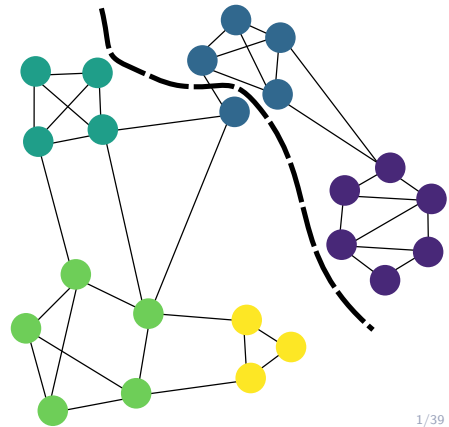
Handling Updates and Failures

Dynamic Graph Algorithms and Distributed Computing on Dynamic Networks

Antoine El-Hayek
Supervised by Prof. Monika Henzinger

Defense Talk
03.06.2026



Introduction

Three examples of changing networks:

- A train network gets new lines and lose others
- A grid database gets updated and completed
- Connections between neurons change

Introduction

Three examples of changing networks:

- A train network gets new lines and lose others
- A grid database gets updated and completed
- Connections between neurons change

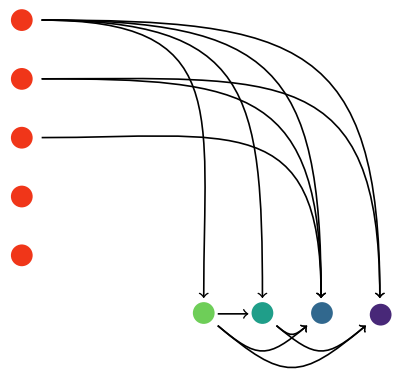
Questions on these networks:

- What algorithms can we run on them?
- What algorithms to manage them?

Part I

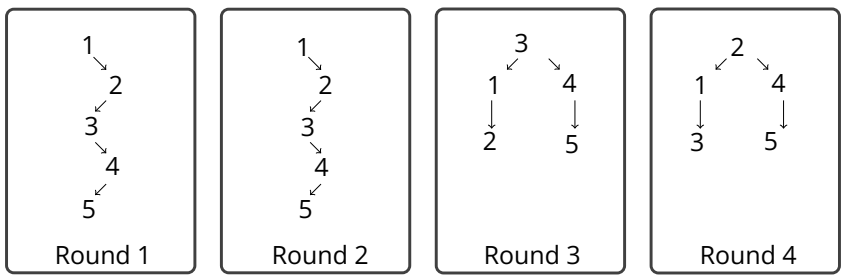
Broadcast in Dynamic Trees and Forests with a Deterministic Adversary

Joint work with:
Monika Henzinger
Stefan Schmid



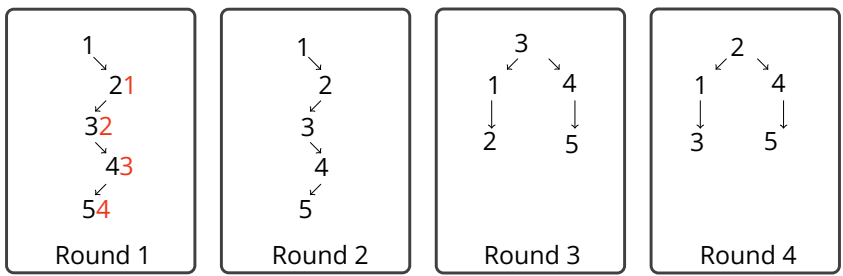
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



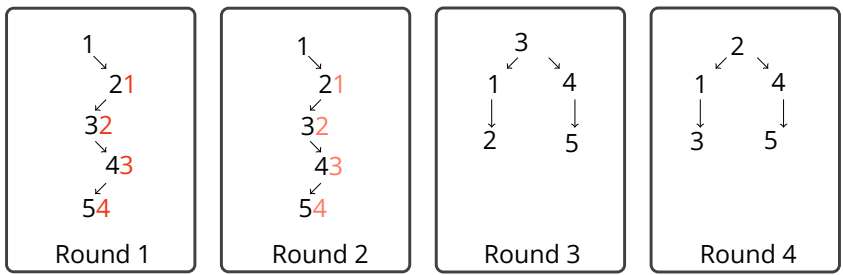
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



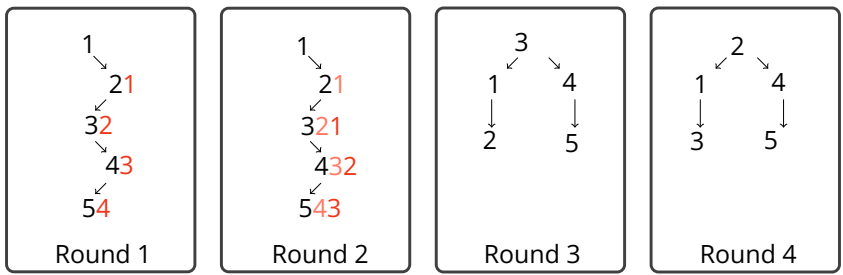
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



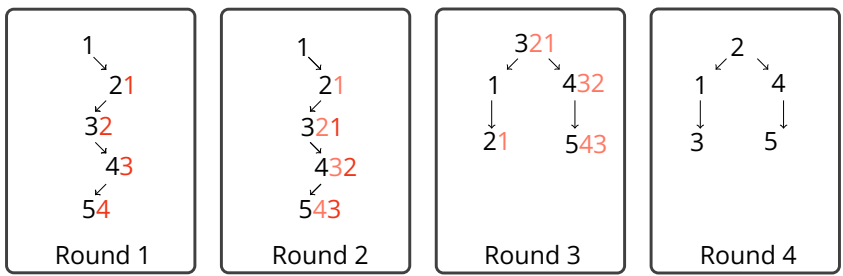
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



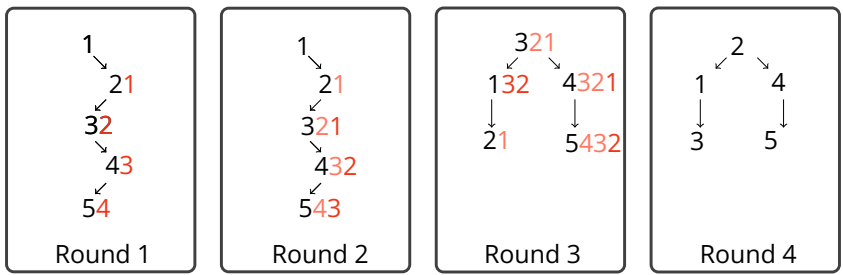
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



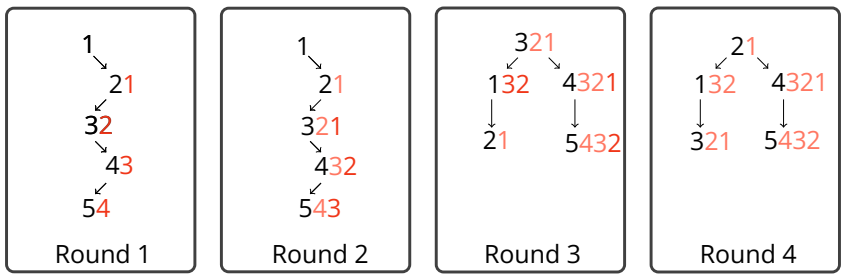
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



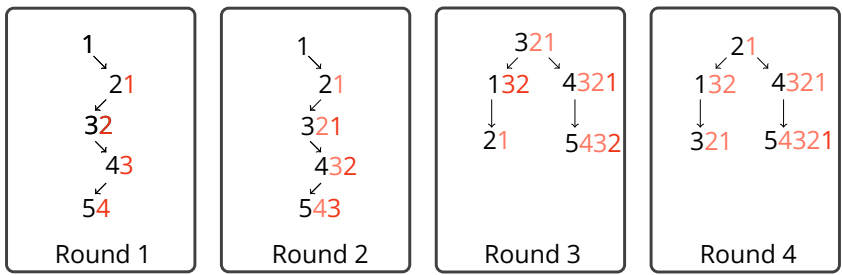
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



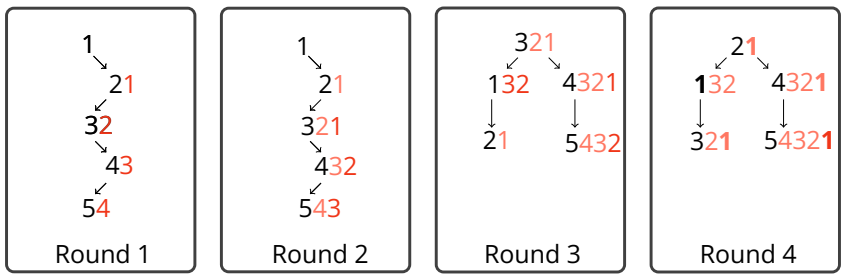
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



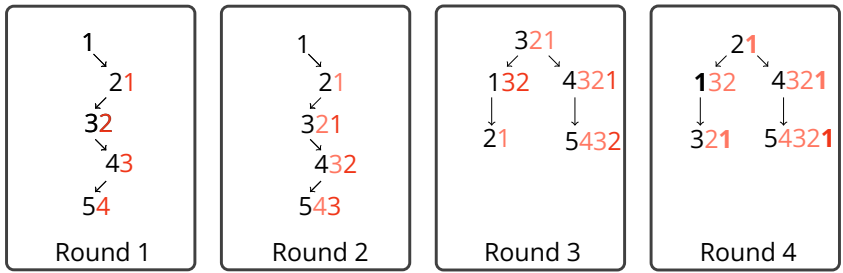
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



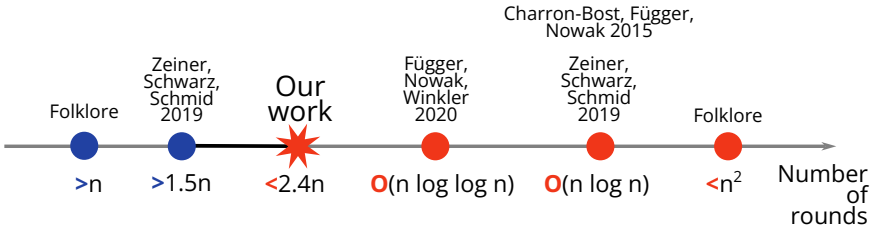
Problem Definition

- n agents, each with a message $\{1, 2, \ldots, n\}$.
- Broadcast: at least one agent sends its message to everyone
- In each round: communication happens in a directed rooted tree
- Agents can forward messages from other agents as well



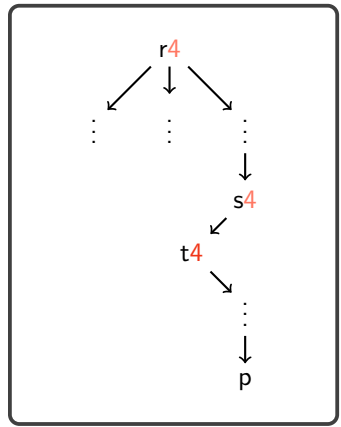
Question: In the worst case, how many rounds until broadcast?

Prior work


Folklore
Zeiner, Schwarz, Schmid 2019
Our work
Függer, Nowak, Winkler 2020
Charron-Bost, Függer, Nowak 2015
Zeiner, Schwarz, Schmid 2019
Folklore
>n
>1.5n
<2.4n
O(n log log n)
O(n log n)
<n2
Number of rounds

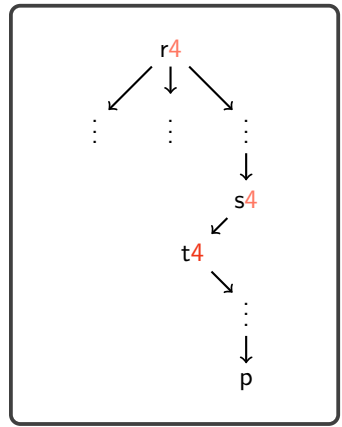
The key observation: primal

- Any message received by the root before the start of a round, is received by at least one new process during the round.



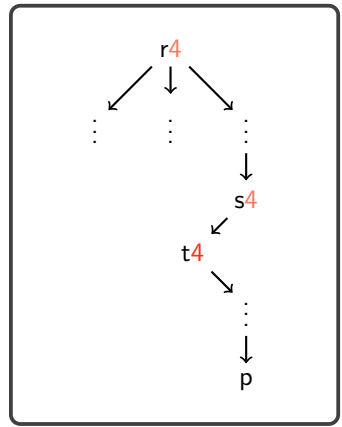
The key observation: primal

- Any message received by the root before the start of a round, is received by at least one new process during the round.
- If a message has been received by n roots, then everyone has received the message



The key observation: primal

- Any message received by the root before the start of a round, is received by at least one new process during the round.
- If a message has been received by n roots, then everyone has received the message
- We will keep track of the messages the root has received before each round



The auxiliary graph

Create a new graph:

- one node for each message
- one node for each round.

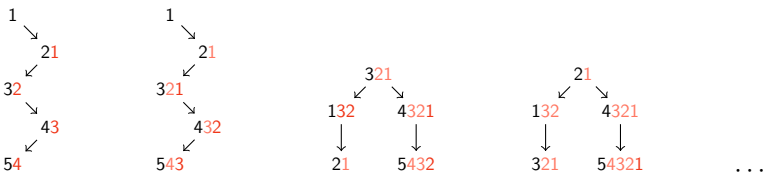
For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.

The auxiliary graph

Create a new graph:

- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.

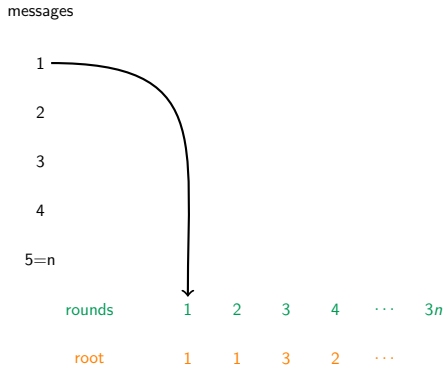


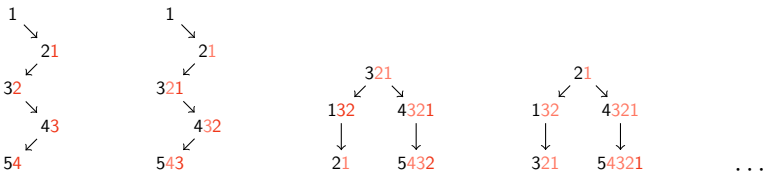
The auxiliary graph

Create a new graph:

- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.



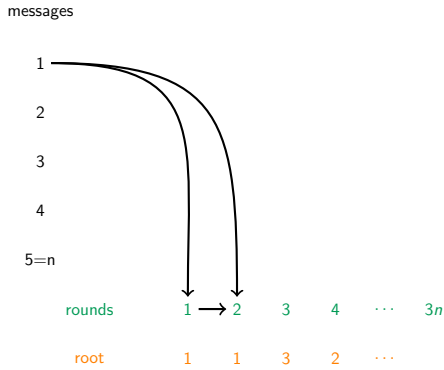


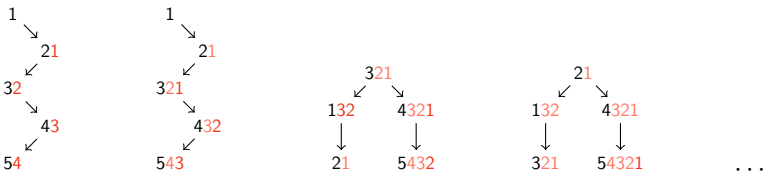
The auxiliary graph

Create a new graph:

- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.



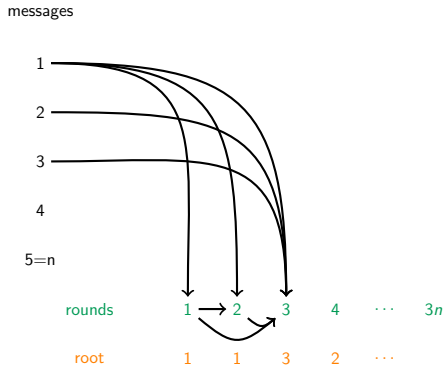


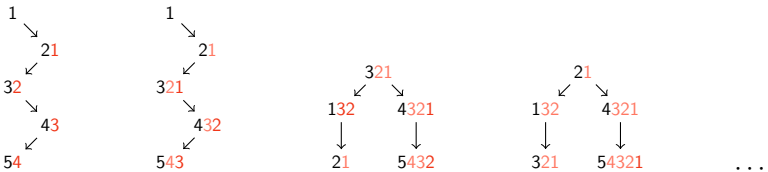
The auxiliary graph

Create a new graph:

- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.



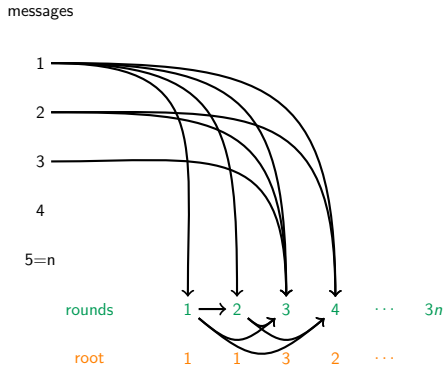


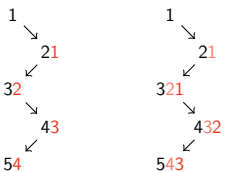
The auxiliary graph

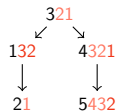
Create a new graph:

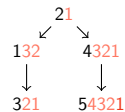
- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.









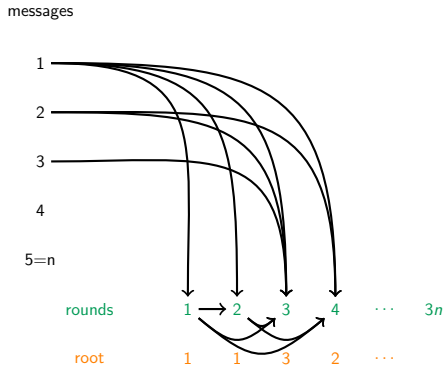
...

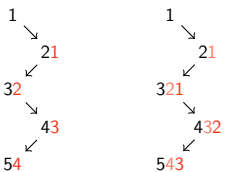
The auxiliary graph

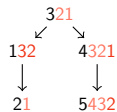
Create a new graph:

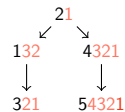
- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.









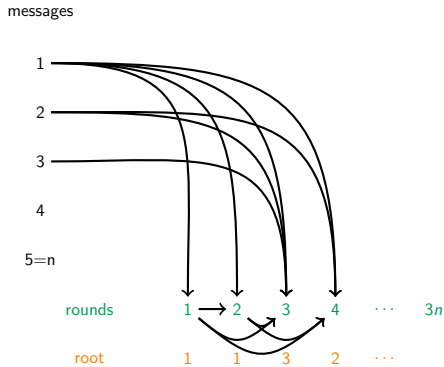
...

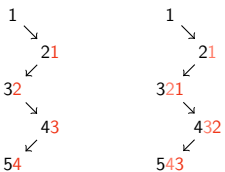
The auxiliary graph

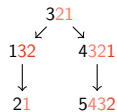
Create a new graph:

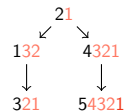
- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.









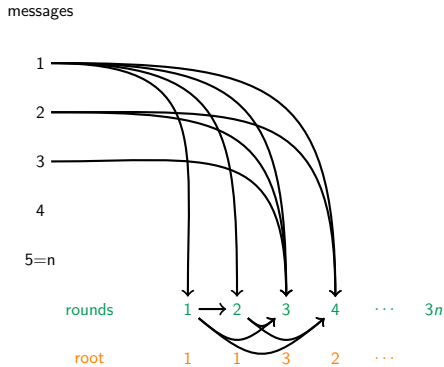
...

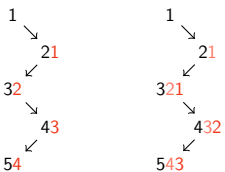
The auxiliary graph

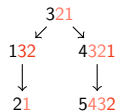
Create a new graph:

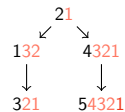
- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.









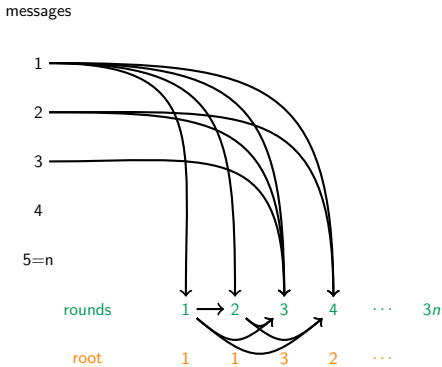
...

The auxiliary graph

Create a new graph:

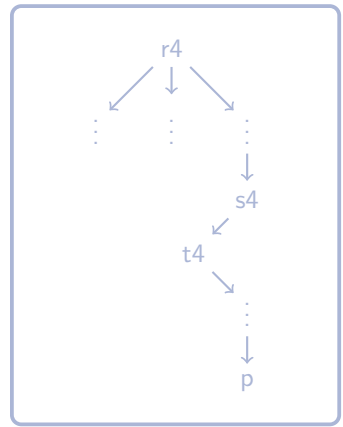
- one node for each message
- one node for each round.

For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.



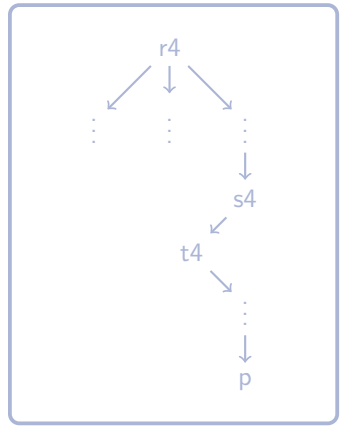
The Key Observation: Primal

- Any message received by the root before the start of a round, is received by at least one new process during the round.
- If a message has been received by n roots, then everyone has received the message
- We will keep track of the messages the root has received before each round



The Key Observation: Primal

- Any message received by the root before the start of a round, is received by at least one new process during the round.
- If a message has been received by n roots, then everyone has received the message
- We will keep track of the messages the root has received before each round
- If a node in the auxiliary graph has out-degree n, then broadcast is complete



The Key Observation: Primal and Dual

- Any message received by the root before the start of a round, is received by at least one new process during the round.
- If a message has been received by n roots, then everyone has received the message
- We will keep track of the messages the root has received before each round
- If a node in the auxiliary graph has out-degree n, then broadcast is complete

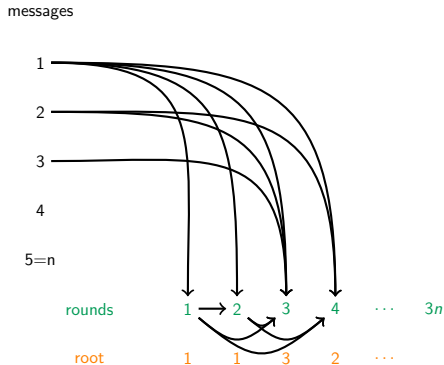
- In the auxiliary graph, the t-th round has in-degree at least t

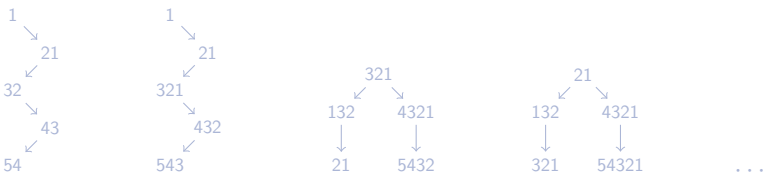
The Auxiliary Graph

Create a new graph:

- one node for each message
- one node for each round.

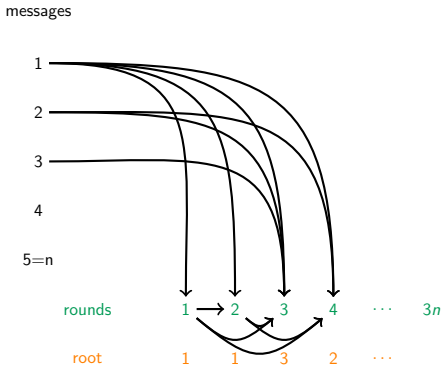
For each round t, add an edge from every message the root has received, and from every round $t' < t$ if the root of t has received the message of the root of t'.





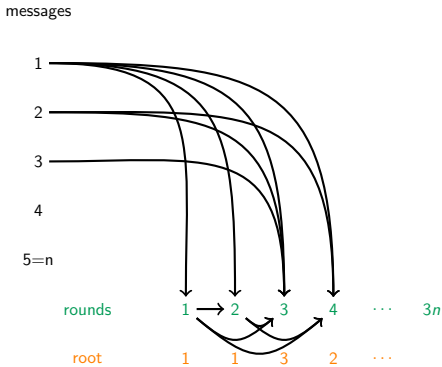
Analysing the auxiliary graph

- Consider $3n$ rounds



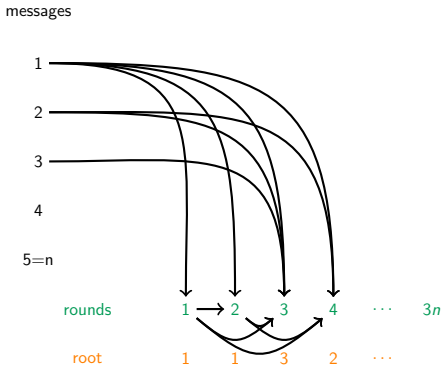
Analysing the auxiliary graph

- Consider $3n$ rounds
- If a node has out-degree at least n, then the corresponding message has reached everyone.



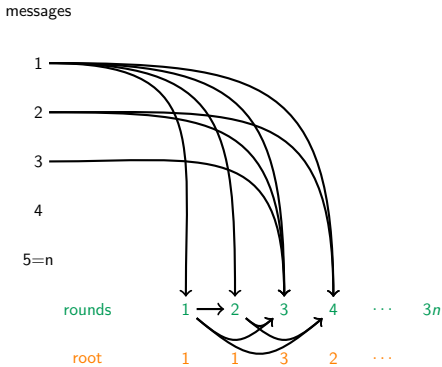
Analysing the auxiliary graph

- Consider $3n$ rounds
- If a node has out-degree at least n, then the corresponding message has reached everyone.
- Round t has in-degree at least t.



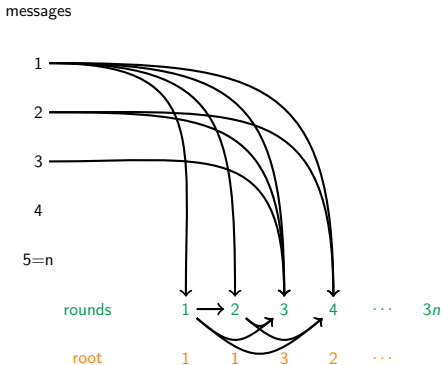
Analysing the auxiliary graph

- Consider $3n$ rounds
- If a node has out-degree at least n, then the corresponding message has reached everyone.
- Round t has in-degree at least t.
- The total number of edges is larger than $\sum_{t=1}^{3n} t = \frac{9n^2}{2} > 4n^2$.



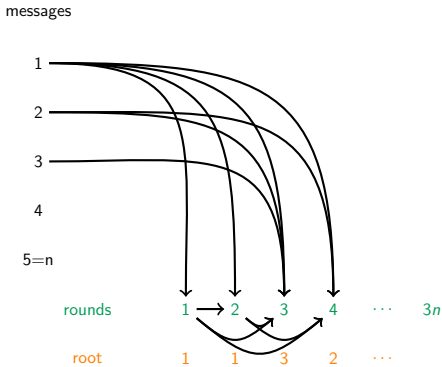
Analysing the auxiliary graph

- Consider $3n$ rounds
- If a node has out-degree at least n, then the corresponding message has reached everyone.
- Round t has in-degree at least t.
- The total number of edges is larger than $\sum_{t=1}^{3n} t = \frac{9n^2}{2} > 4n^2$.
- We have $4n$ nodes total



Analysing the auxiliary graph

- Consider $3n$ rounds
- If a node has out-degree at least n, then the corresponding message has reached everyone.
- Round t has in-degree at least t.
- The total number of edges is larger than $\sum_{t=1}^{3n} t = \frac{9n^2}{2} > 4n^2$.
- We have $4n$ nodes total
- By the pigeonhole principle, there must be a node with out-degree at least n



The main result

The Upper Bound

An upper bound for Broadcast on rooted trees is $\lceil (1+\sqrt{2})n \rceil \sim 2.4n$.

[1] Zeiner, M., Schwarz, M., and Schmid, U. (2019). On linear-time data dissemination in dynamic rooted trees. Discrete Applied Mathematics, 255, 307-319.

The main result

The Upper Bound

An upper bound for Broadcast on rooted trees is $\lceil (1+\sqrt{2})n \rceil \sim 2.4n$.

The Lower Bound

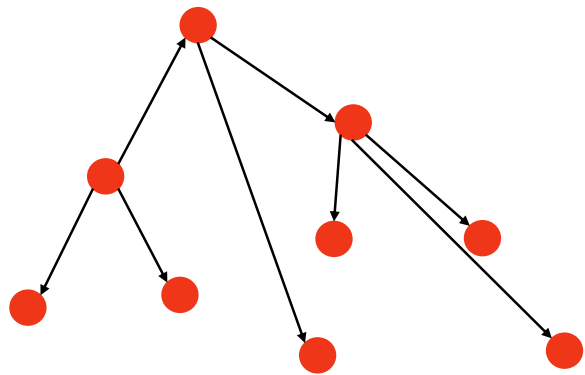
A lower bound for Broadcast on rooted trees is $\sim 1.5n$ [1].

[1] Zeiner, M., Schwarz, M., and Schmid, U. (2019). On linear-time data dissemination in dynamic rooted trees. Discrete Applied Mathematics, 255, 307-319.

Part II

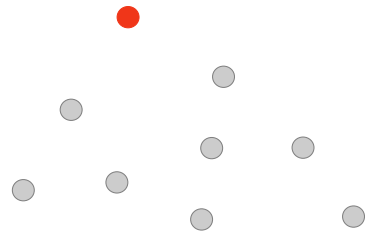
Broadcast in Dynamic Trees with a Random Adversary

Joint work with:
Monika Henzinger
Stefan Schmid



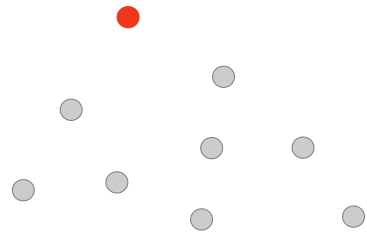
Problem Definition

- One node starts with a piece of information.



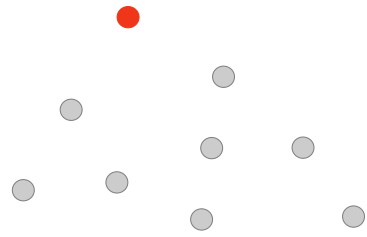
Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.



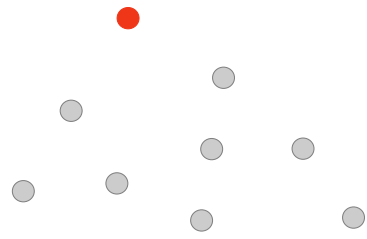
Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.



Problem Definition

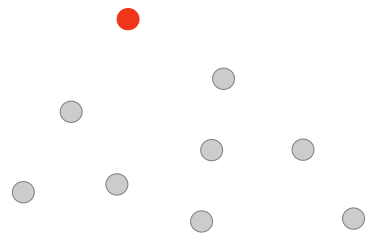
- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.



Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.

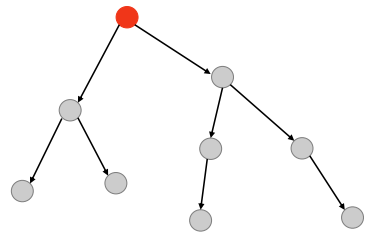
Question: How many rounds does broadcast need?



Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.

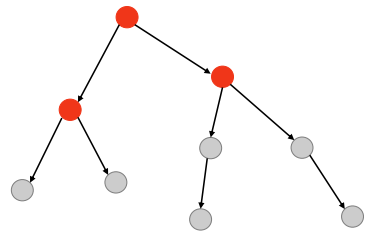
Question: How many rounds does broadcast need?



Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.

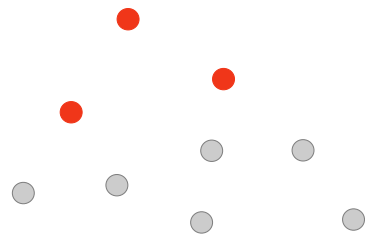
Question: How many rounds does broadcast need?



Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.

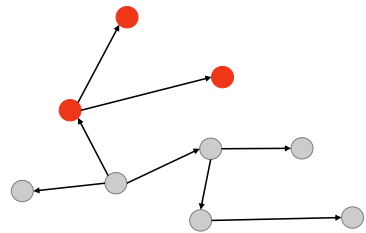
Question: How many rounds does broadcast need?



Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.

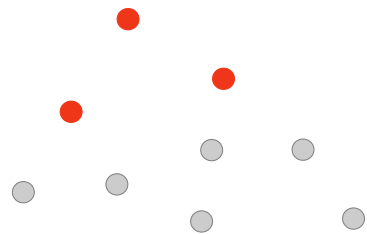
Question: How many rounds does broadcast need?



Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.

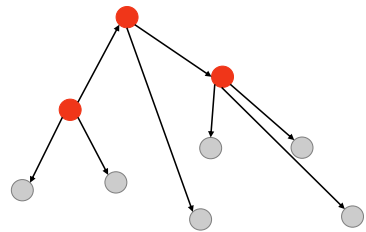
Question: How many rounds does broadcast need?



Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.

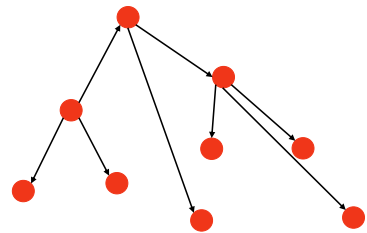
Question: How many rounds does broadcast need?



Problem Definition

- One node starts with a piece of information.
- This node must send it to everyone else.
- Nodes can forward the message.
- In each round, the communication network is a rooted tree, chosen uniformly at random among all rooted trees.

Question: How many rounds does broadcast need?



Counting Trees

Theorem 1

There are n^{n-1} rooted trees on n vertices.

Counting Trees

Theorem 1

There are n^{n-1} rooted trees on n vertices.

Theorem 2

Let E be a subset of edges of a tree. Then there are $n^{n-1-|E|}$ rooted trees on n vertices that contain E.

Counting Trees

Theorem 1

There are n^{n-1} rooted trees on n vertices.

Theorem 2

Let E be a subset of edges of a tree. Then there are $n^{n-1-|E|}$ rooted trees on n vertices that contain E.

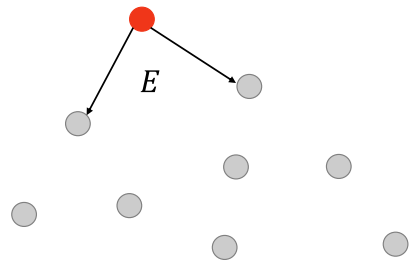
Hence: if T is a rooted tree chosen uniformly at random, then:
$\mathbb{P}(E \subseteq T) = n^{-|E|}$

Counting Trees

$\mathbb{P}(E \subseteq T) = n^{-|E|}$

Example:

- $n = 9$
- $|E| = 2$
- $\mathbb{P}(E \subseteq T) = 9^{-2}$

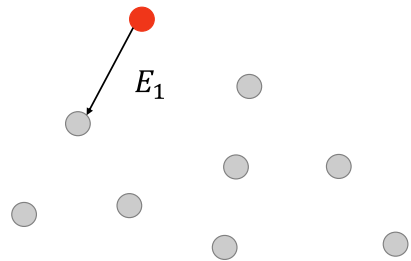


Counting Trees

$\mathbb{P}(E \subseteq T) = n^{-|E|}$

Example:

- $n = 9$
- $|E_1| = 1$
- $\mathbb{P}(E \subseteq T) = 9^{-1}$

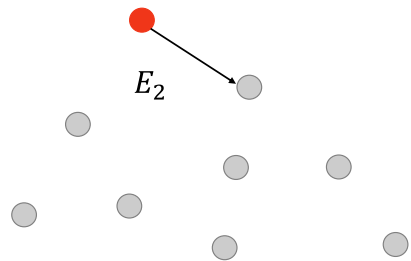


Counting Trees

$\mathbb{P}(E \subseteq T) = n^{-|E|}$

Example:

- $n = 9$
- $|E_2| = 1$
- $\mathbb{P}(E \subseteq T) = 9^{-1}$

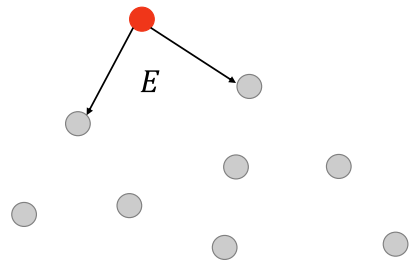


Counting Trees

$\mathbb{P}(E \subseteq T) = n^{-|E|}$

Example:

- $n = 9$
- $|E| = 2$
- $\mathbb{P}(E \subseteq T) = 9^{-2}$



Counting Trees

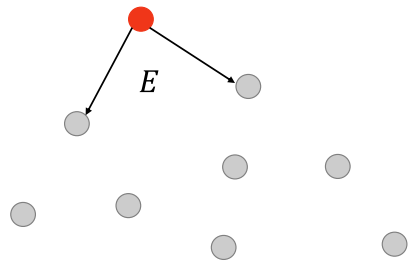
$\mathbb{P}(E \subseteq T) = n^{-|E|}$

Example:

- $n = 9$
- $|E| = 2$
- $\mathbb{P}(E \subseteq T) = 9^{-2}$

Observation:

- $\mathbb{P}(E \subseteq T) = \mathbb{P}(E_1 \subseteq T) \times \mathbb{P}(E_2 \subseteq T)$



Counting Trees

$\mathbb{P}(E \subseteq T) = n^{-|E|}$

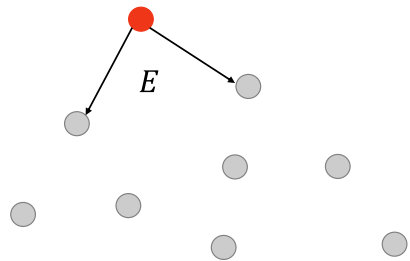
Example:

- $n = 9$
- $|E| = 2$
- $\mathbb{P}(E \subseteq T) = 9^{-2}$

Observation:

- $\mathbb{P}(E \subseteq T) = \mathbb{P}(E_1 \subseteq T) \times \mathbb{P}(E_2 \subseteq T)$

$\Rightarrow$ The events "$E_1 \subseteq T$" and "$E_2 \subseteq T$" are independent !



Independence

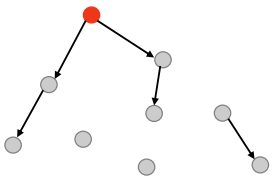
Theorem 3

Edges are either independent or incompatible.

Independence

Theorem 3

Edges are either independent or incompatible.



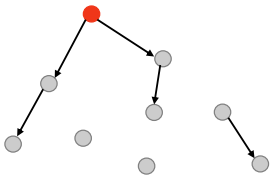
Independent

$\mathbb{P}(E \subseteq T) = n^{-|E|}$

Independence

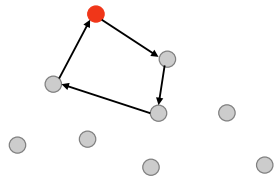
Theorem 3

Edges are either independent or incompatible.



Independent

$\mathbb{P}(E \subseteq T) = n^{-|E|}$



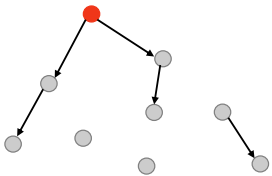
Incompatible

$\mathbb{P}(E \subseteq T) = 0$

Independence

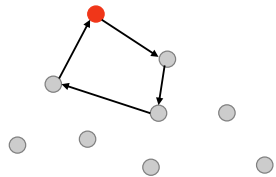
Theorem 3

Edges are either independent or incompatible.



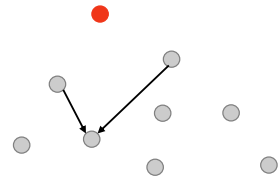
Independent

$\mathbb{P}(E \subseteq T) = n^{-|E|}$



Incompatible

$\mathbb{P}(E \subseteq T) = 0$



Incompatible

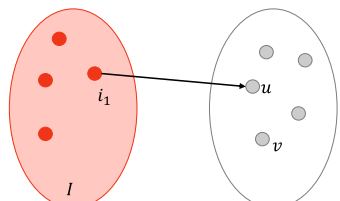
$\mathbb{P}(E \subseteq T) = 0$

Independence

Theorem 3

Edges are either independent or incompatible.

$\mathbb{P}(i_1 \to u) = \frac{1}{n}$

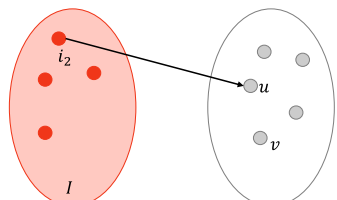


Independence

Theorem 3

Edges are either independent or incompatible.

$\mathbb{P}(i_2 \to u) = \frac{1}{n}$

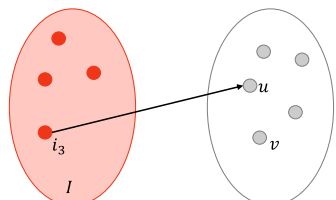


Independence

Theorem 3

Edges are either independent or incompatible.

$\mathbb{P}(i_3 \to u) = \frac{1}{n}$

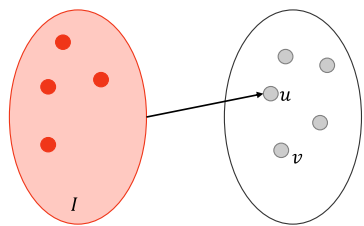


Independence

Theorem 3

Edges are either independent or incompatible.

$\mathbb{P}(I \to u) = \sum_k \mathbb{P}(i_k \to u) = \frac{|I|}{n}$
because of incompatibility

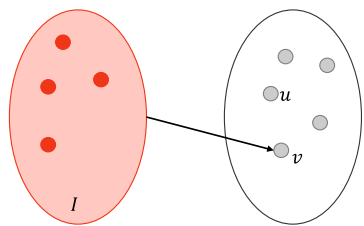


Independence

Theorem 3

Edges are either independent or incompatible.

$\mathbb{P}(I \to v) = \frac{|I|}{n}$

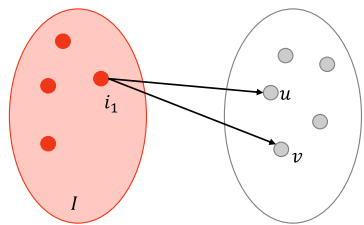


Independence

Theorem 3

Edges are either independent or incompatible.

$\mathbb{P}(i_1 \to u \bigcap i_1 \to v) = \frac{1}{n} \times \frac{1}{n}$
because of independence

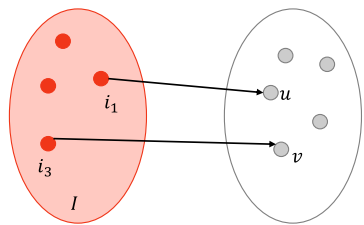


Independence

Theorem 3

Edges are either independent or incompatible.

$\mathbb{P}(i_1 \to u \bigcap i_3 \to v) = \frac{1}{n} \times \frac{1}{n}$
because of independence

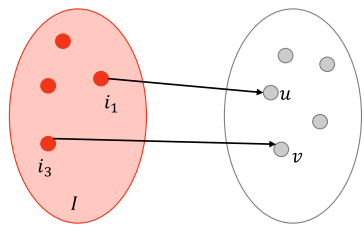


Independence

Theorem 3

Edges are either independent or incompatible.

$\mathbb{P}(i_j \to u \bigcap i_k \to v) = \frac{1}{n} \times \frac{1}{n}$
because of independence

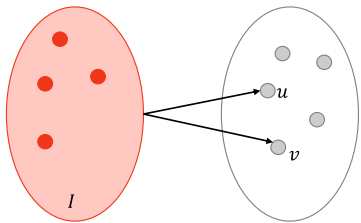


Independence

Theorem 3

Edges are either independent or incompatible.

$$\begin{aligned}\mathbb{P}(I \to u \bigcap I \to v) &= \sum_{j,k} \mathbb{P}(i_j \to u \bigcap i_k \to v) \\ &\qquad\qquad \text{(by incompatibility)} \\ &= \frac{|I|^2}{n^2} \\ &= \mathbb{P}(I \to u) \times \mathbb{P}(I \to v)\end{aligned}$$

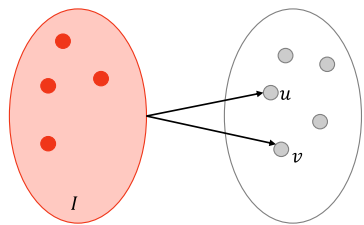


Independence

Theorem 3

Edges are either independent or incompatible.

$$\begin{aligned}\mathbb{P}(I \to u \bigcap I \to v) &= \sum_{j,k} \mathbb{P}(i_j \to u \bigcap i_k \to v) \\ &\qquad\qquad \text{(by incompatibility)} \\ &= \frac{|I|^2}{n^2} \\ &= \mathbb{P}(I \to u) \times \mathbb{P}(I \to v)\end{aligned}$$



The events "$I \to u$" and "$I \to v$" are independent.

Runtime Analysis

- In each round, each uninformed node gets informed with probability $\frac{|I|}{n}$, *independently from other nodes*.

Runtime Analysis

- In each round, each uninformed node gets informed with probability $\frac{|I|}{n}$, *independently from other nodes*.
- We have a stochastic dynamical system:

$$|I_0| = 1$$
$$|I_{t+1}| = |I_t| + B\left(n - I_t, \frac{|I_t|}{n}\right)$$

where $B(\ell, p)$ is the Bernoulli distribution of parameters ℓ and p.

Runtime Analysis

- In each round, each uninformed node gets informed with probability $\frac{|I|}{n}$, *independently from other nodes*.
- We have a stochastic dynamical system:

$$|I_0| = 1$$
$$|I_{t+1}| = |I_t| + B\left(n - I_t, \frac{|I_t|}{n}\right)$$

  where $B(\ell, p)$ is the Bernoulli distribution of parameters ℓ and p.
- If $t > 2\log n$, we have that $|I_t| > n - 1$ with high probability.

The Main Result

Theorem 1

Broadcast on uniformly chosen rooted trees takes $O(\log n)$ rounds, with high probability.

The Main Result

Theorem 1

Broadcast on uniformly chosen rooted trees takes $O(\log n)$ rounds, with high probability.

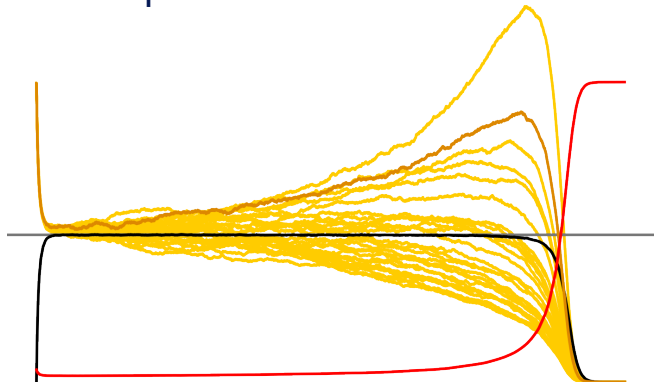
Theorem 2

If an adversary can choose up to $k < \frac{n}{3}$ many edges to appear in the tree, broadcast takes $O(k + \log n)$ rounds, with high probability.

Part III

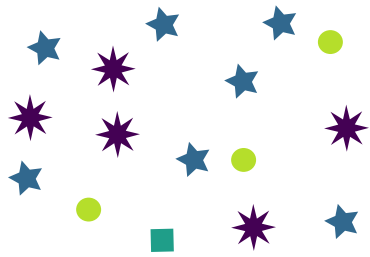
Undecided States Dynamic in Population Protocols

Joint work with:
Robert Elsässer
Stefan Schmid



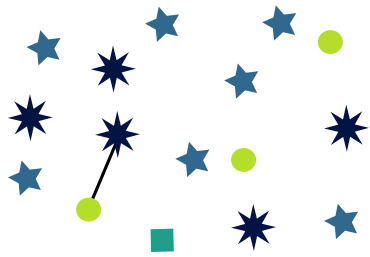
Problem Definition

- n agents, each with limited memory



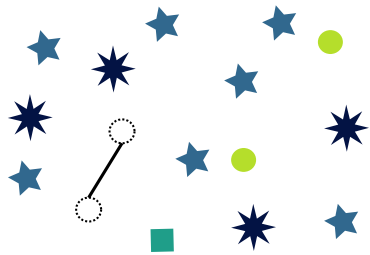
Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction



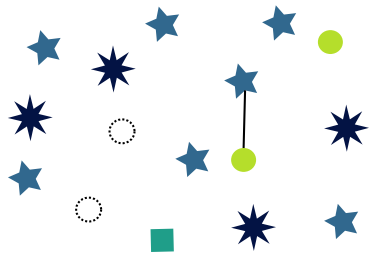
Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided



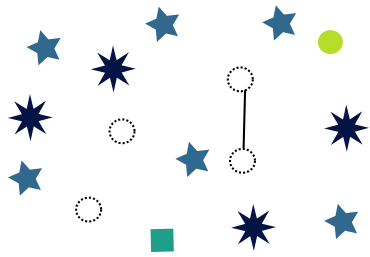
Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided



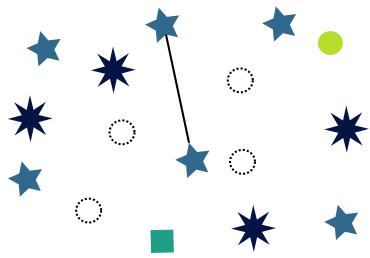
Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided



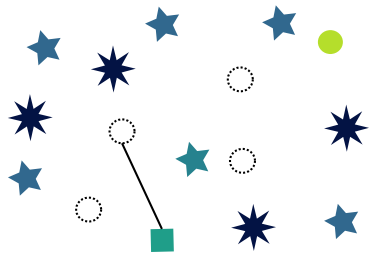
Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided



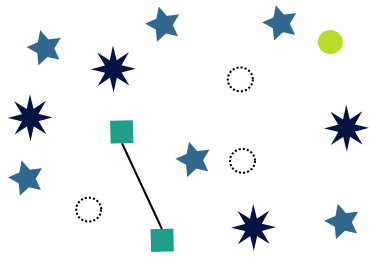
Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided
- If a decided agent meets an undecided agent, the undecided takes the opinion of the decided agent



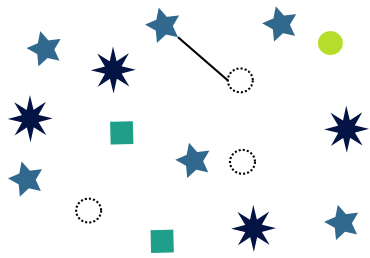
Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided
- If a decided agent meets an undecided agent, the undecided takes the opinion of the decided agent



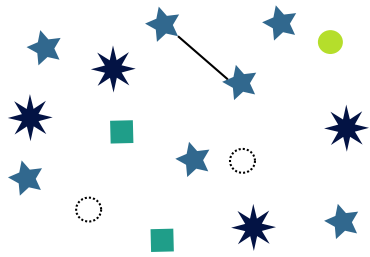
Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided
- If a decided agent meets an undecided agent, the undecided takes the opinion of the decided agent



Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided
- If a decided agent meets an undecided agent, the undecided takes the opinion of the decided agent



Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided
- If a decided agent meets an undecided agent, the undecided takes the opinion of the decided agent

Question: How long until all agents share the same opinion?

Problem Definition

- n agents, each with limited memory
- *Random* scheduler: at each time step, two agents are chosen uniformly at random for interaction
- If two agents with different opinions meet, they become undecided
- If a decided agent meets an undecided agent, the undecided takes the opinion of the decided agent

Question: How long until all agents share the same opinion?

- [AABBHK23] $\sim O(k \cdot n \cdot \log n)$ interactions.
- [EES25] $\sim \Omega(k \cdot n \cdot \log n)$ interactions (Our work)

Undecided States Dynamics Simulation

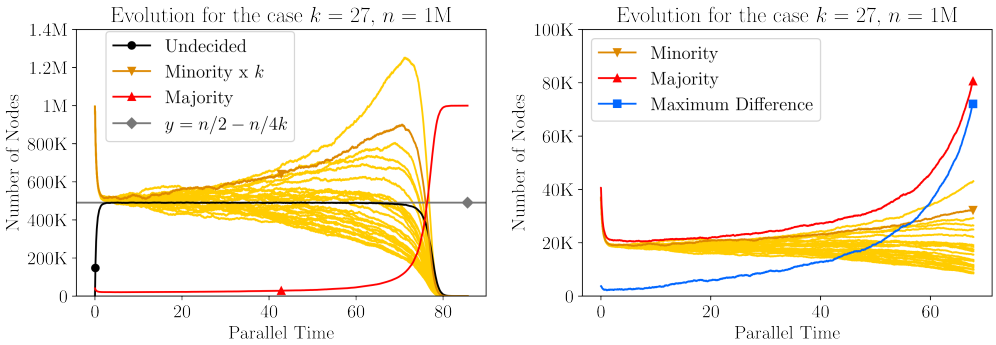
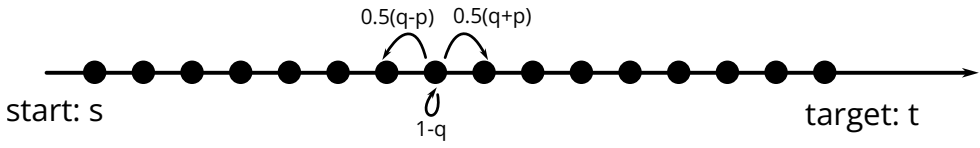


Figure 1: Simulation for $n = 10^9$, $k = 27$

Our Main Tool: Drift Analysis

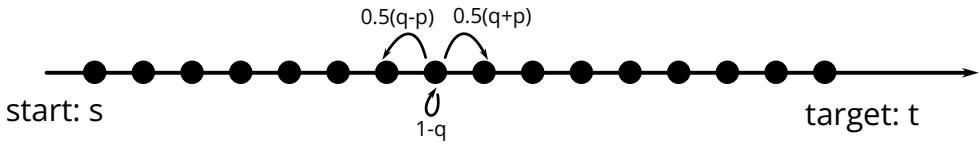
Consider a random walk:



Our Main Tool: Drift Analysis

Consider a random walk:

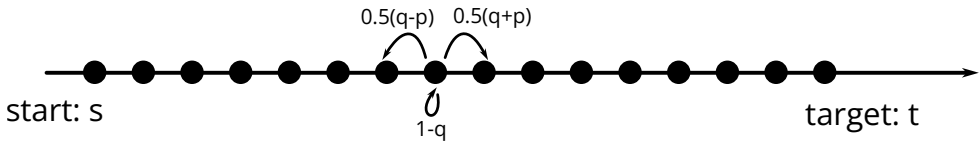
- with bias p



Our Main Tool: Drift Analysis

Consider a random walk:

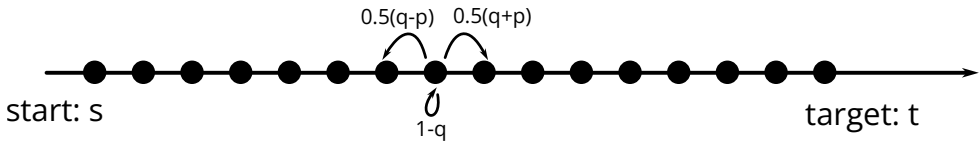
- with bias p
- The probability moving is q



Our Main Tool: Drift Analysis

Consider a random walk:

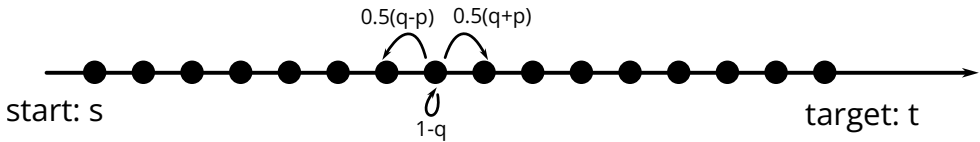
- with bias p
- The probability moving is q
- Assume you start at s and want to reach target t



Our Main Tool: Drift Analysis

Consider a random walk:

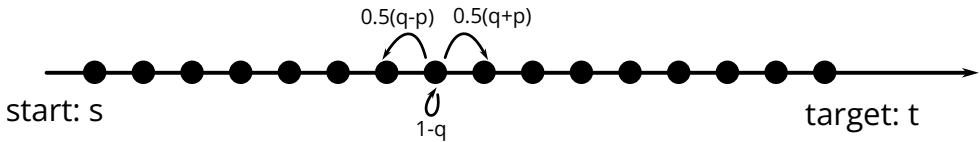
- with bias p
- The probability moving is q
- Assume you start at s and want to reach target t
- Then, if $t - s >> \sqrt{(t-s)\frac{q}{p}}$ and with high probability, this happens in time $\Omega(\frac{t-s}{p})$.



Our Main Tool: Drift Analysis

Consider a random walk:

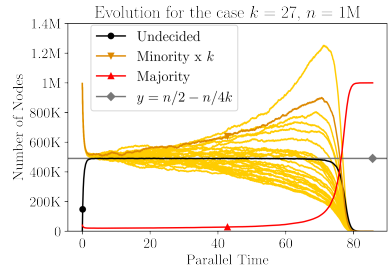
- with bias p
- The probability moving is q
- Assume you start at s and want to reach target t
- Then, if $t - s >> \sqrt{(t-s)\frac{q}{p}}$ and with high probability, this happens in time $\Omega(\frac{t-s}{p})$.
- Conversely, if $p \leq 0$, with high probability this doesn't happen in time $\Omega(n^2)$

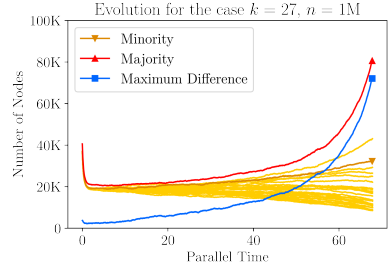


Undecided States Dynamics Analysis

Using drift analysis (random walks), we show that:

- The number of undecided nodes does not exceed $\frac{n}{2} - \frac{n}{4k}$

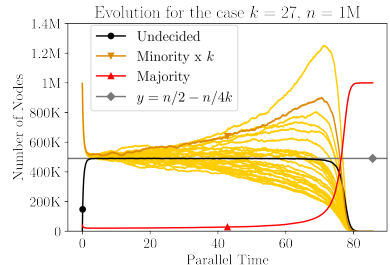


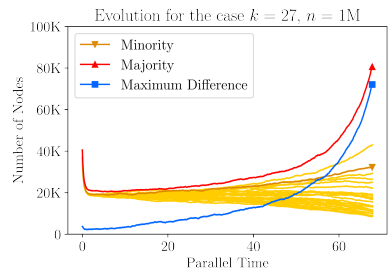


Undecided States Dynamics Analysis

Using drift analysis (random walks), we show that:

- The number of undecided nodes does not exceed $\frac{n}{2} - \frac{n}{4k}$
- It takes kn interactions for the majority opinion to double from $\frac{n}{k}$ to $2\frac{n}{k}$

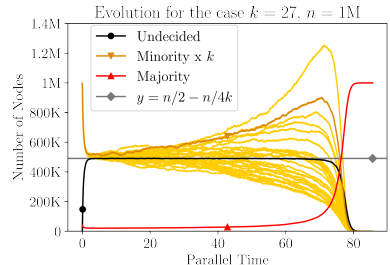


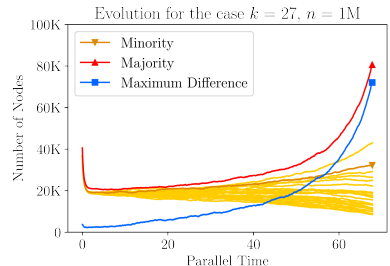


Undecided States Dynamics Analysis

Using drift analysis (random walks), we show that:

- The number of undecided nodes does not exceed $\frac{n}{2} - \frac{n}{4k}$
- It takes kn interactions for the majority opinion to double from $\frac{n}{k}$ to $2\frac{n}{k}$
- While the maximum opinion is smaller than $2\frac{n}{k}$, the maximum difference takes kn interactions to double

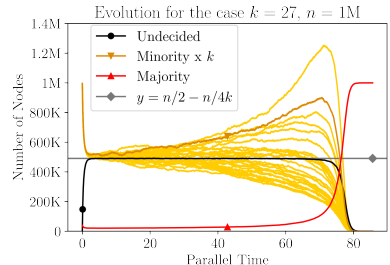


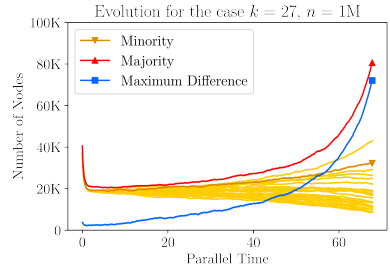


Undecided States Dynamics Analysis

Using drift analysis (random walks), we show that:

- The number of undecided nodes does not exceed $\frac{n}{2} - \frac{n}{4k}$
- It takes kn interactions for the majority opinion to double from $\frac{n}{k}$ to $2\frac{n}{k}$
- While the maximum opinion is smaller than $2\frac{n}{k}$, the maximum difference takes kn interactions to double
- While the maximum difference is $o(\frac{n}{k})$, surely the maximum opinion is at most $\frac{n}{k}$





The main result

The Lower Bound

A lower bound for the convergence of the Undecided States Dynamics is $\Omega(kn \log \frac{\sqrt{n}}{k \log n})$ interactions, for any $k = o\left(\frac{\sqrt{n}}{\log n}\right)$.

[2] Amir, Talley, James Aspnes, Petra Berenbrink, Felix Biermeier, Christopher Hahn, Dominik Kaaser, and John Lazarsfeld. "Fast convergence of k-opinion undecided state dynamics in the population protocol model." In Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, pp. 13-23. 2023.

The main result

The Lower Bound

A lower bound for the convergence of the Undecided States Dynamics is $\Omega(kn \log \frac{\sqrt{n}}{k \log n})$ interactions, for any $k = o\left(\frac{\sqrt{n}}{\log n}\right)$.

The Upper Bound[2]

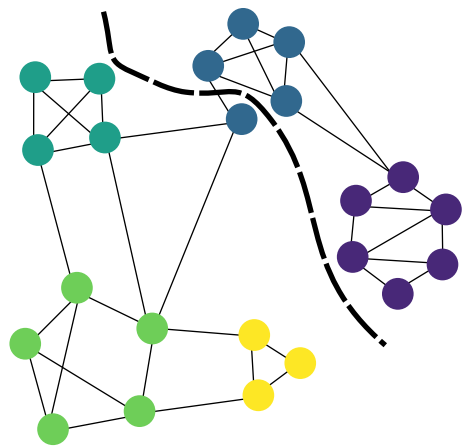
An upper bound for the convergence of the Undecided States Dynamics is $O(kn \log n)$ interactions, for $k \leq n^{\frac{1}{2}-\epsilon}$.

[2] Amir, Talley, James Aspnes, Petra Berenbrink, Felix Biermeier, Christopher Hahn, Dominik Kaaser, and John Lazarsfeld. "Fast convergence of k-opinion undecided state dynamics in the population protocol model." In Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, pp. 13-23. 2023.

Part IV

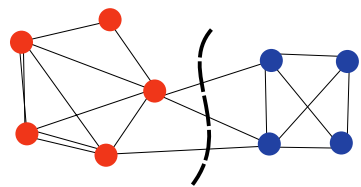
Fully-Dynamic Minimum Cut

Joint work with:
Monika Henzinger
Jason Li



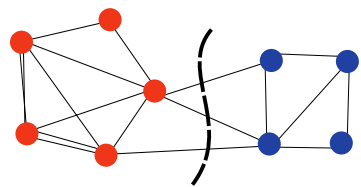
The Minimum Cut Problem

- Given a Graph $G = (V, E)$, partition V into V_1 and V_2 minimizing the number of crossing edges.
- Parallel edges are allowed
- Dynamic updates: Insertion and deletion of edges
- We denote by λ the value of the minimum cut.



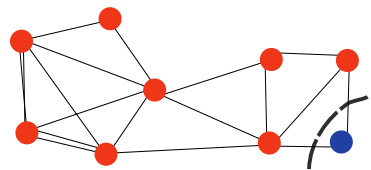
The Minimum Cut Problem

- Given a Graph $G = (V, E)$, partition V into V_1 and V_2 minimizing the number of crossing edges.
- Parallel edges are allowed
- Dynamic updates: Insertion and deletion of edges
- We denote by λ the value of the minimum cut.



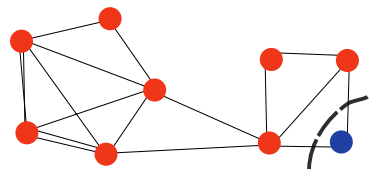
The Minimum Cut Problem

- Given a Graph $G = (V, E)$, partition V into V_1 and V_2 minimizing the number of crossing edges.
- Parallel edges are allowed
- Dynamic updates: Insertion and deletion of edges
- We denote by λ the value of the minimum cut.



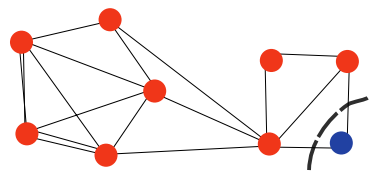
The Minimum Cut Problem

- Given a Graph $G = (V, E)$, partition V into V_1 and V_2 minimizing the number of crossing edges.
- Parallel edges are allowed
- Dynamic updates: Insertion and deletion of edges
- We denote by λ the value of the minimum cut.



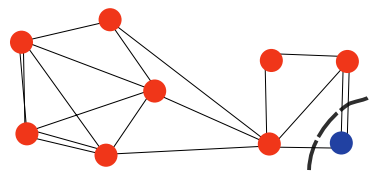
The Minimum Cut Problem

- Given a Graph $G = (V, E)$, partition V into V_1 and V_2 minimizing the number of crossing edges.
- Parallel edges are allowed
- Dynamic updates: Insertion and deletion of edges
- We denote by λ the value of the minimum cut.



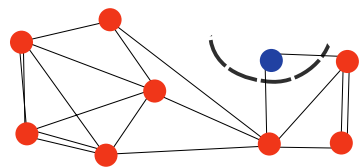
The Minimum Cut Problem

- Given a Graph $G = (V, E)$, partition V into V_1 and V_2 minimizing the number of crossing edges.
- Parallel edges are allowed
- Dynamic updates: Insertion and deletion of edges
- We denote by λ the value of the minimum cut.



The Minimum Cut Problem

- Given a Graph $G = (V, E)$, partition V into V_1 and V_2 minimizing the number of crossing edges.
- Parallel edges are allowed
- Dynamic updates: Insertion and deletion of edges
- We denote by λ the value of the minimum cut.



Minimum Cut results

For **static** minimum cut:

Reference	**Running Time**	**Deterministic**	**Weighted**
Karger (2000)	Near-linear time	No	Yes
Kawarabayashi, Thorup (2015)	Near-linear time	Yes	No
Henzinger, Rao, Wang (2020)	Near-linear time	Yes	No
Henzinger, Li, Rao, Wang (2024)	Near-linear time	Yes	Yes

Dynamic Minimum Cut Results

For **Dynamic Exact** minimum cut:

Reference	**Upd. Time**	**Deterministic**	**Conditions**
Henzinger (1997)	$O(\lambda \log n)$	Yes	Incremental
Thorup (2007)	$\tilde{O}(\lambda^{14.5}\sqrt{n})$	Yes	None
Goranci, Henzinger, Thorup (2018)	$\tilde{O}(1)$	Yes	Incremental
Goranci, Henzinger, Nanongkai, Saranurak, Thorup, Wulff-Nilsen (2023)	$\tilde{O}(m^{30/31})$	Yes	None
Jin, Sun, Thorup (2024)	$n^{o(1)}$	Yes	$\lambda \leq (\log n)^{o(1)}$
de Vos, Christiansen (2025)	$\tilde{O}(\lambda^{5.5}\sqrt{n})$ $\tilde{O}(m^{11/12})$	Yes	None
El-Hayek, Henzinger, Li (2026)	$n^{o(1)}$	Yes	$\lambda \leq 2^{\Theta(\log^{3/4-c} n)}$

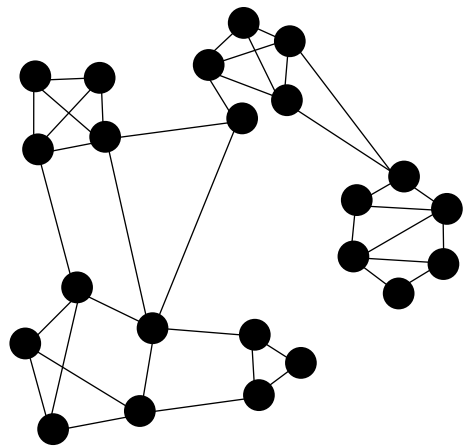
Dynamic Minimum Cut Results

For **Dynamic Approximate** minimum cut:

Reference	**Upd. Time**	**Approx. Ratio**	**Weighted**	**Det.**
Thorup, Karger (2000)	$\tilde{O}(\lambda^2)$	$\sqrt{2+o(1)}$	No	Yes
Thorup (2007)	$\tilde{O}(\sqrt{n})$	$1+\frac{1}{\log^{0.05} n}$	No	Yes
Goranci, Räcke, Saranurak, Tan (2021)	$n^{o(1)}$	$n^{o(1)}$	No	Yes
van den Brand, Chen, Kyng, Liu, Meierhans, Gutenberg, Sachdeva (2024)	$n^{o(1)}$	$n^{o(1)}$	Yes	Yes
El-Hayek, Henzinger, Li (2025)	$n^{o(1)}$	$1+\frac{1}{\log^{0.25} n}$	No	No
El-Hayek, Henzinger, Li (2026)	$n^{o(1)}$	$1+2^{-\Theta(\log^{3/4-c} n)}$	Yes	No

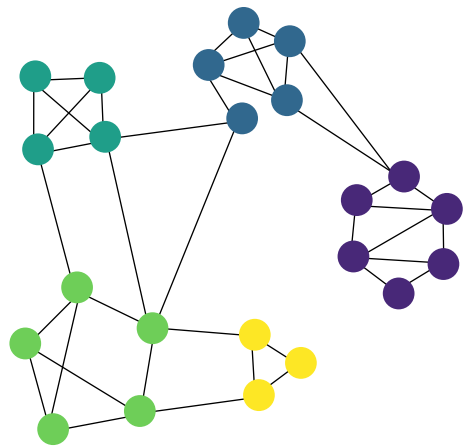
A First Heuristic

1. Find expander decomposition of G
2. Contract each expander into a node to create a contracted graph
3. Repeat 1.2 until two nodes remain
4. Return value of final cut



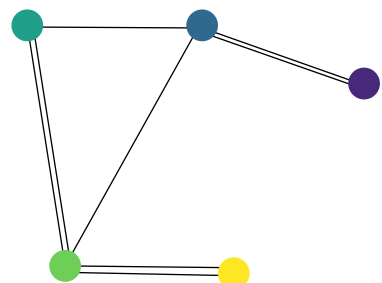
A First Heuristic

1. **Find expander decomposition of** G
2. Contract each expander into a node to create a contracted graph
3. Repeat 1.2 until two nodes remain
4. Return value of final cut



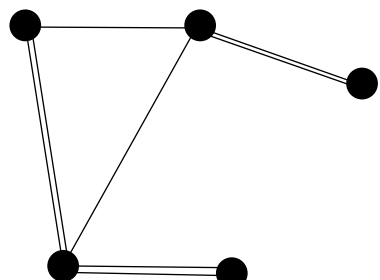
A First Heuristic

1. Find expander decomposition of G
2. **Contract each expander into a node to create a contracted graph**
3. Repeat 1.2 until two nodes remain
4. Return value of final cut



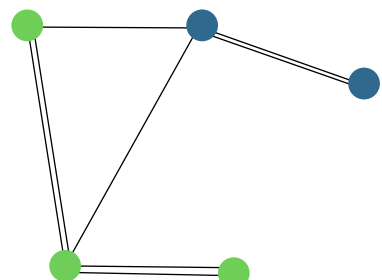
A First Heuristic

1. Find expander decomposition of G
2. Contract each expander into a node to create a contracted graph
3. **Repeat 1.2 until two nodes remain**
4. Return value of final cut



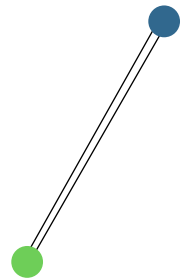
A First Heuristic

1. **Find expander decomposition of** G
2. Contract each expander into a node to create a contracted graph
3. Repeat 1.2 until two nodes remain
4. Return value of final cut



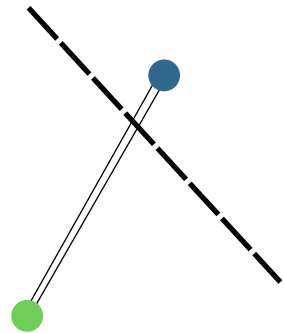
A First Heuristic

1. Find expander decomposition of G
2. **Contract each expander into a node to create a contracted graph**
3. Repeat 1.2 until two nodes remain
4. Return value of final cut



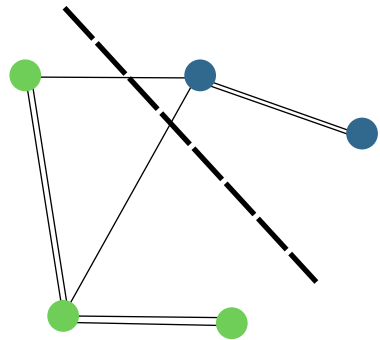
A First Heuristic

1. Find expander decomposition of G
2. Contract each expander into a node to create a contracted graph
3. Repeat 1.2 until two nodes remain
4. **Return value of final cut**



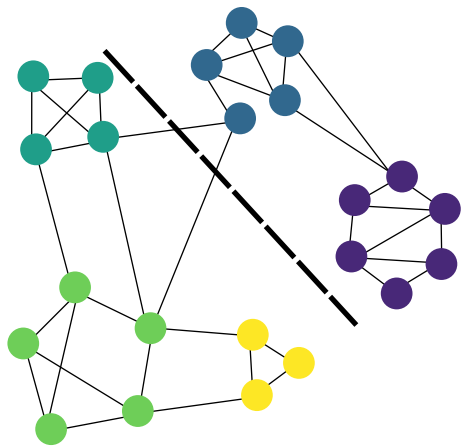
A First Heuristic

1. Find expander decomposition of G
2. Contract each expander into a node to create a contracted graph
3. Repeat 1.2 until two nodes remain
4. **Return value of final cut**



A First Heuristic

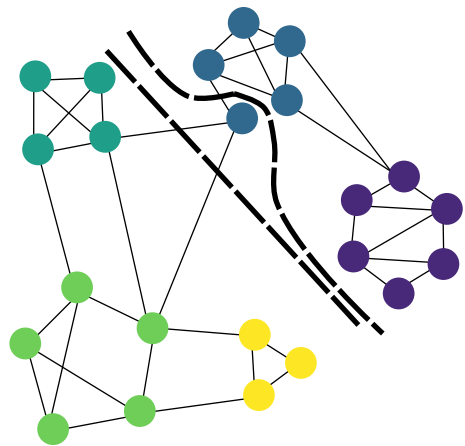
1. Find expander decomposition of G
2. Contract each expander into a node to create a contracted graph
3. Repeat 1.2 until two nodes remain
4. **Return value of final cut**



A First Heuristic

1. Find expander decomposition of G
2. Contract each expander into a node to create a contracted graph
3. Repeat 1.2 until two nodes remain
4. **Return value of final cut**

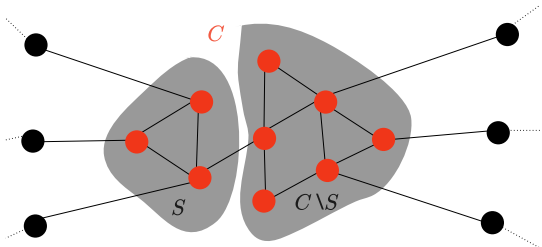
Problem: This does not return the correct value of the minimum cut.



$(1-\epsilon)$-Boundary Sparseness

A subset $S \subset C$ of a cluster C is $(1-\epsilon)$-boundary sparse if:

$$|E(S, C \setminus S)| < (1-\epsilon)\min\{|E(S, V \setminus C)|, |E(S, C \setminus S)|\}$$

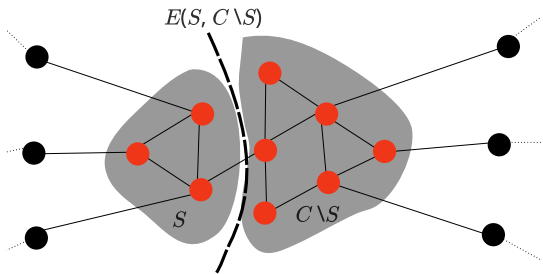


a $(1-\epsilon)$ boundary-sparse cut

$(1-\epsilon)$-Boundary Sparseness

A subset $S \subset C$ of a cluster C is $(1-\epsilon)$-boundary sparse if:

$$|E(S, C \setminus S)| < (1-\epsilon) \min\{|E(S, V \setminus C)|, |E(S, C \setminus S)|\}$$

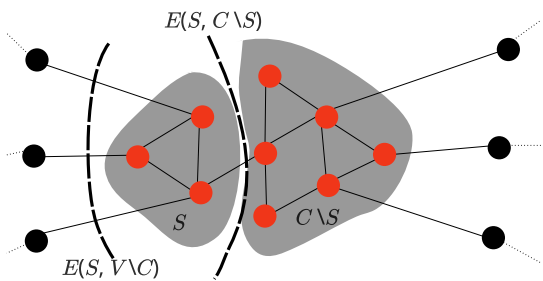


a $(1-\epsilon)$ boundary-sparse cut

$(1-\epsilon)$-Boundary Sparseness

A subset $S \subset C$ of a cluster C is $(1-\epsilon)$-boundary sparse if:

$$|E(S, C \setminus S)| < (1-\epsilon) \min\{|E(S, V \setminus C)|, |E(S, C \setminus S)|\}$$

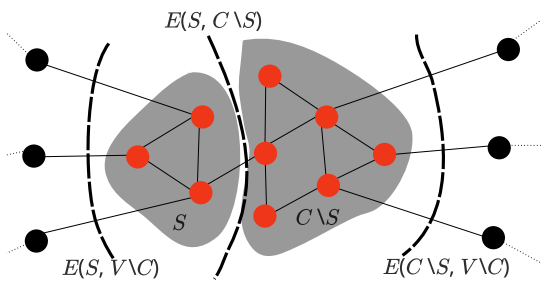


a $(1-\epsilon)$ boundary-sparse cut

$(1-\epsilon)$-Boundary Sparseness

A subset $S \subset C$ of a cluster C is $(1-\epsilon)$-boundary sparse if:

$$|E(S, C \setminus S)| < (1-\epsilon) \min\{|E(S, V \setminus C)|, |E(S, C \setminus S)|\}$$

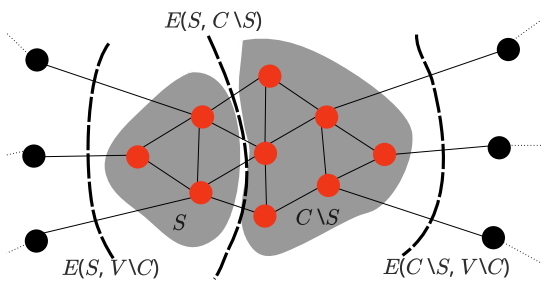


a $(1-\epsilon)$ boundary-sparse cut

$(1-\epsilon)$-Boundary Sparseness

A subset $S \subset C$ of a cluster C is $(1-\epsilon)$-boundary sparse if:

$$|E(S, C \setminus S)| < (1-\epsilon)\min\{|E(S, V \setminus C)|, |E(S, C \setminus S)|\}$$

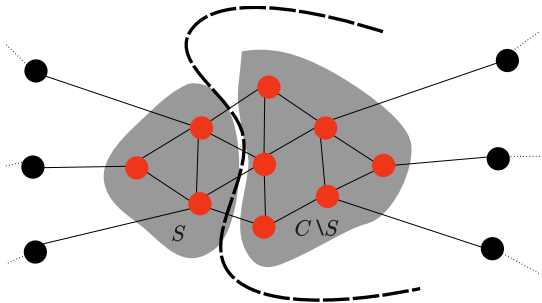


a **non** $(1-\epsilon)$-boundary-sparse cut

$(1-\epsilon)$-Boundary Sparseness

A subset $S \subset C$ of a cluster C is $(1-\epsilon)$-boundary sparse if:

$$|E(S, C \setminus S)| < (1-\epsilon)\min\{|E(S, V \setminus C)|, |E(S, C \setminus S)|\}$$

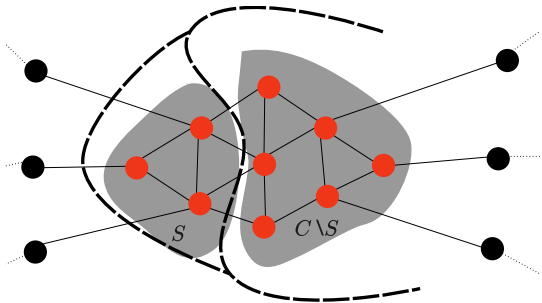


a **non** $(1-\epsilon)$-boundary-sparse cut

$(1-\epsilon)$-Boundary Sparseness

A subset $S \subset C$ of a cluster C is $(1-\epsilon)$-boundary sparse if:

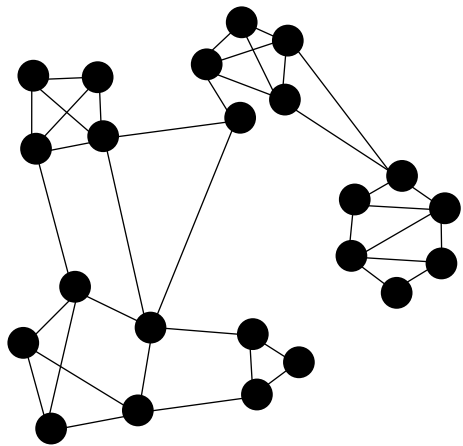
$$|E(S, C \setminus S)| < (1-\epsilon)\min\{|E(S, V \setminus C)|, |E(S, C \setminus S)|\}$$



a **non** $(1-\epsilon)$-boundary-sparse cut

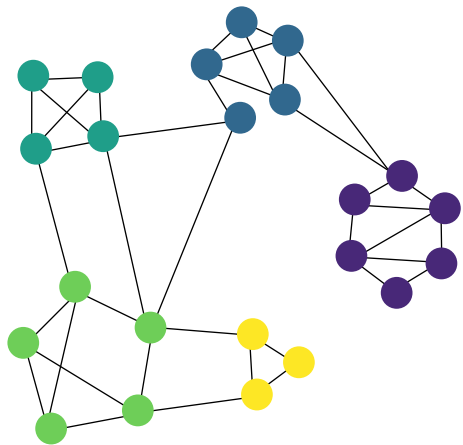
Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



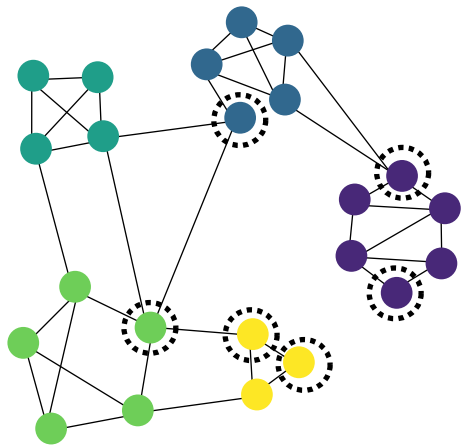
Our Algorithm

1. **Find expander decomposition of** G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



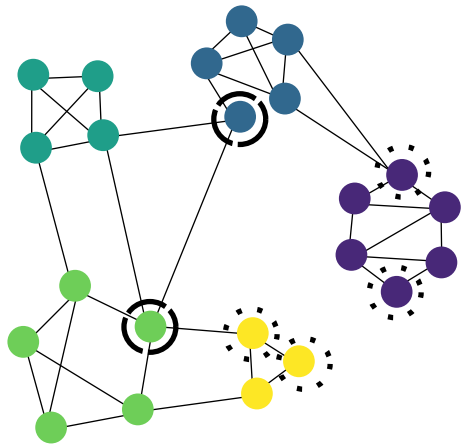
Our Algorithm

1. Find expander decomposition of G
2. **Find cuts with LocalKCut**
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



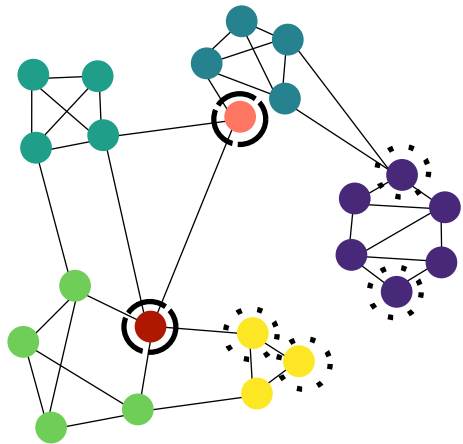
Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. **Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts**
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



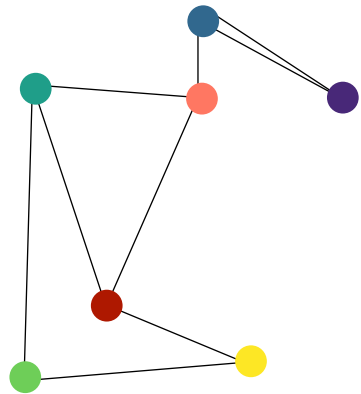
Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. **Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts**
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



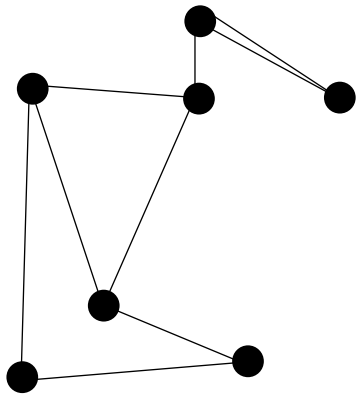
Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. **Contract each cluster into a node to create a contracted graph**
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



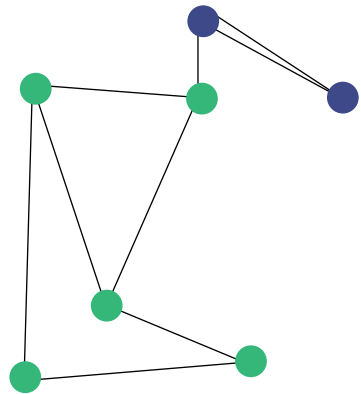
Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. **Repeat 1.2.3.4 until two nodes remain**
6. Return value of final cut



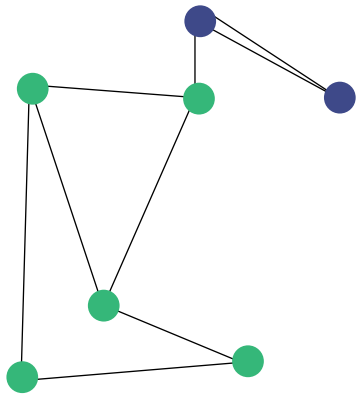
Our Algorithm

1. **Find expander decomposition of** G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



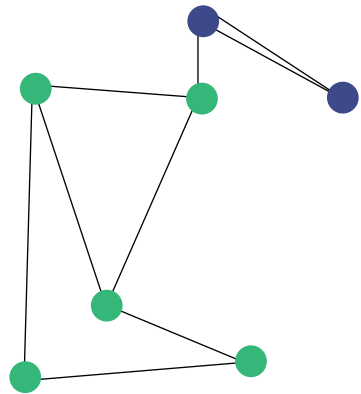
Our Algorithm

1. Find expander decomposition of G
2. **Find cuts with LocalKCut**
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



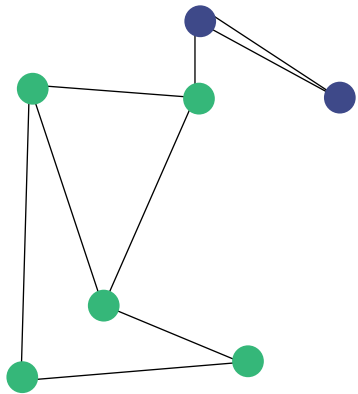
Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. **Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts**
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut



Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. **Contract each cluster into a node to create a contracted graph**
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut

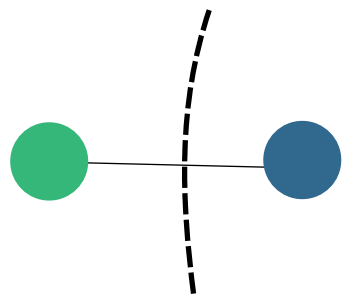


Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. **Contract each cluster into a node to create a contracted graph**
5. Repeat 1.2.3.4 until two nodes remain
6. Return value of final cut

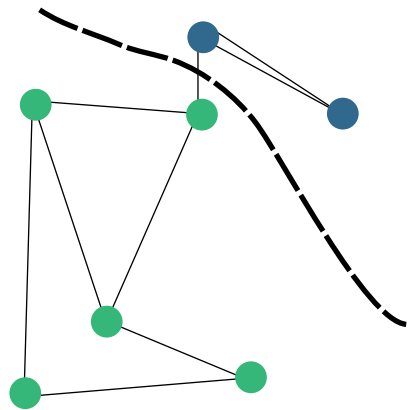
Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. **Return value of final cut**



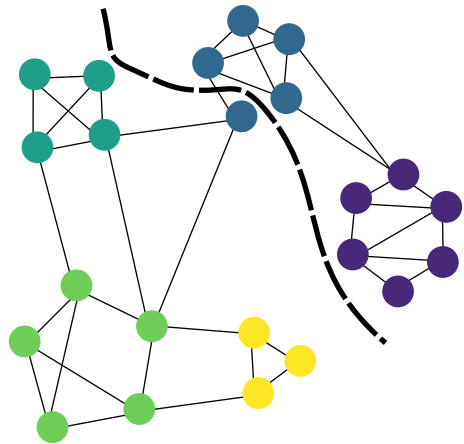
Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. **Return value of final cut**



Our Algorithm

1. Find expander decomposition of G
2. Find cuts with LocalKCut
3. Cut expanders along the $(1-\epsilon)$-Boundary Sparse cuts
4. Contract each cluster into a node to create a contracted graph
5. Repeat 1.2.3.4 until two nodes remain
6. **Return value of final cut**



The main results

Theorem 1

For **unweighted** graphs, we give a **deterministic** fully-dynamic algorithm that can maintain the **exact** minimum cut in $n^{o(1)}$ update time as long as $\lambda \leq 2^{\Theta(\log^{3/4-c} n)}$.

The main results

Theorem 1

For **unweighted** graphs, we give a **deterministic** fully-dynamic algorithm that can maintain the **exact** minimum cut in $n^{o(1)}$ update time as long as $\lambda \leq 2^{\Theta(\log^{3/4-c} n)}$.

Theorem 2

For **positively, real-weighted** graphs, we give a **randomised** fully-dynamic algorithm that can maintain a $\left(1 + 2^{-\Theta(\log^{3/4-c} n)}\right)$-**approximate** minimum cut in $n^{o(1)}$ update time.

Part V
What's Next?

Population Protocols

- What about the optimal space complexity protocols?

Population Protocols

- What about the optimal space complexity protocols?
- What about other problems, like partition or counting?

Population Protocols

- What about the optimal space complexity protocols?
- What about other problems, like partition or counting?
- What if all interactions are not possible, and there exists an underlying graph?

Dynamic Graph Algorithms

On the minimum cut problem:

- Can we extend the exact algorithm to all values of λ?
- What about the directed case?
- What about the exact weighted case? The deterministic weighted case?

Thank you

And thank you for all my past and ongoing collaborators:

- Monika Henzinger, my supervisor
- Stefan Schmid
- Robert Elsässer
- Jason Li
- Julien Dallot
- Tom-Lukas Breitkopf
- Gregor Bankhamer