Topic 11: Sketching Graphs

David Woodruff

Outline

- Sketching Model
  - Estimating the Euclidean norm of a vector
  - Finding a non-zero coordinate of a vector
- Graph sketching motivation
  - Boruvka's spanning tree algorithm
  - Finding a spanning tree from a sketch

Sketching

- Random linear projection $S: R^n \rightarrow R^k$ that preserves properties of any $x \in R^n$ with high probability ,where $k \ll n$

$$\begin{pmatrix} \quad S \quad \end{pmatrix} \begin{pmatrix} \\ x \\ \\ \end{pmatrix} = \begin{pmatrix} Sx \end{pmatrix} \longrightarrow \text{answer}$$

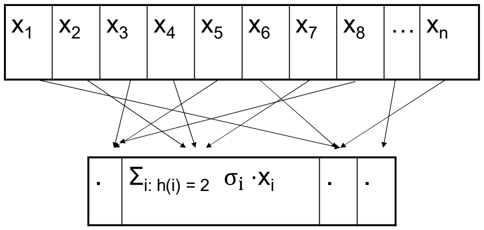
- The matrix S does not depend on x, e.g., S is a random matrix (typically, we require the entries of S be O(log n) bits)

Outline

- Sketching Model
  - Estimating the Euclidean norm of a vector
  - Finding a non-zero coordinate of a vector
- Graph sketching motivation
  - Boruvka's spanning tree algorithm
  - Finding a spanning tree from a sketch

Estimating the Norm of a Vector

- For a vector $x \in R^n$, its (squared) Euclidean norm is $|x|^2 = \sum_i x_i^2$
- Want to output a number Z for which $(1-\epsilon)|x|^2 \le Z \le (1+\epsilon)|x|^2$
- Choose a 2-wise independent hash function h:[n] -> [k]
- Choose a 4-wise independent hash function $\sigma : [n] \to \{-1,1\}$



CountSketch

- CountSketch is a linear map $S: R^n \to R^k$
- A row i of S is a hash bucket, and $(Sx)_i$ is the value in the bucket
- Output $|Sx|^2$

- $E[|Sx|^2] = E[\sum_j (\sum_i \delta(h(i) = j)\sigma(i)\, x_i)^2]$

$$= \sum_j \sum_{i,i'} x_i x_{i'} E[\delta(h(i) = j)\delta(h(i') = j)\sigma(i)\sigma(i')]$$

$$= \sum_j \sum_{i,i'} x_i x_{i'}\, E[\delta(h(i) = j)]E[\delta(h(i') = j)]E[\sigma(i)]E[\sigma(i')]$$

$$= \frac{\sum_j \sum_i x_i^2}{k} = |x|^2$$

Estimating the Norm from CountSketch

- In recitation, you will show $Var[|Sx|^2] = O(|x|^4/k)$

- By Chebyshev's inequality,

$$\Pr\left[\left||Sx|^2 - |x|^2\right| > \epsilon|x|^2\right] \le \frac{Var\left[|Sx|^2\right]}{\epsilon^2|x|^4} \le \frac{1}{10} \text{ provided } k = \Theta(\frac{1}{\epsilon^2})$$

- If S has $k = \Theta(\frac{1}{\epsilon^2})$ rows, can estimate $|x|^2$ from Sx up to a $(1+\epsilon)$-factor with probability at least 9/10

Outline

- Sketching Model
  - Estimating the Euclidean norm of a vector
  - Finding a non-zero coordinate of a vector
- Graph sketching motivation
  - Boruvka's spanning tree algorithm
  - Finding a spanning tree from a sketch

A 1-Sparse Recovery Algorithm

- Underlying n-dimensional vector x initialized to 0^n
- Stream of updates $x_i \leftarrow x_i + \Delta_j$ for Δ_j in {-1,1}
  - Promised that at all times, $-\mathrm{poly}(n) \le x_i \le \mathrm{poly}(n)$
- Want a procedure which with probability 1-1/poly(n),
  - if x is 1-sparse, i.e., has exactly one non-zero entry x_i, it returns (x_i, i)
  - otherwise output FAIL

A 1-Sparse Recovery Algorithm

- Choose 2-universal hash functions $h_1, \ldots, h_{2\log n}$ each mapping [n] to {0,1}
  - Let $x_j^{i,0} = x_j$ if $h_i(j) = 0$, else $x_j^{i,0} = 0$
  - Let $x_j^{i,1} = x_j$ if $h_i(j) = 1$, else $x_j^{i,0} = 0$
  - For each i ∈ {1, 2, …, 2log n}, $x = x^{i,0} + x^{i,1}$
- Let $S^{i,b}$ be a CountSketch, and maintain $S^{i,b} \cdot x^{i,b}$ for i ∈ [n] and b ∈ {0,1}

- Output:
  - If there's an i with $\left|S^{i,0} \cdot x^{i,0}\right|^2 > 0$ and $\left|S^{i,1} \cdot x^{i,1}\right|^2 > 0$, output FAIL
  - If for all i, $\left|S^{i,0} \cdot x^{i,0}\right|^2 = \left|S^{i,1} \cdot x^{i,1}\right|^2 = 0$, output FAIL
  - For each i, if there's a b in {0,1} with $\left|S^{i,b} \cdot x^{i,b}\right|^2 > 0$, let $T^i = h^{-1}(b)$, else let T^i = [n]
  - If $\left|\cap_i T^i\right| = 1$, output the item in $\cap_i T^i$, else output FAIL

A 1-Sparse Recovery Algorithm

- If there's an i with $\left|S^{i,0} \cdot x^{i,0}\right|^2 > 0$ and $\left|S^{i,1} \cdot x^{i,1}\right|^2 > 0$, output FAIL
- If for all i, $\left|S^{i,0} \cdot x^{i,0}\right|^2 = \left|S^{i,1} \cdot x^{i,1}\right|^2 = 0$, output FAIL
- For each i, if there's a b ∈ {0,1} with $\left|S^{i,b} \cdot x^{i,b}\right|^2 > 0$, let $T^i = h_i^{-1}(b)$, else T^i = [n]
- If $\left|\cap_i T^i\right| = 1$, output the item in $\cap_i T^i$, else output FAIL

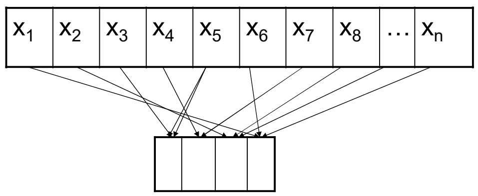
- With probability 1, if x = 0^n, output FAIL. *Why?*
- With probability 1-1/poly(n), if x has more than one non-zero entry, output FAIL. *Why?*
- If x is 1-sparse with non-zero entry j, for any $j' \neq j$, $\Pr[h_i(j) = h_i(j')] = 1/2$
- $\Pr\left[j' \in \cap_i T^i\right] \le \frac{1}{2^{2\log n}} \le \frac{1}{n^2}$, so $\Pr\left[\exists j' \neq j \text{ with } j' \in \cap_i T^i\right] \le \frac{n}{n^2} = \frac{1}{n}$
- $j \in T^i$ for every i, so with probability $1 - \frac{1}{n}$, we return j

Outputting a Non-Zero Coordinate of a Vector

- Maintain $S^{i,b} \cdot x^{i,b}$ for i ∈ [n] and b ∈ {0,1} in the stream
  - Easy to maintain with positive and negative updates to coordinates
  - $O(\log^2 n)$ bits of space
- With probability 1-1/n,
  - If x is 1-sparse, correctly output single non-zero coordinate j
  - Otherwise, correctly output FAIL
- Call this algorithm 1-Sparse-Finder
- *Can we use 1-Sparse-Finder to find a non-zero item of x if x is not 1-sparse?*

Outputting a Non-Zero Coordinate of a k-Sparse Vector

- If x is k-sparse, i.e., has k non-zero entries, use hashing
- Let h be a 2-universal hash function from [n] to [10k]



- In the j-th hash bucket, run 1-Sparse Finder

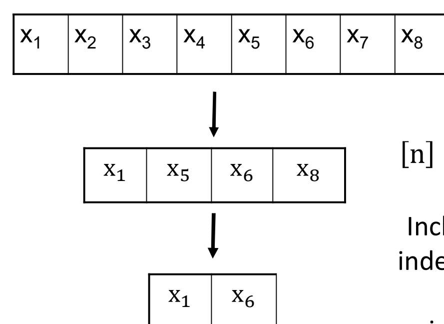
k-Sparse Algorithm Analysis

- In each bucket, we find a non-zero entry i or output FAIL, with probability 1-1/poly(n)
- *What if all the non-zero items of x collide in a bucket?*
- Consider a non-zero entry i of x
- Since h is 2-universal, with probability at least 1-k/(10k) = 9/10,
  $h(i) \neq h(j)$ for all $j \neq i$ for which $x_j \neq 0$
- With 9/10 probability, we output a non-zero entry i of x
- We know when we fail to output a non-zero entry (except with probability 1/poly(n))

Reducing the Space

- Have a procedure which, if x is k-sparse, either outputs a non-zero item i of x or says FAIL
- Output a non-zero item with probability at least 9/10
- Use O(k log n) bits of space
- Good if k is small, but how can we reduce the space for large k?

Subsampling



Uniformly sample the coordinates as nested subsets

$[n] = S_0 \supseteq S_1 \supseteq S_2 \supseteq \cdots \supseteq S_{\log_2 n}$

Include each item from S_{i-1} in S_i independently with probability 1/2

x_{S_i} is x restricted to coordinates in S_i

Algorithm for Finding a Non-Zero Item

- *If x has k non-zero entries, what's the expected number of non-zero entries in x_{S_i}?*
  - For each non-zero entry j, let $Z_j = 1$ if $j \in S_i$, and $Z_j = 0$ otherwise
  - $Z = \sum_j Z_j$, and so $E[Z] = k \cdot E[Z_j] = \frac{k}{2^i}$
- *What's the variance of Z?*
  - $\mathrm{Var}[Z] = \sum_j \mathrm{Var}[Z_j] = k \cdot \mathrm{Var}[Z_1] = k\left(\frac{1}{2^i}\right)\left(1 - \frac{1}{2^i}\right) \le \frac{k}{2^i}$
- If $i = \lfloor \log_2 k \rfloor - 5$, then $32 \le E[Z] < 64$ and $\mathrm{Var}[Z] < 64$
- By Chebyshev, $\Pr[|Z - E[Z]| \ge 32] \le \frac{\mathrm{Var}[Z]}{32^2} \le \frac{1}{16}$
- If we run a k'-sparse algorithm with k' = 96 on x_{S_i}, we recover a non-zero item of x_{S_i} with probability at least 1-1/16 – 1/10 > 4/5, or output FAIL
- *But we don't know i?*

Algorithm for Finding a Non-Zero Item

- Run a k'=96-sparse vector algorithm on every x_{S_i} !
- For each x_{S_i}, our algorithm either returns a non-zero item of x_{S_i}, and hence of x, or outputs FAIL
- For $i = \lfloor \log_2 k \rfloor - 5$, with probability at least 4/5, we output a non-zero item of x_{S_i}, and hence of x
- Space is $(\log_2 n) \cdot O(k' \log n) = O(\log^3 n)$ bits!

Outline

- Sketching Model
  - Estimating the Euclidean norm of a vector
  - Finding a non-zero coordinate of a vector
- Graph sketching motivation
  - Boruvka's spanning tree algorithm
  - Finding a spanning tree from a sketch

Sketching Graphs

Are there sketches for graphs? Here A_G is the n x n adjacency matrix

- $(A_G)_{i,j} = 1$ if {i,j} is an edge, and $(A_G)_{i,j} = 0$ otherwise

$$\begin{pmatrix} & S & \end{pmatrix} \begin{pmatrix} & & \\ & A_G & \\ & & \end{pmatrix} = \begin{pmatrix} & SA_G & \end{pmatrix} \longrightarrow \text{answer}$$

- Is there a distribution on random matrices S with poly(log n) rows so that you can output a spanning tree of G given SA_G, with high probability?

Application: Graph Streams

- Want to process a graph stream, where you see the edges of a graph $e_1, \ldots, e_m$ one at a time, in an arbitrary order. Assume the vertices are labeled 1, 2, …, n.

e_3 | e_{10} | e_1 | e_2 | e_5 | e_7 | e_6 | …

- Can only make 1 pass over the stream
- Trivially store stream using $O(n^2)$ bits of memory.
- Want to instead use $n \cdot \mathrm{poly}(\log n)$ bits of memory
- How would you compute a spanning forest?

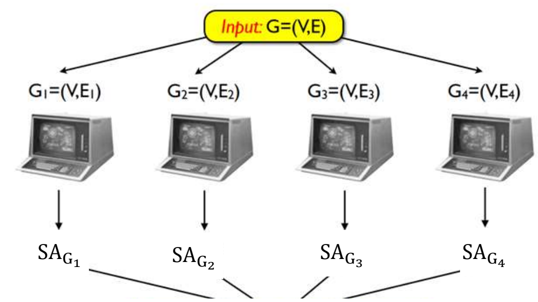
Computing a Spanning Forest

- For each edge e in the stream
  - If ___________________, store edge e
- ___________________ is “doesn’t form a cycle”
- Store at most n-1 edges, so O(n log n) bits of memory
- *But what if you are allowed to delete edges? This is called a dynamic stream*

Handling Deletions with Sketching

- Given $S \cdot A_G$, replace it with $S \cdot A_G - S \cdot A_e = S \cdot A_{G-e}$
- Memory required to store $S \cdot A_G$ is (# of rows of S)$\cdot$ $n \cdot \log n$ bits
  - Also need to store S, which is (# of rows of S)$\cdot$ $n \cdot \log n$
- Goal: find a distribution on matrices S with a small # of rows so that given $S \cdot A_G$, can output a spanning tree of G with high probability
- Theorem: there is a distribution on S with only poly(log n) rows!

Parallel Computing



$$SA_G = SA_{G_1} + SA_{G_2} + SA_{G_3} + SA_{G_4}$$

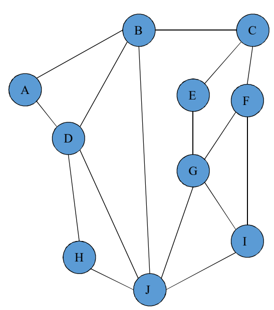
Outline

- Sketching Model
  - Estimating the Euclidean norm of a vector
  - Finding a non-zero coordinate of a vector
- Graph sketching motivation
  - Boruvka's spanning tree algorithm
  - Finding a spanning tree from a sketch

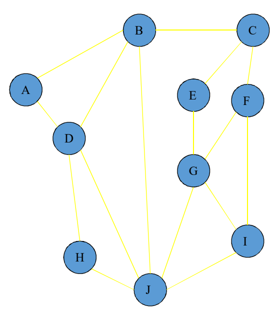
Boruvka's Spanning Tree Algorithm (Modified)

- For simplicity, assume the input graph is connected
- Initialize edgeset E' to Ø
- Create a list L of n groups of vertices, each initialized to a single vertex
- While the list has more than one group
  - For each group G, put an edge e from a vertex in G to a vertex not in G into E'
  - Merge any groups connected by an edge in the previous step
- Find a spanning tree among the edges in E'

Input Graph

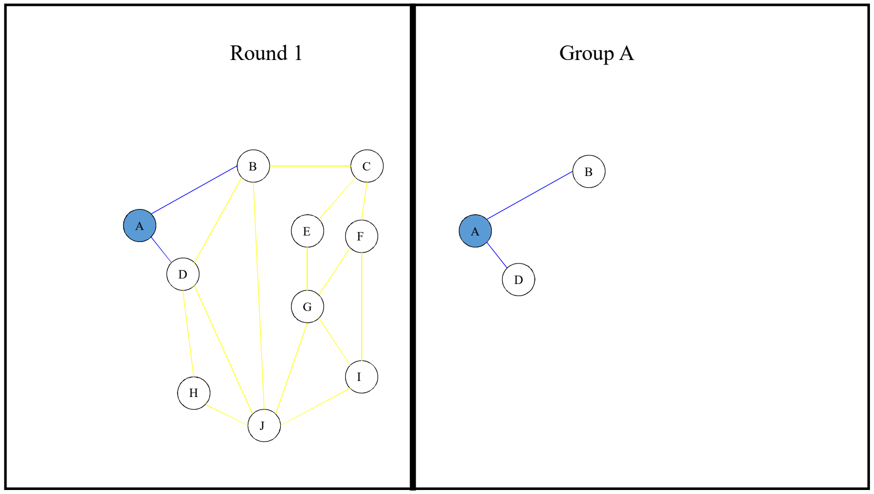


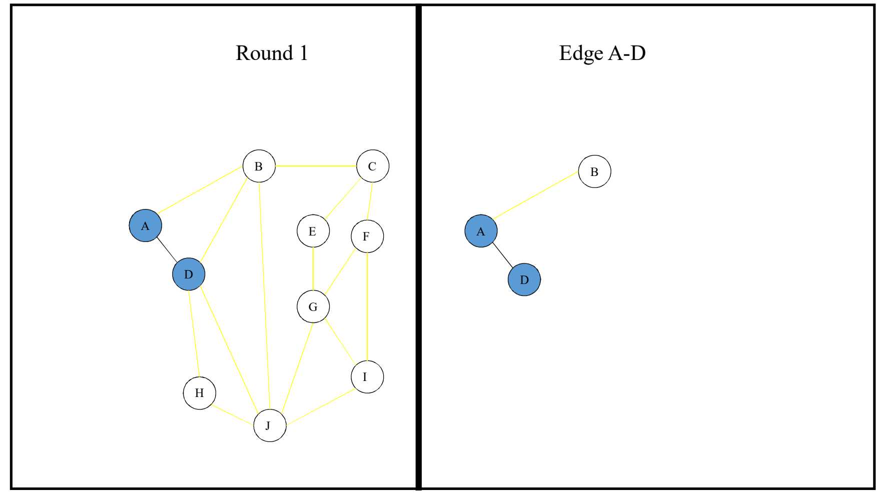
Groups at Beginning of Round 1

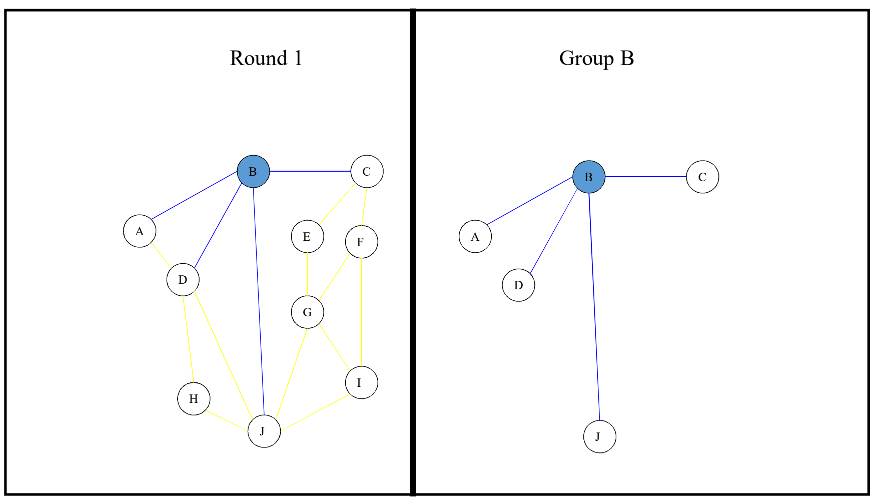


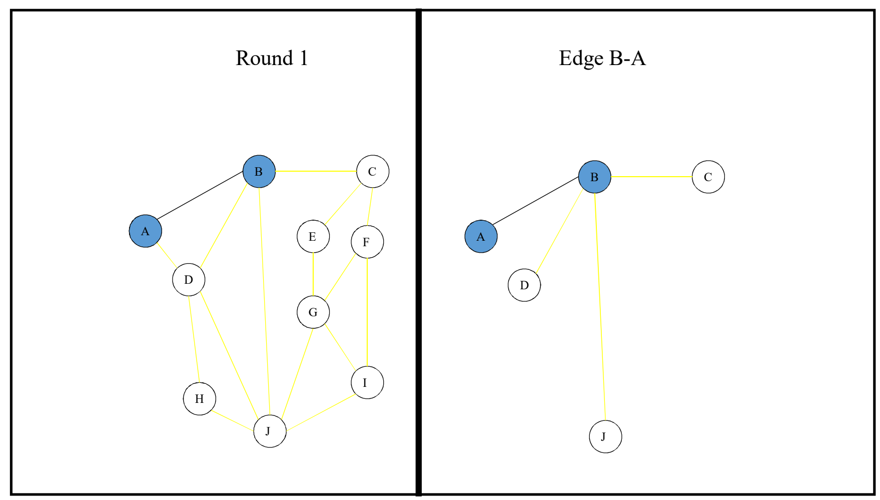
List of Groups

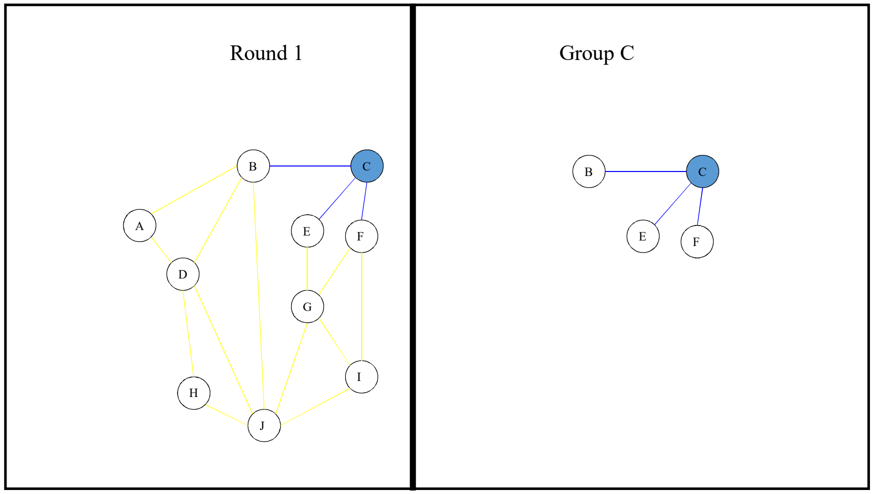
- A
- B
- C
- D
- E
- F
- G
- H
- I
- J

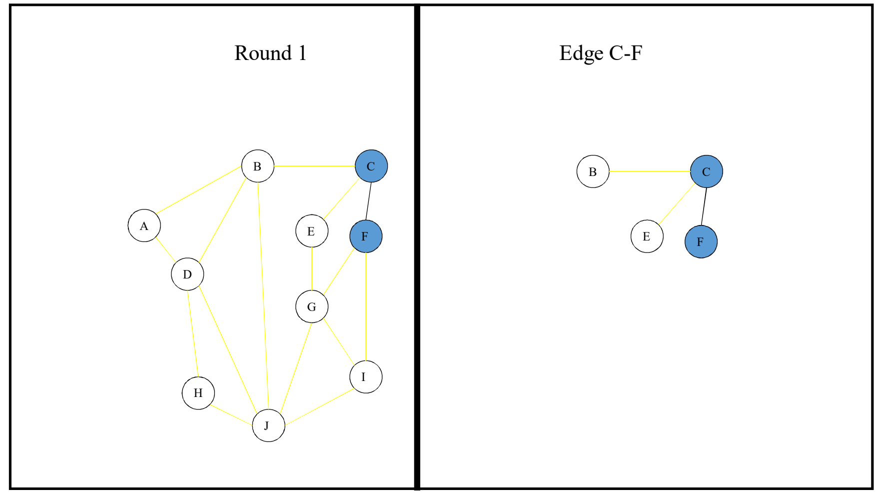

Round 1
Group A
A B C D E F G H I J
A B D

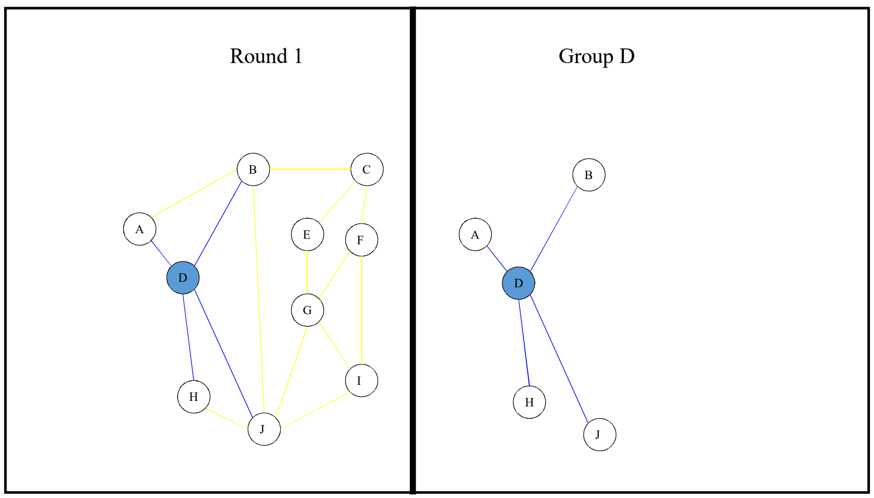

Round 1
Edge A-D
A B C D E F G H I J
A B D

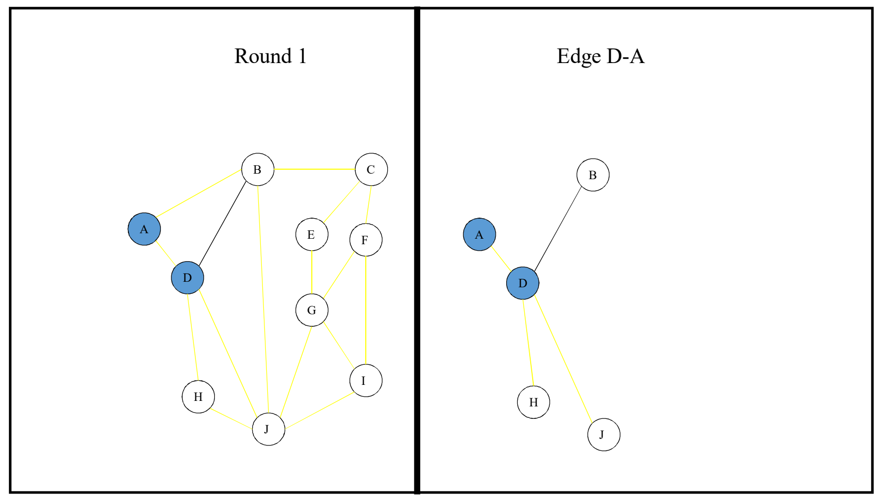

Round 1
Group B
A B C D E F G H I J
A B C D J

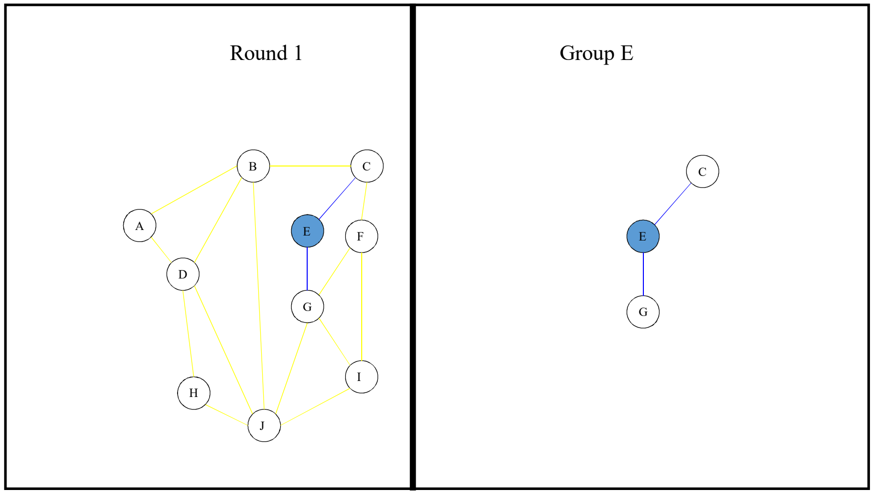

Round 1
Edge B-A
A B C D E F G H I J
A B C D J

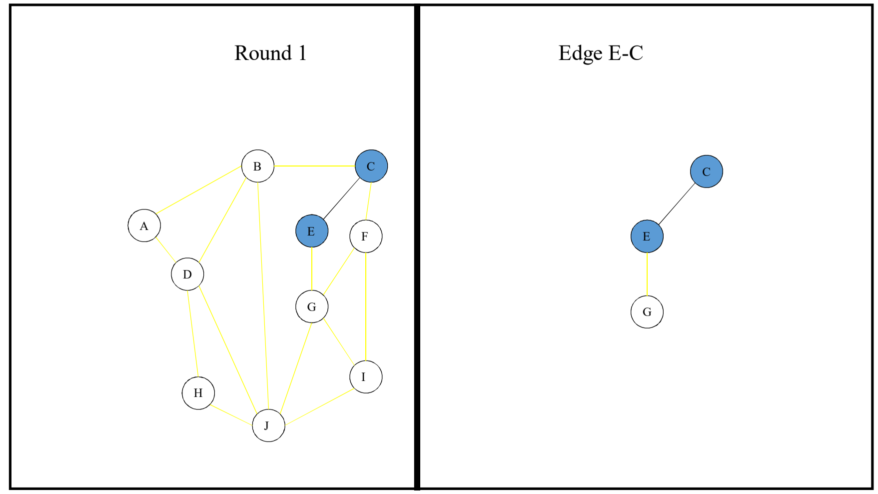

Round 1
A
B
C
D
E
F
G
H
I
J
Group C
B
C
E
F

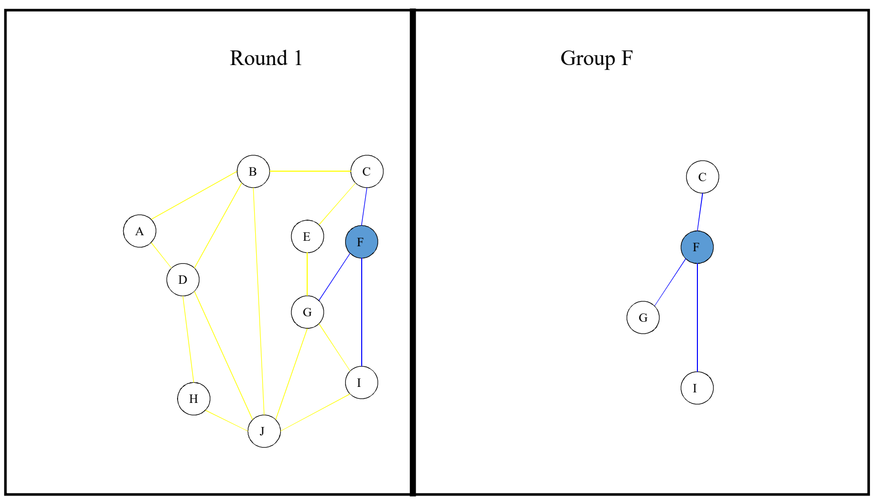

Round 1
A
B
C
D
E
F
G
H
I
J
Edge C-F
B
C
E
F

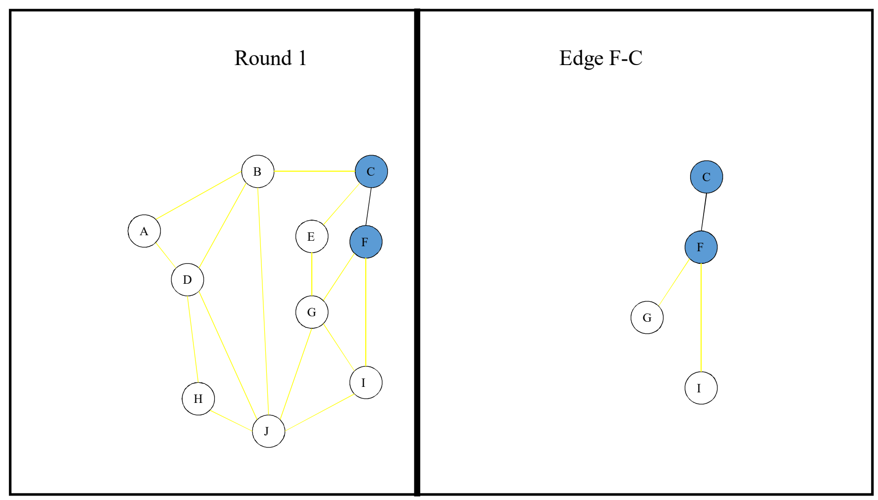

Round 1
A
B
C
D
E
F
G
H
I
J
Group D
A
B
D
H
J

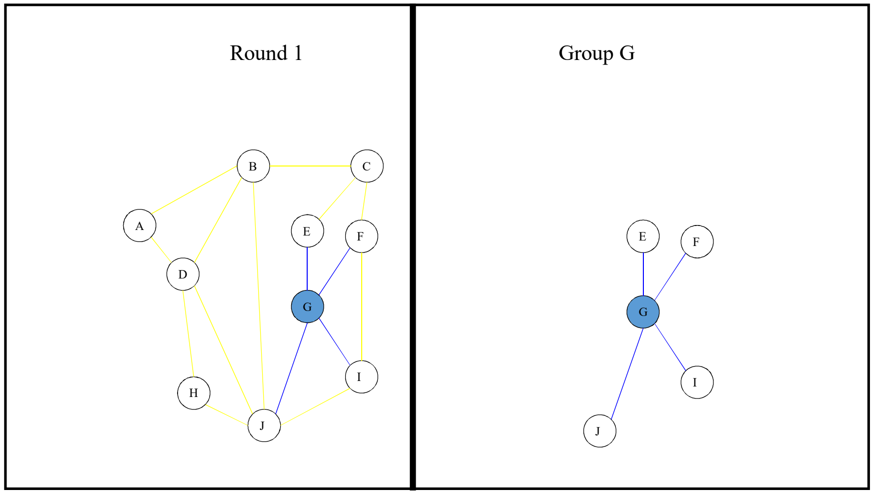

Round 1
A
B
C
D
E
F
G
H
I
J
Edge D-A
A
B
D
H
J

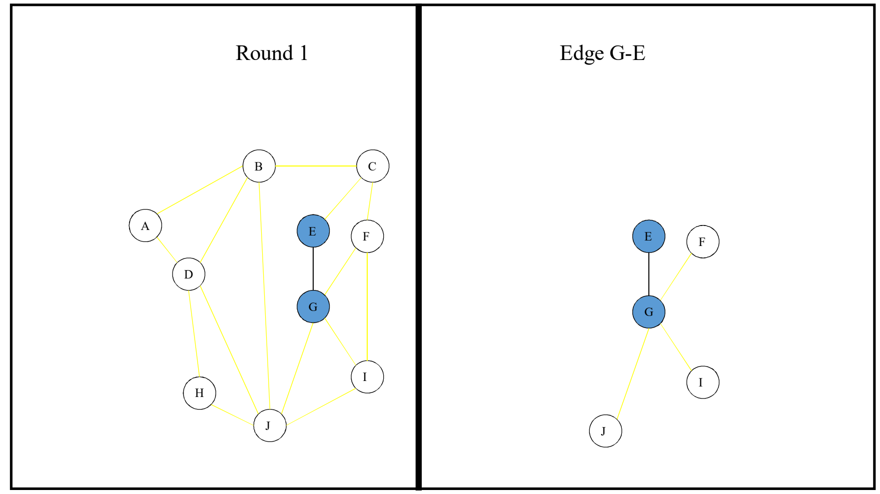

Round 1
Group E
A
B
C
D
E
F
G
H
I
J
C
E
G

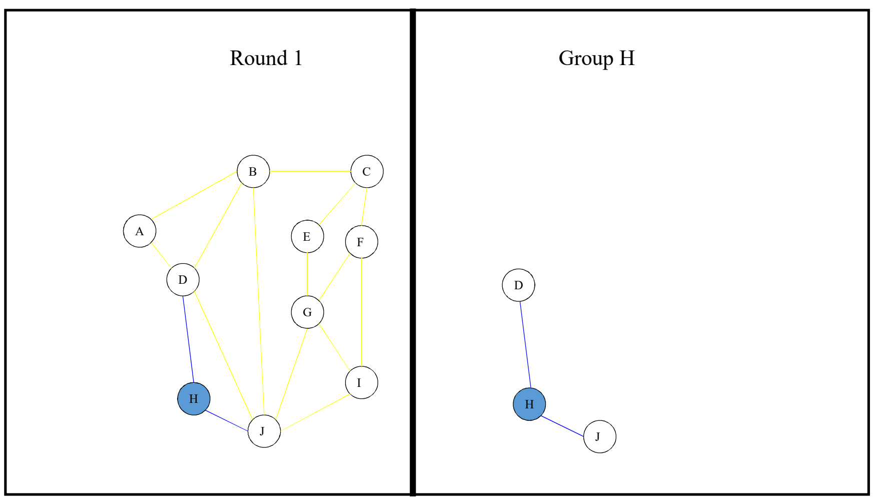

Round 1
Edge E-C
A
B
C
D
E
F
G
H
I
J
C
E
G

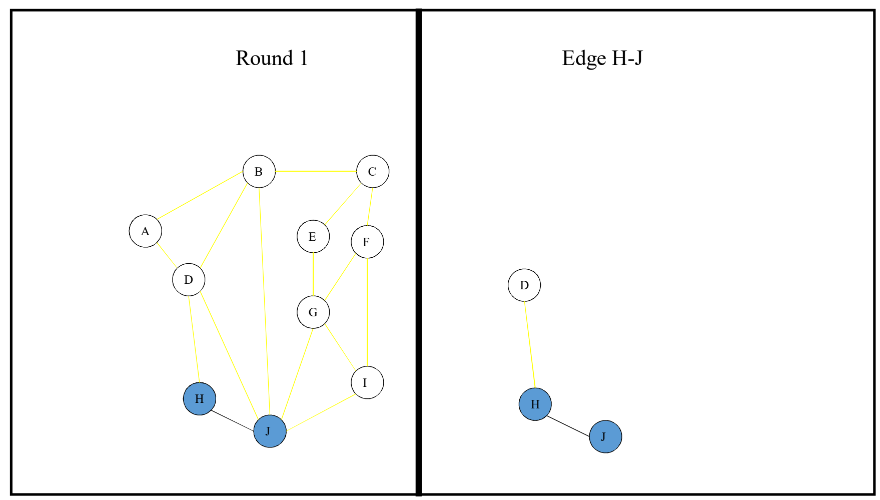

Round 1
Group F
A
B
C
D
E
F
G
H
I
J
C
F
G
I


Round 1
Edge F-C
A
B
C
D
E
F
G
H
I
J
C
F
G
I

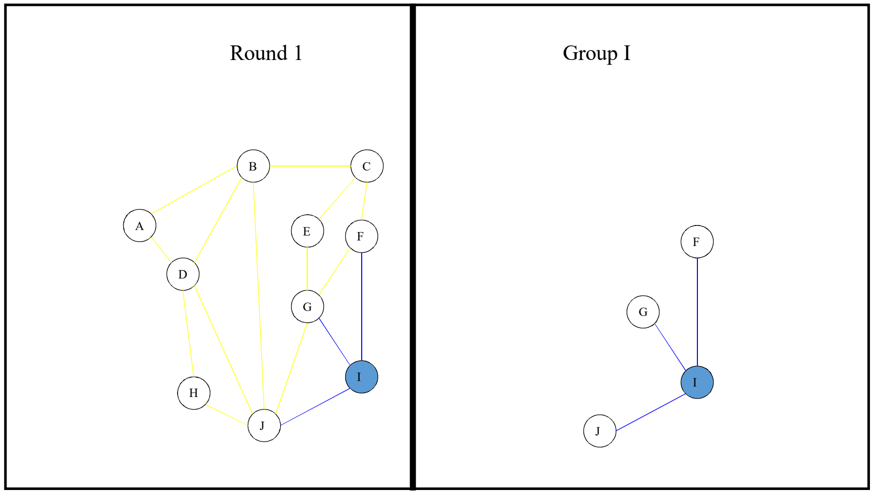

Round 1
B
C
A
E
F
D
G
I
H
J
Group G
E
F
G
I
J

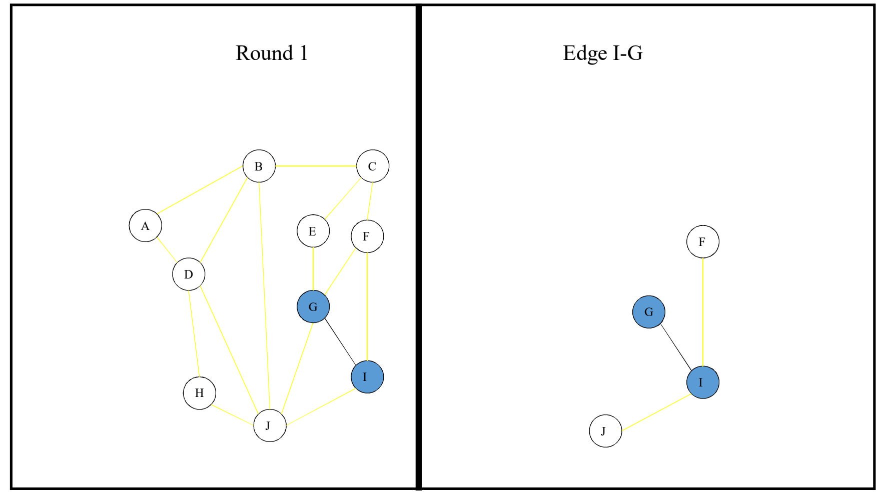

Round 1
B
C
A
E
F
D
G
I
H
J
Edge G-E
E
F
G
I
J

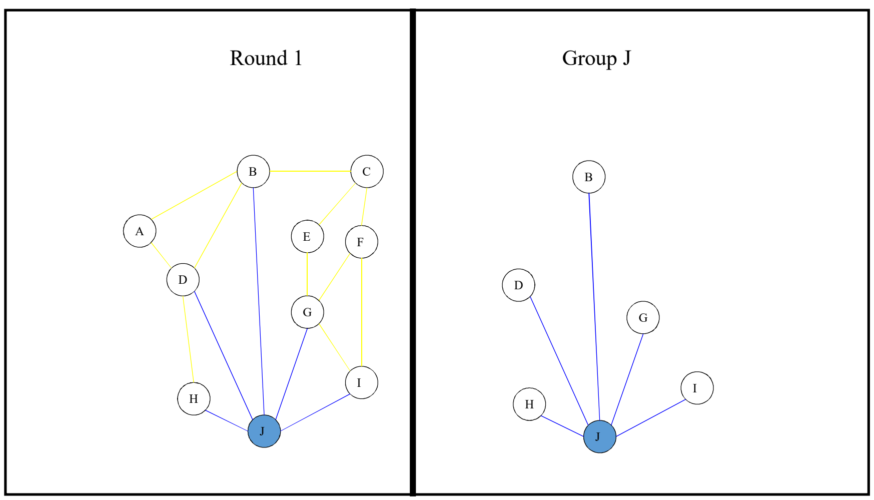

Round 1
B
C
A
E
F
D
G
I
H
J
Group H
D
H
J

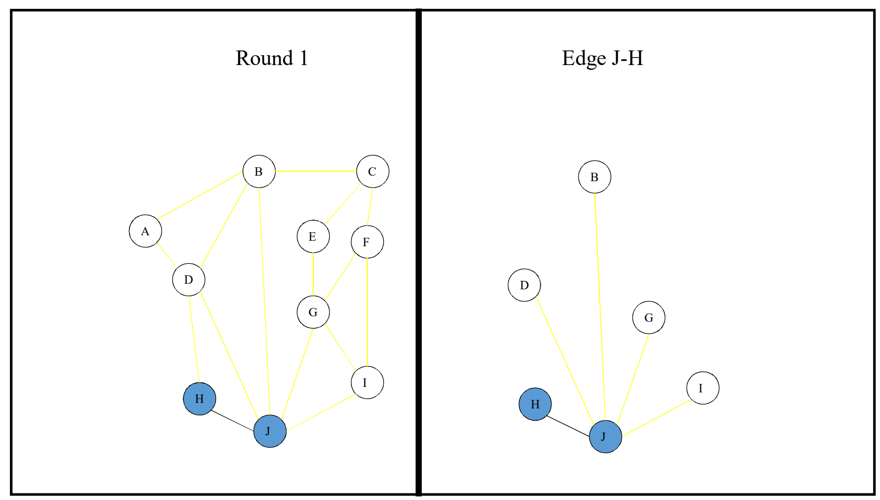

Round 1
B
C
A
E
F
D
G
I
H
J
Edge H-J
D
H
J

7/14/2019

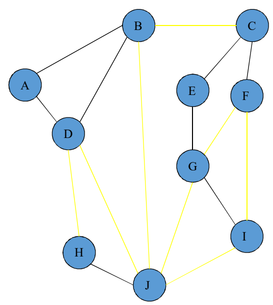

Round 1
Group I
A
B
C
D
E
F
G
H
I
J
F
G
I
J


Round 1
Edge I-G
A
B
C
D
E
F
G
H
I
J
F
G
I
J


Round 1
Group J
A
B
C
D
E
F
G
H
I
J
B
D
G
H
I
J


Round 1
Edge J-H
A
B
C
D
E
F
G
H
I
J
B
D
G
H
I
J

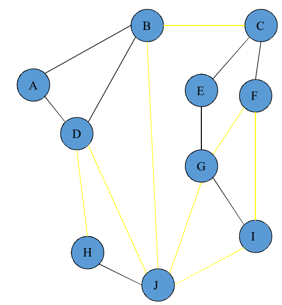
Round 1 Ends



List of Edges Added

- A-D
- B-A
- C-F
- D-B
- E-C
- F-C
- G-E
- H-J
- I-G
- J-H

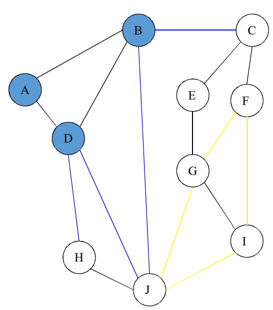
Groups at Beginning of Round 2



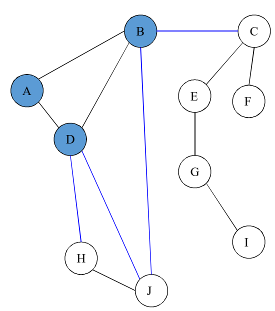
List of Groups

- D-A-B
- F-C-E-G-I
- H-J

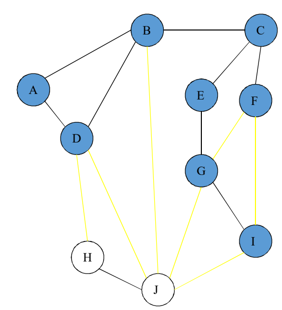
Round 2



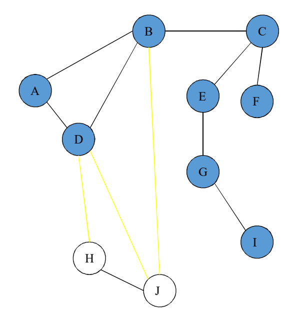
Group D-A-B

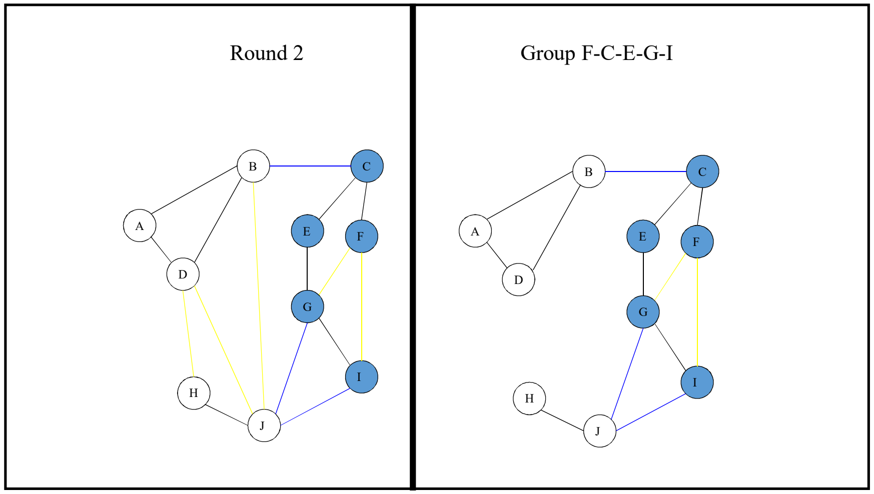


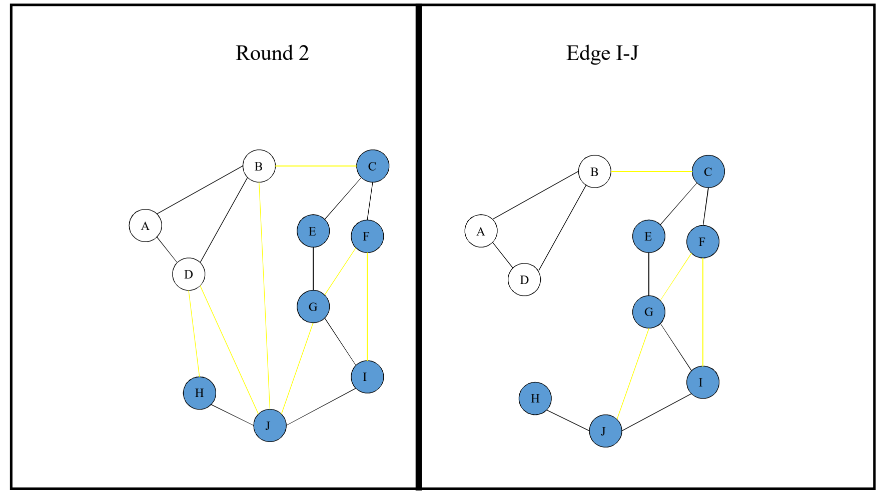
Round 2

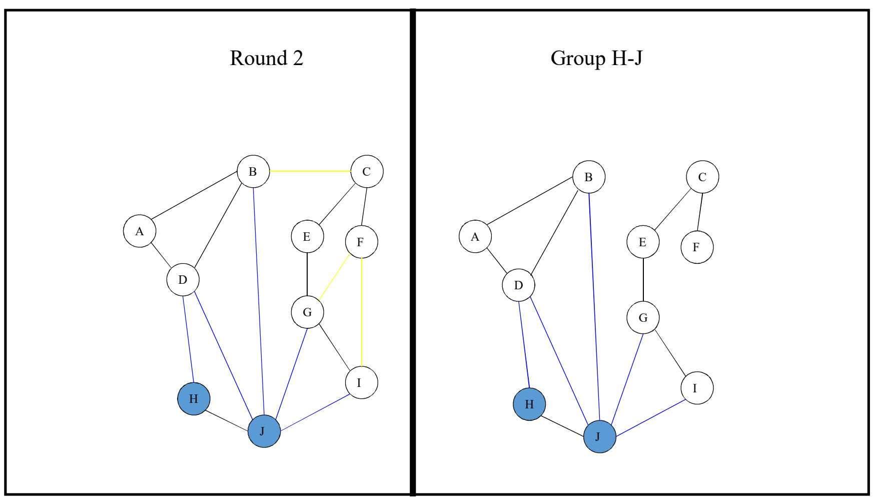


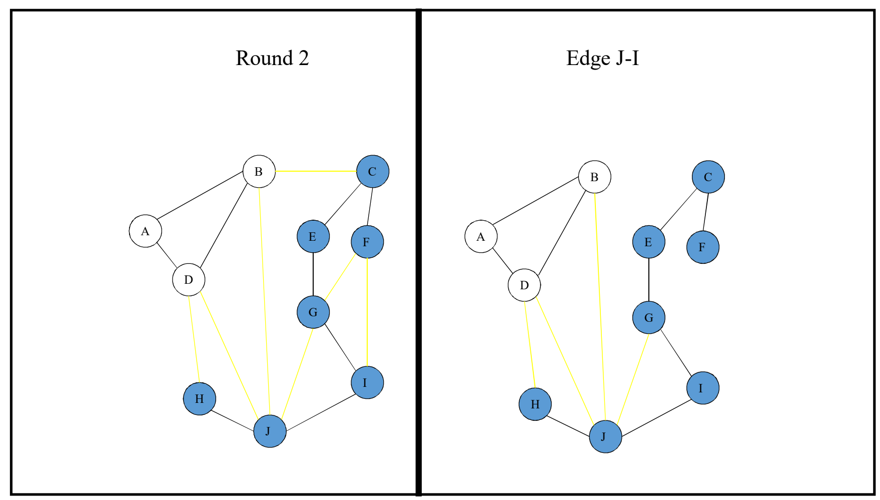
Edge B-C



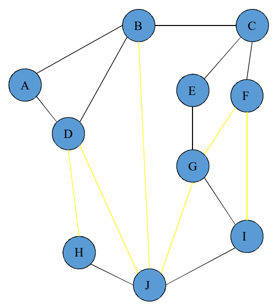

Round 2
Group F-C-E-G-I


Round 2
Edge I-J


Round 2
Group H-J


Round 2
Edge J-I

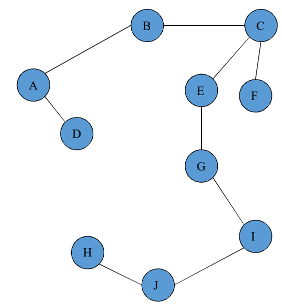
Round 2 Ends



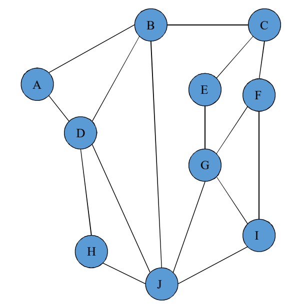
List of Edges Added

- B-C
- I-J
- J-I

Spanning Tree



Input Graph



Analysis

- If $G_1, G_2, \ldots, G_r$ are the groups of vertices in an iteration, for each G_i, there is a G_j, $i \neq j$, and an edge {u,v} from a vertex $u \in G_i$ to a vertex $v \in G_j$
  - Otherwise, graph is disconnected
- If t groups at start of an iteration, at most t/2 groups at end of iteration
  - Consider graph H with vertex set $G_1, G_2, \ldots, G_r$ and r edges, where the edges correspond to the groups we connect
  - Number of groups at most number of connected components in H. *Why?*
- After $\log_2 n$ iterations, one group left
  - At most n + n/2 + n/4 + … + 1 $\leq$ 2n edges chosen in E'
- E' contains a spanning tree
  - Invariant: the vertices in each group in each iteration are connected

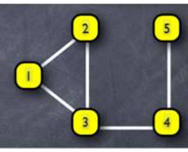
Outline

- Sketching Model
  - Estimating the Euclidean norm of a vector
  - Finding a non-zero coordinate of a vector
- Graph sketching motivation
  - Boruvka's spanning tree algorithm
  - Finding a spanning tree from a sketch

Representing a Graph

- For node i, let a_i be a vector indexed by node pairs
- If {i,j} is an edge, $a_i[i,j] = 1$ if j > i, and $a_i[i,j] = -1$ if j < i
- If {i,j} is not an edge, $a_i[i,j] = 0$

$$\begin{array}{rcccccccccc} & \{1,2\} & \{1,3\} & \{1,4\} & \{1,5\} & \{2,3\} & \{2,4\} & \{2,5\} & \{3,4\} & \{3,5\} & \{4,5\} \\ \mathbf{a}_1 = & (1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0) \\ \mathbf{a}_2 = & (-1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0) \end{array}$$



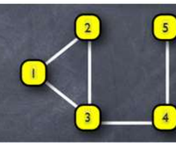
Representing a Graph

- Lemma: for any subset S of nodes,

$$\text{Support}(\textstyle\sum_{i \in S} a_i) = E(S, V \backslash S)$$

- Proof: for edge {i,j}, if i, j ∈ S, the sum of entries on {i,j}-th column is 0

$$\begin{array}{rcccccccccc} & \{1,2\} & \{1,3\} & \{1,4\} & \{1,5\} & \{2,3\} & \{2,4\} & \{2,5\} & \{3,4\} & \{3,5\} & \{4,5\} \\ \mathbf{a}_1 = & (1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0) \\ \mathbf{a}_2 = & (-1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0) \end{array}$$



Spanning Tree Algorithm

- Compute O(log n) sketches $C_1(a_j), \ldots, C_{O(\log n)}(a_j)$ for each a_j
- Each sketch $C_i(a_j)$ can output a non-zero item of a_j with probability at least 4/5, otherwise returns FAIL
- Idea: Run Boruvka's algorithm in sketch space
- For each node j, use $C_i(a_j)$ to get incident edge on each node j
- For i = 2, …, O(log n)
  - To get incident edge on group $G \subseteq V$, use

$$\sum_{j \in S} C_i \mathbf{a}_j = C_i \left(\sum_{j \in S} \mathbf{a}_j \right) \longrightarrow e \in \text{support}(\sum_{j \in S} \mathbf{a}_j) = E(S, V \setminus S)$$

Spanning Tree Wrapup

- O(n log n) total sketches $C_i(a_j)$, as i and j vary, so $O(n \log^4 n)$ space
- Note: a 1/5 fraction of sketches fail in each iteration in expectation, but Boruvka's algorithm guarantees that on the remaining 4/5 fraction of vertices, the number of connected components halves
- Expected number of iterations still O(log n)
- Since sketches are linear, they can be maintained with insertions and deletions of edges in a stream
- Overall $O(n \log^4 n)$ bits of space to output a spanning tree!
  - Can be improved to $O(n \log^3 n)$ bits