begin{bmatrix} \\ v \\ \\ \end{bmatrix} = \begin{bmatrix} Mv \end{bmatrix}$$

- *Random linear projection* M: $\mathbb{R}^N \rightarrow \mathbb{R}^D$ where $D \ll N$ that preserves properties of any $v \in \mathbb{R}^N$ with high probability.

$$\begin{bmatrix} & & M & & \end{bmatrix} \begin{bmatrix} \\ v \\ \\ \end{bmatrix} = \begin{bmatrix} Mv \end{bmatrix} \longrightarrow \text{answer}$$

- *Random linear projection* M: $\mathbb{R}^N \rightarrow \mathbb{R}^D$ where $D \ll N$ that preserves properties of any $v \in \mathbb{R}^N$ with high probability.

$$\begin{bmatrix} & & M & & \end{bmatrix} \begin{bmatrix} \\ \\ v \\ \\ \\ \end{bmatrix} = \begin{bmatrix} Mv \end{bmatrix} \longrightarrow \text{answer}$$

- *Many results* for numerical statistics and geometric properties... *extensive theory* with connections to hashing, compressed sensing, dimensionality reduction, metric embeddings... *widely applicable* since parallelizable and suitable for stream processing.

- *Random linear projection* M: $\mathbb{R}^N \rightarrow \mathbb{R}^D$ where $D \ll N$ that preserves properties of any $v \in \mathbb{R}^N$ with high probability.

$$\begin{bmatrix} & M & \end{bmatrix} \begin{bmatrix} \\ v \\ \\ \end{bmatrix} = \begin{bmatrix} Mv \end{bmatrix} \longrightarrow \text{answer}$$

- *Many results* for numerical statistics and geometric properties... *extensive theory* with connections to hashing, compressed sensing, dimensionality reduction, metric embeddings... *widely applicable* since parallelizable and suitable for stream processing.

- *Example "l_0 Sampling" Sketch* Can be used to sample uniformly from non-zero entries of the vector where D=polylog(N).

Jowhari, Saglam, Tardos [PODS 11], *Kapralov et al.* [FOCS 17]

- ***Random linear projection*** M: $\mathbb{R}^N \rightarrow \mathbb{R}^D$ where $D \ll N$ that preserves properties of any $v \in \mathbb{R}^N$ with high probability.

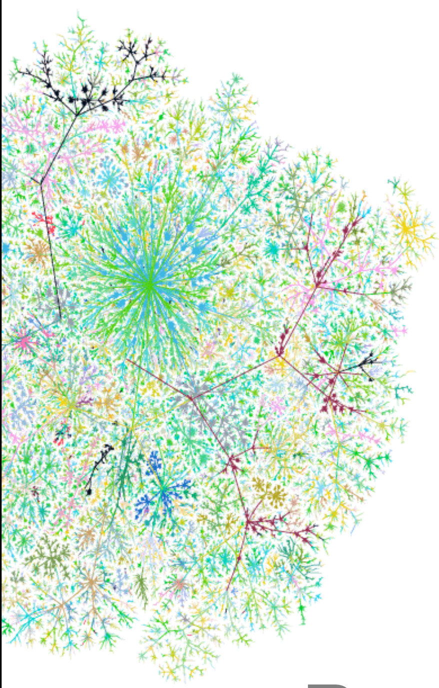
$$\begin{bmatrix} & M & \end{bmatrix} \begin{bmatrix} \\ v \\ \\ \end{bmatrix} = \begin{bmatrix} Mv \end{bmatrix} \longrightarrow \text{answer}$$

- *Many results* for numerical statistics and geometric properties... *extensive theory* with connections to hashing, compressed sensing, dimensionality reduction, metric embeddings... *widely applicable* since parallelizable and suitable for stream processing.

- ***Example "l_0 Sampling" Sketch*** Can be used to sample uniformly from non-zero entries of the vector where D=polylog(N).

  Jowhari, Saglam, Tardos [PODS 11], *Kapralov et al.* [FOCS 17]

? ***Question*** What about analyzing massive graphs via sketches?



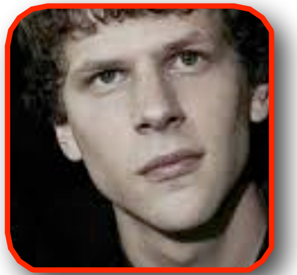
Part II

Sketching

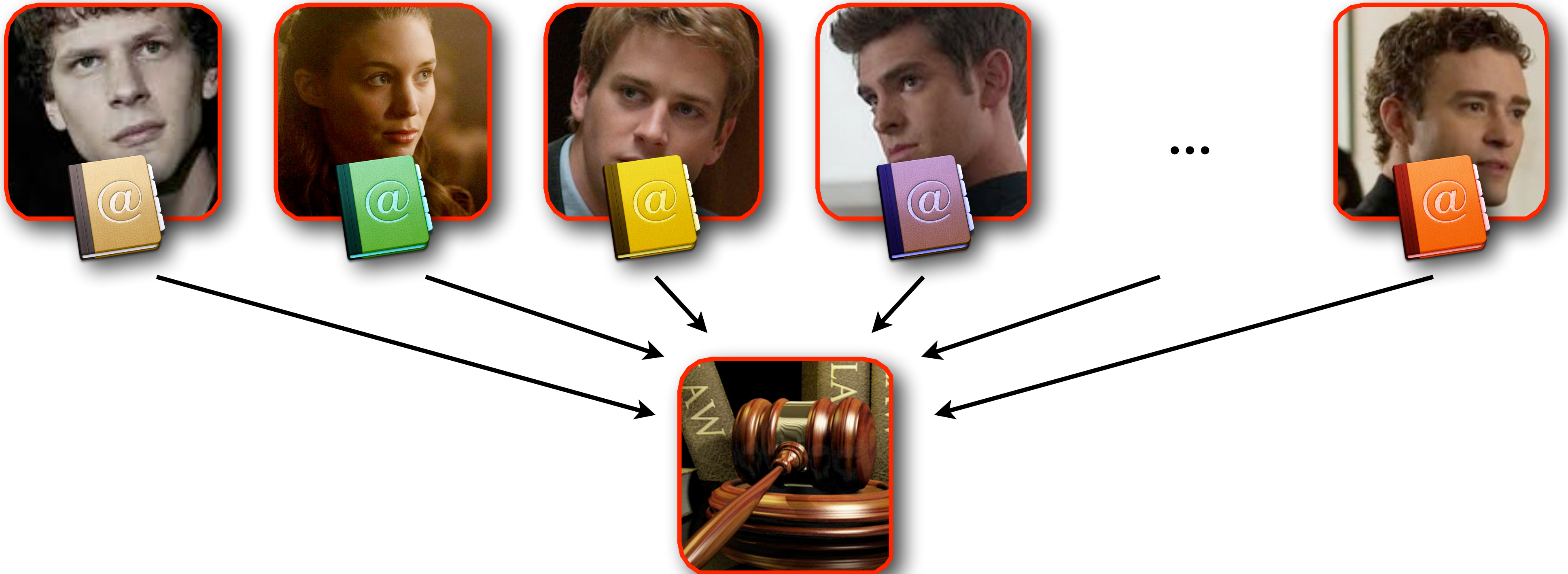
What is sketching?

Surprising connectivity example

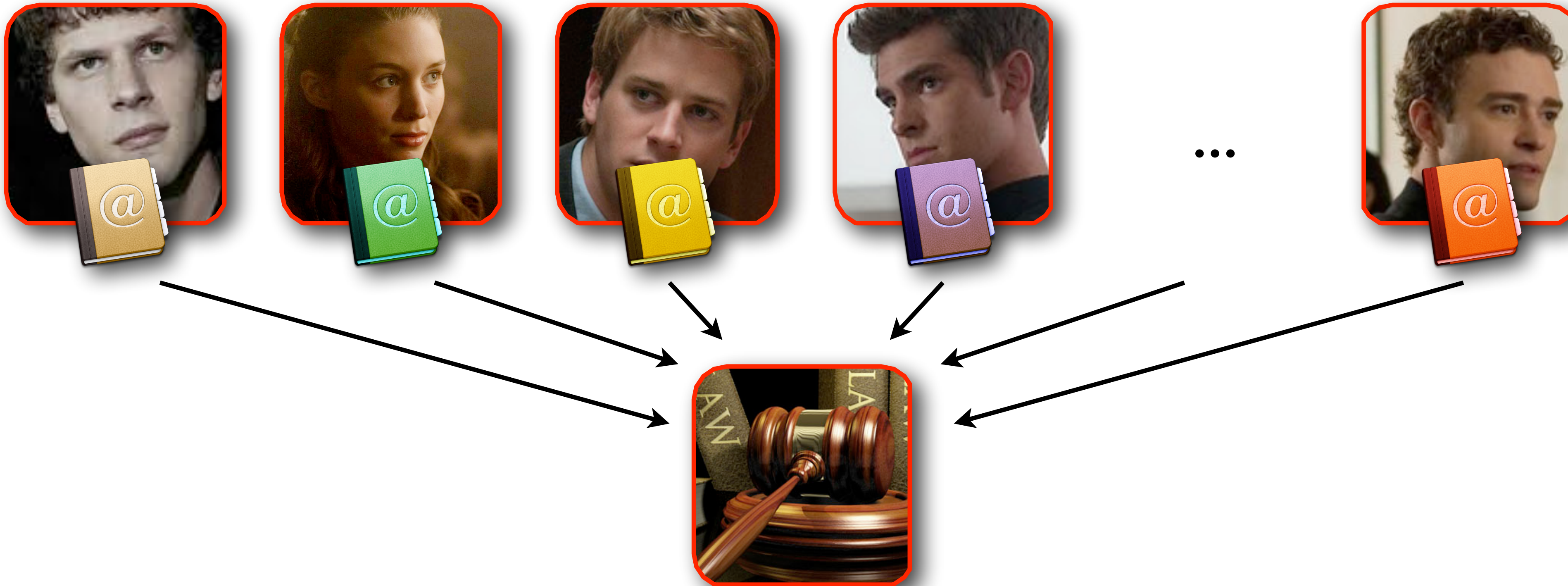
Revisiting graph cuts and sparsification

    ... 

- *Communication Problem* n players each have a list their friends. Simultaneously, they each send a message to a central player who deduces if underlying graph is connected.

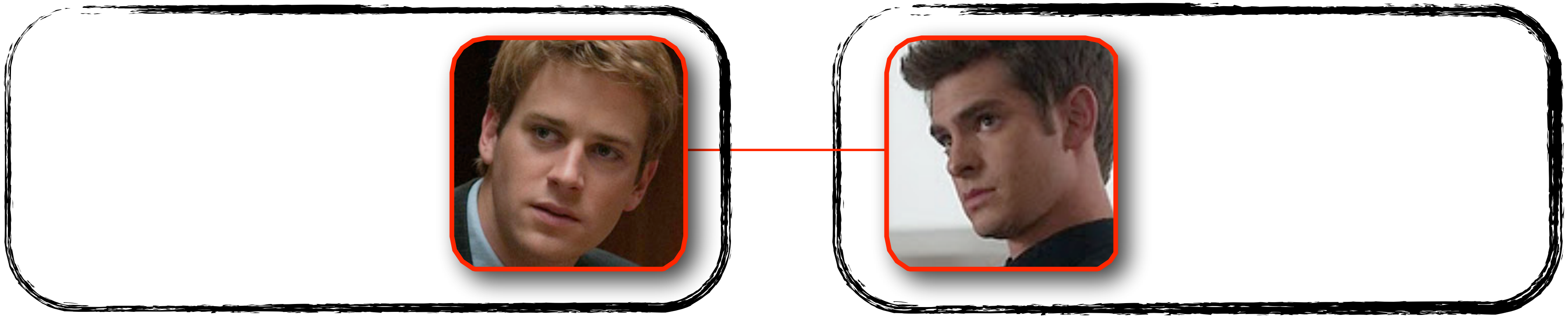


- *Communication Problem* n players each have a list their friends. Simultaneously, they each send a message to a central player who deduces if underlying graph is connected.

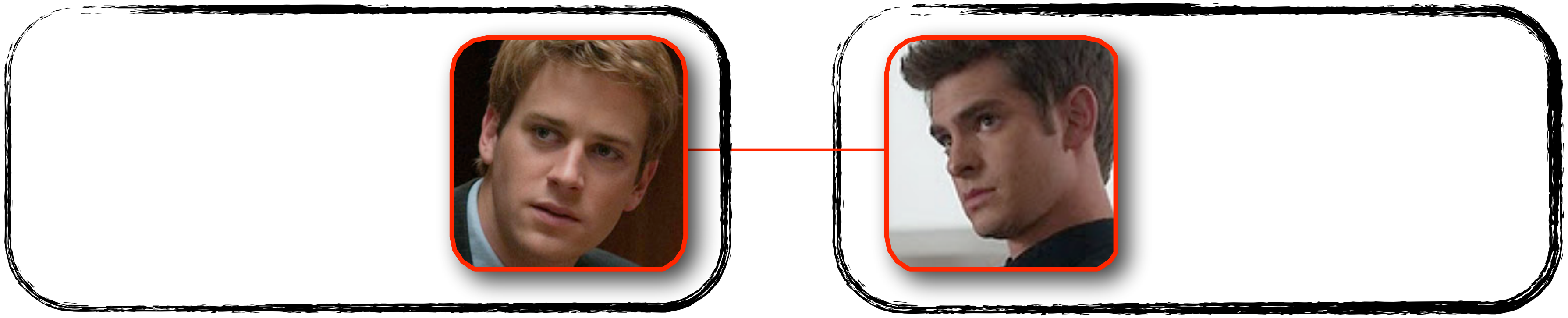


- *Communication Problem* n players each have a list their friends. Simultaneously, they each send a message to a central player who deduces if underlying graph is connected.
- *Thm* O(polylog n) bit message from each player suffices.

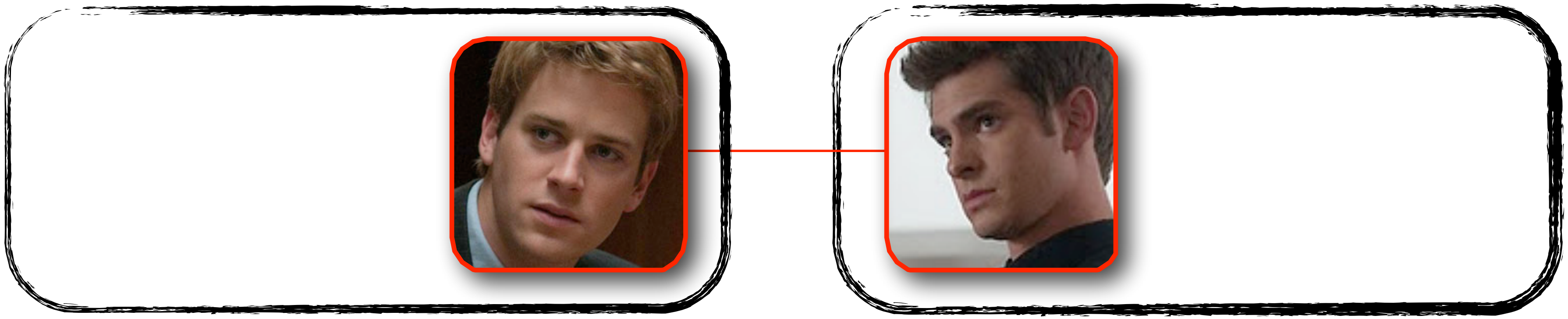
Ahn, Guha, McGregor [SODA 12]



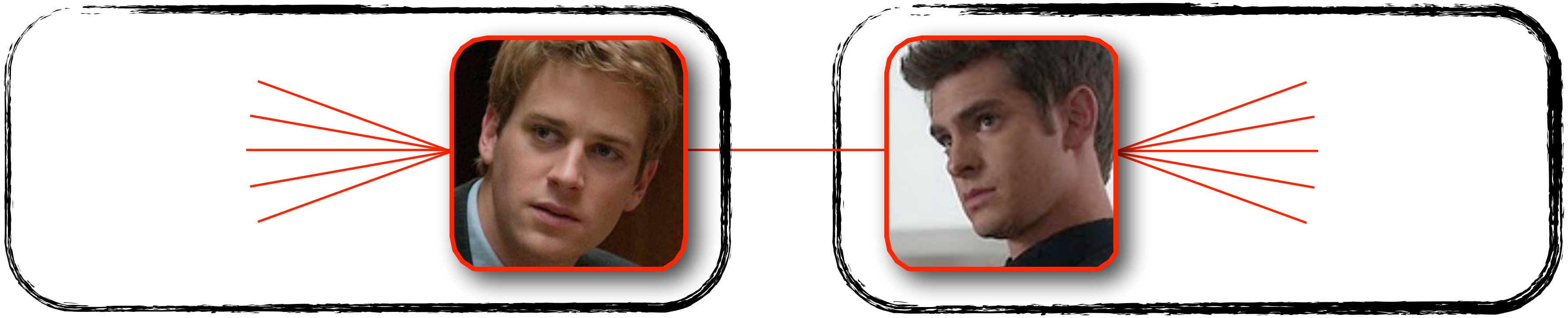
- *Can't be possible!* What if there's a *bridge* (u,v) in the graph, i.e., a friendship that is critical to ensuring the graph is connected.



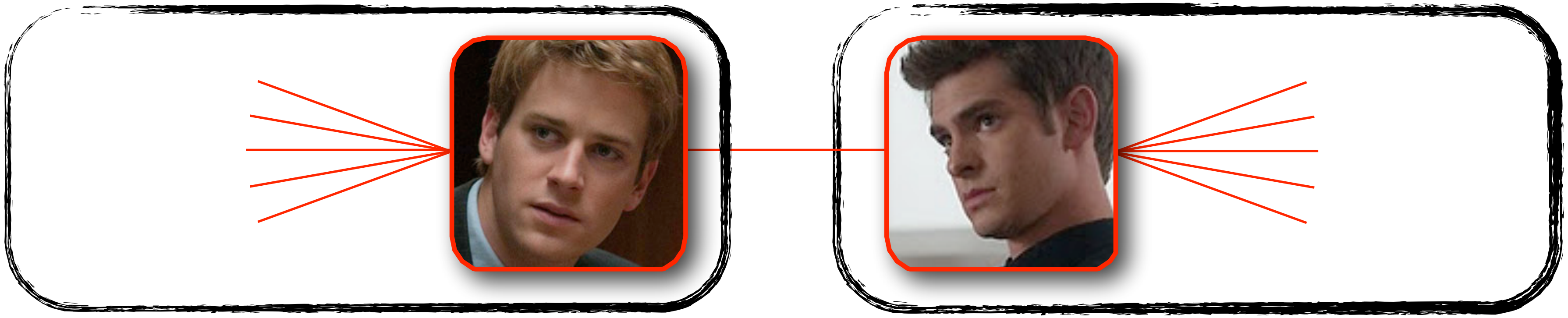
- *Can't be possible!* What if there's a *bridge* (u,v) in the graph, i.e., a friendship that is critical to ensuring the graph is connected.
- It *appears* like at least one player needs to send Ω(n) bits.



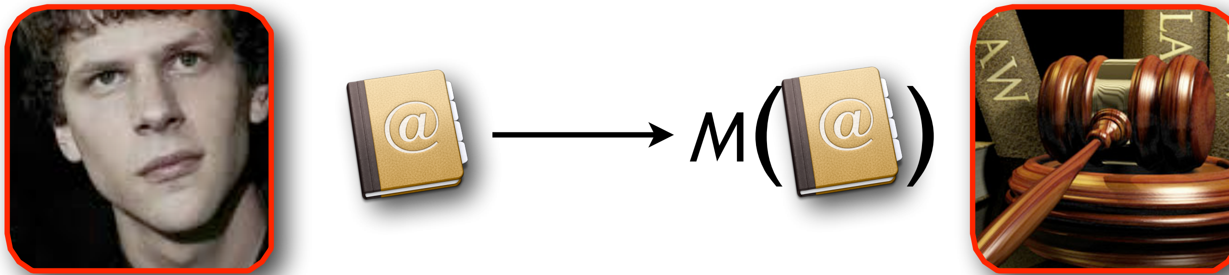
- *Can’t be possible!* What if there’s a *bridge* (u,v) in the graph, i.e., a friendship that is critical to ensuring the graph is connected.
- It *appears* like at least one player needs to send Ω(n) bits.
  - Central player needs to know about the special friendship.



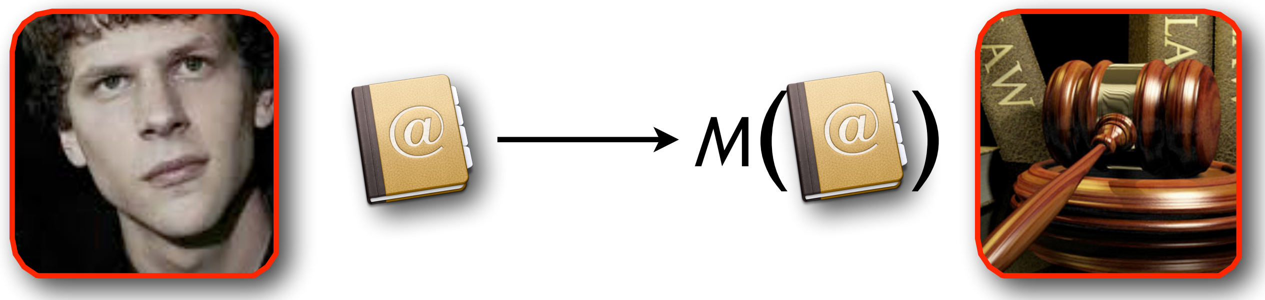
- *Can't be possible!* What if there's a *bridge* (u,v) in the graph, i.e., a friendship that is critical to ensuring the graph is connected.
- It *appears* like at least one player needs to send $\Omega(n)$ bits.
  - Central player needs to know about the special friendship.
  - Participant doesn't know which friendships are special.



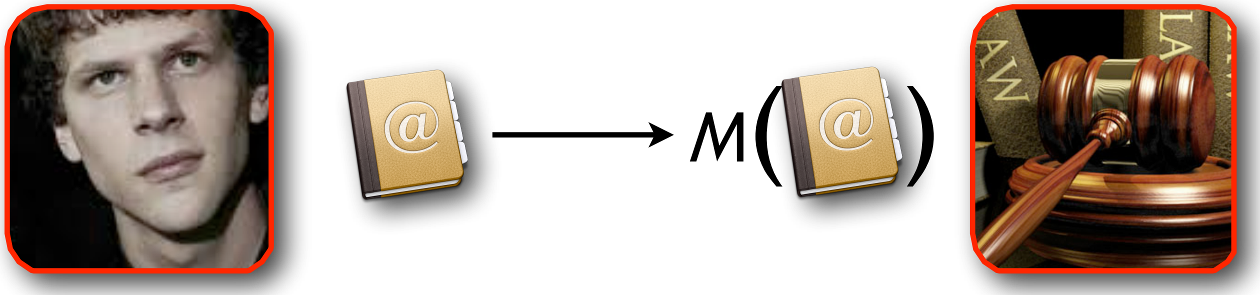
- *Can't be possible!* What if there's a *bridge* (u,v) in the graph, i.e., a friendship that is critical to ensuring the graph is connected.
- It *appears* like at least one player needs to send Ω(n) bits.
  - Central player needs to know about the special friendship.
  - Participant doesn't know which friendships are special.
  - Participants may have Ω(n) friends.



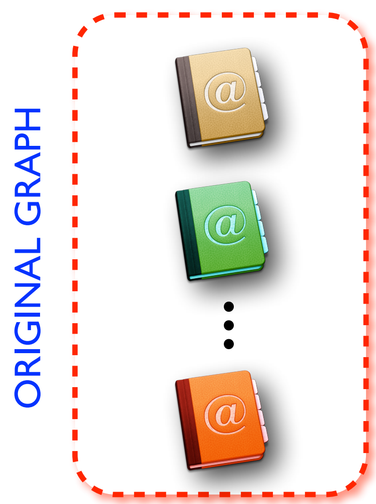
- Players send carefully-designed sketches of address books.

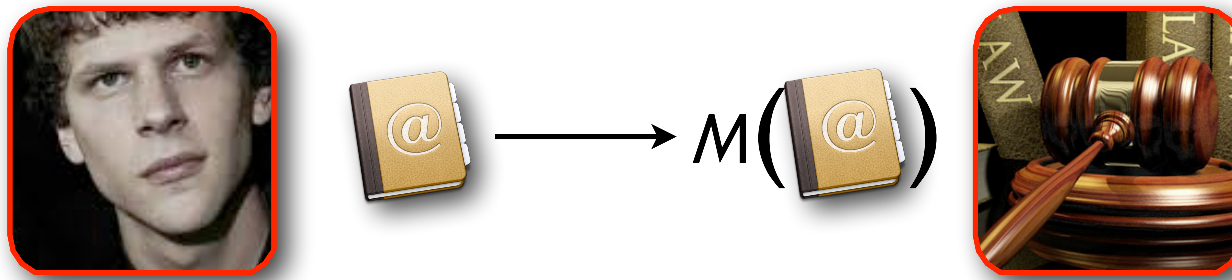


- Players send carefully-designed sketches of address books.
- <u>*Homomorphic Compression*</u> Instead of running algorithm on original data, run algorithm on sketched data.

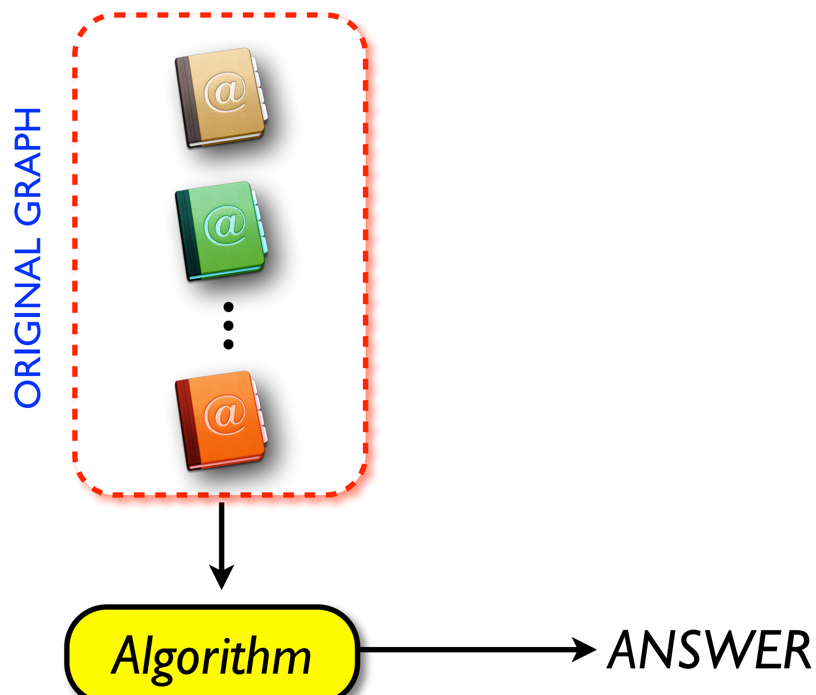


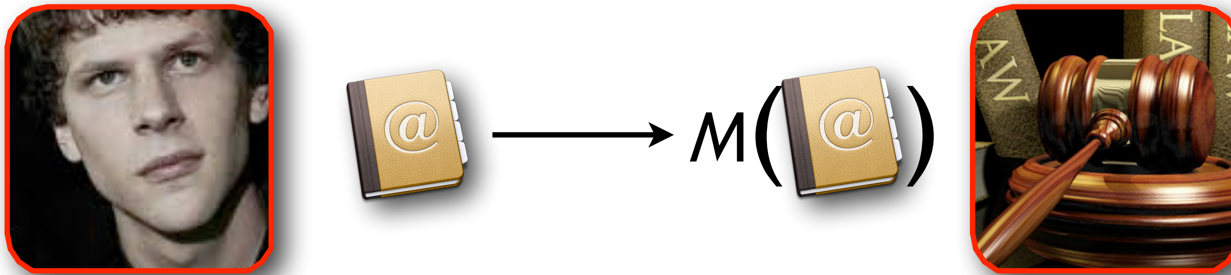
- Players send carefully-designed sketches of address books.
- *Homomorphic Compression* Instead of running algorithm on original data, run algorithm on sketched data.



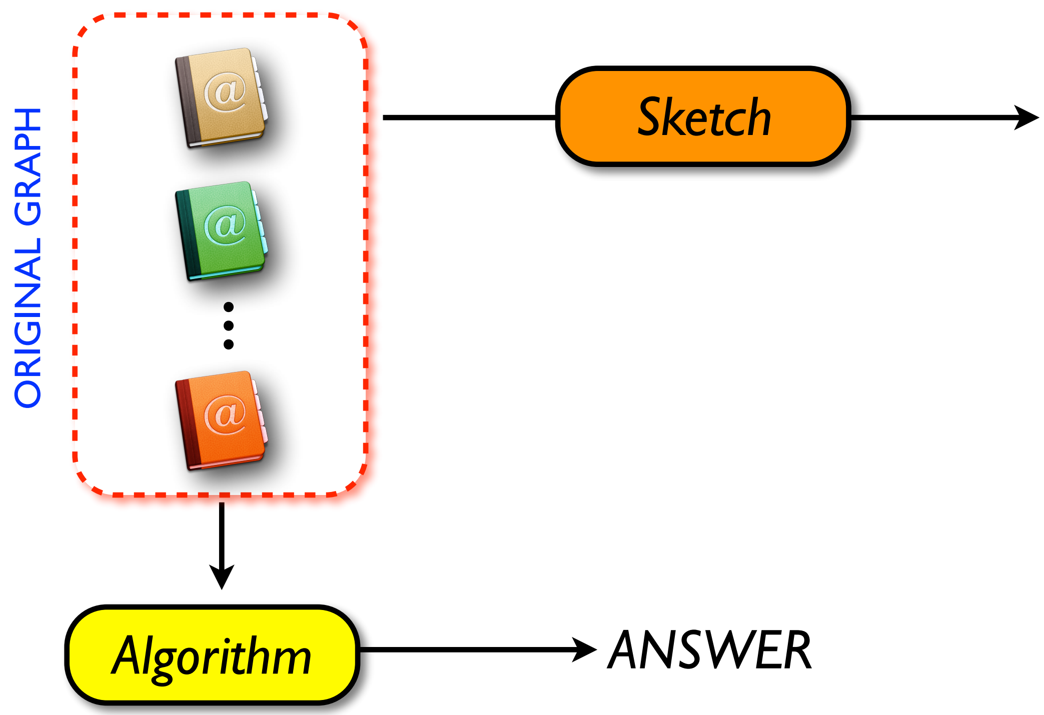


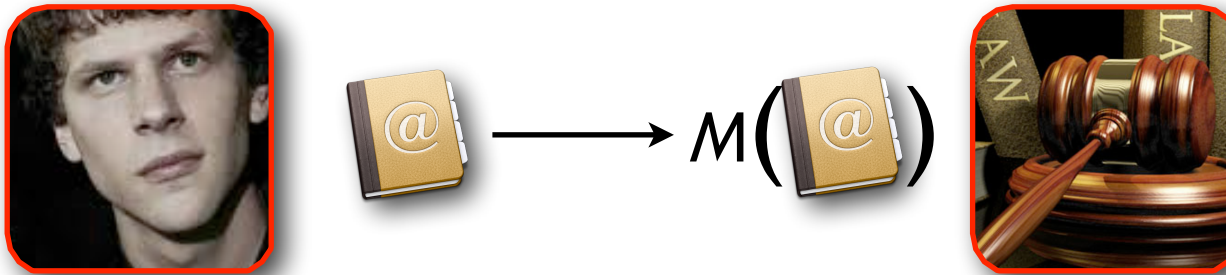
- Players send carefully-designed sketches of address books.
- *Homomorphic Compression* Instead of running algorithm on original data, run algorithm on sketched data.



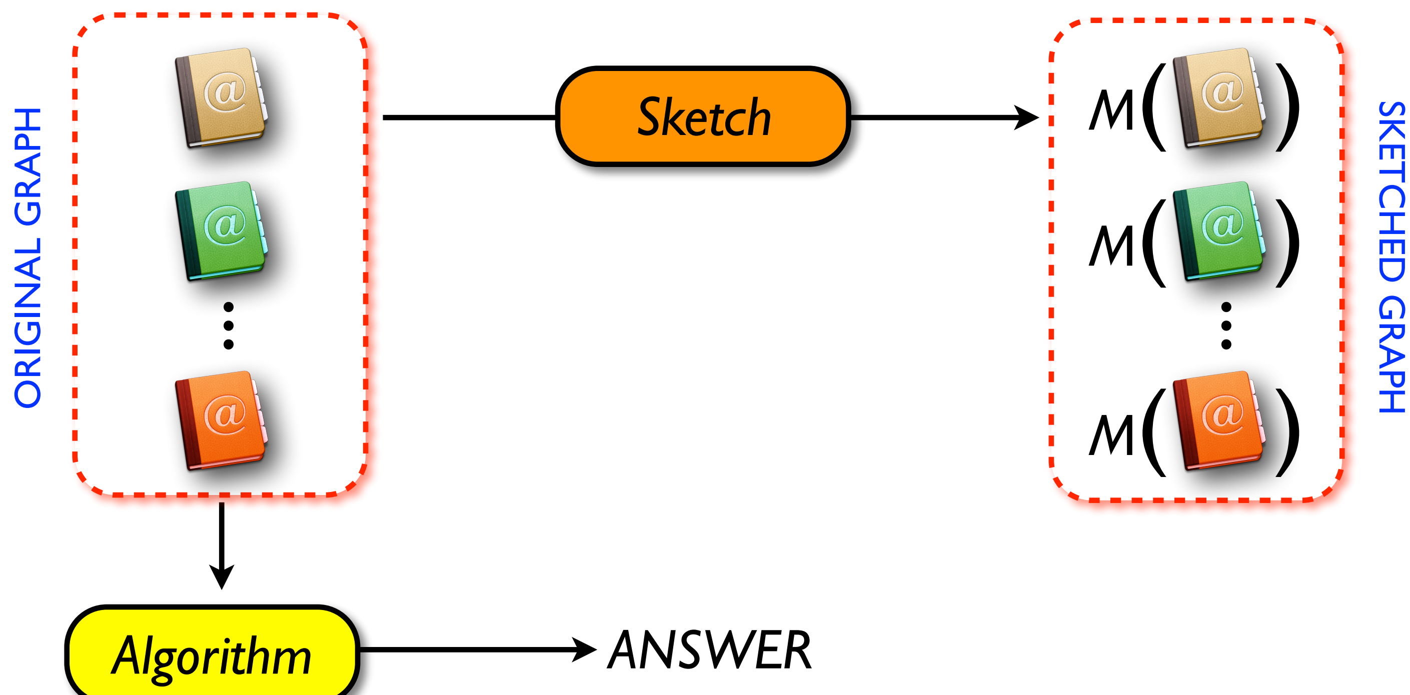


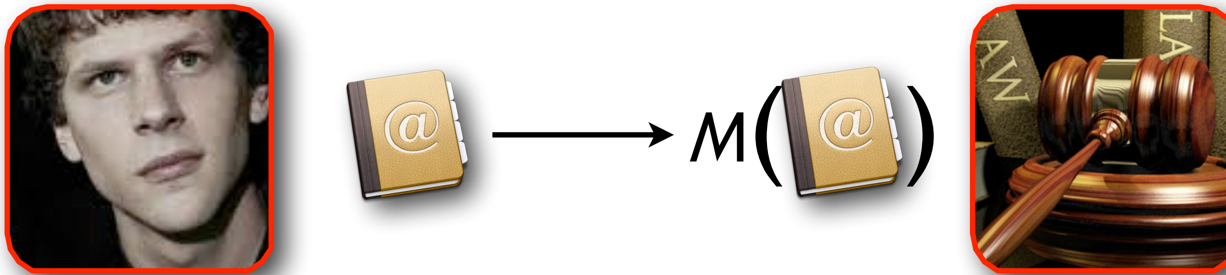
- Players send carefully-designed sketches of address books.
- *Homomorphic Compression* Instead of running algorithm on original data, run algorithm on sketched data.



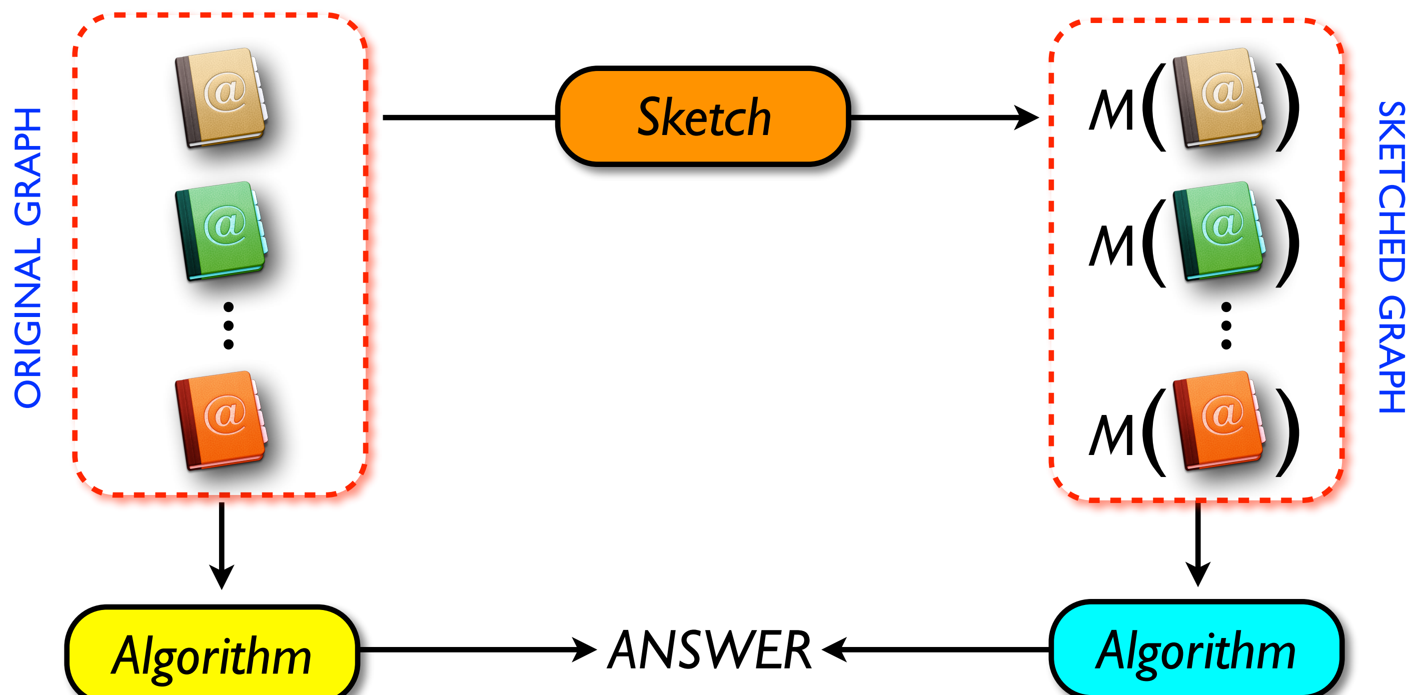


- Players send carefully-designed sketches of address books.
- *Homomorphic Compression* Instead of running algorithm on original data, run algorithm on sketched data.





- Players send carefully-designed sketches of address books.
- *Homomorphic Compression* Instead of running algorithm on original data, run algorithm on sketched data.



Ingredient 1: Basic Algorithm

Ingredient 1: Basic Algorithm

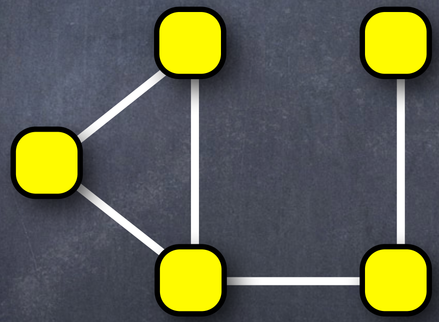
- Algorithm (Spanning Forest):

Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge

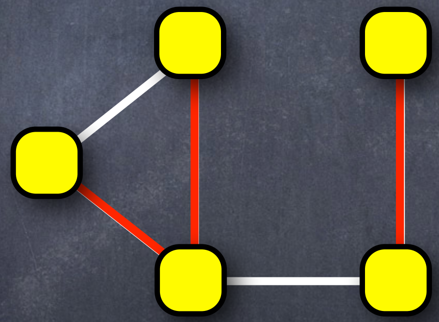
Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge



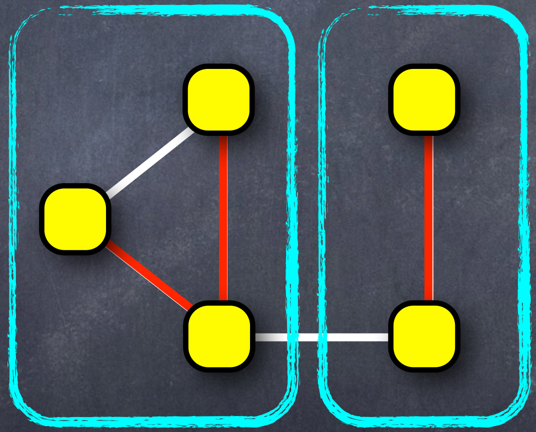
Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge



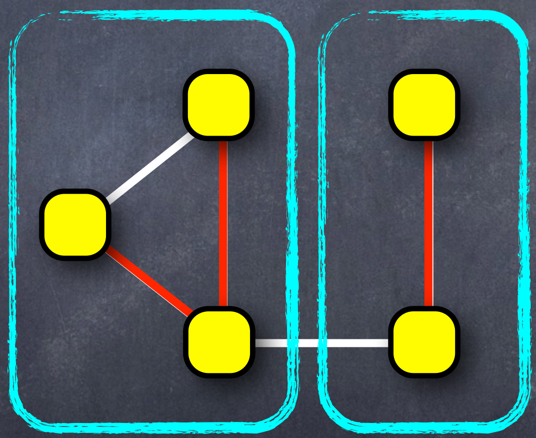
Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge



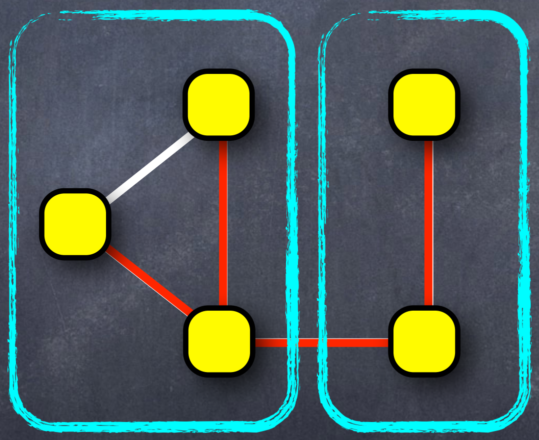
Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge
  - For each connected comp: pick incident edge



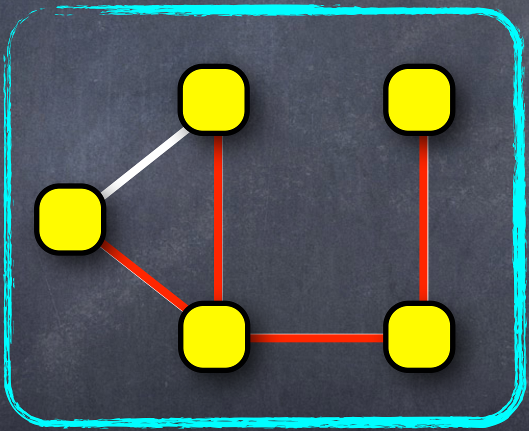
Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge
  - For each connected comp: pick incident edge



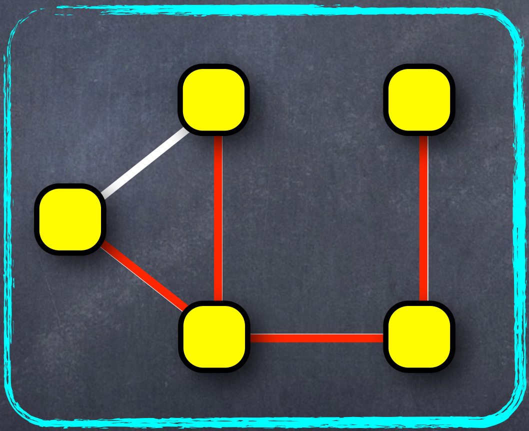
Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge
  - For each connected comp: pick incident edge



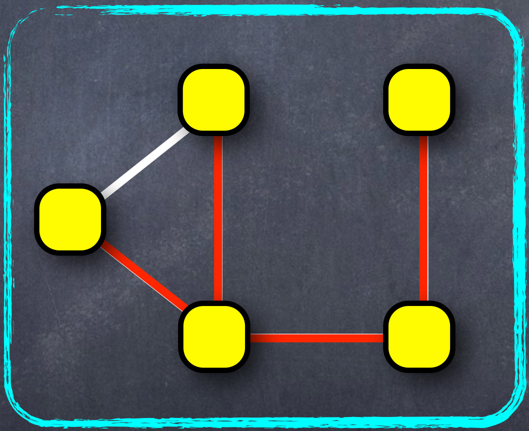
Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge
  - For each connected comp: pick incident edge
  - Repeat until no edges between connected comp.



Ingredient 1: Basic Algorithm

- Algorithm (Spanning Forest):
  - For each node: pick incident edge
  - For each connected comp: pick incident edge
  - Repeat until no edges between connected comp.



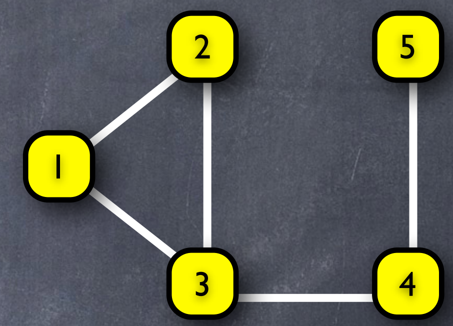
- Lemma After O(log n) rounds selected edges include spanning forest.

Ingredient 2: Sketching Neighborhoods

Ingredient 2: Sketching Neighborhoods

- For node i, let a_i be vector indexed by node pairs. Non-zero entries: $a_i[i,j]=1$ if $j>i$ and $a_i[i,j]=-1$ if $j<i$.

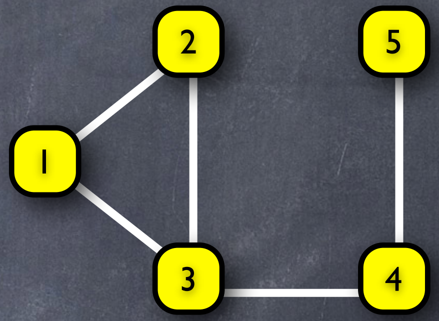
	{1,2}	{1,3}	{1,4}	{1,5}	{2,3}	{2,4}	{2,5}	{3,4}	{3,5}	{4,5}
$\mathbf{a}_1 =$	1	1	0	0	0	0	0	0	0	0



Ingredient 2: Sketching Neighborhoods

- For node i, let a_i be vector indexed by node pairs. Non-zero entries: $a_i[i,j]=1$ if $j>i$ and $a_i[i,j]=-1$ if $j<i$.

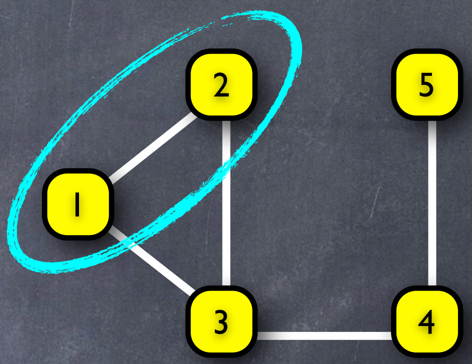
	{1,2}	{1,3}	{1,4}	{1,5}	{2,3}	{2,4}	{2,5}	{3,4}	{3,5}	{4,5}
$\mathbf{a}_1 =$	1	1	0	0	0	0	0	0	0	0
$\mathbf{a}_2 =$	-1	0	0	0	1	0	0	0	0	0



Ingredient 2: Sketching Neighborhoods

- For node i, let a_i be vector indexed by node pairs. Non-zero entries: $a_i[i,j]=1$ if $j>i$ and $a_i[i,j]=-1$ if $j<i$.

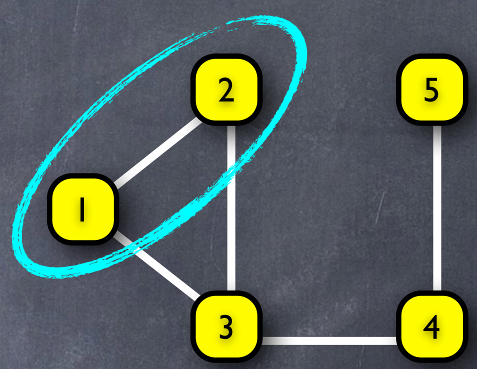
	{1,2}	{1,3}	{1,4}	{1,5}	{2,3}	{2,4}	{2,5}	{3,4}	{3,5}	{4,5}
$\mathbf{a}_1 =$	1	1	0	0	0	0	0	0	0	0
$\mathbf{a}_2 =$	−1	0	0	0	1	0	0	0	0	0



Ingredient 2: Sketching Neighborhoods

- For node i, let a_i be vector indexed by node pairs. Non-zero entries: $a_i[i,j]=1$ if $j>i$ and $a_i[i,j]=-1$ if $j<i$.

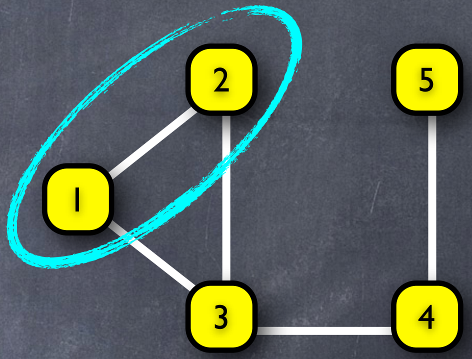
	{1,2}	{1,3}	{1,4}	{1,5}	{2,3}	{2,4}	{2,5}	{3,4}	{3,5}	{4,5}
$\mathbf{a}_1 =$	1	1	0	0	0	0	0	0	0	0
$\mathbf{a}_2 =$	−1	0	0	0	1	0	0	0	0	0
$\mathbf{a}_1 + \mathbf{a}_2 =$	0	1	0	0	1	0	0	0	0	0



Ingredient 2: Sketching Neighborhoods

- For node i, let a_i be vector indexed by node pairs. Non-zero entries: $a_i[i,j]=1$ if $j>i$ and $a_i[i,j]=-1$ if $j<i$.

	{1,2}	{1,3}	{1,4}	{1,5}	{2,3}	{2,4}	{2,5}	{3,4}	{3,5}	{4,5}
$\mathbf{a}_1 =$	1	1	0	0	0	0	0	0	0	0
$\mathbf{a}_2 =$	−1	0	0	0	1	0	0	0	0	0
$\mathbf{a}_1 + \mathbf{a}_2 =$	0	1	0	0	1	0	0	0	0	0

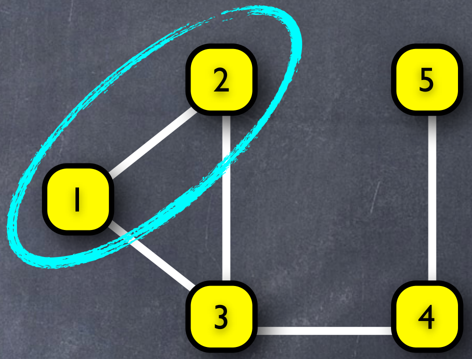


- Lemma For any subset of nodes $S \subset V$, non-zero entries of $\sum_{j \in S} a_j$ are edges across cut (S,V\S)

Ingredient 2: Sketching Neighborhoods

- For node i, let a_i be vector indexed by node pairs. Non-zero entries: $a_i[i,j]=1$ if $j>i$ and $a_i[i,j]=-1$ if $j<i$.

	{1,2}	{1,3}	{1,4}	{1,5}	{2,3}	{2,4}	{2,5}	{3,4}	{3,5}	{4,5}
$\mathbf{a}_1 =$	1	1	0	0	0	0	0	0	0	0
$\mathbf{a}_2 =$	-1	0	0	0	1	0	0	0	0	0
$\mathbf{a}_1 + \mathbf{a}_2 =$	0	1	0	0	1	0	0	0	0	0



- Lemma For any subset of nodes $S \subset V$, non-zero entries of $\sum_{j \in S} a_j$ are edges across cut (S,V\S)
- Player j sends $M(a_j)$ where M is "l_0 sampling" sketch.

Recipe: Sketch & Compute on Sketches

Recipe: Sketch & Compute on Sketches

- Player with Address Books: Player j sends Ma_j

Recipe: Sketch & Compute on Sketches

- Player with Address Books: Player j sends Ma_j
- Central Player: "Runs Algorithm in Sketch Space"

Recipe: Sketch & Compute on Sketches

- Player with Address Books: Player j sends Ma_j
- Central Player: "Runs Algorithm in Sketch Space"
  - Use Ma_j to get incident edge on each node j

Recipe: Sketch & Compute on Sketches

- Player with Address Books: Player j sends Ma_j
- Central Player: "Runs Algorithm in Sketch Space"
  - Use Ma_j to get incident edge on each node j
  - For i=2 to log n:
    - To get incident edge on component $S \subset V$ use:

Recipe: Sketch & Compute on Sketches

- Player with Address Books: Player j sends Ma_j
- Central Player: "Runs Algorithm in Sketch Space"
  - Use Ma_j to get incident edge on each node j
  - For i=2 to log n:
    - To get incident edge on component $S \subset V$ use:

$$\sum_{j \in S} M\mathbf{a}_j = M\left(\sum_{j \in S} \mathbf{a}_j\right)$$

Recipe: Sketch & Compute on Sketches

- Player with Address Books: Player j sends Ma_j
- Central Player: "Runs Algorithm in Sketch Space"
  - Use Ma_j to get incident edge on each node j
  - For i=2 to log n:
    - To get incident edge on component $S \subset V$ use:

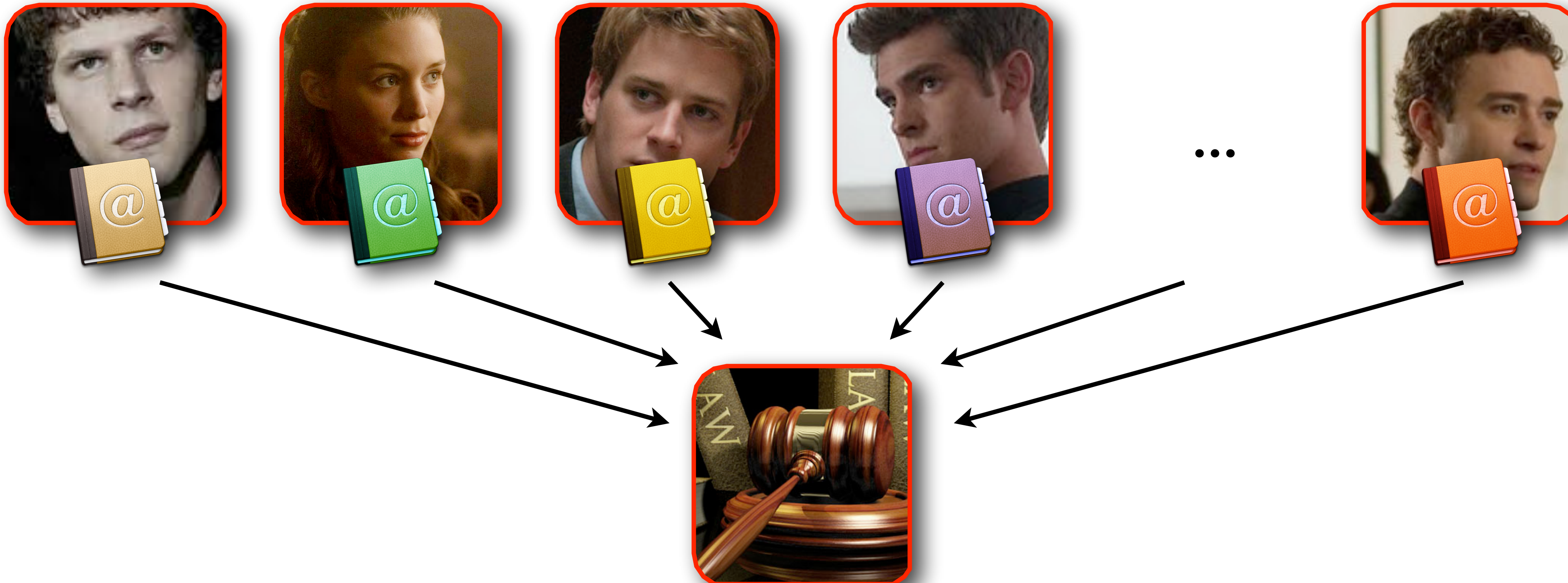
$$\sum_{j \in S} M\mathbf{a}_j = M(\sum_{j \in S} \mathbf{a}_j) \longrightarrow \text{non-zero elt. of } \sum_{j \in S} a_j = \text{edge across cut}$$

Recipe: Sketch & Compute on Sketches

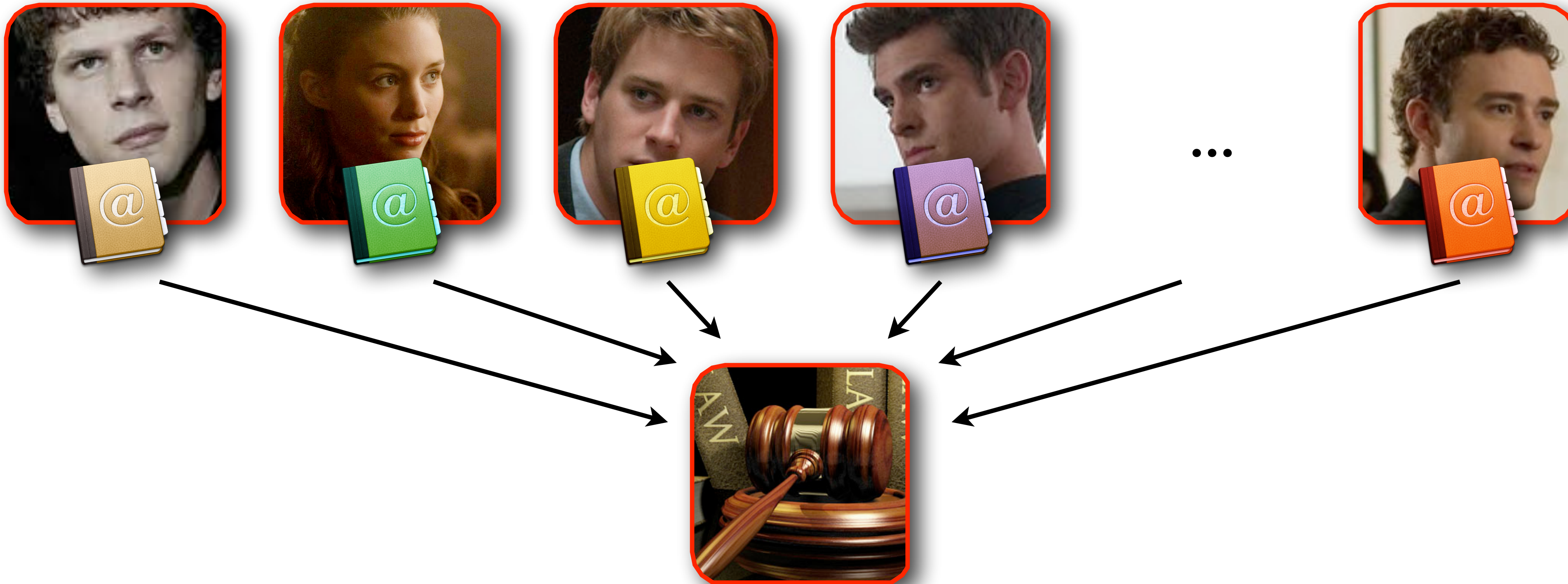
- Player with Address Books: Player j sends Ma_j
- Central Player: "Runs Algorithm in Sketch Space"
  - Use Ma_j to get incident edge on each node j
  - For i=2 to log n:
    - To get incident edge on component $S \subset V$ use:

$$\sum_{j \in S} M\mathbf{a}_j = M\Big(\sum_{j \in S} \mathbf{a}_j\Big) \longrightarrow \text{non-zero elt. of } \sum_{j \in S} a_j = \text{edge across cut}$$

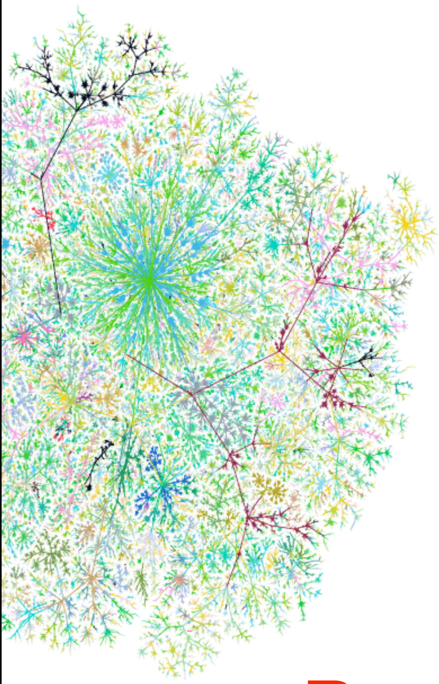
Detail: Actually each player sends log n independent sketches $M_1 a_j$, $M_2 a_j$, … and central player uses $M_i a_j$ when emulating i^{th} iteration of the algorithm.



- *Thm* O(polylog n) bit message from each player suffices.



- *Thm* O(polylog n) bit message from each player suffices.
- *Various extensions* For example, with Õ(k) bit messages, can find all edges that participate in cuts of size less than k.



Part II

Sketching

What is sketching?

Surprising connectivity example

Revisiting graph cuts and sparsification

- *Thm* O(ε^{-2} polylog n) bit messages suffice for central player to construct sparsifier and approx all graph cuts.

Guha, McGregor, Tench [PODS 15], *Kapralov et al.* [STOC 14]

- *Thm* O(ε^{-2} polylog n) bit messages suffice for central player to construct sparsifier and approx all graph cuts.

Guha, McGregor, Tench [PODS 15], *Kapralov et al.* [STOC 14]

- *Main Ideas*

1. For a graph G, can find all edges in small cuts.

- *Thm* $O(\varepsilon^{-2}$ polylog n) bit messages suffice for central player to construct sparsifier and approx all graph cuts.

  Guha, McGregor, Tench [PODS 15], *Kapralov et al.* [STOC 14]

- *Main Ideas*

  1. For a graph G, can find all edges in small cuts.
  2. For large cuts, suffices to sample edges with prob. 1/2.

- *Thm* O(ε^{-2} polylog n) bit messages suffice for central player to construct sparsifier and approx all graph cuts.

  Guha, McGregor, Tench [PODS 15], *Kapralov et al.* [STOC 14]

- *Main Ideas*

1. For a graph G, can find all edges in small cuts.
2. For large cuts, suffices to sample edges with prob. 1/2.
3. So, sparsifying G reduces to sparsifying sampled graph G'.

- *Thm* O(ε^{-2} polylog n) bit messages suffice for central player to construct sparsifier and approx all graph cuts.

  Guha, McGregor, Tench [PODS 15], *Kapralov et al.* [STOC 14]

- *Main Ideas*

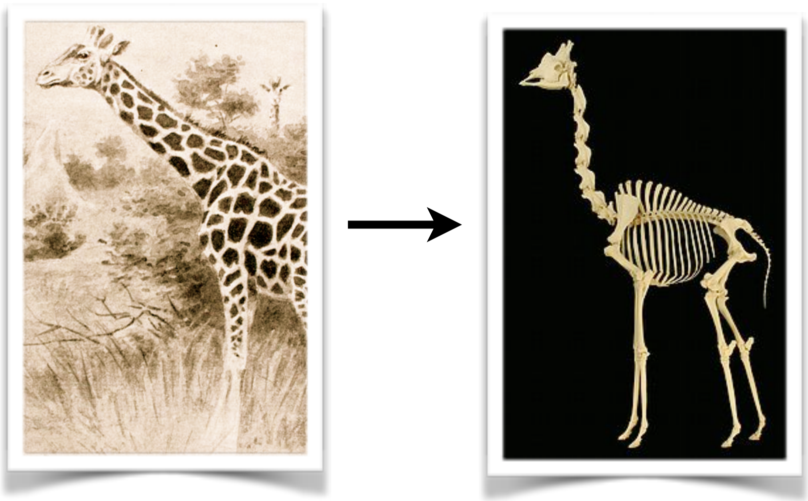
  1. For a graph G, can find all edges in small cuts.
  2. For large cuts, suffices to sample edges with prob. 1/2.
  3. So, sparsifying G reduces to sparsifying sampled graph G'.
  4. To sparsify G' recurse… Can do recursion in parallel.

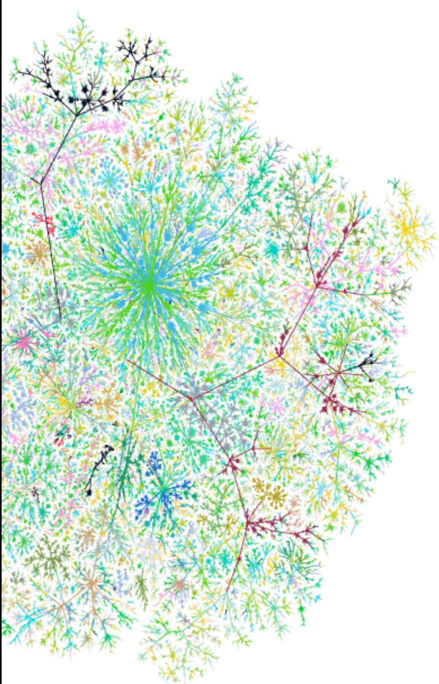
- *Thm* O(ε^{-2} polylog n) bit messages suffice for central player to construct sparsifier and approx all graph cuts.

Guha, McGregor, Tench [PODS 15], *Kapralov et al.* [STOC 14]

- *Main Ideas*

1. For a graph G, can find all edges in small cuts.
2. For large cuts, suffices to sample edges with prob. 1/2.
3. So, sparsifying G reduces to sparsifying sampled graph G'.
4. To sparsify G' recurse… Can do recursion in parallel.





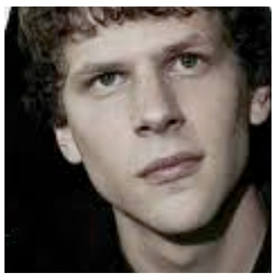
Part III

Streaming

Revisiting Matching
Correlation Clustering
Coloring Graphs
Coverage and Submodular Maximization

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.



Mark and Erica are now friends.

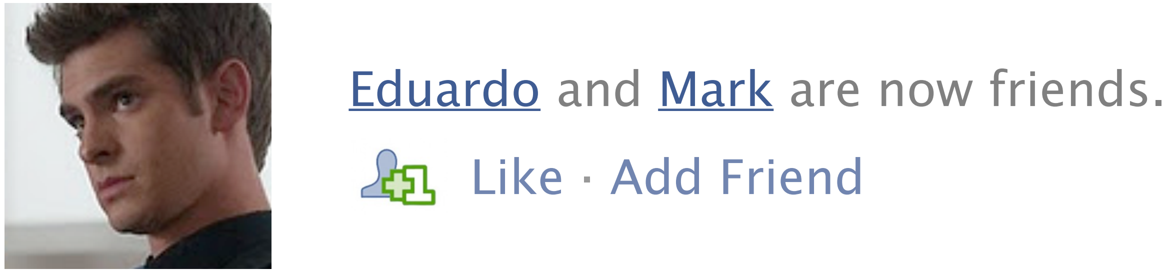
Like · Add Friend

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Mark and Erica are no longer friends.

Like · Add Friend

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.



- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Tyler and Cameron are friends with Mark.

Like · Add Friend

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Sean and Mark are now friends.

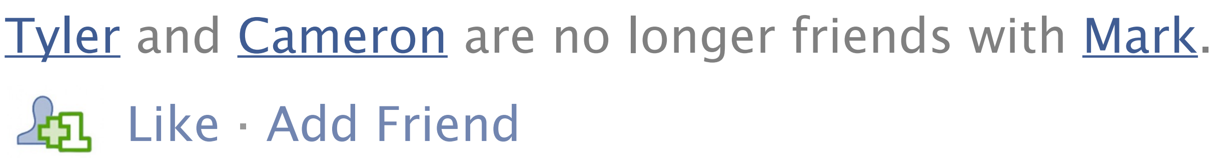
Like · Add Friend

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Eduardo and Mark are no longer friends.

Like · Add Friend

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.



- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Lawyers are now friends with everyone.

Like · Add Friend

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Lawyers are now friends with everyone.

Like · Add Friend

- *Goal* Using small memory, compute properties of the graph.

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Lawyers are now friends with everyone.

Like · Add Friend

- *Goal* Using small memory, compute properties of the graph.
- All the earlier algorithms apply in insert-delete model:

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Lawyers are now friends with everyone.

Like · Add Friend

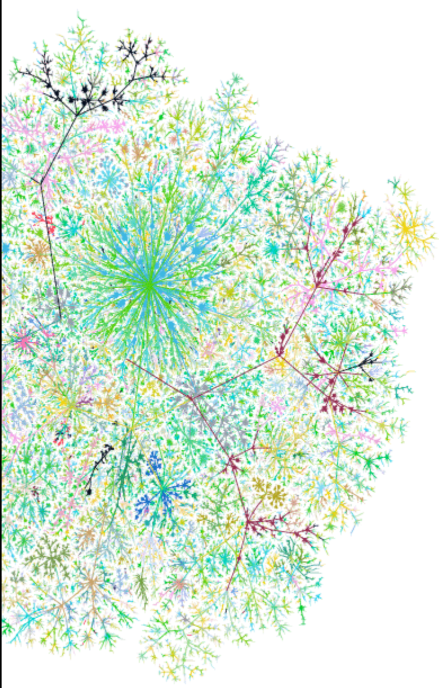
- *Goal* Using small memory, compute properties of the graph.
- All the earlier algorithms apply in insert-delete model:
  - Maintain sketch Mx where x is characteristic vector of edges.

- *Two Main Graph Stream Models*
  - *Insert-Only Model:* Input is a stream of edges.
  - *Insert-Delete Model:* Edge insertions and edge deletions.

Lawyers are now friends with everyone.

Like · Add Friend

- *Goal* Using small memory, compute properties of the graph.
- All the earlier algorithms apply in insert-delete model:
  - Maintain sketch Mx where x is characteristic vector of edges.
  - When e inserted, update sketch $Mx \leftarrow Mx + (e^{th}$ column of M)



Part III

Streaming

Revisiting Matching

Correlation Clustering

Coloring Graphs

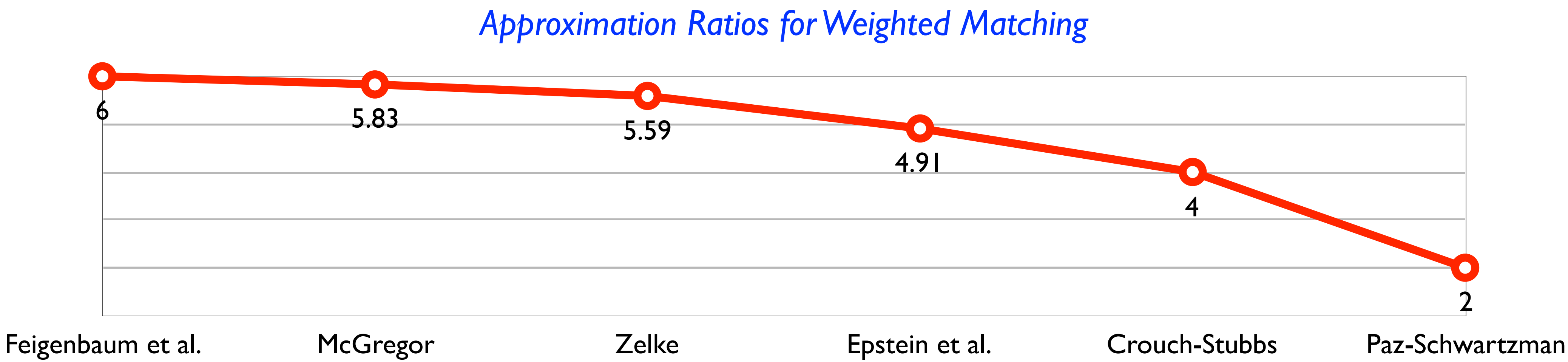
Coverage and Submodular Maximization

- *Unweighted Matching* Greedy algorithm returns 2-approx using $\tilde{O}(n)$ space. Embarrassingly, this is best known one-pass result!

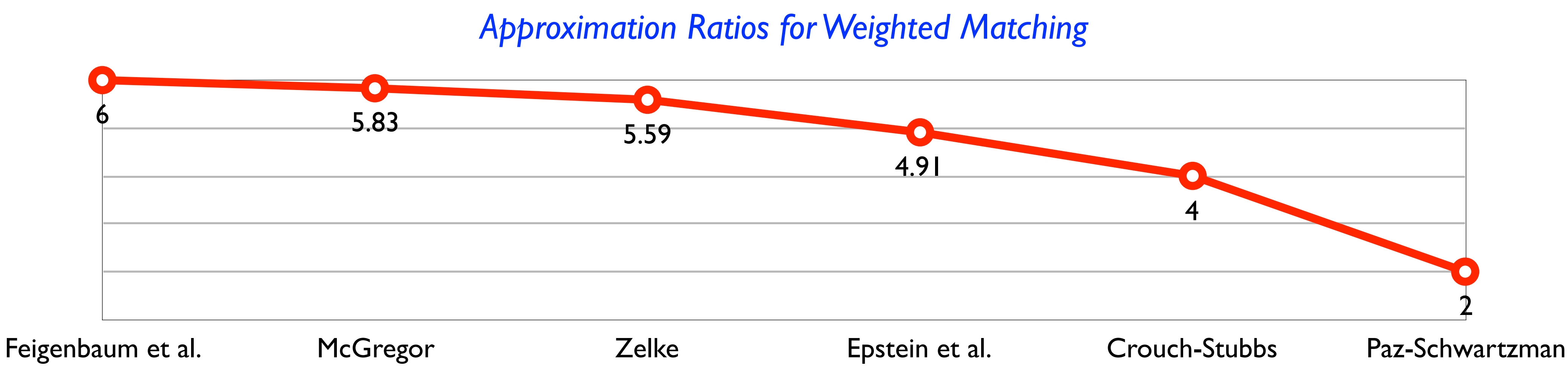
- *Unweighted Matching* Greedy algorithm returns 2-approx using Õ(n) space. Embarrassingly, this is best known one-pass result!

Approximation Ratios for Weighted Matching

- *Unweighted Matching* Greedy algorithm returns 2-approx using Õ(n) space. Embarrassingly, this is best known one-pass result!



- *Unweighted Matching* Greedy algorithm returns 2-approx using Õ(n) space. Embarrassingly, this is best known one-pass result!

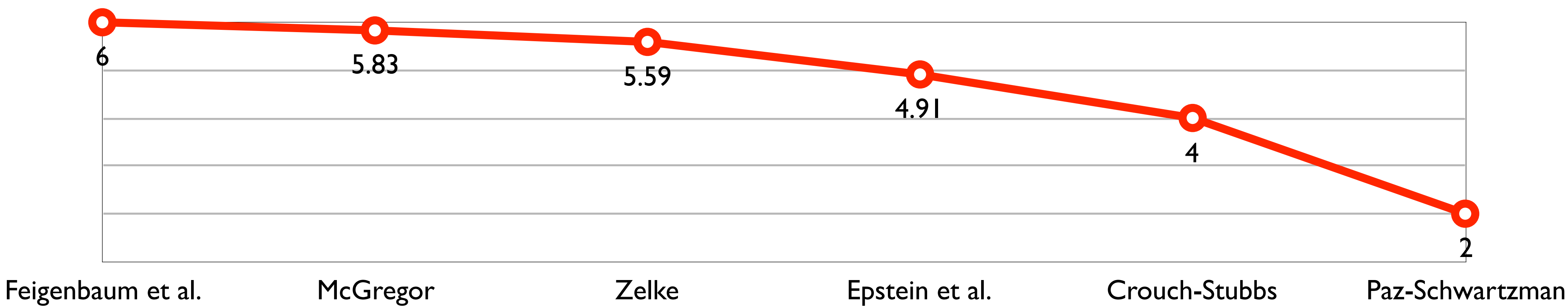


- *Weighted Matching* 2+ε approx in Õ(n/ε) space.

Paz, Schwartzman [SODA 17]

- *Unweighted Matching* Greedy algorithm returns 2-approx using Õ(n) space. Embarrassingly, this is best known one-pass result!

Approximation Ratios for Weighted Matching



- *Weighted Matching* 2+ε approx in Õ(n/ε) space.

  Paz, Schwartzman [SODA 17]

? Improve result for sparse graphs? Graph has arboricity α if all subgraphs have average degree < α. Planar graph has α=3.

- *Thm* $\alpha+2+\varepsilon$ approx of matching size in O(polylog n) space.

Cormode et al. [ESA 17], *McGregor, Vorotnikova* [SOSA 18]

- *Thm* α+2+ε approx of matching size in O(polylog n) space.
  Cormode et al. [ESA 17], *McGregor, Vorotnikova* [SOSA 18]

- *Define* Edge {u,v} is special if ≤ α edges incident to u and ≤ α edges incident to v later than {u,v}. Let s be # special edges.

- *Thm* $\alpha+2+\varepsilon$ approx of matching size in O(polylog n) space.
  Cormode et al. [ESA 17], *McGregor, Vorotnikova* [SOSA 18]
- *Define* Edge {u,v} is special if $\leq \alpha$ edges incident to u and $\leq \alpha$ edges incident to v later than {u,v}. Let s be # special edges.
- *Lemma* match(G)$\leq$s$\leq$(2+α)match(G).

- ***Thm*** $\alpha+2+\varepsilon$ approx of matching size in O(polylog n) space.
  Cormode et al. [ESA 17], *McGregor, Vorotnikova* [SOSA 18]

- ***Define*** Edge {u,v} is special if $\leq \alpha$ edges incident to u and $\leq \alpha$ edges incident to v later than {u,v}. Let s be # special edges.

- ***Lemma*** match(G)$\leq$s$\leq$(2+α)match(G).

  - ***Proof Ingredients*** Graph of special edges has degrees $\leq \alpha+1$. Low arboricity bounds number of almost special edges.

- *Thm* α+2+ε approx of matching size in O(polylog n) space.

  Cormode et al. [ESA 17], *McGregor, Vorotnikova* [SOSA 18]

- *Define* Edge {u,v} is special if ≤ α edges incident to u and ≤ α edges incident to v later than {u,v}. Let s be # special edges.

- *Lemma* match(G)≤s≤(2+α)match(G).

  - *Proof Ingredients* Graph of special edges has degrees ≤ α+1. Low arboricity bounds number of almost special edges.

- *Algorithm* Estimate s up to a factor 1+ε

- *Thm* $\alpha+2+\varepsilon$ approx of matching size in O(polylog n) space.
  Cormode et al. [ESA 17], *McGregor, Vorotnikova* [SOSA 18]
- *Define* Edge {u,v} is special if $\leq \alpha$ edges incident to u and $\leq \alpha$ edges incident to v later than {u,v}. Let s be # special edges.
- *Lemma* match(G)$\leq$s$\leq$(2+α)match(G).
  - *Proof Ingredients* Graph of special edges has degrees $\leq \alpha+1$. Low arboricity bounds number of almost special edges.
- *Algorithm* Estimate s up to a factor $1+\varepsilon$

  a) Suppose we have guess g that is 2-approximates s

- *Thm* α+2+ε approx of matching size in O(polylog n) space.
  Cormode et al. [ESA 17], *McGregor, Vorotnikova* [SOSA 18]
- *Define* Edge {u,v} is special if ≤ α edges incident to u and ≤ α edges incident to v later than {u,v}. Let s be # special edges.
- *Lemma* match(G)≤s≤(2+α)match(G).
  - *Proof Ingredients* Graph of special edges has degrees ≤ α+1. Low arboricity bounds number of almost special edges.
- *Algorithm* Estimate s up to a factor 1+ε

  a) Suppose we have guess g that is 2-approximates s

  b) Sample each edge w/p ≈ ε^{-2} (log n)/g. If you subsequently see >α edges incident to either endpoint, drop the edge.

- *Thm* $\alpha+2+\varepsilon$ approx of matching size in O(polylog n) space.
  Cormode et al. [ESA 17], *McGregor, Vorotnikova* [SOSA 18]

- *Define* Edge {u,v} is special if $\leq \alpha$ edges incident to u and $\leq \alpha$ edges incident to v later than {u,v}. Let s be # special edges.

- *Lemma* $\text{match}(G) \leq s \leq (2+\alpha)\text{match}(G)$.

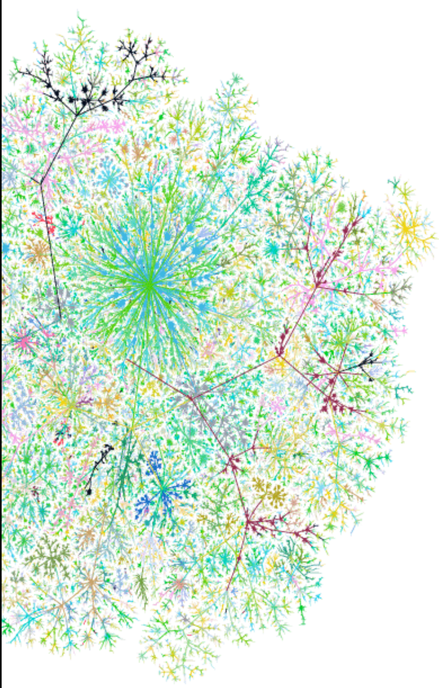
  - *Proof Ingredients* Graph of special edges has degrees $\leq \alpha+1$. Low arboricity bounds number of almost special edges.

- *Algorithm* Estimate s up to a factor $1+\varepsilon$

  a) Suppose we have guess g that is 2-approximates s

  b) Sample each edge w/p $\approx \varepsilon^{-2} (\log n)/g$. If you subsequently see $>\alpha$ edges incident to either endpoint, drop the edge.

- Can show a) the current sample size is always small and b) size of final sample and g yields good approx for s.



Part III

Streaming

Revisiting Matching

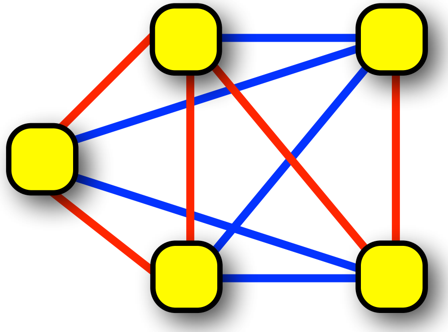
Correlation Clustering

Coloring Graphs

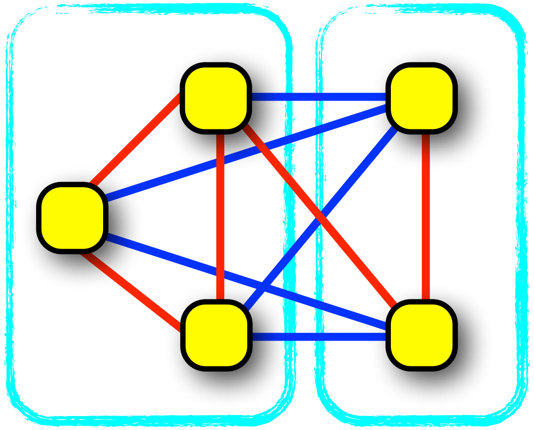
Coverage and Submodular Maximization

- Consider a complete graph where edges are labelled attractive or repulsive. Given a node partition, an attractive edge is sad if it is cut and a repulsive edge is sad if it is not cut.

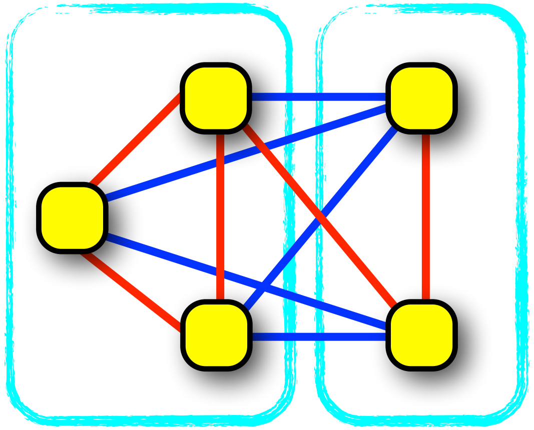
- Consider a complete graph where edges are labelled attractive or repulsive. Given a node partition, an attractive edge is sad if it is cut and a repulsive edge is sad if it is not cut.



- Consider a complete graph where edges are labelled attractive or repulsive. Given a node partition, an attractive edge is sad if it is cut and a repulsive edge is sad if it is not cut.



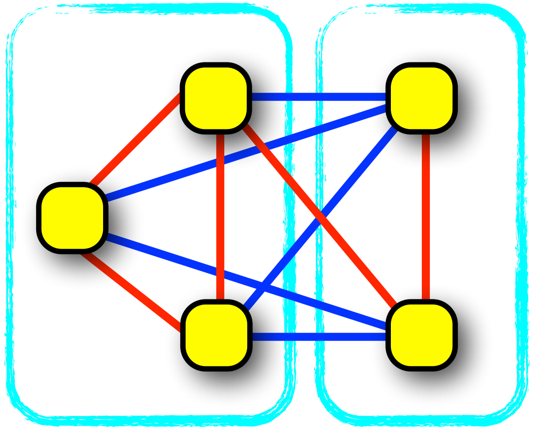
- Consider a complete graph where edges are labelled attractive or repulsive. Given a node partition, an attractive edge is sad if it is cut and a repulsive edge is sad if it is not cut.



- *Correlation Clustering* Find partition minimizing # sad edges.

See tutorial *Bonchi, Garcia-Soriano, Liberty* [KDD 14]

- Consider a complete graph where edges are labelled attractive or repulsive. Given a node partition, an attractive edge is sad if it is cut and a repulsive edge is sad if it is not cut.



- *Correlation Clustering* Find partition minimizing # sad edges.

  See tutorial *Bonchi, Garcia-Soriano, Liberty* [KDD 14]

- *3-Approx Algorithm* a) Pick random node. b) Form cluster with it and its attracted neighbors. c) Remove cluster from graph and repeat until nodes remain. *Ailon, Charikar, Newman* [J.ACM 08]

- *Emulating algorithm in two passes:*

- *Emulating algorithm in two passes:*
  - *Preprocess* Randomly order nodes, $v_1, v_2, \ldots, v_n$.

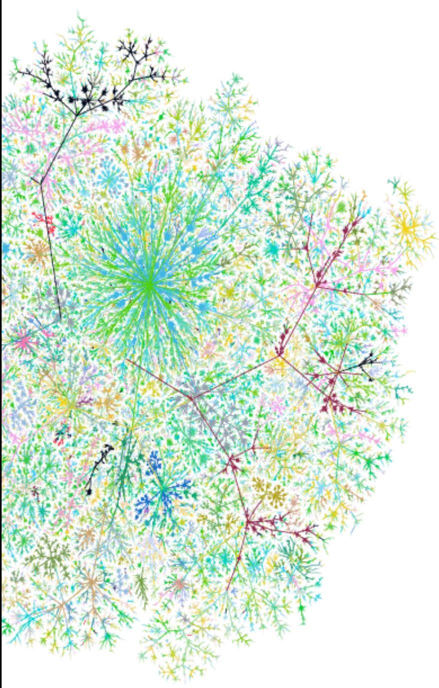
- *Emulating algorithm in two passes:*
  - *Preprocess* Randomly order nodes, $v_1, v_2, \ldots, v_n$.
  - *First Pass* Store all attractive edges incident to $\{v_1, \ldots, v_{\sqrt{n}}\}$. Now can emulate first $\sqrt{n}$ iterations of the algorithm.

- *Emulating algorithm in two passes:*
  - *Preprocess* Randomly order nodes, $v_1, v_2, \ldots, v_n$.
  - *First Pass* Store all attractive edges incident to $\{v_1, \ldots, v_{\sqrt{n}}\}$. Now can emulate first $\sqrt{n}$ iterations of the algorithm.
  - *Second Pass* Store all remaining attractive edges. Now can emulate remaining steps of the algorithm.

- *Emulating algorithm in two passes:*
    - *Preprocess* Randomly order nodes, $v_1, v_2, \ldots, v_n$.
    - *First Pass* Store all attractive edges incident to $\{v_1, \ldots, v_{\sqrt{n}}\}$. Now can emulate first $\sqrt{n}$ iterations of the algorithm.
    - *Second Pass* Store all remaining attractive edges. Now can emulate remaining steps of the algorithm.
- *Thm* Algorithm uses $\tilde{O}(n^{1.5})$ space. *Ahn et al.* [ICML 16]

- *Emulating algorithm in two passes:*
  - *Preprocess* Randomly order nodes, $v_1, v_2, \ldots, v_n$.
  - *First Pass* Store all attractive edges incident to $\{v_1, \ldots, v_{\sqrt{n}}\}$. Now can emulate first $\sqrt{n}$ iterations of the algorithm.
  - *Second Pass* Store all remaining attractive edges. Now can emulate remaining steps of the algorithm.
- *Thm* Algorithm uses $\tilde{O}(n^{1.5})$ space. *Ahn et al.* [ICML 16]
  - *Proof Idea* At most $n^{1.5}$ edges stored in first pass. In second, pass, can show remaining node have at most $n^{0.5}$ neighbors.

- *Emulating algorithm in two passes:*
    - *Preprocess* Randomly order nodes, $v_1, v_2, \ldots, v_n$.
    - *First Pass* Store all attractive edges incident to $\{v_1, \ldots, v_{\sqrt{n}}\}$. Now can emulate first $\sqrt{n}$ iterations of the algorithm.
    - *Second Pass* Store all remaining attractive edges. Now can emulate remaining steps of the algorithm.
- *Thm* Algorithm uses $\tilde{O}(n^{1.5})$ space. *Ahn et al.* [ICML 16]
    - *Proof Idea* At most $n^{1.5}$ edges stored in first pass. In second, pass, can show remaining node have at most $n^{0.5}$ neighbors.
- With more work, can get $\tilde{O}(n)$ space with $O(\log \log n)$ passes. Can also find maximal independent sets.



Part III

Streaming

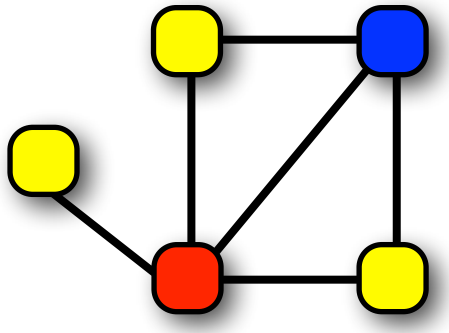
Revisiting Matching

Correlation Clustering

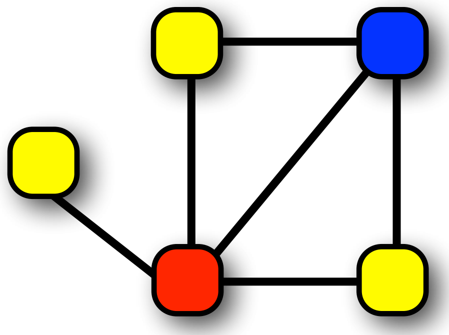
Coloring Graphs

Coverage and Submodular Maximization

- *Coloring* With min number of colors, assign a color to every node such that no edge has monochromatic endpoints.

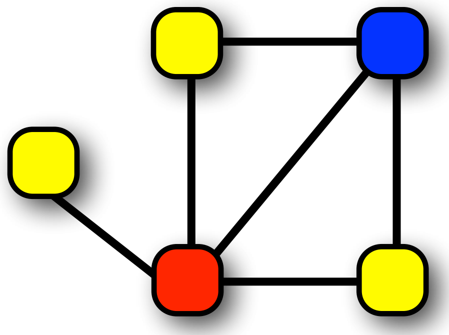


- *Coloring* With min number of colors, assign a color to every node such that no edge has monochromatic endpoints.



- *Thm* Can color a graph in $\Delta+1$ colors where Δ is max degree.

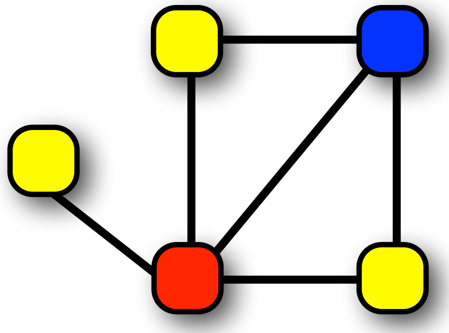
- *Coloring* With min number of colors, assign a color to every node such that no edge has monochromatic endpoints.



- *Thm* Can color a graph in $\Delta+1$ colors where Δ is max degree.

? How can we do this in a few passes with $\tilde{O}(n)$ space?

- *Coloring* With min number of colors, assign a color to every node such that no edge has monochromatic endpoints.

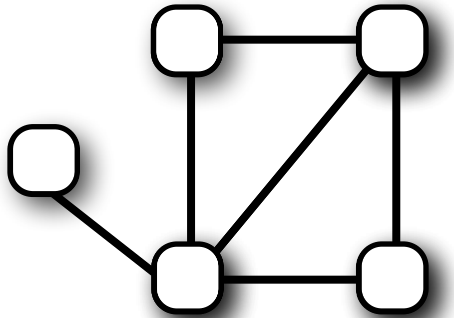


- *Thm* Can color a graph in $\Delta+1$ colors where Δ is max degree.

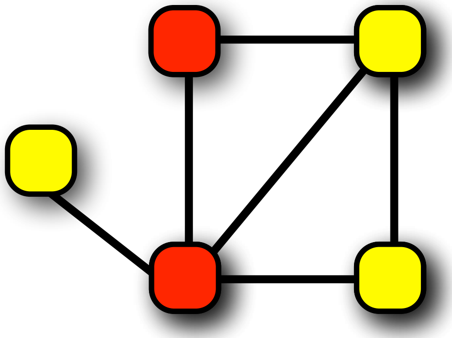
? How can we do this in a few passes with $\tilde{O}(n)$ space?

- $O(\Delta \log \log n)$ passes via independent sets. Let's do better!

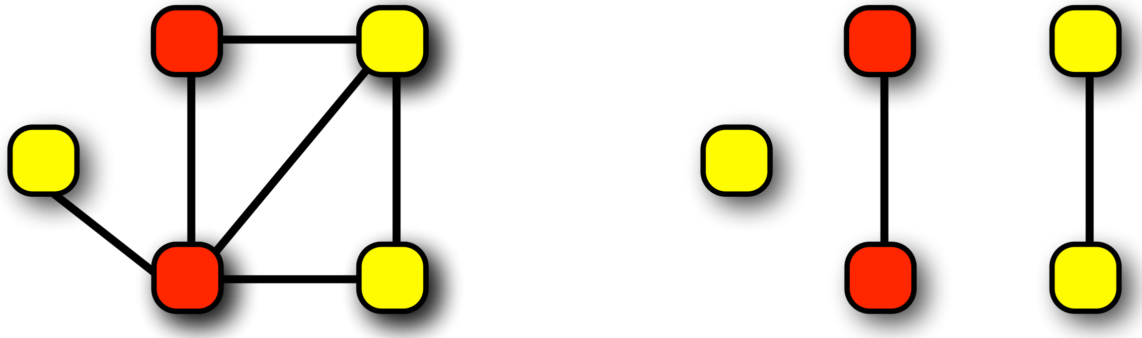
- (1+ε)Δ *Coloring* a) Randomly color with Δ/r colors. b) Store edges E' with monochromatic endpoints. c) Shade colors such that E' edges no longer monochromatic. *Bera, Ghosh* [ArXiv 18]



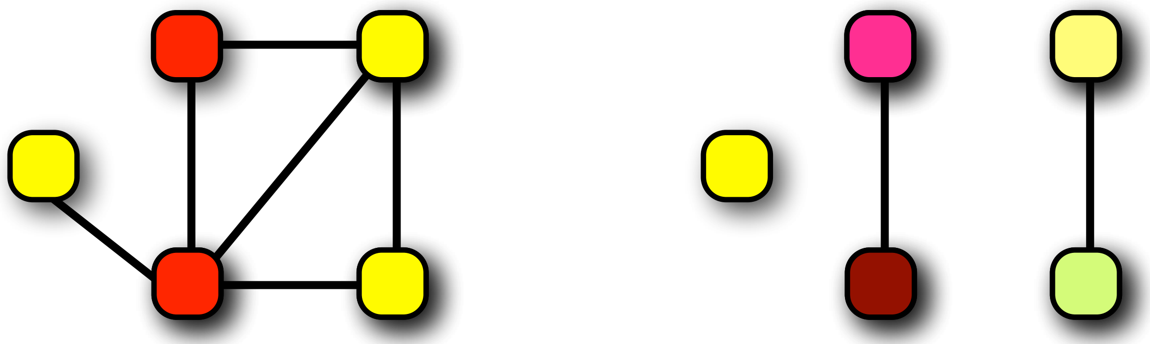
- <u>$(1+\varepsilon)\Delta$ *Coloring*</u> a) Randomly color with Δ/r colors. b) Store edges E' with monochromatic endpoints. c) Shade colors such that E' edges no longer monochromatic. *Bera, Ghosh* [ArXiv 18]



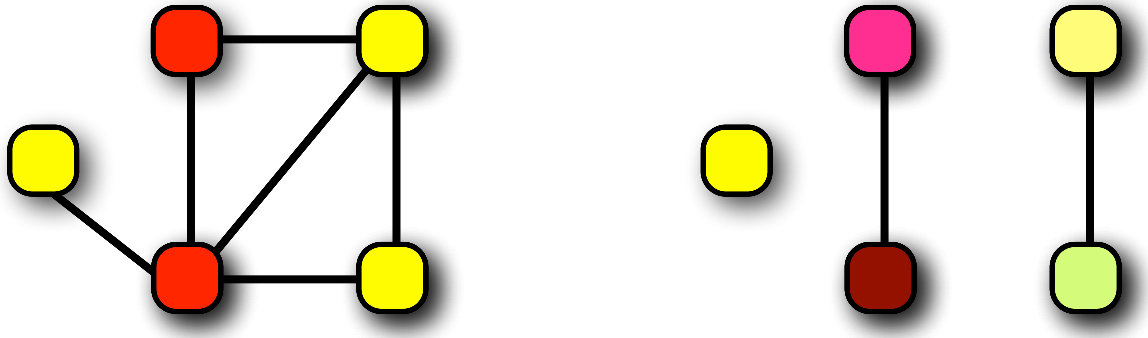
- <u>(1+ε)Δ *Coloring*</u> a) Randomly color with Δ/r colors. b) Store edges E' with monochromatic endpoints. c) Shade colors such that E' edges no longer monochromatic. *Bera, Ghosh* [ArXiv 18]



- <u>(1+ε)Δ *Coloring*</u> a) Randomly color with Δ/r colors. b) Store edges E' with monochromatic endpoints. c) Shade colors such that E' edges no longer monochromatic. *Bera, Ghosh* [ArXiv 18]

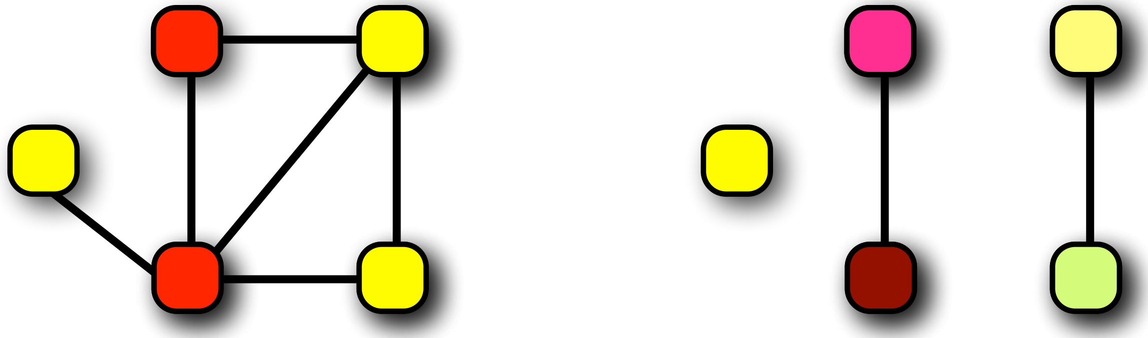


- *(1+ε)Δ Coloring* a) Randomly color with Δ/r colors. b) Store edges E' with monochromatic endpoints. c) Shade colors such that E' edges no longer monochromatic. *Bera, Ghosh* [ArXiv 18]



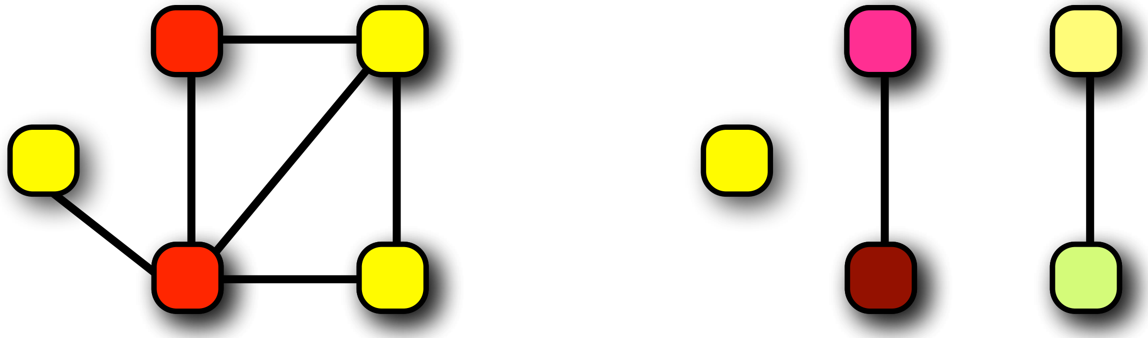
- *Space Analysis* |E'|=O(nr) since probability edge in E' is r/Δ.

- *(1+ε)Δ Coloring* a) Randomly color with Δ/r colors. b) Store edges E’ with monochromatic endpoints. c) Shade colors such that E’ edges no longer monochromatic. *Bera, Ghosh* [ArXiv 18]

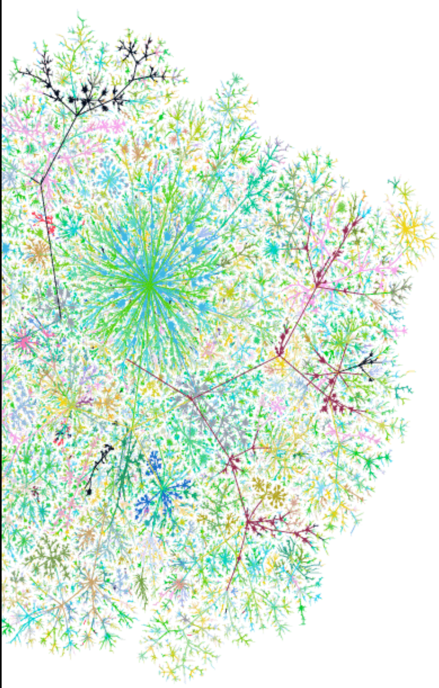


- *Space Analysis* |E’|=O(nr) since probability edge in E’ is r/Δ.
- *Colors Analysis* If $r \approx \varepsilon^{-2} \log n$, max degree in E’ is $\Delta_{E'} < (1+\varepsilon)r$ and final number of colors is $(1+\Delta_{E'})\Delta/r = (1+\varepsilon)\Delta$.

- *(1+ε)Δ Coloring* a) Randomly color with Δ/r colors. b) Store edges E' with monochromatic endpoints. c) Shade colors such that E' edges no longer monochromatic. *Bera, Ghosh* [ArXiv 18]



- *Space Analysis* $|E'|=O(nr)$ since probability edge in E' is r/Δ.
- *Colors Analysis* If $r \approx \varepsilon^{-2} \log n$, max degree in E' is $\Delta_{E'} < (1+\varepsilon)r$ and final number of colors is $(1+\Delta_{E'})\Delta/r = (1+\varepsilon)\Delta$.
- *Δ+1 Coloring Idea* For node v, pick $S_v \subset \{1,\ldots,\Delta+1\}$ of $O(\log n)$ random colors. May assume v's color in S_v. *Assadi et al.* [ArXiv 18]

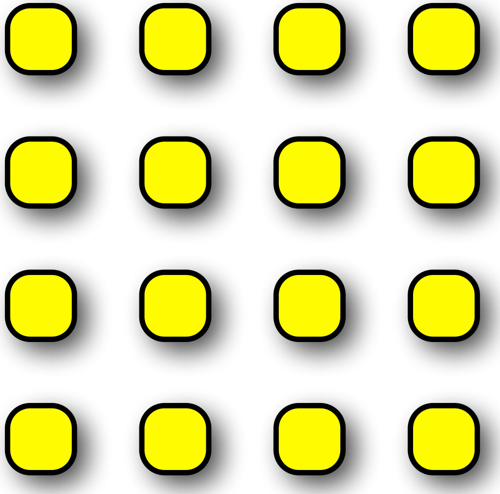


Part III

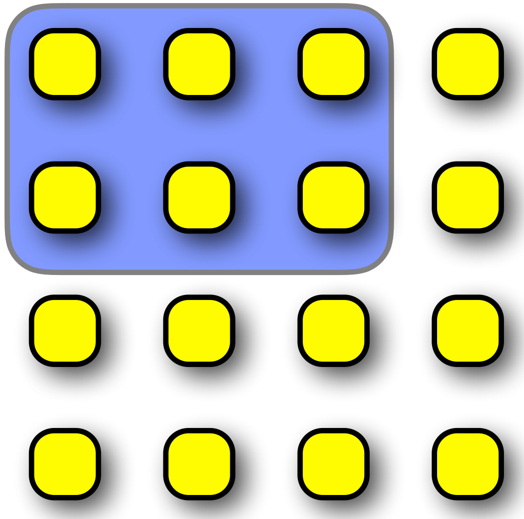
Streaming

Revisiting Matching
Correlation Clustering
Coloring Graphs
Coverage and Submodular Maximization

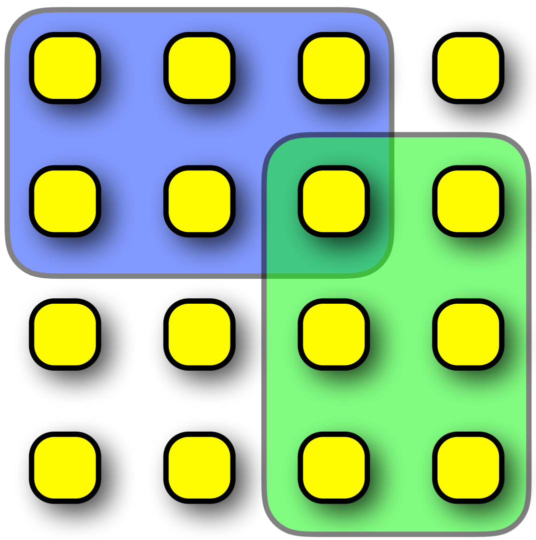
- ***Max-k-Coverage*** Given a stream of subsets $S_1, \ldots, S_m$ of $[n]$, find C that maximizes $f(C)=|\cup_{i\in C} S_i|$ subject to $|C|\leq k$.



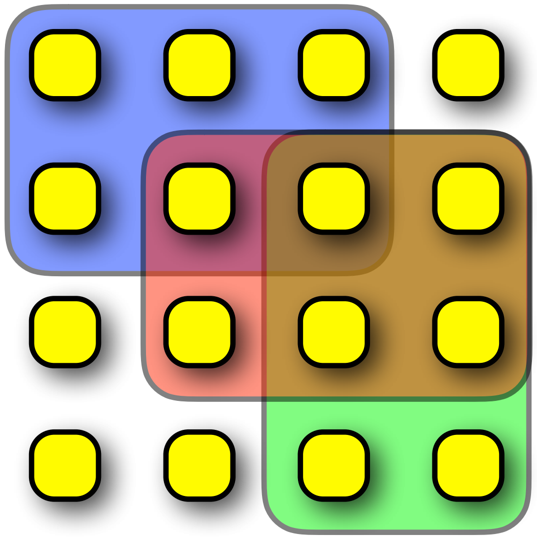
- *Max-k-Coverage* Given a stream of subsets $S_1, \ldots, S_m$ of $[n]$, find C that maximizes $f(C)=|\cup_{i\in C} S_i|$ subject to $|C|\leq k$.



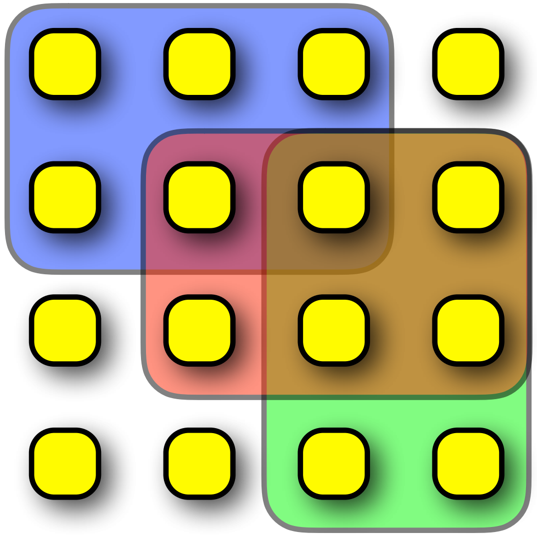
- *Max-k-Coverage* Given a stream of subsets $S_1, \ldots, S_m$ of $[n]$, find C that maximizes $f(C)=|\cup_{i \in C} S_i|$ subject to $|C| \leq k$.



- *Max-k-Coverage* Given a stream of subsets $S_1, \ldots, S_m$ of $[n]$, find C that maximizes $f(C)=|\cup_{i \in C} S_i|$ subject to $|C| \leq k$.



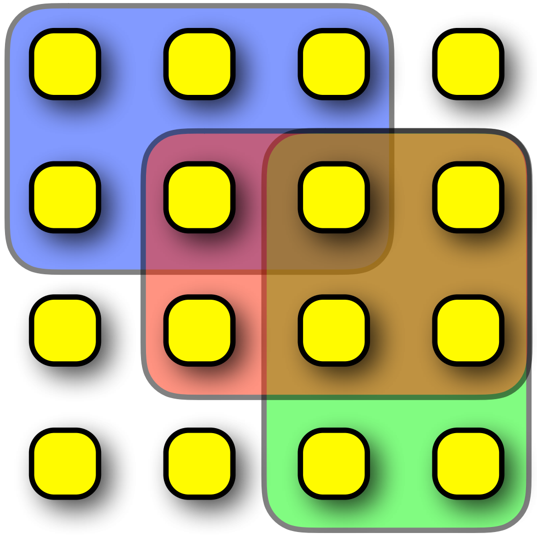
- ***Max-k-Coverage*** Given a stream of subsets $S_1, \ldots, S_m$ of $[n]$, find C that maximizes $f(C)=|\cup_{i \in C} S_i|$ subject to $|C| \leq k$.



- ***Submodular Functions*** f is sub-modular if for $A \subset B$ and $x \notin B$,

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

- *Max-k-Coverage* Given a stream of subsets $S_1, \ldots, S_m$ of $[n]$, find C that maximizes $f(C)=|\cup_{i\in C} S_i|$ subject to $|C|\leq k$.



- *Submodular Functions* f is sub-modular if for $A\subset B$ and $x\notin B$,

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

- *Thm* $(1-\varepsilon)/2$ approx. of max-coverage in $\tilde{O}(\varepsilon^{-3}k)$ space.

McGregor, Vu [ICDT 17]

- *Algorithm* Guess g such that $OPT \leq g \leq (1+\varepsilon)OPT$. Add first $\leq k$ sets that each cover at least $g/(2k)$ new elements.

- *Algorithm* Guess g such that OPT≤g≤(1+ε)OPT. Add first ≤k sets that each cover at least g/(2k) new elements.
- *Approx Ratio* If k sets added, we cover g/2≥OPT/2. If less sets added, each set not added covers <g/(2k) new elements and hence we covered OPT-g/2≥OPT(1-ε)/2.

- *Algorithm* Guess g such that $OPT \leq g \leq (1+\varepsilon)OPT$. Add first $\leq k$ sets that each cover at least $g/(2k)$ new elements.
- *Approx Ratio* If k sets added, we cover $g/2 \geq OPT/2$. If less sets added, each set not added covers $<g/(2k)$ new elements and hence we covered $OPT-g/2 \geq OPT(1-\varepsilon)/2$.
- *Reducing Space* Above algorithm requires $\tilde{O}(\varepsilon^{-1} OPT)$ space. Can use subsampling to such that $OPT = \tilde{O}(\varepsilon^{-2} k)$.

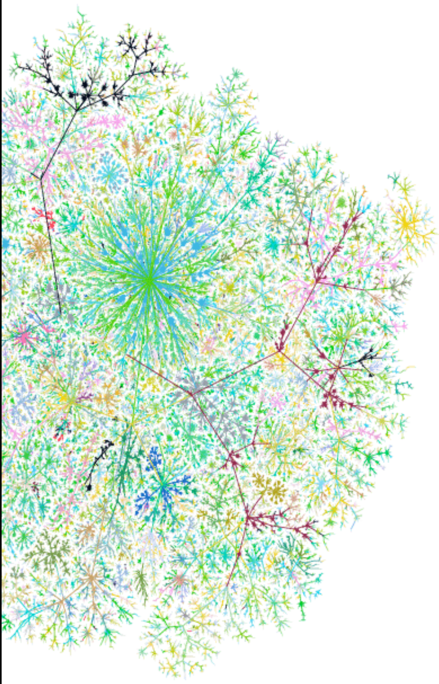
- *Algorithm* Guess g such that $OPT \leq g \leq (1+\varepsilon)OPT$. Add first $\leq k$ sets that each cover at least $g/(2k)$ new elements.

- *Approx Ratio* If k sets added, we cover $g/2 \geq OPT/2$. If less sets added, each set not added covers $<g/(2k)$ new elements and hence we covered $OPT-g/2 \geq OPT(1-\varepsilon)/2$.

- *Reducing Space* Above algorithm requires $\tilde{O}(\varepsilon^{-1} OPT)$ space. Can use subsampling to such that $OPT = \tilde{O}(\varepsilon^{-2} k)$.

- *Generalizations* Constant passes for $\approx 1-1/e$ approx. Extends to other monotone submodular function. Other work on non-monotone functions, beyond cardinality constraints, etc.

McGregor, Vu [ICDT 17], *Bateni et al.* [SPAA 17], *Assadi* [PODS 17]



Thanks! Over to Sudipto…