Distributed Approximation of k-edge-connected Subgraphs

Michal Dory, Technion

k-edge-connectivity

A graph G is **k-edge-connected**
= resistant to any $k - 1$ edge failures

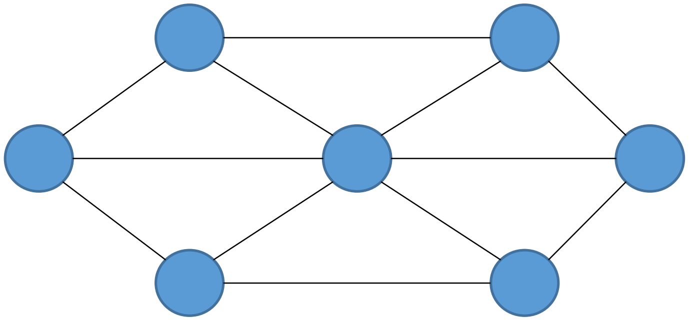
The problem:

Find the **minimum** k-edge-connected spanning subgraph (**k-ECSS**)

k-edge-connectivity

A graph G is **k-edge-connected**
= resistant to any $k - 1$ edge failures

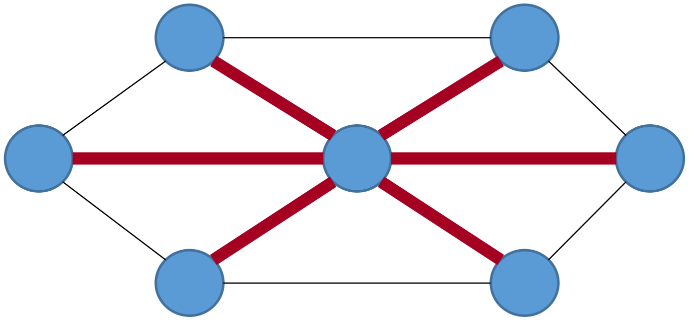
$k = 1$ ⇨ Find a minimum spanning tree (MST)



k-edge-connectivity

A graph G is **k-edge-connected**
= resistant to any $k-1$ edge failures

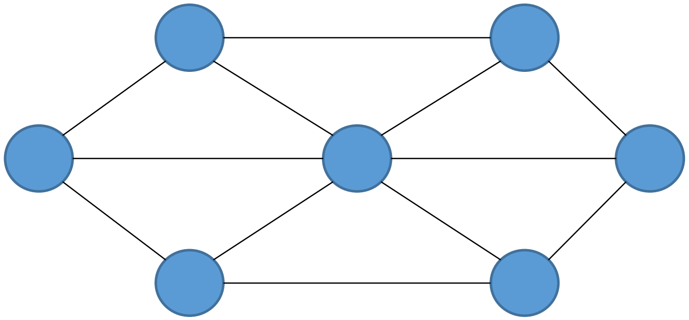
$k=1$ ➡ Find a minimum spanning tree (MST)



k-edge-connectivity

A graph G is **k-edge-connected**
= resistant to any $k-1$ edge failures

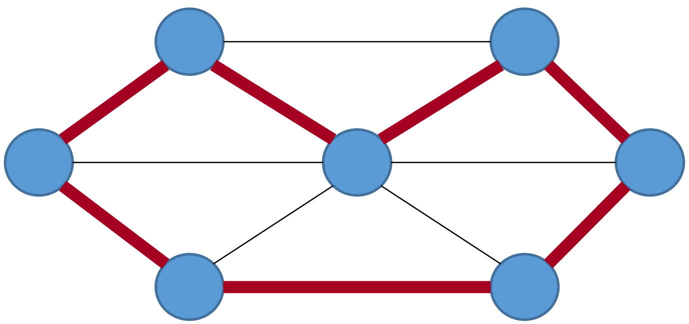
$k = 2$ ➡ Find a minimum 2-ECSS



k-edge-connectivity

A graph G is **k-edge-connected**
= resistant to any $k - 1$ edge failures

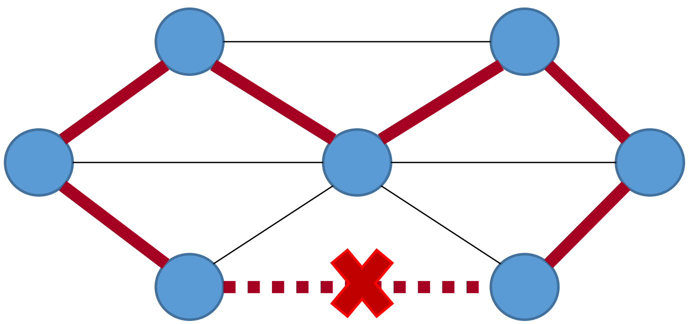
$k = 2$ ➡ Find a minimum 2-ECSS



k-edge-connectivity

A graph G is **k-edge-connected**
= resistant to any $k-1$ edge failures

$k=2$ ➡ Find a minimum 2-ECSS

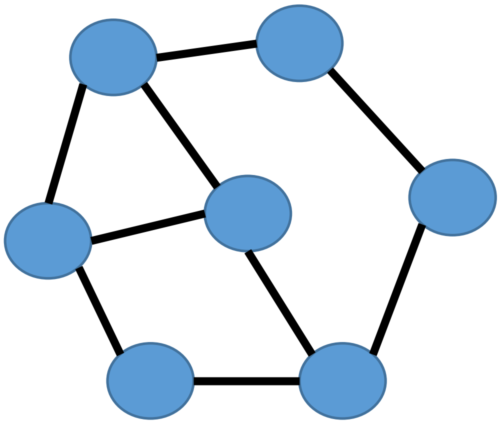


k-ECSS

- A central problem in network design.
- Well-studied in the sequential setting.
- The goal: solve in the **distributed** setting.

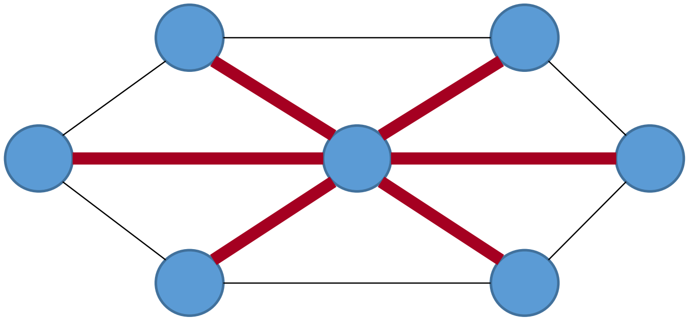
The **CONGEST** model

- ***n*** vertices
- $\Theta(\log n)$-bit messages
- **synchronous** rounds



Previous work

- The minimum spanning tree (MST) problem is well-studied in the CONGEST model.
- Takes $\tilde{O}(D + \sqrt{n})$ rounds (D = diameter) [Kutten and Peleg, 95]



Unweighted k-ECSS

	rounds	approximation	
k	$\tilde{O}(k(D+\sqrt{n}))$	2	Thurimella, 95
$k=2$	$O(D)$	2	Censor-Hillel and Dory, 17

Unweighted k-ECSS

	rounds	approximation	
k	$\tilde{O}(k(D+\sqrt{n}))$	2	Thurimella, 95
$k=2$	$O(D)$	2	Censor-Hillel and Dory, 17

This work

$k=3$	$O(D\log^3 n)$	$O(\log n)$	

Weighted k-ECSS

	rounds	approximation	
k	$O(knD)$	$O(\log k)$	Nutov and Sadeh, 2009
$k = 2$	$O(n)$	3	Censor-Hillel and Dory, 17

Weighted k-ECSS

	rounds	approximation	
k	$O(knD)$	$O(\log k)$	Nutov and Sadeh, 2009
$k=2$	$O(n)$	3	Censor-Hillel and Dory, 17

This work

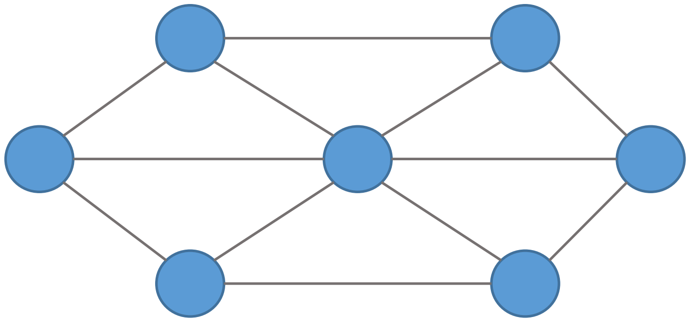
$k=2$	$\tilde{O}(D+\sqrt{n})$	$O(\log n)$	
k	$\tilde{O}(kn)$	$O(k \log n)$	

Our Results

	rounds	approximation
Weighted 2-ECSS	$\tilde{O}(D + \sqrt{n})$	$O(\log n)$
Weighted k-ECSS	$\tilde{O}(kn)$	$O(k \log n)$
Unweighted 3-ECSS	$O(D \log^3 n)$	$O(\log n)$

General structure

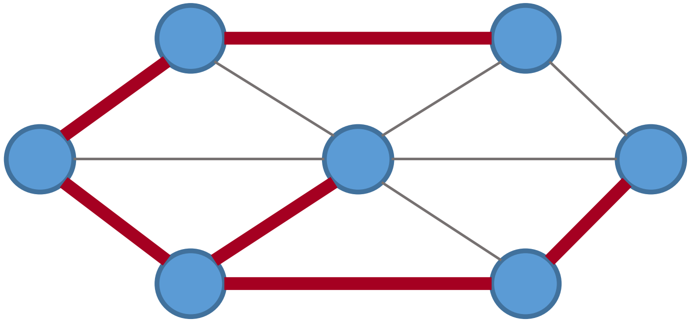
- We augment the connectivity gradually



$$k = 0$$

General structure

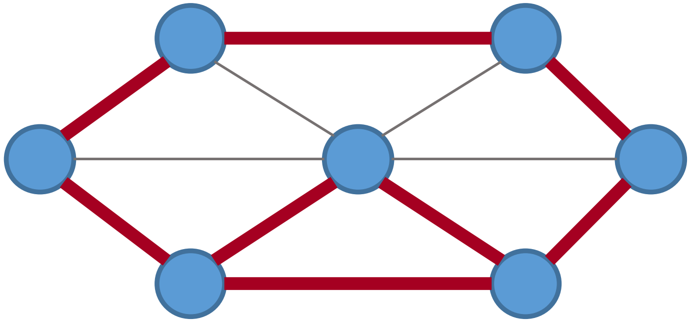
- We augment the connectivity gradually



$k = 1$

General structure

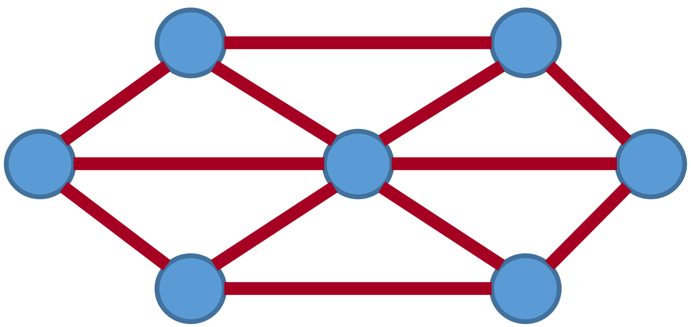
- We augment the connectivity gradually



$k = 2$

General structure

- We augment the connectivity gradually



$k = 3$

Aug_k

The input:

- k-edge-connected graph $G = (\mathrm{V}, \mathrm{E})$,
- $(k-1)$-ECSS H

The output:

Minimum weight set of edges $A \subseteq E$, such that $H \cup A$ is ***k*-edge-connected**.

Aug_k

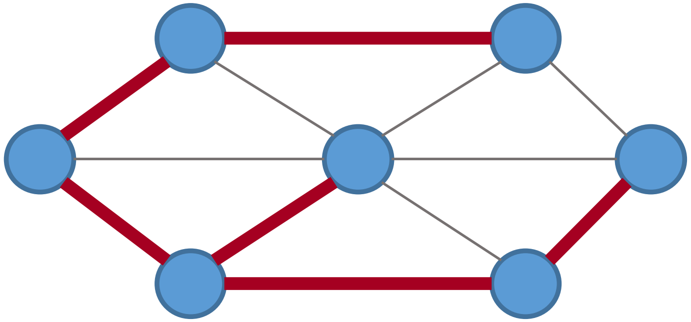
A_i = $\boldsymbol{\alpha_i}$-approximation algorithm for Aug_i
for $1 \leq i \leq k$

$\Downarrow$

$(\sum \boldsymbol{\alpha_i})$-approximation algorithm for k-ECSS

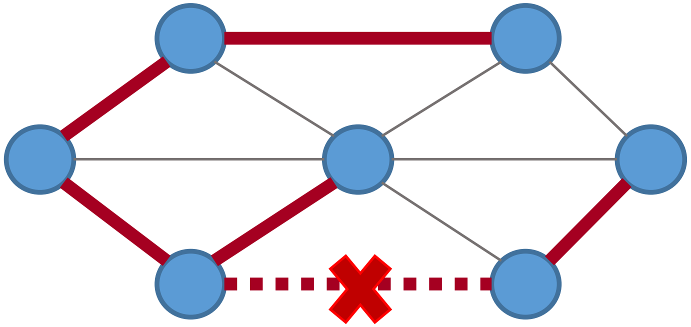
Solving Aug_k

- An edge e **covers** a cut C in H, if $(H \backslash \mathrm{C}) \cup \{e\}$ is connected



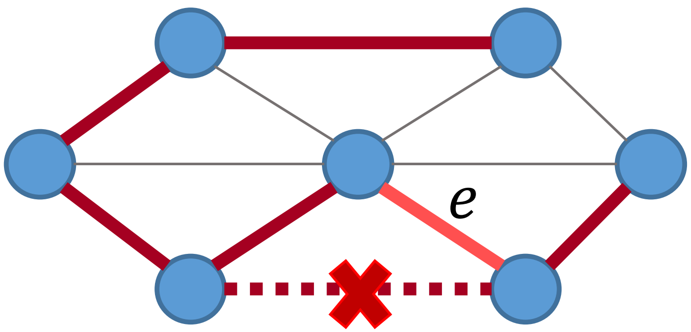
Solving Aug_k

- An edge e **covers** a cut C in H, if $(H \backslash \mathrm{C}) \cup \{e\}$ is connected



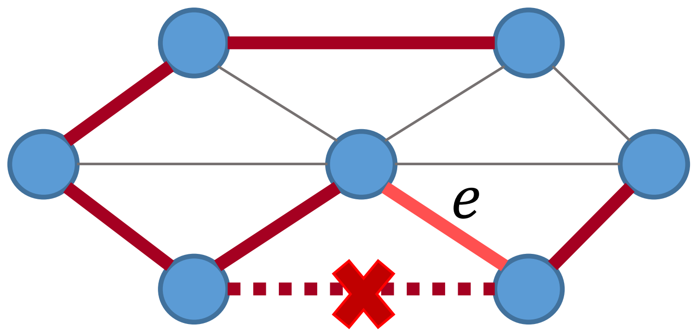
Solving Aug_k

- An edge e **covers** a cut C in H, if $(H \backslash \mathrm{C}) \cup \{e\}$ is connected



Solving Aug_k

- An edge e **covers** a cut C in H, if $(H \backslash C) \cup \{e\}$ is connected



- To solve Aug_k our goal is to cover all cuts of size $(k-1)$ in H

Cost-effectiveness

- For an edge e, let C_e be all the cuts of size $(k-1)$ in H **covered** by e
- The **cost-effectiveness** of e is

$$\rho(e) = \frac{|C_e|}{w(e)}$$

Sequential Greedy Algorithm

- At each step, add to the augmentation the edge with **maximum cost-effectiveness**.
- Continue until all the cuts of size $(k-1)$ are covered.

Gives an $O(\log n)$-approximation

Distributed Algorithm – take 1

- At each step, add to the augmentation the edge with **maximum cost-effectiveness**.
- Continue until all the cuts of size $(k - 1)$ are covered.

Distributed Algorithm – take 2

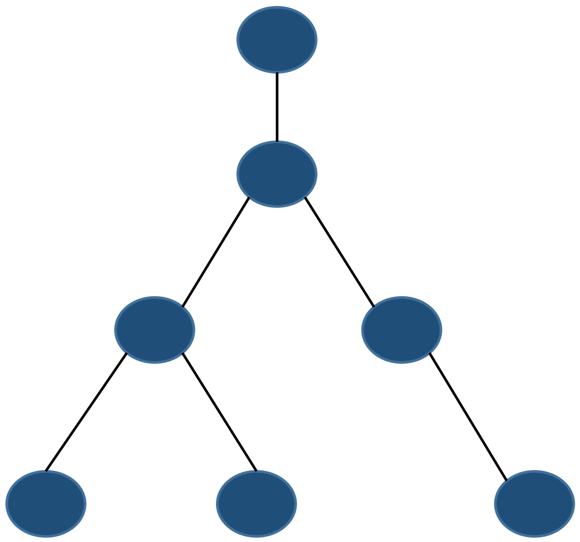
- At each step, add to the augmentation **all** the edges with **maximum cost-effectiveness**.
- Continue until all the cuts of size $(k-1)$ are covered.

Distributed Algorithm

- We would like to add edges simultaneously.
- How to **break the symmetry**?
- How to compute **cost-effectiveness**?

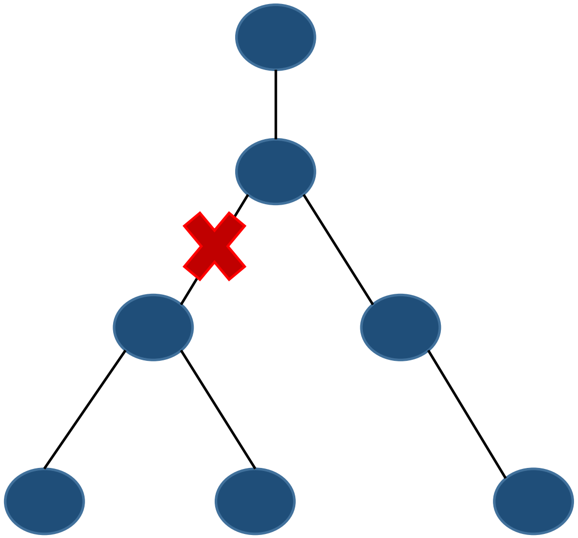
2-ECSS

- A cut = a tree edge



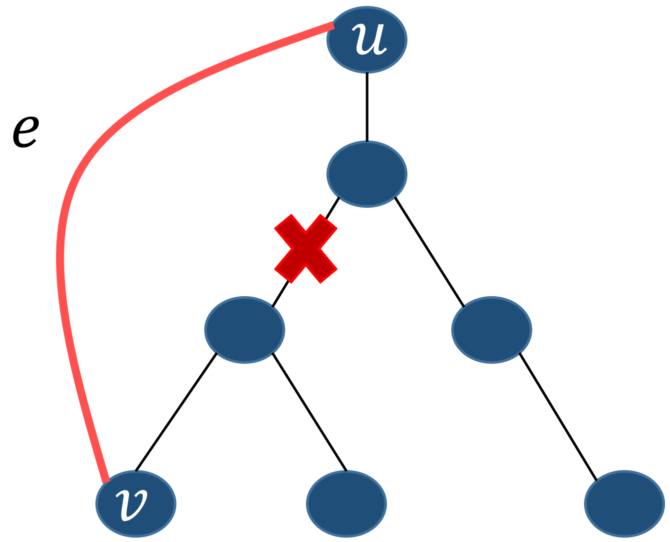
2-ECSS

- A cut = a tree edge



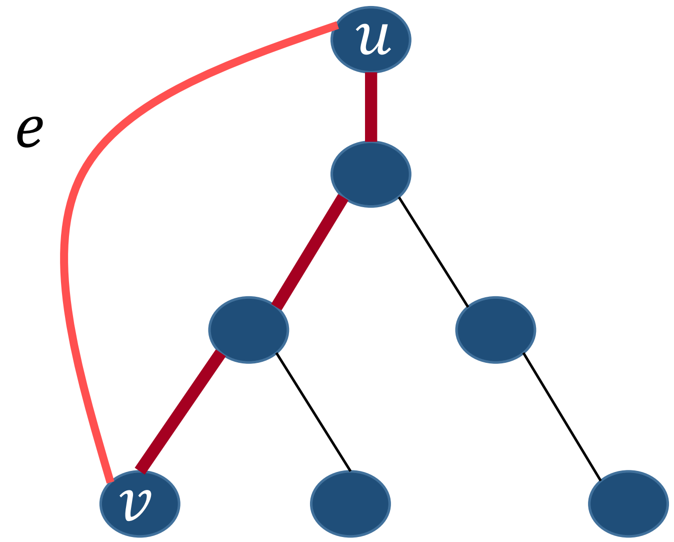
2-ECSS

- A cut = a tree edge



2-ECSS

- A cut = a tree edge



The Algorithm

- At each step, find all the edges with **maximum cost-effectiveness**, they are the ***candidates***.
- Each candidate edge chooses a random number $r \in [0,1]$.
- Each uncovered tree edge **votes** to the first candidate that covers it.
- An edge is added to the augmentation **if it gets at least $\mathbf{\frac{1}{8}}$ of the votes** of the tree edges it covers.
- We continue until all tree edges are covered.

The Algorithm

- At each step, find all the edges with **maximum cost-effectiveness**, they are the ***candidates***.
- Each candidate edge chooses a random number $r \in [0,1]$.
- Each uncovered tree edge **votes** to the first candidate that covers it.
- An edge is added to the augmentation **if it gets at least $\frac{1}{8}$ of the votes** of the tree edges it covers.
- We continue until all tree edges are covered.

The Algorithm

- At each step, find all the edges with **maximum cost-effectiveness**, they are the ***candidates***.
- Each candidate edge chooses a random number $r \in [0,1]$.
- Each uncovered tree edge **votes** to the first candidate that covers it.
- An edge is added to the augmentation **if it gets at least $\frac{1}{8}$ of the votes** of the tree edges it covers.
- We continue until all tree edges are covered.

The Algorithm

- At each step, find all the edges with **maximum cost-effectiveness**, they are the ***candidates***.
- Each candidate edge chooses a random number $r \in [0,1]$.
- Each uncovered tree edge **votes** to the first candidate that covers it.
- An edge is added to the augmentation **if it gets at least $\mathbf{\frac{1}{8}}$ of the votes** of the tree edges it covers.
- We continue until all tree edges are covered.

The Algorithm

- At each step, find all the edges with **maximum cost-effectiveness**, they are the ***candidates***.
- Each candidate edge chooses a random number $r \in [0,1]$.
- Each uncovered tree edge **votes** to the first candidate that covers it.
- An edge is added to the augmentation **if it gets at least $\frac{1}{8}$ of the votes** of the tree edges it covers.
- We continue until all tree edges are covered.

The Algorithm

- At each step, find all the edges with **maximum cost-effectiveness**, they are the ***candidates***.
- Each candidate edge chooses a random number $r \in [0,1]$.
- Each uncovered tree edge **votes** to the first candidate that covers it.
- An edge is added to the augmentation **if it gets at least $\frac{1}{8}$ of the votes** of the tree edges it covers.
- We continue until all tree edges are covered.

The Algorithm

- Gives an $O(\log n)$-approximation
- The number of iterations is $O(\log^2 n)$
- How to implement each iteration?

The Algorithm

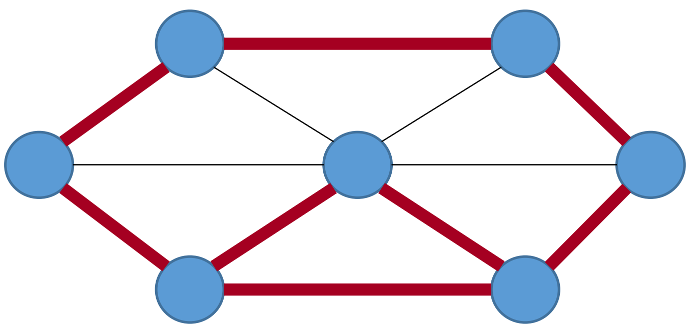
- At each step, find all the edges with **maximum cost-effectiveness**, they are the ***candidates***.
- Each candidate edge chooses a random number $r \in [0,1]$.
- Each uncovered tree edge **votes** to the first candidate that covers it.
- An edge is added to the augmentation **if it gets at least $\mathbf{\frac{1}{8}}$ of the votes** of the tree edges it covers.
- We continue until all tree edges are covered.

The Algorithm

- We need to do many **global computations** in parallel: computing cost-effectiveness, computing the number of votes…
- To achieve this, we **decompose the tree into fragments**, following a decomposition presented for solving the fault-tolerant MST problem [Ghaffari and Parter, 2016].

Conclusion

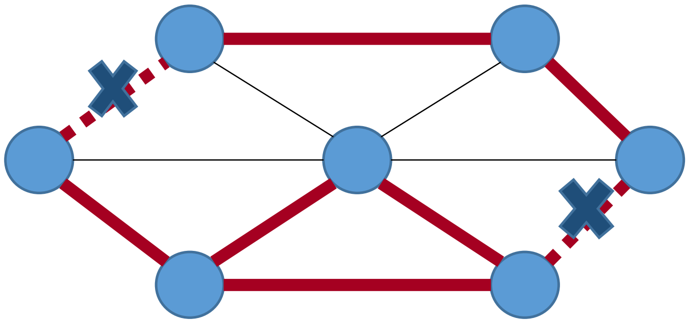
- $O\left((D+\sqrt{n})\log^2 n\right)$-round, $O(\log n)$-approximation for 2-ECSS



$k = 2$

What about minimum k-ECSS?

- Now the cost-effectiveness of an edge depends on the number of cuts it covers.

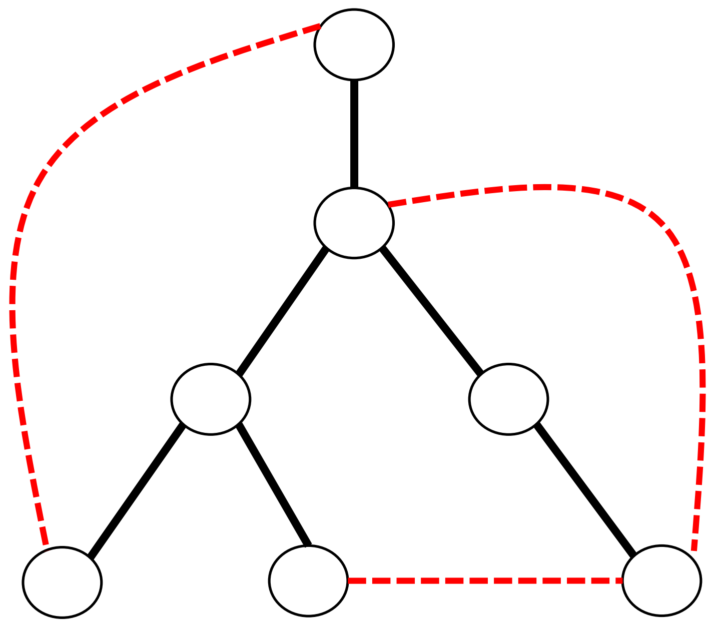


- How to compute cost-effectiveness?
- How to break the symmetry?

3-ECSS

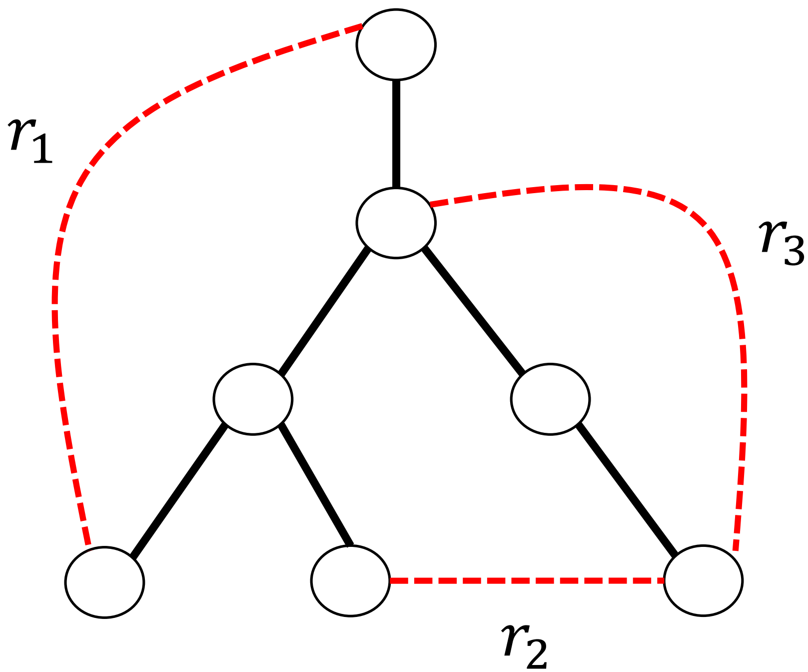
- To compute cost-effectiveness, we use the **cycle space sampling** technique [Pritchard and Thurimella, 2011]
- We give the edges of the graph labels that allow to detect cuts of size 2 efficiently.

3-ECSS



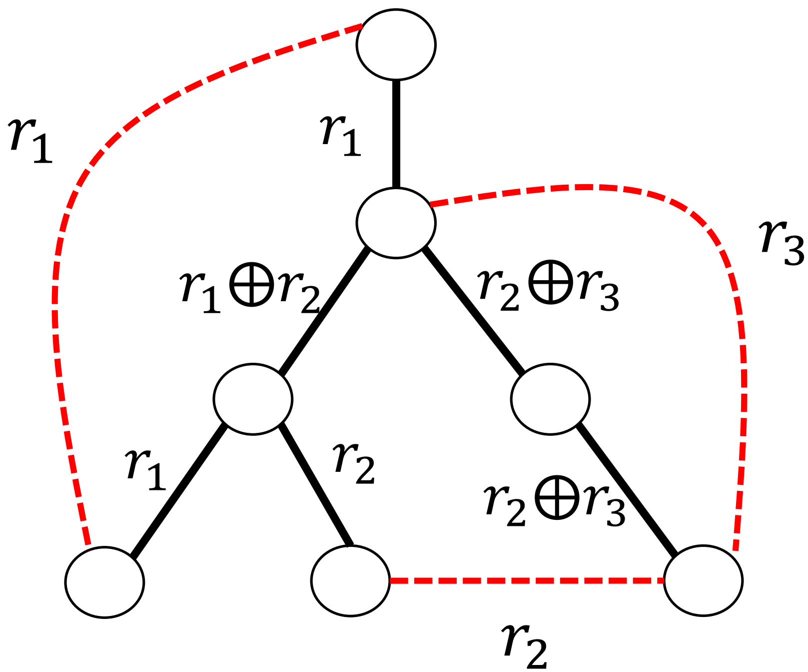
A 2-edge-connected subgraph H

3-ECSS



Each **non-tree edge** chooses a random label

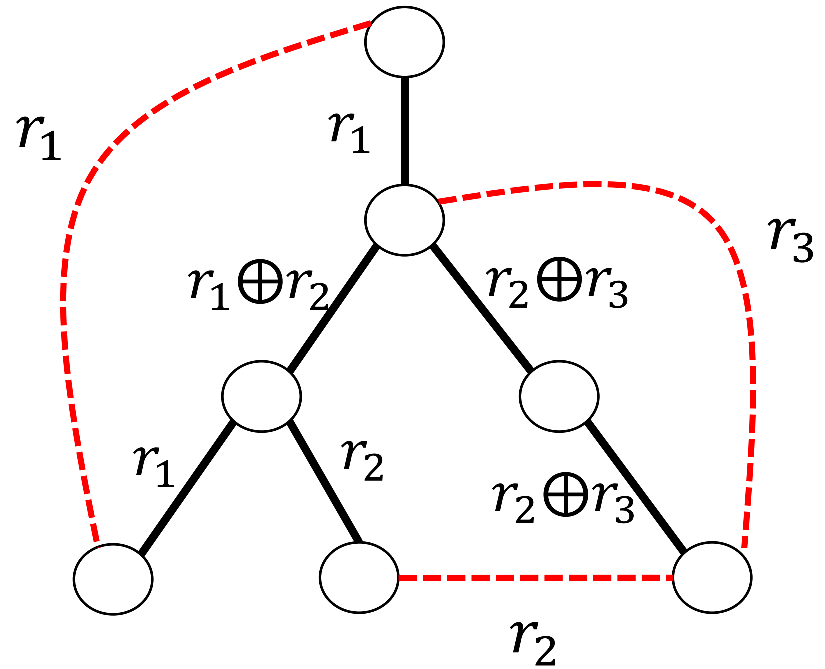
3-ECSS



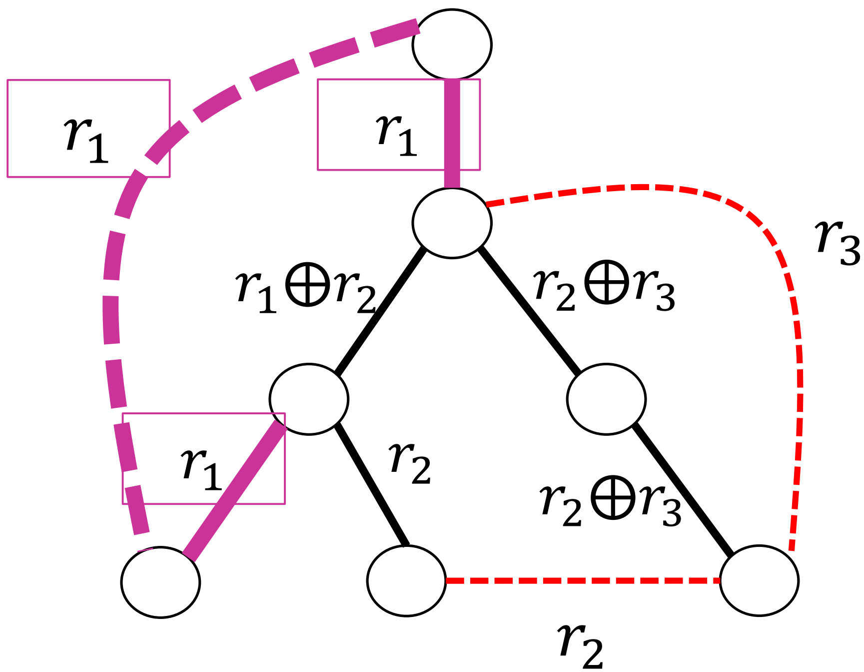
The label of a **tree edge** is the xor of non-tree edges that cover it

3-ECSS

Two edges define a cut
$\Leftrightarrow$
They have the same label

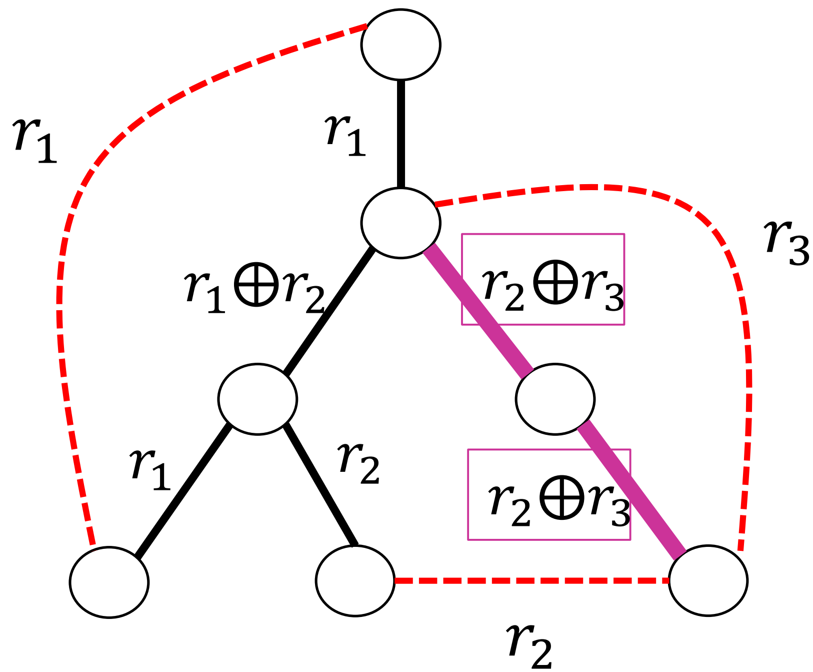


3-ECSS



Two edges define a cut ⇔ they have the same label

3-ECSS

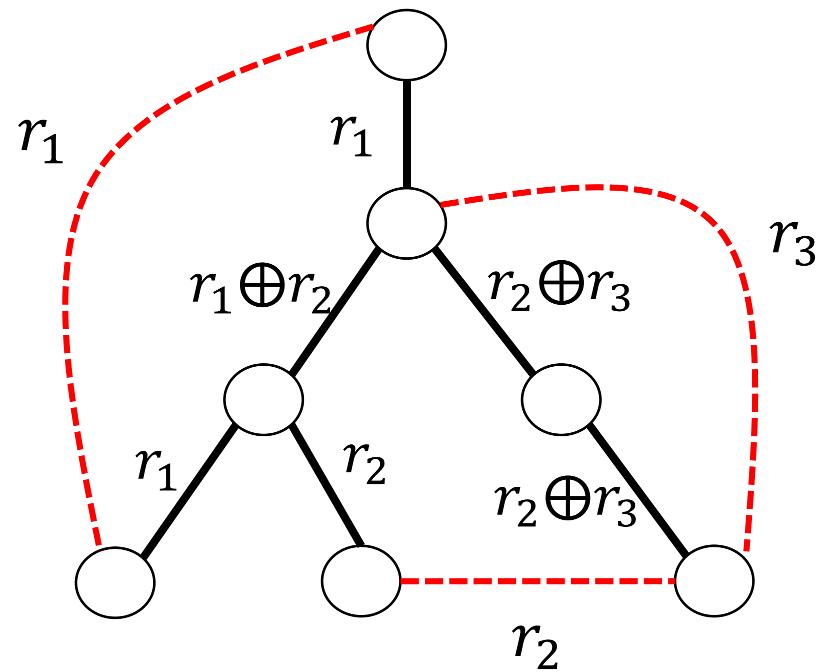


Two edges define a cut ⇔ they have the same label

Detecting cut pairs

Two edges define a cut in <u>two cases:</u>

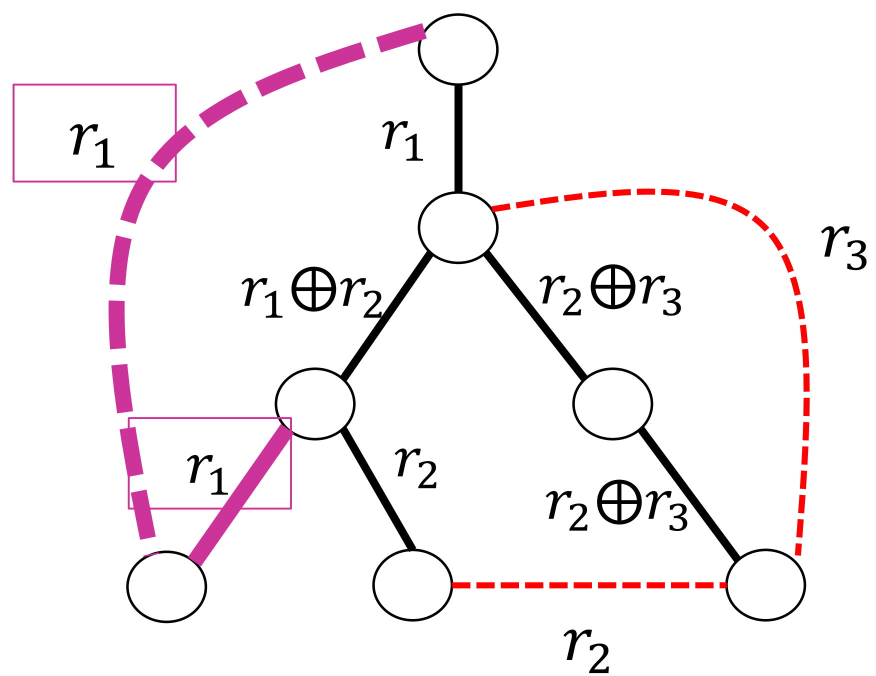
- A tree edge and a unique non-tree edge that covers it
- Two tree edges covered by the exact same non-tree edges



Detecting cut pairs

Two edges define a cut in <u>two cases:</u>

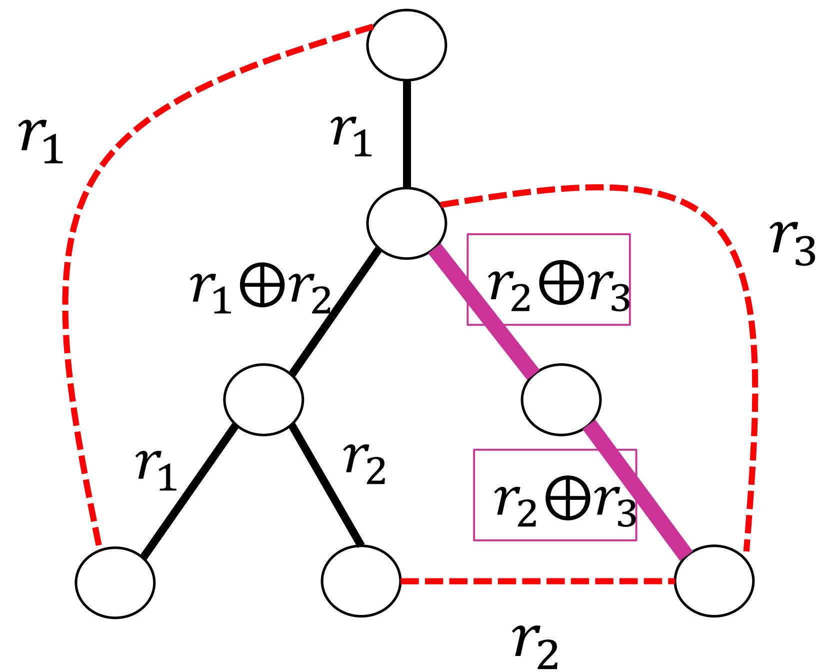
- A tree edge and a unique non-tree edge that covers it
- Two tree edges covered by the exact same non-tree edges



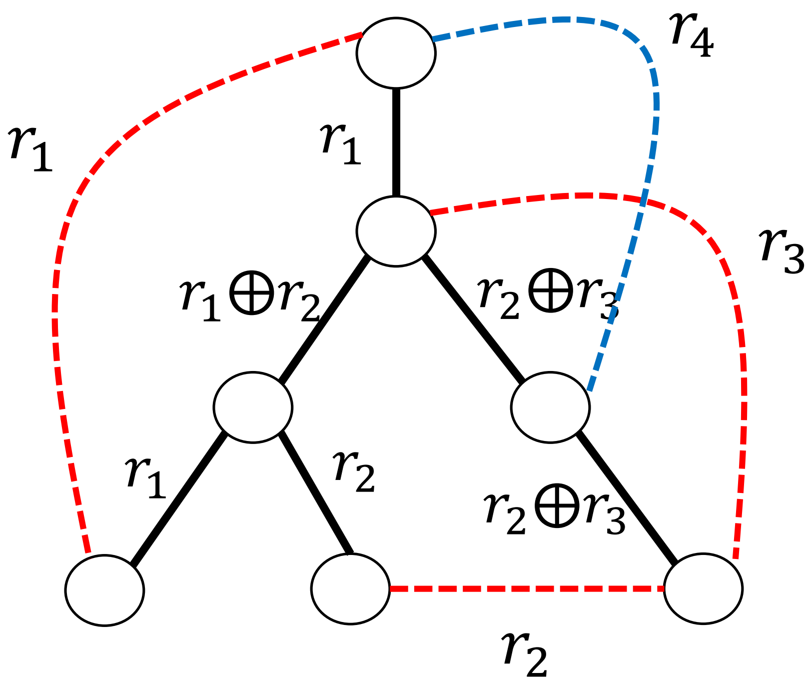
Detecting cut pairs

Two edges define a cut in <u>two cases:</u>

- A tree edge and a unique non-tree edge that covers it
- Two tree edges covered by the exact same non-tree edges

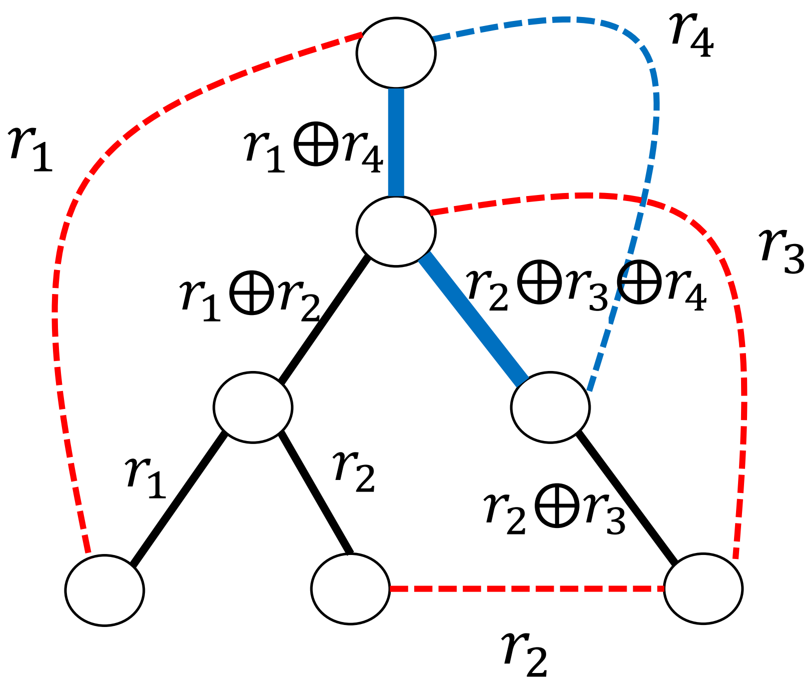


Computing cost-effectiveness



Assume we add a new edge e with the label r_4

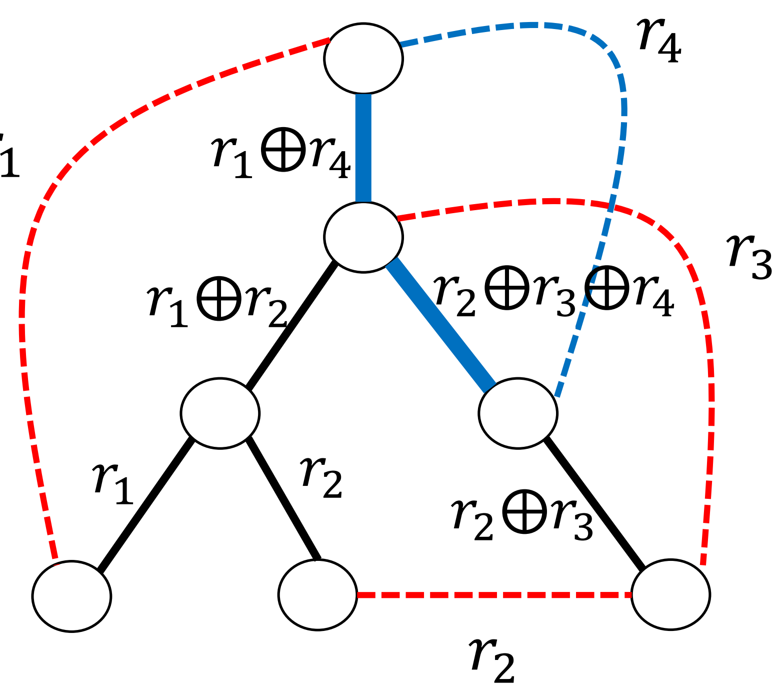
Computing cost-effectiveness



It changes the labels of tree edges it covers

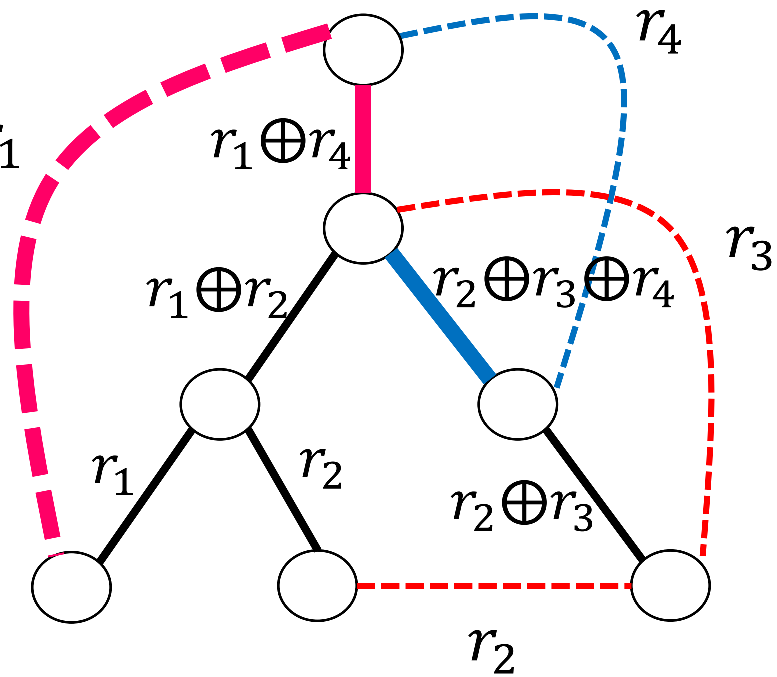
Computing cost-effectiveness

e covers all the cut pairs $\{f, f'\}$ where exactly one of f, f' is in the tree path covered by e



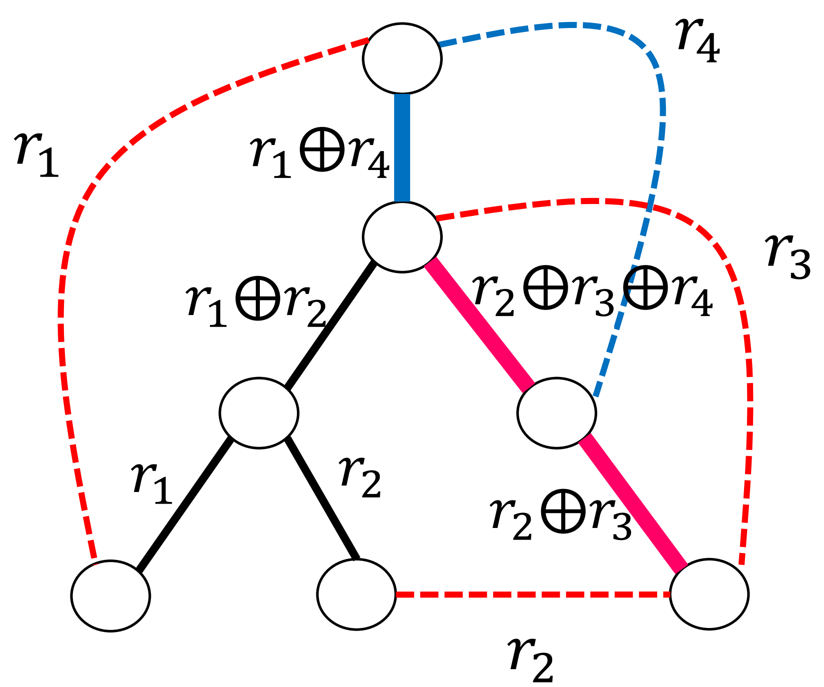
Computing cost-effectiveness

e covers all the cut pairs $\{f, f'\}$ where exactly one of f, f' is in the tree path covered by e



Computing cost-effectiveness

e covers all the cut pairs $\{f, f'\}$ where exactly one of f, f' is in the tree path covered by e



3-ECSS

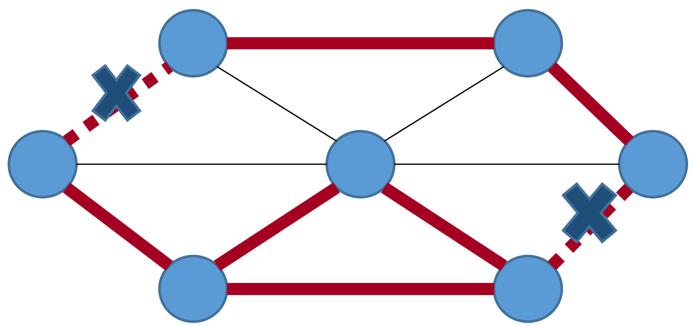
- We compute **cost-effectiveness** in $O(D)$ rounds
- We show a mechanism for **symmetry breaking** that takes $O(\log^3 n)$ iterations

Conclusion:

$O(D \log^3 n)$-round $O(\log n)$-approximation for unweighted 3-ECSS

What about minimum k-ECSS?

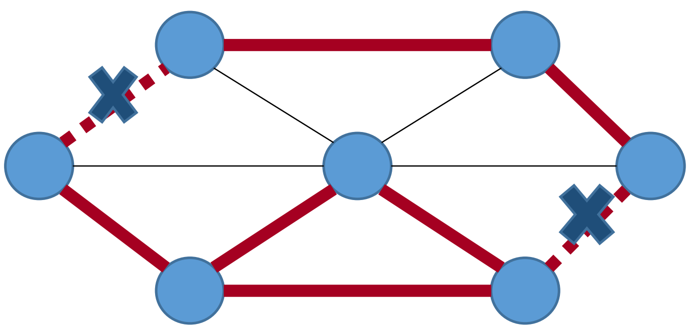
- Now the cost-effectiveness of an edge depends on the number of cuts it covers.



- How to compute cost-effectiveness?
- How to break the symmetry?

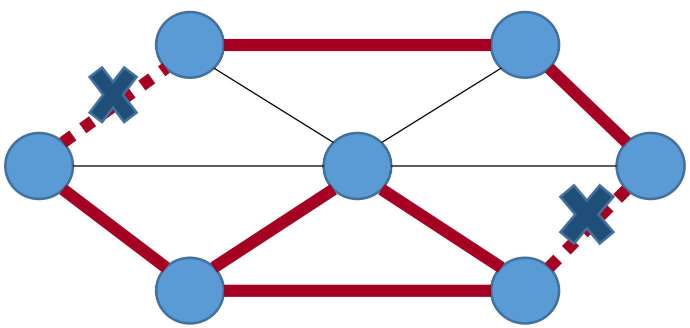
Computing cost-effectiveness

- Minimum k-edge-connected subgraphs are *sparse*.
- In $O(kn)$ rounds we can learn the whole subgraph H and compute cost-effectiveness.



Symmetry Breaking

- *Candidates* = edges with maximum cost-effectiveness
- We would like to add small number of candidates that cover many cuts.



Symmetry Breaking

$\deg(C)$ = number of candidates that cover the cut C

<u>Intuition:</u> if each of the candidates that covers C is added with probability $\frac{1}{\deg(C)}$ we add one candidate to cover C in expectation

Symmetry Breaking

Idea: Candidates are added to the augmentation with probability p

- Initially $p = \frac{1}{m}$
- Every $O(\log n)$ iterations p is increased by a factor of 2

Claim: when $p = \frac{1}{2^i}$ for all cuts C, $\deg(C) \leq 2^i$ w.h.p

The number of iterations is $O(\log^3 n)$.

Summary

	rounds	approximation
Weighted 2-ECSS	$\tilde{O}(D + \sqrt{n})$	$O(\log n)$
Weighted k-ECSS	$\tilde{O}(kn)$	$O(k \log n)$
Unweighted 3-ECSS	$O(D \log^3 n)$	$O(\log n)$

Open questions

Weighted $\boldsymbol{k}$-ECSS

There is a **sublinear** algorithm for $\boldsymbol{k=2}$, what about $k>2$?

Unweighted $\boldsymbol{k}$-ECSS

There is an $O(D\log^3 n)$-round algorithm for $\boldsymbol{k=3}$, what about $k>3$?