

On Coloring Embedded Graphs

Alexandre Nolin

CISPA Helmholtz Center for Information Security (for 8½ hours)

AMG@DISC 2025 – Berlin, 31.10.2025

Primarily based on joint works [FHN24, FHN25] with



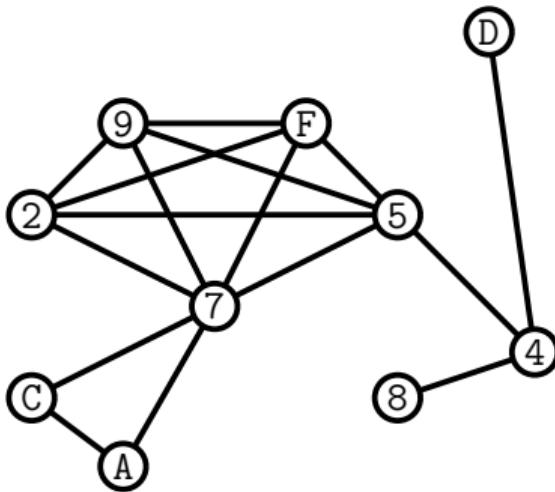
Maxime Flin



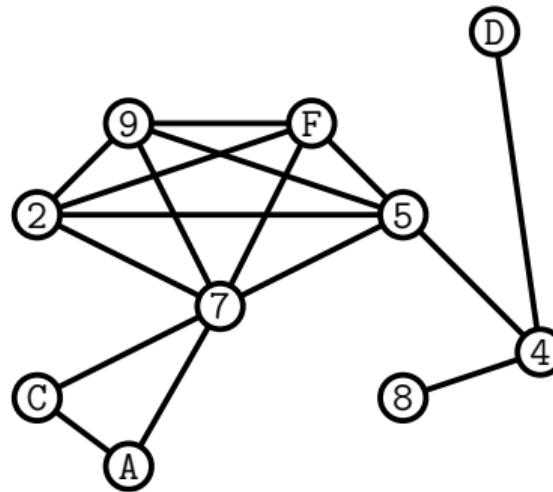
Magnús M. Halldórsson

LOCAL and CONGEST

LOCAL



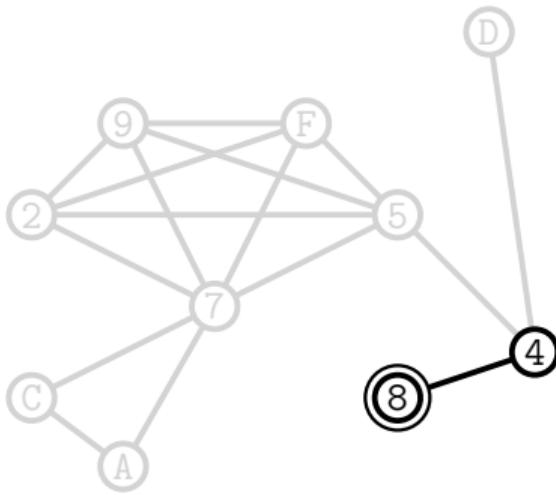
CONGEST



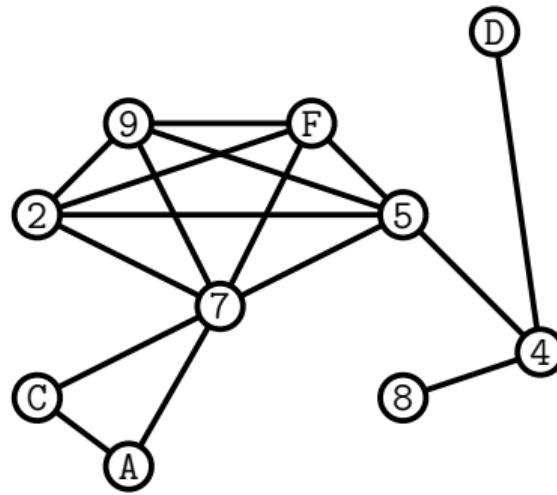
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL



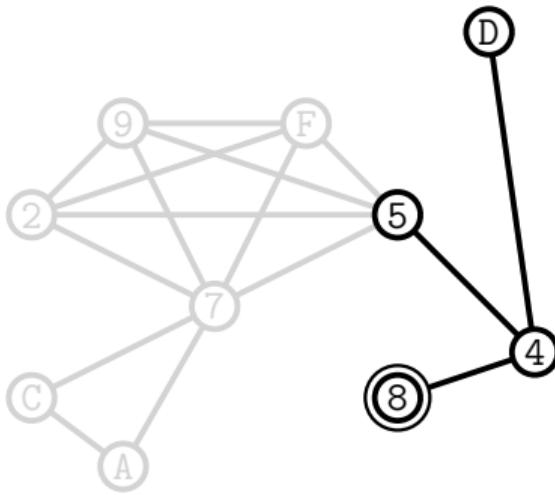
CONGEST



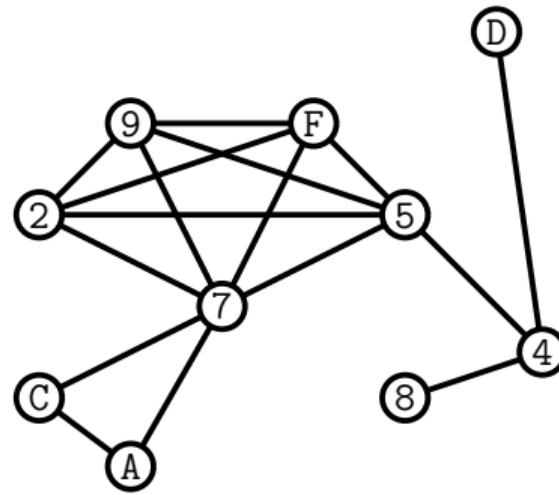
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL



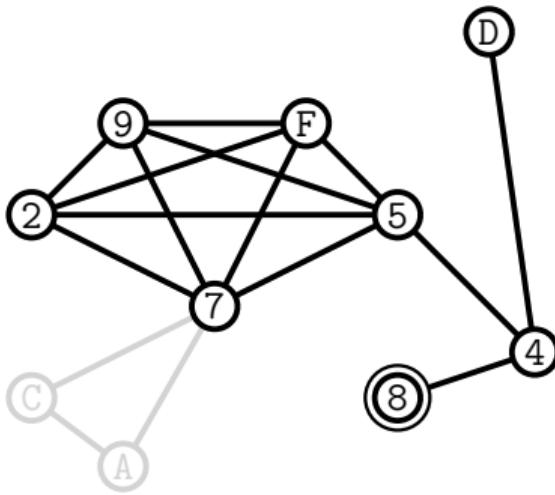
CONGEST



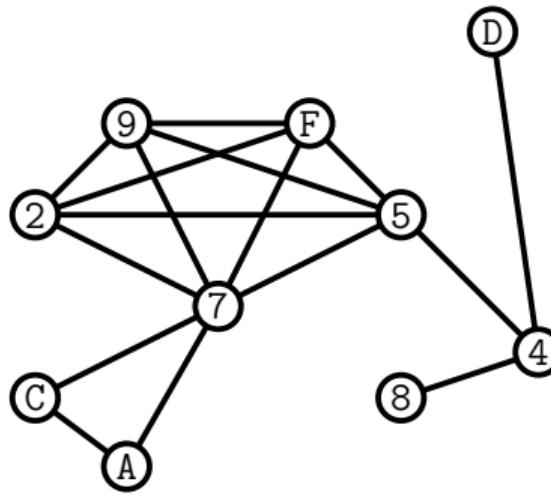
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL



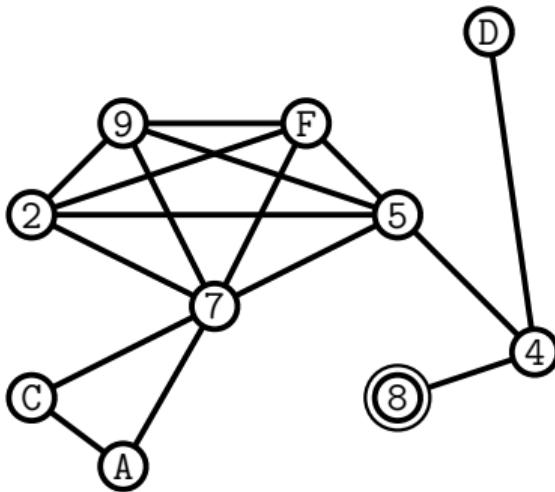
CONGEST



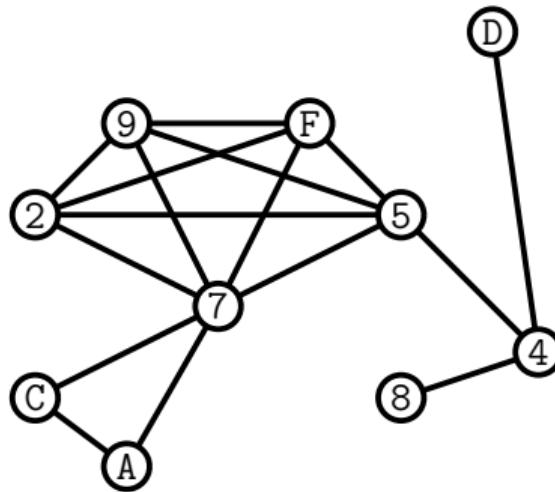
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL



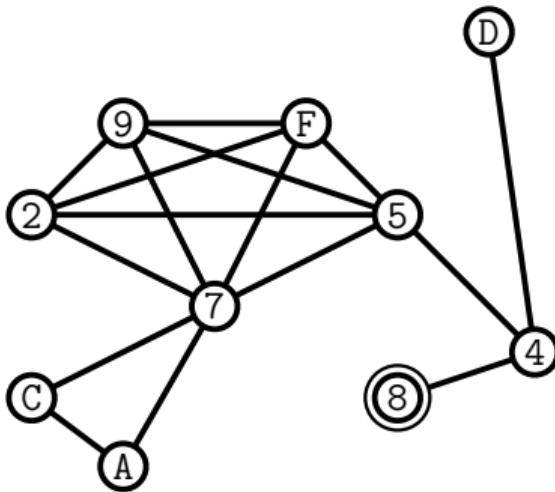
CONGEST



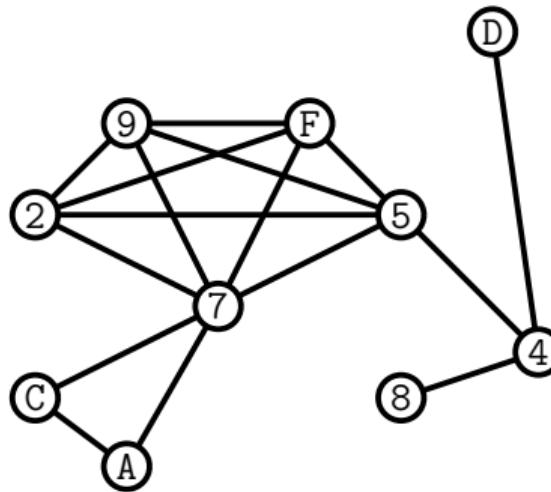
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL



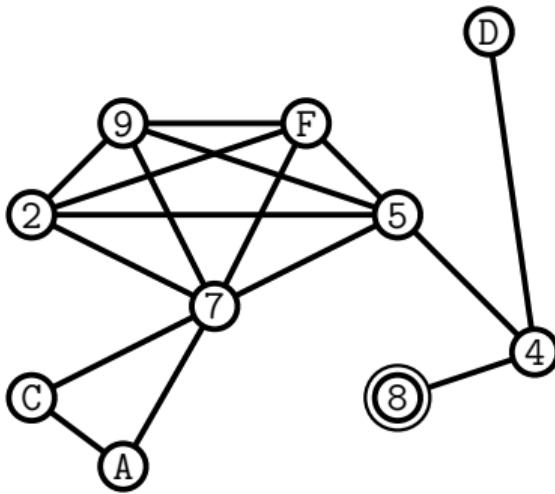
CONGEST



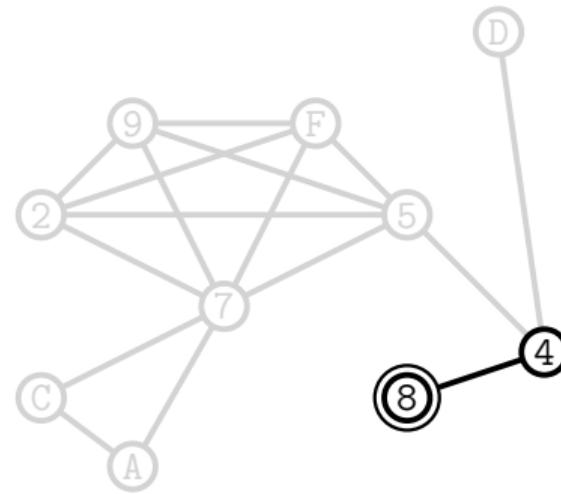
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL



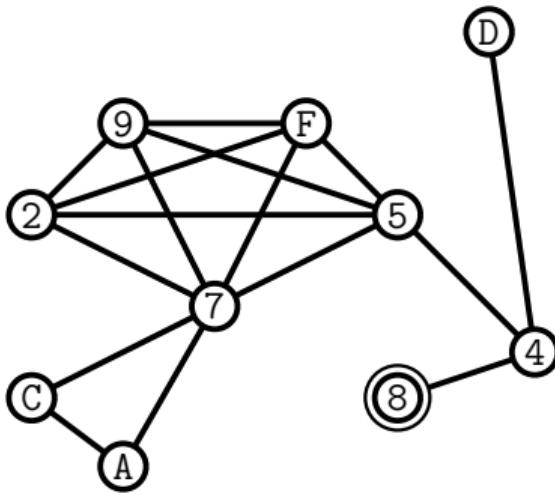
CONGEST



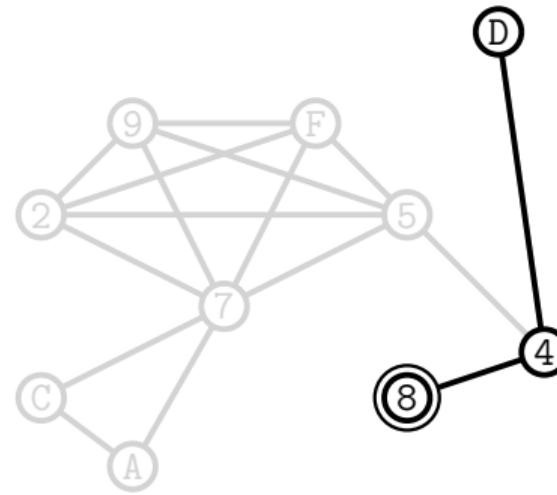
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL



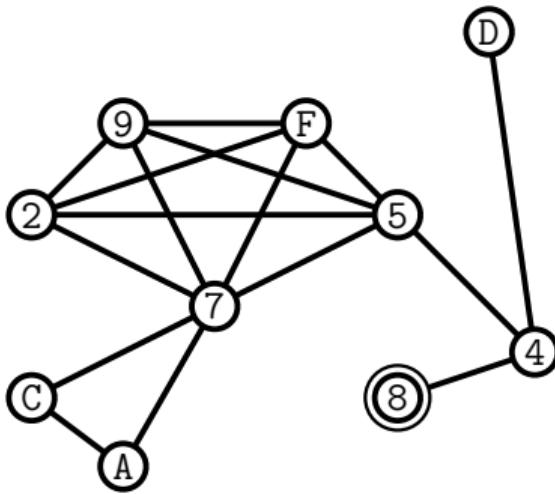
CONGEST



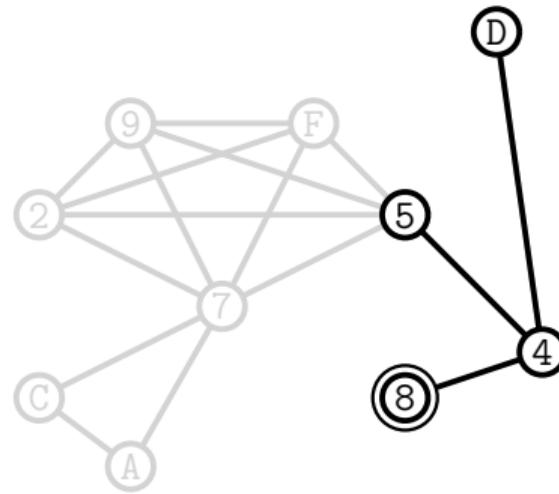
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL



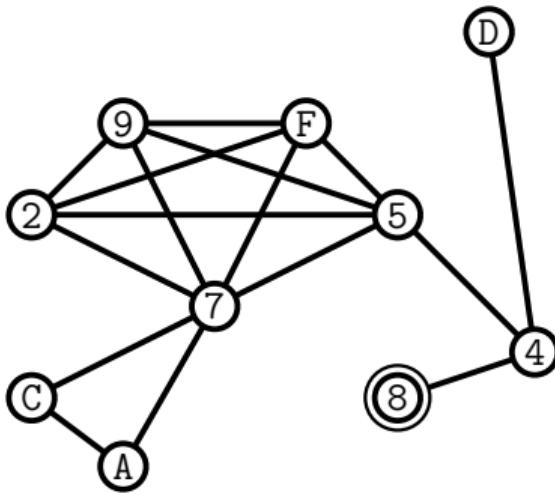
CONGEST



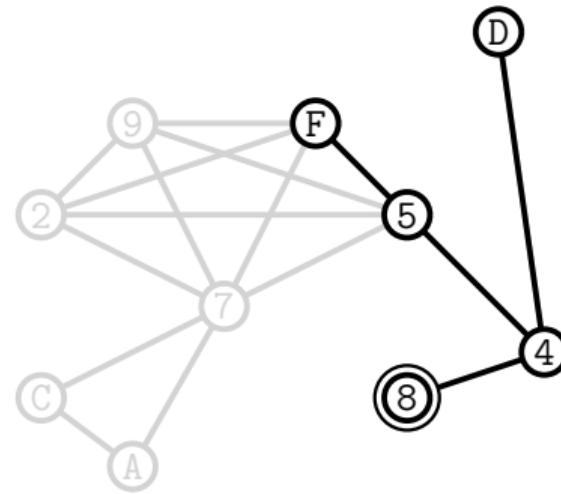
- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

LOCAL and CONGEST

LOCAL

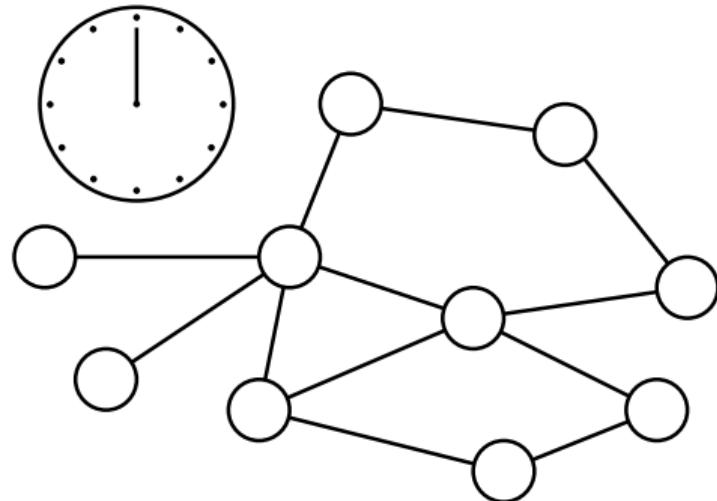


CONGEST



- Both: Synchronous message passing, graph models communication network.
- LOCAL: ∞ -sized messages.
- CONGEST: $O(\log n)$ -sized messages (n upper bound on number of nodes).

$\Delta + 1$ -Coloring problem



Goal: starting from an initially uncolored graph $G = (V, E)$, assign a color to each node s.t. adjacent nodes receive distinct colors.

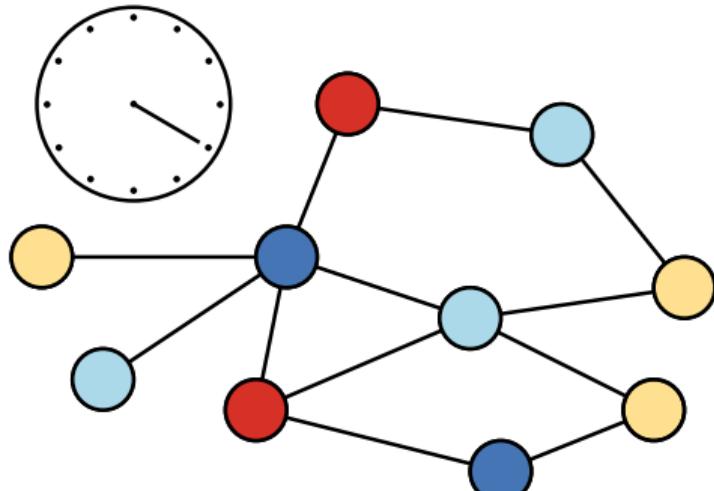
Formally: compute an assignment φ giving colors to the nodes

$$\varphi : V \rightarrow [\Delta + 1]$$

s.t. $\varphi(u) \neq \varphi(v)$ for each edge $uv \in E$.

Δ : maximum degree of the graph, given to the nodes.

$\Delta + 1$ -Coloring problem



Goal: starting from an initially uncolored graph $G = (V, E)$, assign a color to each node s.t. adjacent nodes receive distinct colors.

Formally: compute an assignment φ giving colors to the nodes

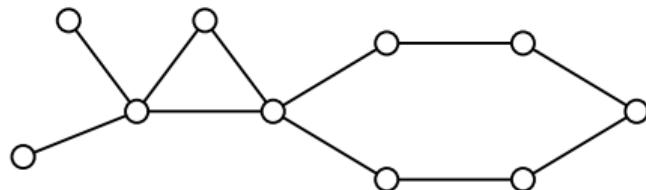
$$\varphi : V \rightarrow [\Delta + 1]$$

s.t. $\varphi(u) \neq \varphi(v)$ for each edge $uv \in E$.

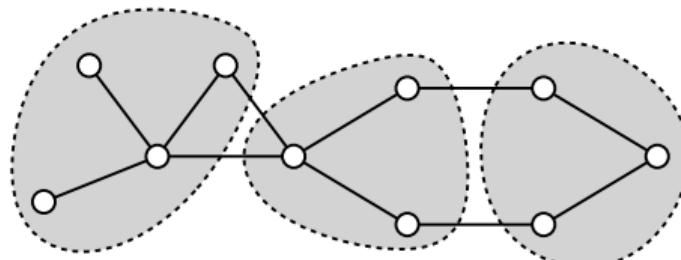
Δ : maximum degree of the graph, given to the nodes.

Embedded graphs

Power graphs

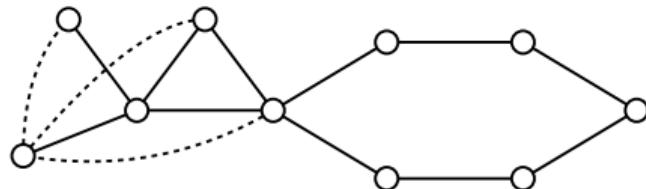


Cluster graphs

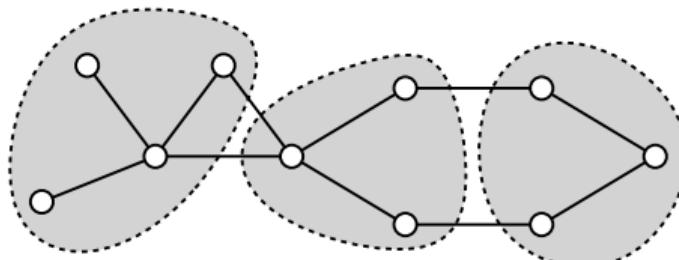


Embedded graphs

Power graphs

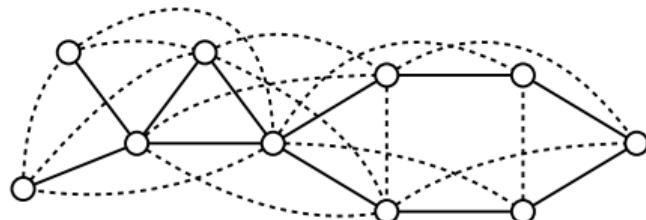


Cluster graphs

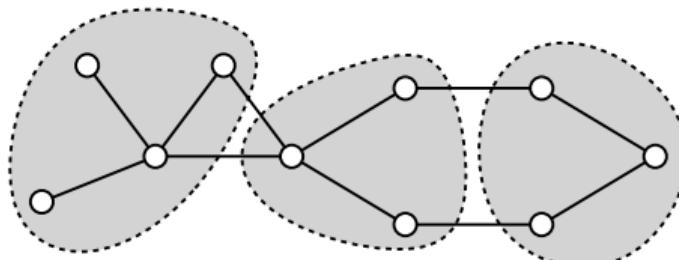


Embedded graphs

Power graphs

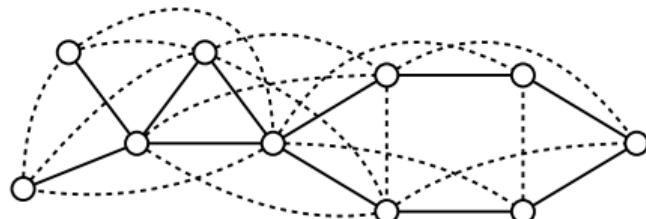


Cluster graphs

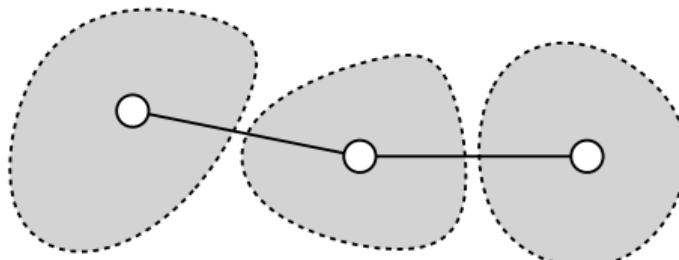


Embedded graphs

Power graphs

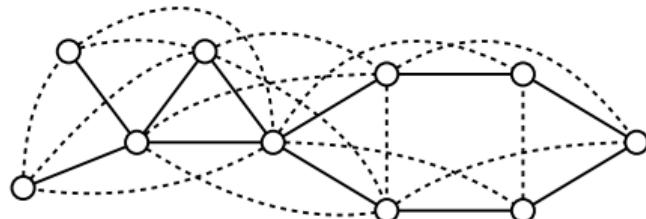


Cluster graphs

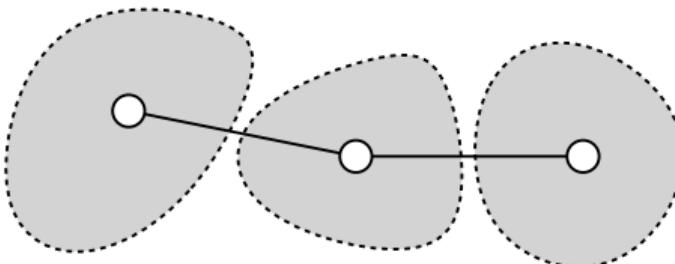


Embedded graphs

Power graphs



Cluster graphs

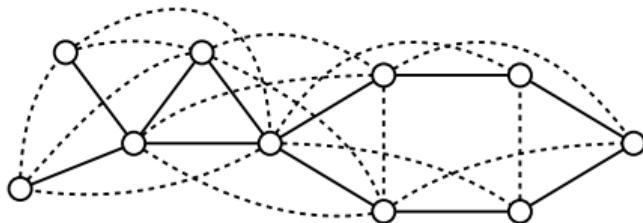


The graph to color H is not the communication graph G , but

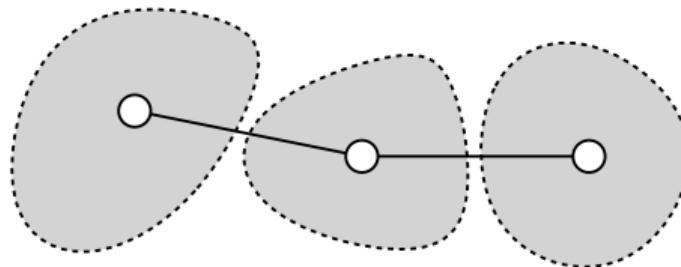
- for any node or edge of H , there is a machine in G “in charge” of that node or edge,
- the machines in charge of a node and its edges have a small depth spanning tree connecting them.

Embedded graphs

Power graphs



Cluster graphs



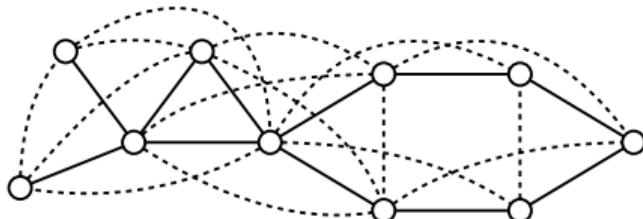
The graph to color H is not the communication graph G , but

- for any node or edge of H , there is a machine in G “in charge” of that node or edge,
- the machines in charge of a node and its edges have a small depth spanning tree connecting them.

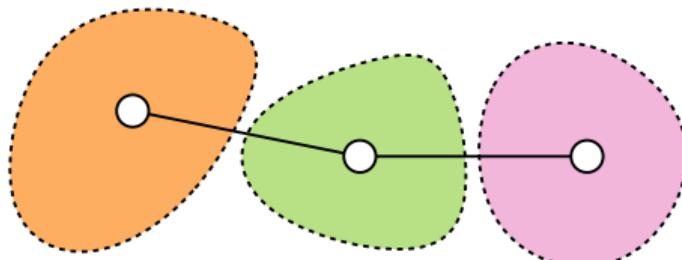
For the rest of the talk, keep cluster graphs as the working example.

Embedded graphs

Power graphs



Cluster graphs



The graph to color H is not the communication graph G , but

- for any node or edge of H , there is a machine in G “in charge” of that node or edge,
- the machines in charge of a node and its edges have a small depth spanning tree connecting them.

For the rest of the talk, keep cluster graphs as the working example.

Agenda for this talk

1. Explain state-of-the-art randomized coloring in LOCAL.
2. Wonder which parts break down in the embedded graph setting.
3. Talk about some nice ideas enabling almost-as-fast* randomized coloring algorithms for embedded graphs in CONGEST as in LOCAL.

Before we actually see some algorithms



Any questions at this point?

Randomized $\Delta + 1$ -coloring: some milestones and the state-of-the-art

Result	Source
$O(\log n)$	Luby [Lub86]
3-coloring rings requires $\Omega(\log^* n)$	Naor [Nao91]
Arguably simpler $O(\log n)$	Johansson [Joh99]
$O(\log \Delta + \sqrt{\log n})$	Schneider and Wattenhofer [SW10]
$O(\log \Delta + \log \log n + T_{\det, d11c}(\text{poly log } n, O(\log n)))$	Barenboim, Elkin, Pettie, and Schneider [BEPS16]
$O(\sqrt{\log \Delta} + T_{\text{rand}, d11c}(n, O(\log n) \cdot 2^{O(\sqrt{\log \Delta})}))$	Harris, Schneider, and Su [HSS18]
$T_{\det, \Delta+1}(n, \Delta) < T_{\text{rand}, \Delta+1}(2^{n^2}, \Delta)$	Chang, Kopelowitz, and Pettie [CKP19]
$O(\log^* \Delta + T_{\det, d11c}(\text{poly log } n, \text{poly log } n))$	Chang, Li, and Pettie [CLP20]

Randomized $\Delta + 1$ -coloring: some milestones and the state-of-the-art

Result	Source
$O(\log n)$	Luby [Lub86]
3-coloring rings requires $\Omega(\log^* n)$	Naor [Nao91]
Arguably simpler $O(\log n)$	Johansson [Joh99]
$O(\log \Delta + \sqrt{\log n})$	Schneider and Wattenhofer [SW10]
$O(\log \Delta + \text{poly log log } n)$	Barenboim, Elkin, Pettie, and Schneider [BEPS16]
$O(\sqrt{\log \Delta} + \text{poly log log } n)$	Harris, Schneider, and Su [HSS18]
$T_{\text{det}, \Delta+1}(n, \Delta) < T_{\text{rand}, \Delta+1}(2^{n^2}, \Delta)$	Chang, Kopelowitz, and Pettie [CKP19]
$\text{poly log log } n$, $O(\log^* n)$ if $\Delta > \log^c n$ (c a large enough universal constant)	Chang, Li, and Pettie [CLP20]

Dealing with congestion

Some previously mentionned results work directly in CONGEST [Lub86, Joh99, BEPS16].

First $\text{poly log log } n$ algorithm by Halldórsson, Kuhn, Maus, and Tonoyan [HKMT21]

First $O(\log^* n)$ algorithm for $\Delta > \log^c n$ by Halldórsson, N., and Tonoyan [HNT22]

Several works on distance-2 coloring [HKM20, HKMN20, FHN23], ultimately achieving $\text{poly log log } n$ complexity.

CONGEST with broadcast communication, complexity $\text{poly log log } n$ by Flin, Ghaffari, Halldórsson, Kuhn, and N. [FGH⁺23]

Deterministic $O(\Delta^4)$ -coloring distance-2 coloring in $O(\log \Delta + \log^* n)$, and more, by Barenboim and Goldenberg [BG24]

Dealing with congestion

Some previously mentionned results work directly in CONGEST [Lub86, Joh99, BEPS16].

First $\text{poly log log } n$ algorithm by Halldórsson, Kuhn, Maus, and Tonoyan [HKMT21]

First $O(\log^* n)$ algorithm for $\Delta > \log^c n$ by Halldórsson, N., and Tonoyan [HNT22]

Several works on distance-2 coloring [HKM20, HKMN20, FHN23], ultimately achieving $\text{poly log log } n$ complexity.

CONGEST with broadcast communication, complexity $\text{poly log log } n$ by Flin, Ghaffari, Halldórsson, Kuhn, and N. [FGH⁺23]

Deterministic $O(\Delta^4)$ -coloring distance-2 coloring in $O(\log \Delta + \log^* n)$, and more, by Barenboim and Goldenberg [BG24]

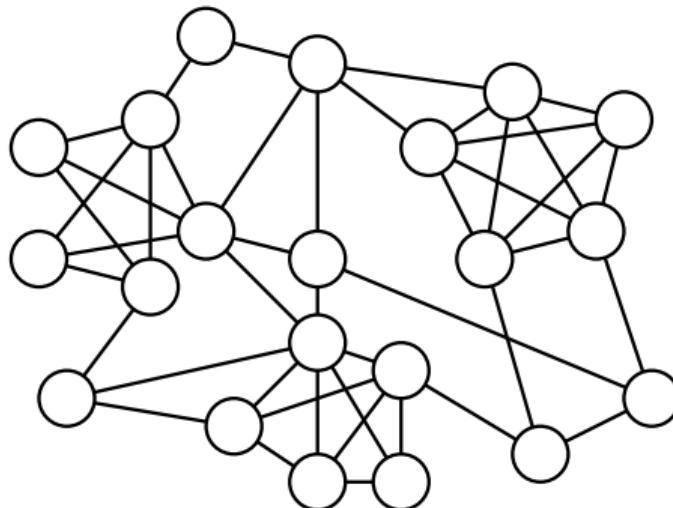
Main works for today: [FHN24, FHN25] achieving $\Delta + 1$ -coloring of virtual graphs (notably cluster graphs) in $\text{poly log log } n$ rounds, $O(\log^* n)$ when $\Delta > \log^c n$.

A state-of-the-art LOCAL algorithm

1. Compute an **Almost-Clique Decomposition**
2. Compute **Put-Aside Sets** in the densest almost-cliques
3. Generate some **Slack**
4. Color most nodes in almost-cliques by **Synchronized Color Trials**
5. Extend the coloring to all nodes except the Put-Aside Sets with **multi-color trials**
6. Extend the coloring to the Put-Aside Sets

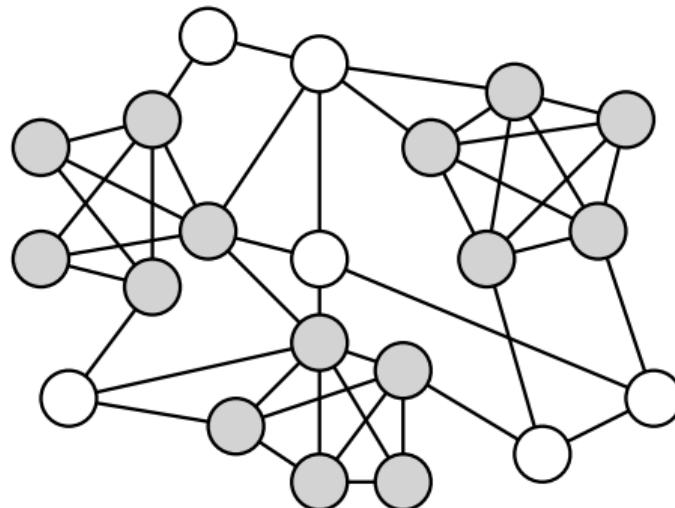
Step 1: Almost-clique decomposition

Partition nodes as **dense** or **sparse** according to the number of edges in their neighborhood.



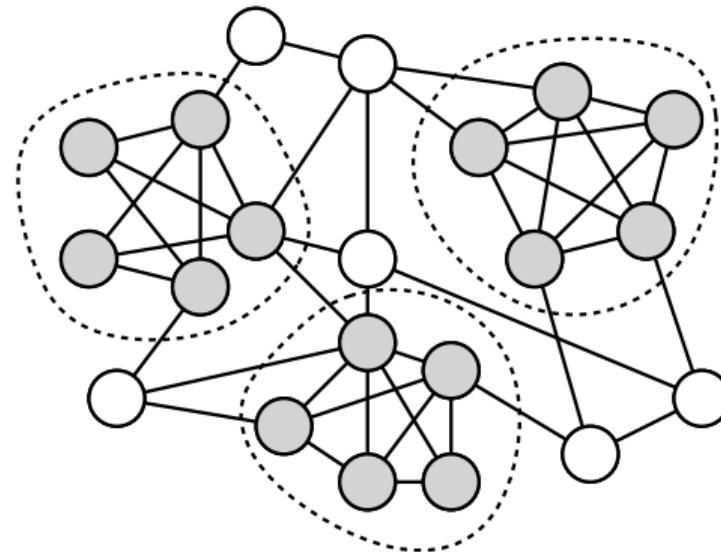
Step 1: Almost-clique decomposition

Partition nodes as **dense** or **sparse** according to the number of edges in their neighborhood.



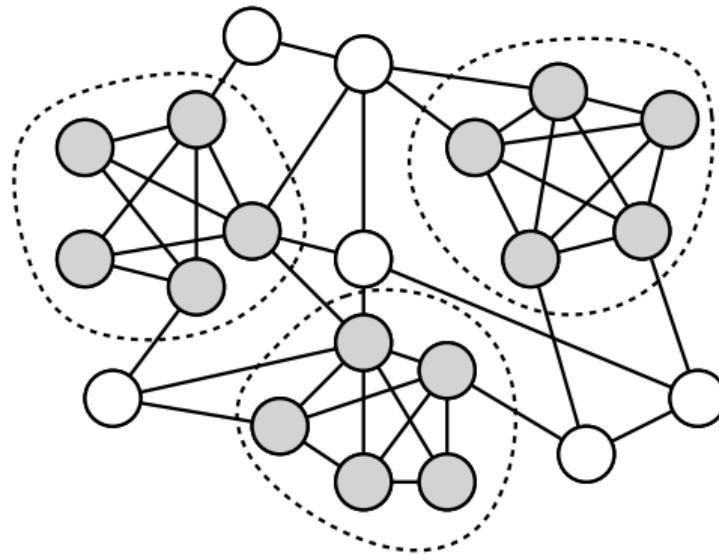
Step 1: Almost-clique decomposition

Partition nodes as **dense** or **sparse** according to the number of edges in their neighborhood.



Step 1: Almost-clique decomposition

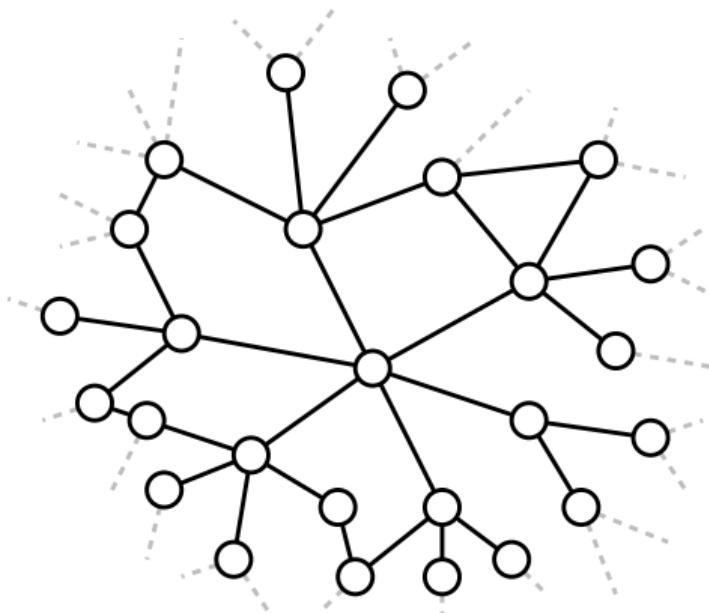
Partition nodes as **dense** or **sparse** according to the number of edges in their neighborhood.



Dense nodes form low-diameter subgraphs called, **almost-cliques**, where members pairwise share $> (1 - \varepsilon)\Delta$ neighbors.

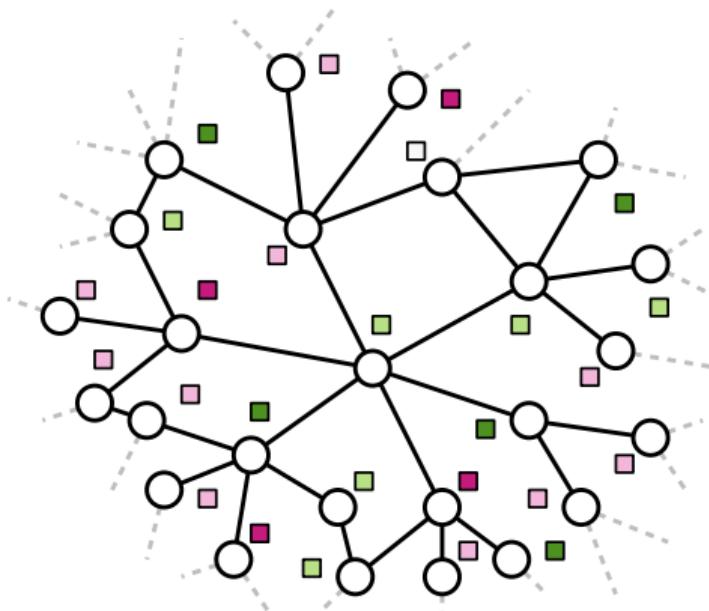
Step 3: Generate slack

Having nodes try a color at random results in *redundancies* (slack) in sparse parts.



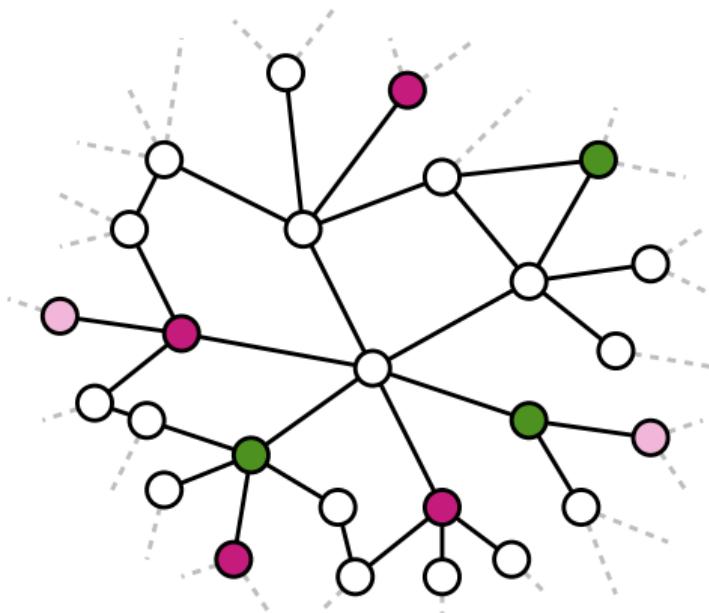
Step 3: Generate slack

Having nodes try a color at random results in *redundancies* (slack) in sparse parts.



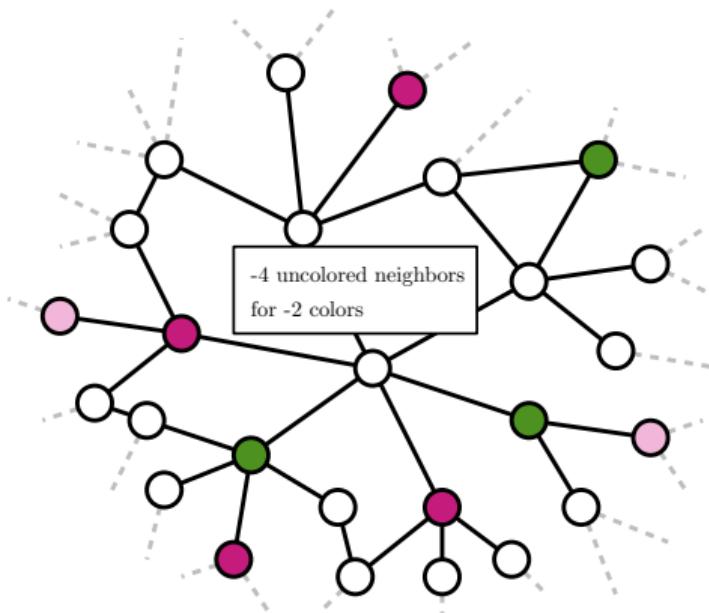
Step 3: Generate slack

Having nodes try a color at random results in *redundancies* (slack) in sparse parts.



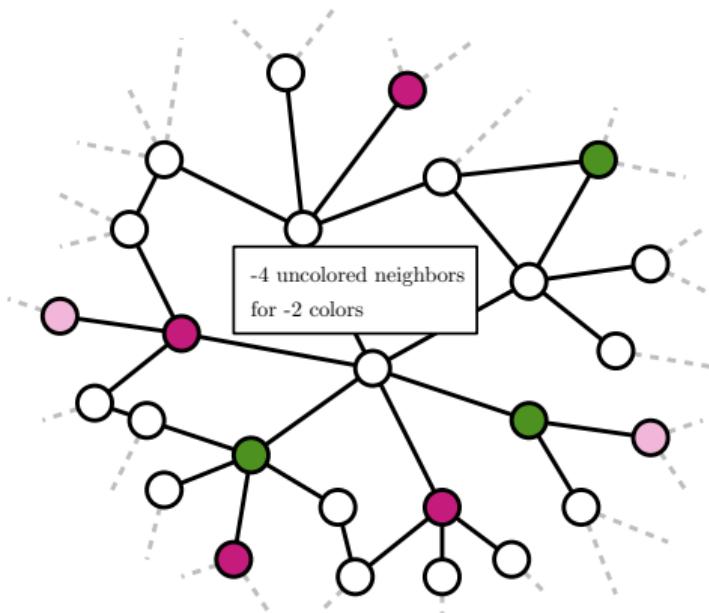
Step 3: Generate slack

Having nodes try a color at random results in *redundancies* (slack) in sparse parts.



Step 3: Generate slack

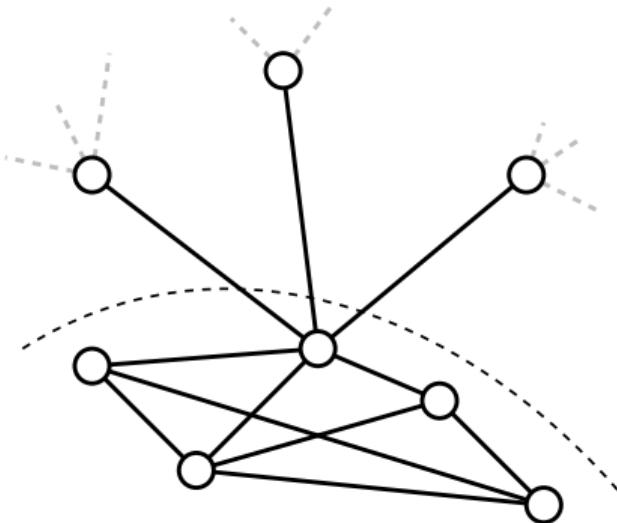
Having nodes try a color at random results in *redundancies* (slack) in sparse parts.



Also works for dense nodes with some neighbors outside their almost-clique.

Step 3: Generate slack

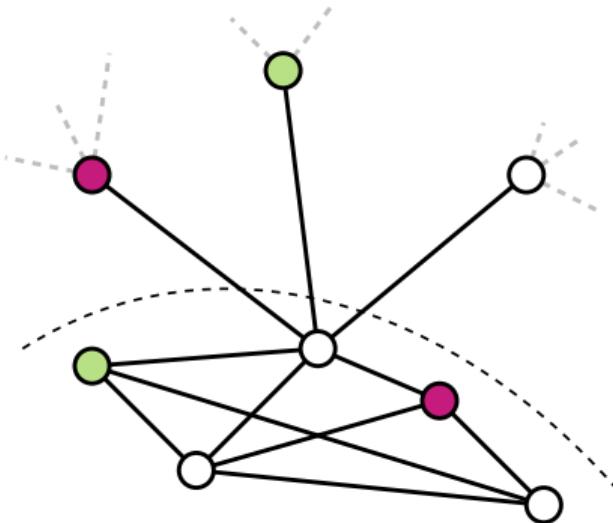
Having nodes try a color at random results in *redundancies* (slack) in sparse parts.



Also works for dense nodes with some neighbors outside their almost-clique.

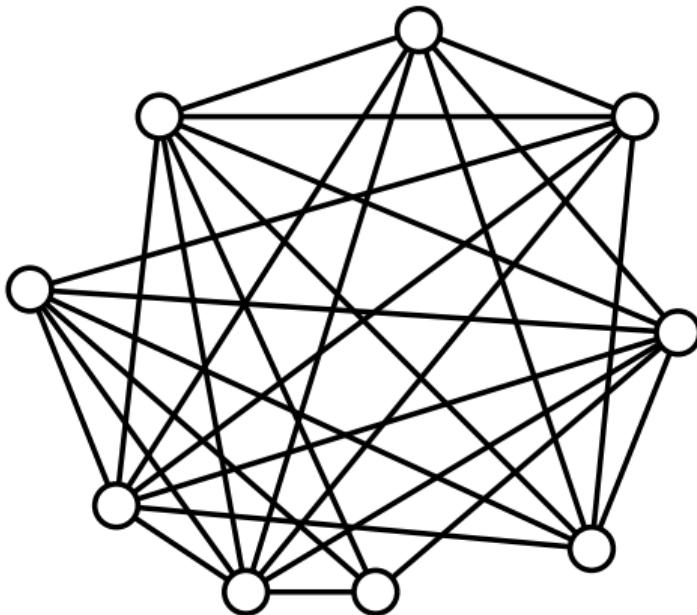
Step 3: Generate slack

Having nodes try a color at random results in *redundancies* (slack) in sparse parts.



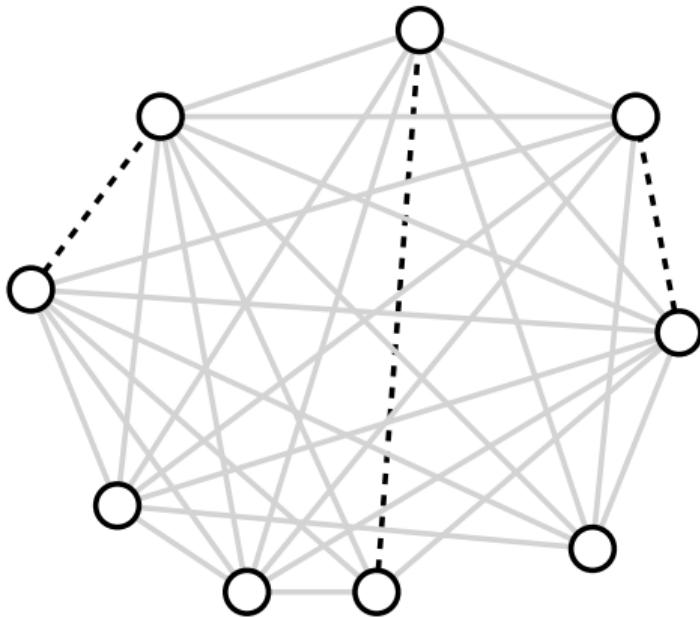
Also works for dense nodes with some neighbors outside their almost-clique.

Step 3 continued: generating slack by colorful matching



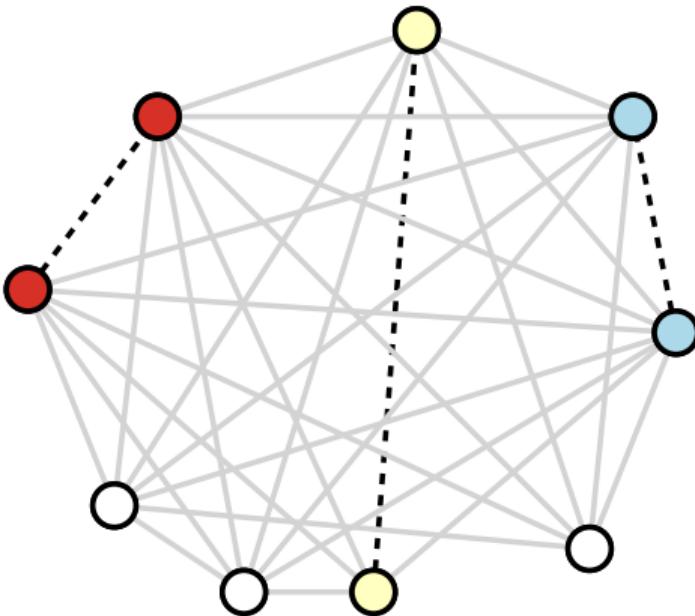
Find missing edges inside almost-cliques, try to assign equal colors to both ends.

Step 3 continued: generating slack by colorful matching



Find missing edges inside almost-cliques, try to assign equal colors to both ends.

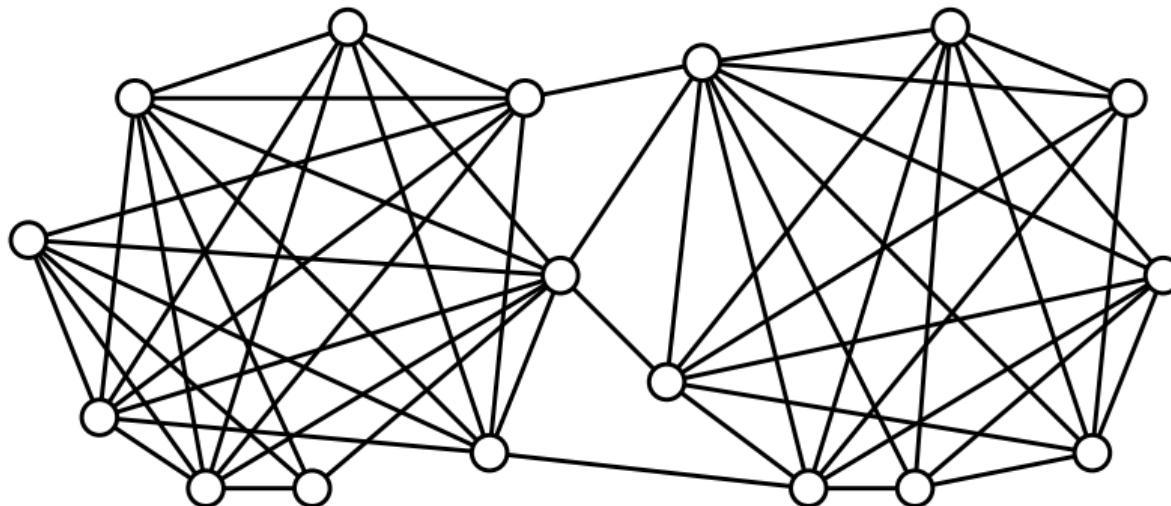
Step 3 continued: generating slack by colorful matching



Find missing edges inside almost-cliques, try to assign equal colors to both ends.

Step 2: Put-aside sets

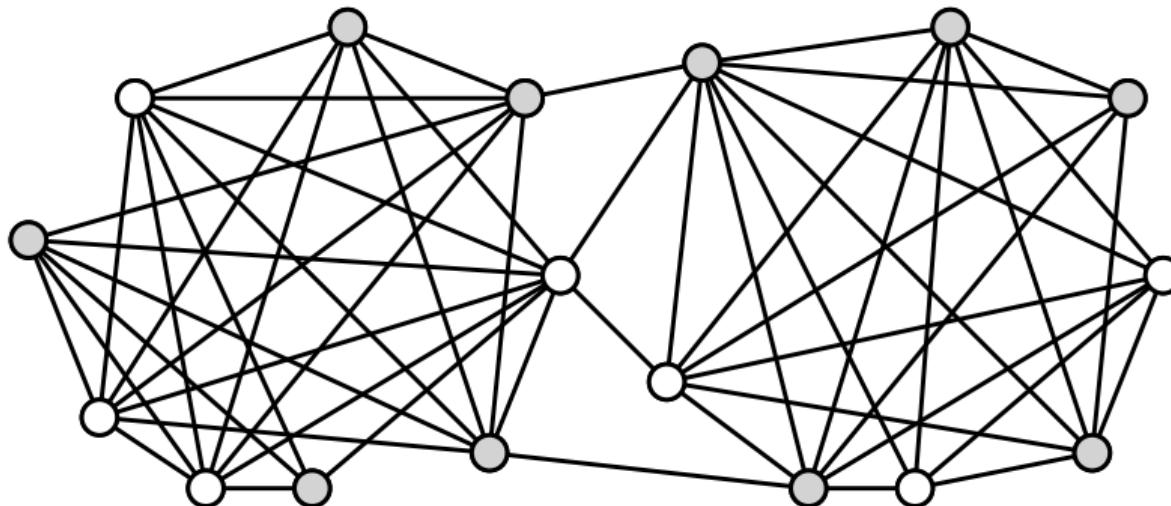
Used in almost-cliques with very few external edges and missing internal edges.



Nodes whose coloring is delayed until the very end. Provides flexibility when color redundancies are not an option.

Step 2: Put-aside sets

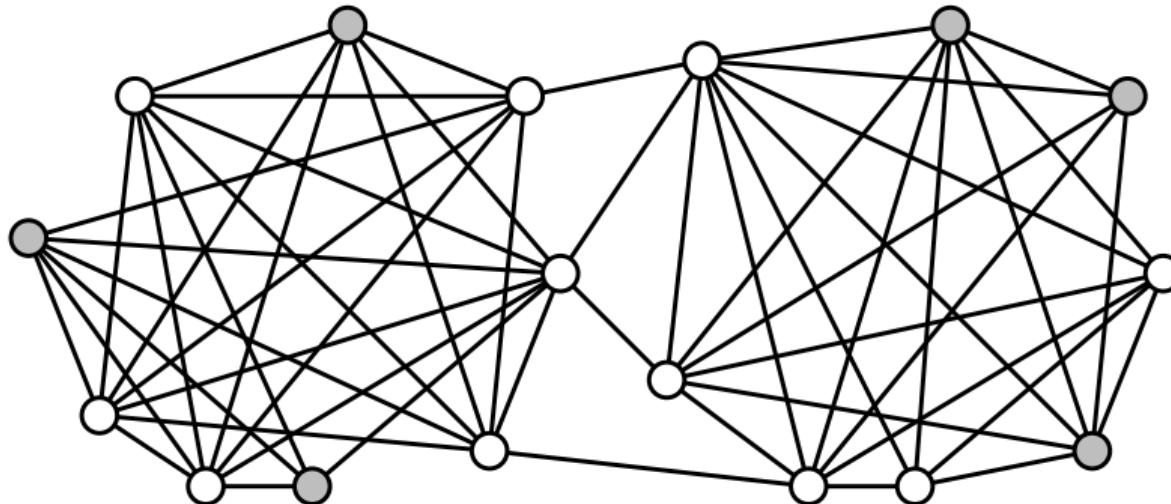
Used in almost-cliques with very few external edges and missing internal edges.



Nodes whose coloring is delayed until the very end. Provides flexibility when color redundancies are not an option.

Step 2: Put-aside sets

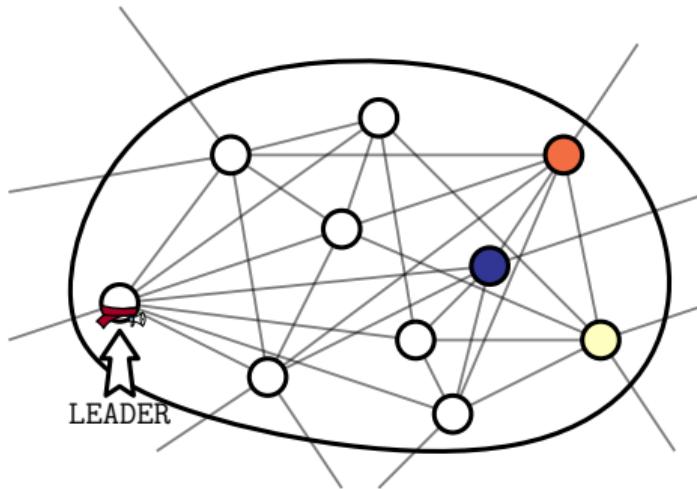
Used in almost-cliques with very few external edges and missing internal edges.



Nodes whose coloring is delayed until the very end. Provides flexibility when color redundancies are not an option.

Step 4: Synchronized Color Trials

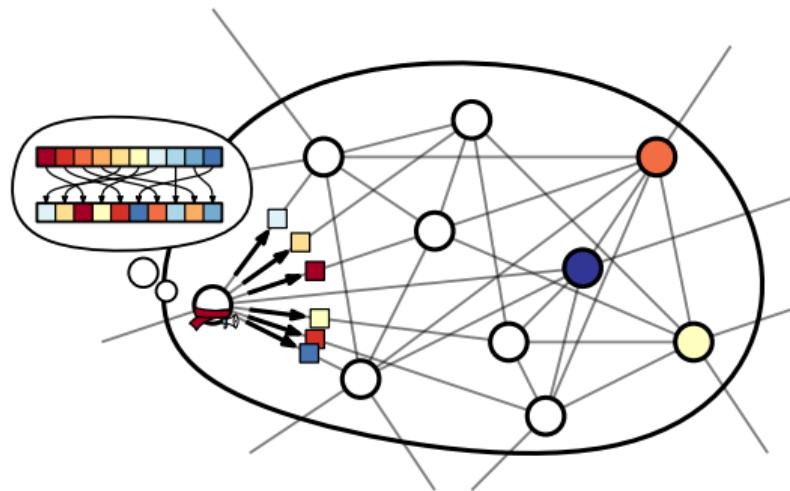
Dense nodes are mostly in competition with nodes in their almost-cliques.



Coordinating colors tried in almost-cliques leaves few nodes uncolored.

Step 4: Synchronized Color Trials

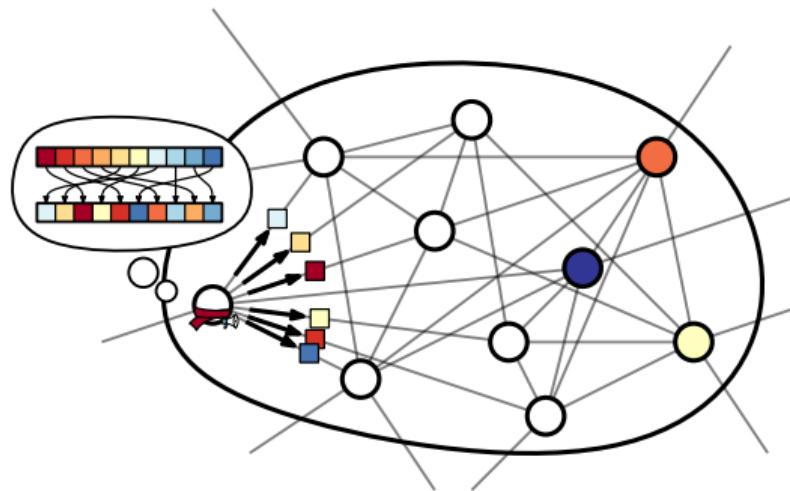
Dense nodes are mostly in competition with nodes in their almost-cliques.



Coordinating colors tried in almost-cliques leaves few nodes uncolored.

Step 4: Synchronized Color Trials

Dense nodes are mostly in competition with nodes in their almost-cliques.



Coordinating colors tried in almost-cliques leaves few nodes uncolored.

(putting a small thing under the rug here, feel free to ask about it later)

Step 5: Coloring almost everything

Everything until now had one goal:

that uncolored nodes have $\times(1 + \Omega(1))$ more available colors than (active) uncolored neighbors.

Technique of Schneider and Wattenhofer [SW10] for coloring in $O(\log^* n)$ rounds in that situation: try increasing amounts of colors (up to $\Theta(\log n)$) as the degree of nodes goes down.

Intuition: suppose every node has $\geq C$ available colors, and $\leq d$ neighbors. Let $x = C/(2d)$. If every node tries x colors at random, it is colored w.p. $\geq 1 - 2^{-x}$.

Step 6: Coloring put-aside sets

But we left a few nodes uncolored...

Step 6: Coloring put-aside sets

But we left a few nodes uncolored...

...but they are organized into low-diameter sets that can be colored independently from one another. $O(1)$ extra rounds to finish.

A state-of-the-art LOCAL algorithm (restated)

1. Compute an **Almost-Clique Decomposition**
2. Compute **Put-Aside Sets** in the densest almost-cliques
3. Generate some **Slack**
4. Color most nodes in almost-cliques by **Synchronized Color Trials**
5. Extend the coloring to all nodes except the Put-Aside Sets with **multi-color trials**
6. Extend the coloring to the Put-Aside Sets

A state-of-the-art LOCAL algorithm (restated)

1. Compute an **Almost-Clique Decomposition**
2. Compute **Put-Aside Sets** in the densest almost-cliques
3. Generate some **Slack**
4. Color most nodes in almost-cliques by **Synchronized Color Trials**
5. Extend the coloring to all nodes except the Put-Aside Sets with **multi-color trials**
6. Extend the coloring to the Put-Aside Sets

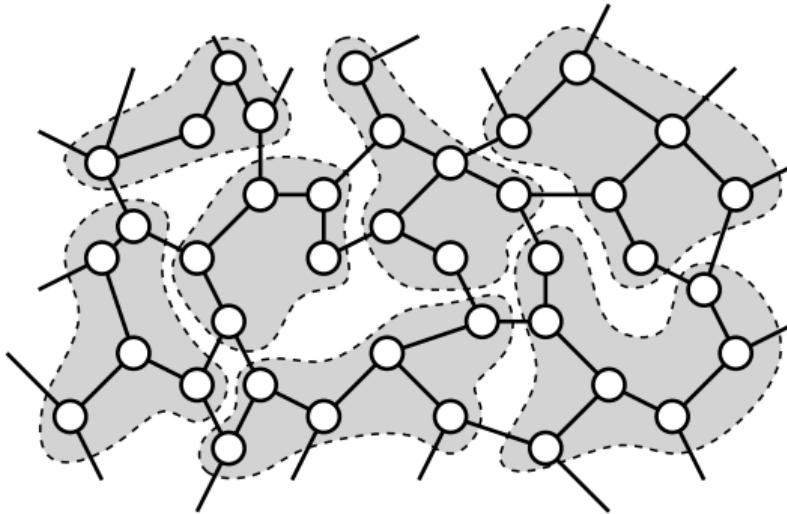
Time for CONGEST and cluster graphs!

Before adding congestion



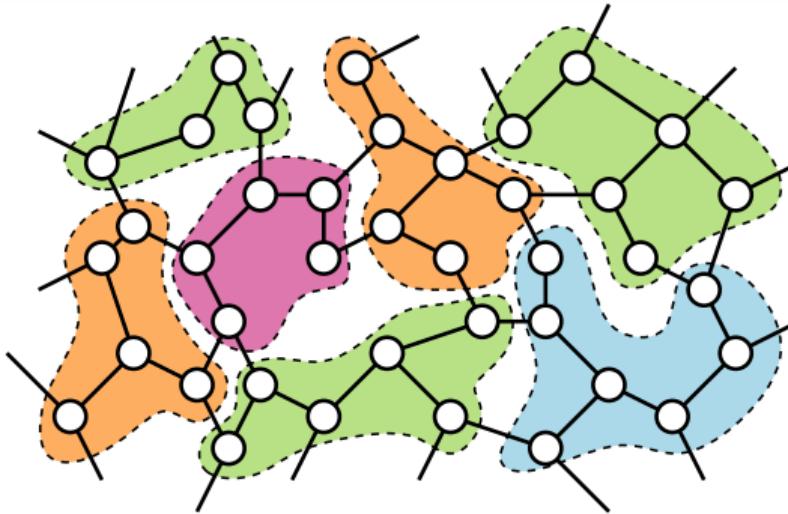
Questions on this LOCAL algorithm?

Recalling our goal



- Nodes are partitioned into constant-diameter clusters.
- Adjacent clusters must receive distinct colors.
- Edges can only carry $O(\log n)$ bit messages
- The maximum degree of the cluster graph, Δ , is given. The degree is **not** with multiplicity.

Recalling our goal



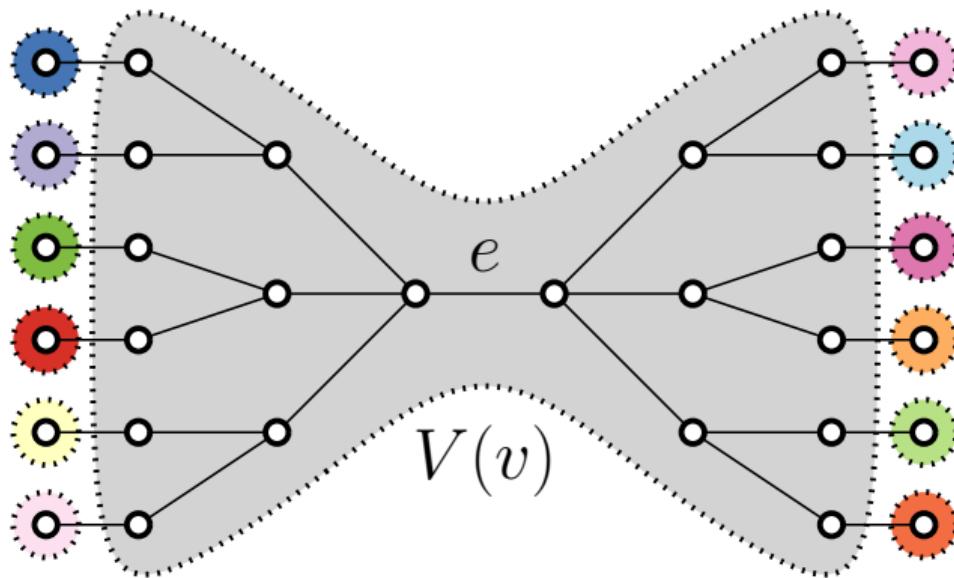
- Nodes are partitioned into constant-diameter clusters.
- Adjacent clusters must receive distinct colors.
- Edges can only carry $O(\log n)$ bit messages
- The maximum degree of the cluster graph, Δ , is given. The degree is **not** with multiplicity.

Group exercise

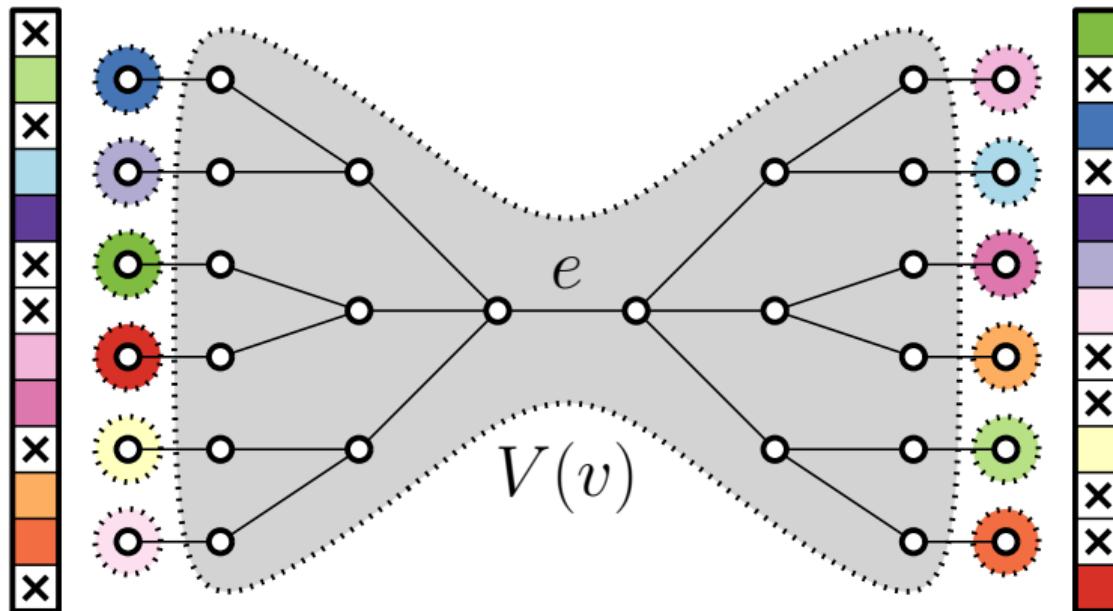
1. Compute an **Almost-Clique Decomposition**
2. Compute **Put-Aside Sets** in the densest almost-cliques
3. Generate some **Slack**
4. Color most nodes in almost-cliques by **Synchronized Color Trials**
5. Extend the coloring to all nodes except the Put-Aside Sets with **multi-color trials**
6. Extend the coloring to the Put-Aside Sets

What could go wrong in CONGEST, especially with cluster graphs?

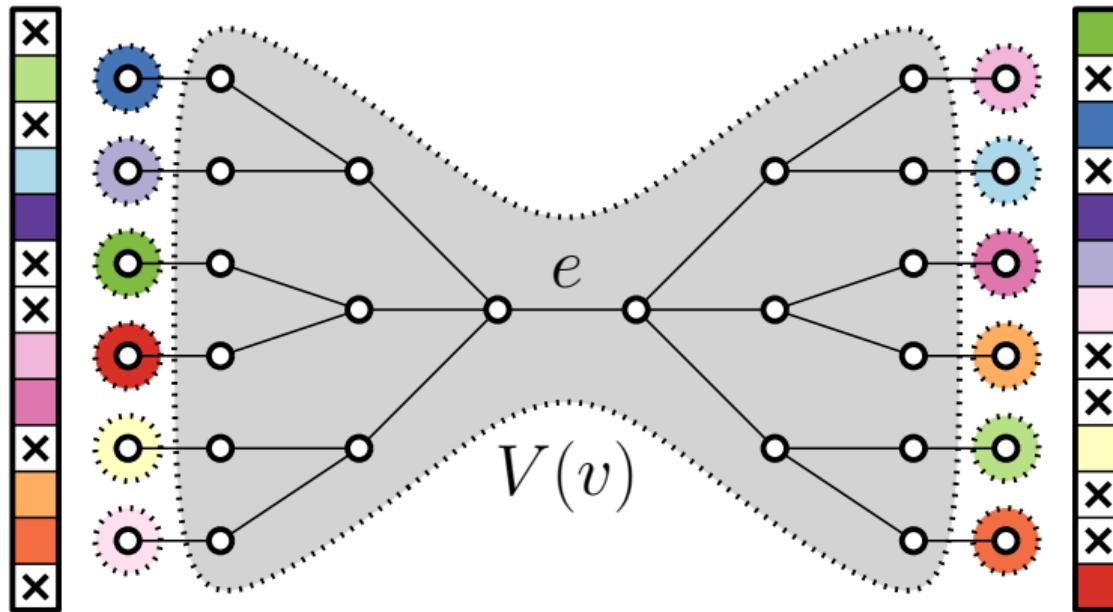
Issue 1: no access to neighbor's colors



Issue 1: no access to neighbor's colors

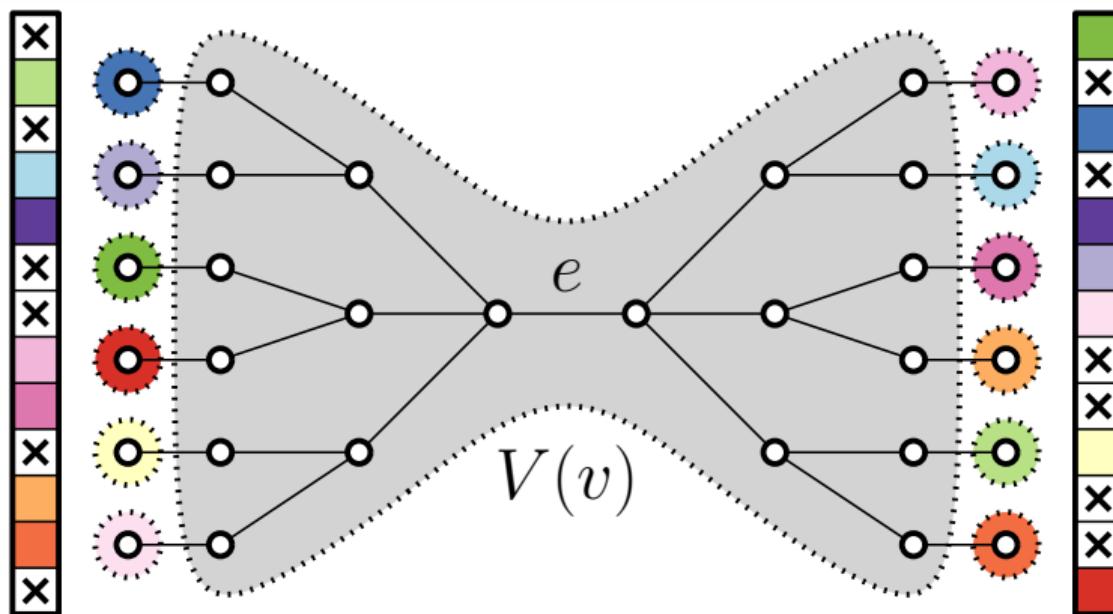


Issue 1: no access to neighbor's colors



“Trying a random **available** color” can be costly.

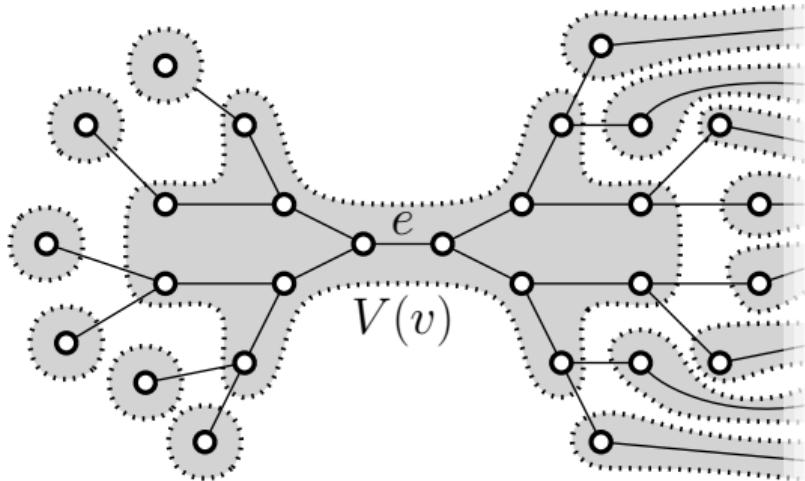
Issue 1: no access to neighbor's colors



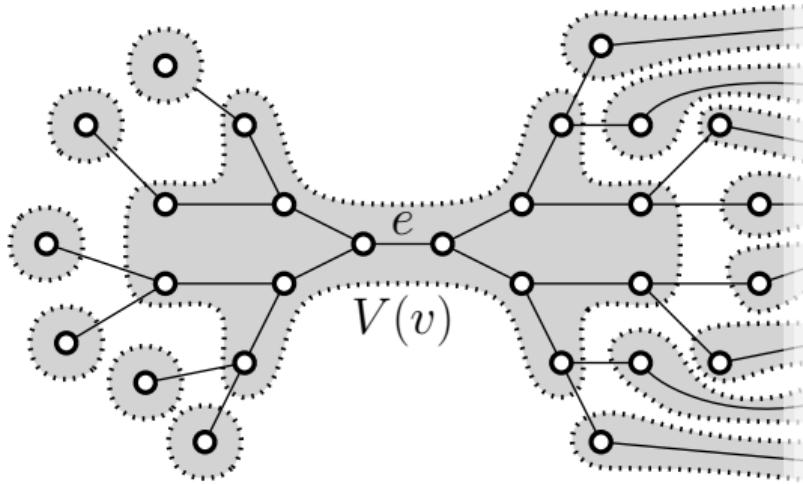
“Trying a random **available** color” can be costly.

Redeeming fact: “trying” a color is still fine.

Issue 2: no access to degree

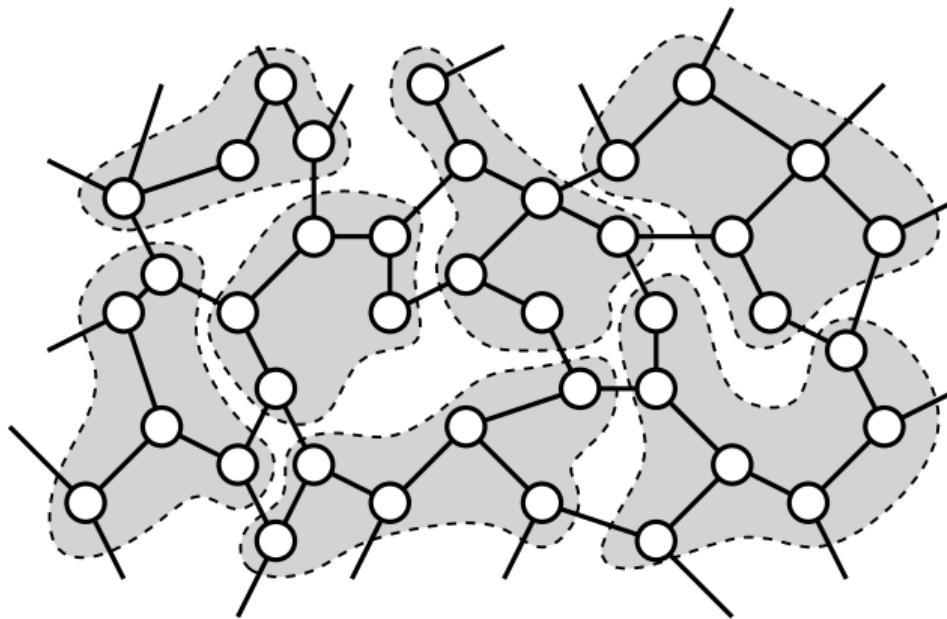


Issue 2: no access to degree

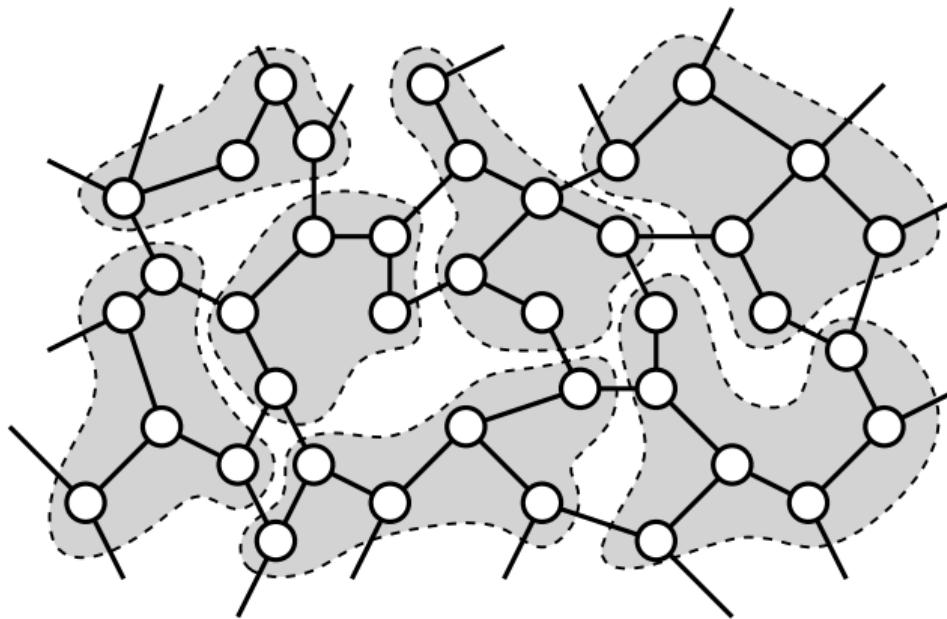


Redeeming fact: we are given Δ , and will be able to **estimate** the degree using randomness.

Issue 3: no access to neighborhood similarities

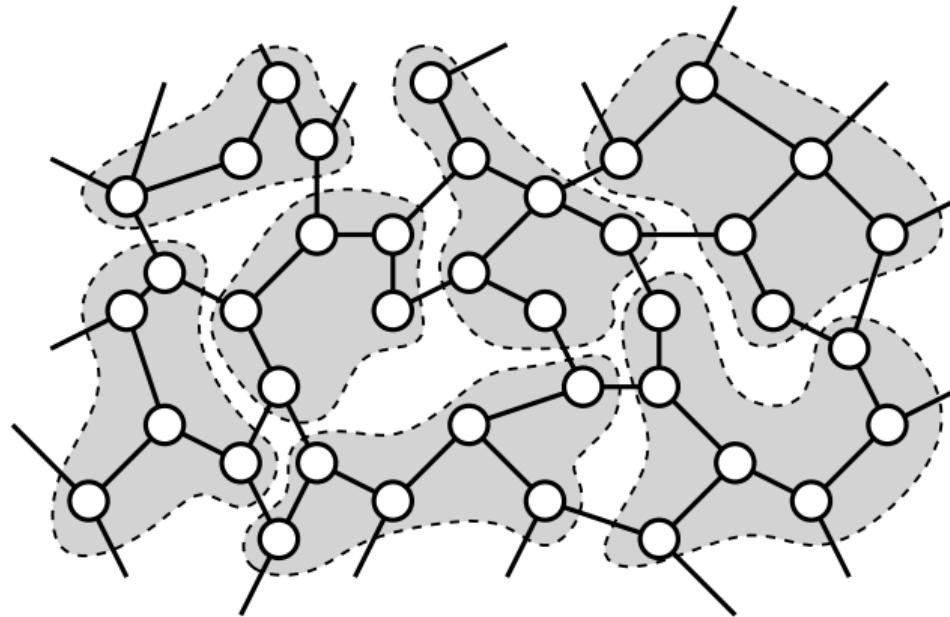


Issue 3: no access to neighborhood similarities

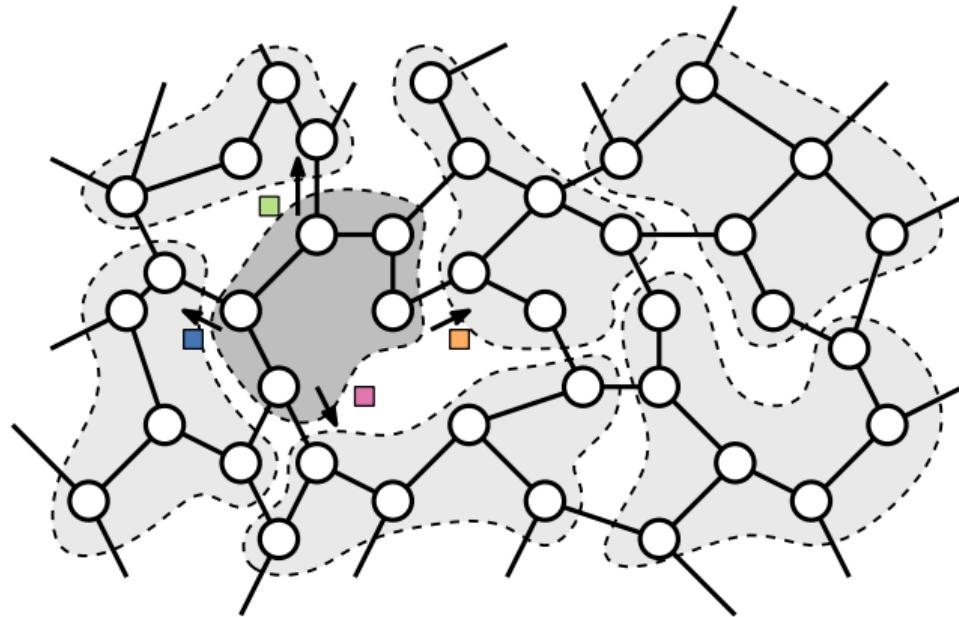


Describing a neighborhood is $\Theta(\Delta \log n)$ bits. Comparing neighborhoods might require channeling this much information through few links.

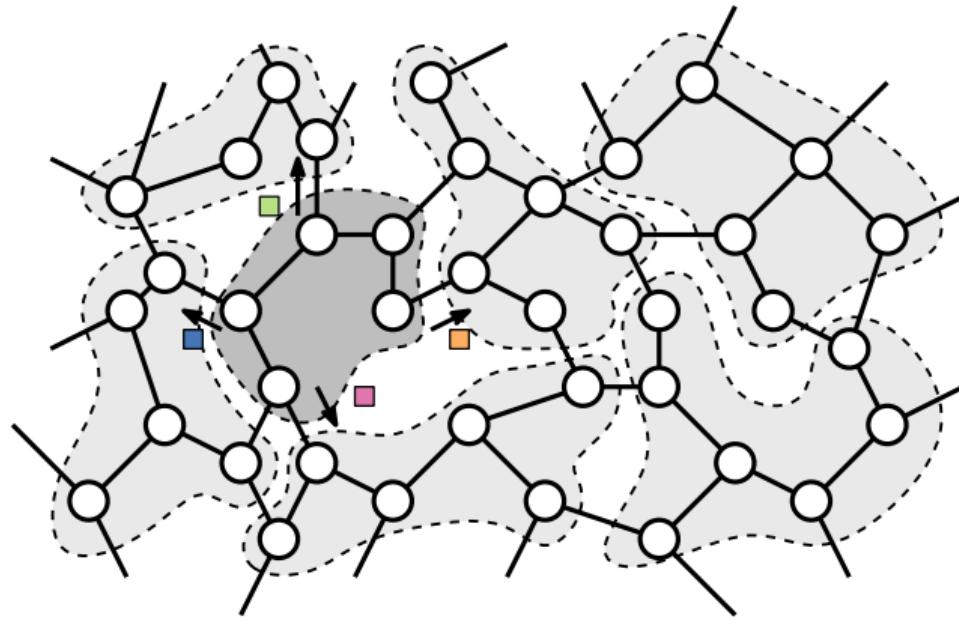
Issue 4: can't centralize decisions at a leader (SCT)



Issue 4: can't centralize decisions at a leader (SCT)

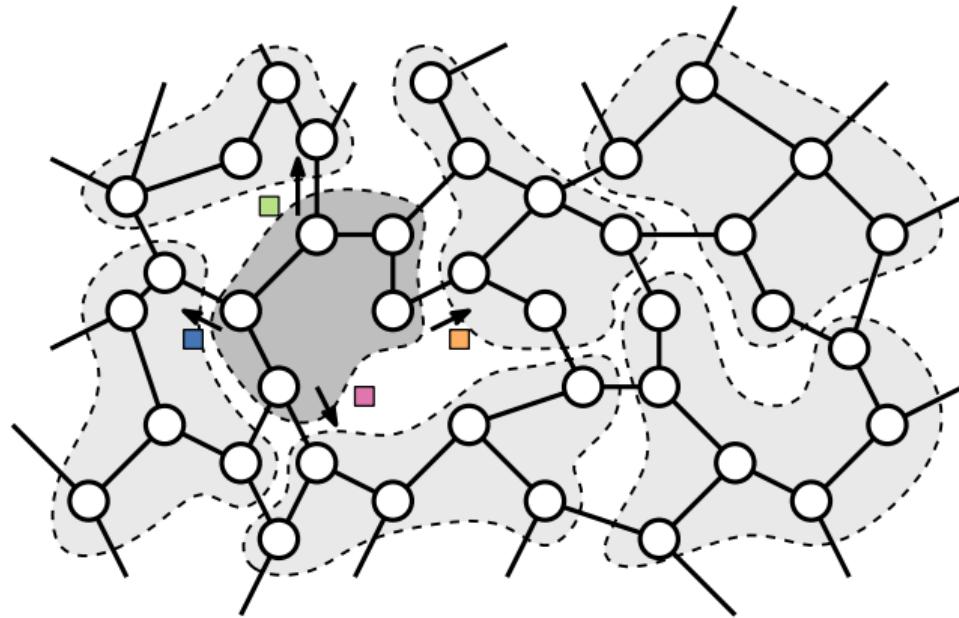


Issue 4: can't centralize decisions at a leader (SCT)



Up to $\Theta(\Delta \log \Delta)$ bits to send, which might need to go through very few links.

Issue 4: can't centralize decisions at a leader (SCT)



Up to $\Theta(\Delta \log \Delta)$ bits to send, which might need to go through very few links.
Routing issue: giving color to the right recipient?

Issue 5: can't send multiple colors

A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

Issue 5: can't send multiple colors

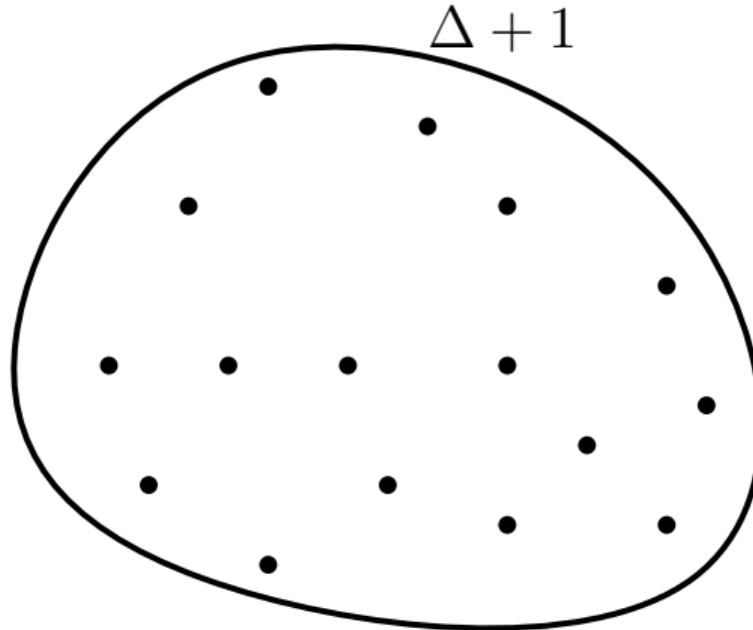
A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

Fix known from earlier work: pseudorandom samples [HN23, HNT22]

Issue 5: can't send multiple colors

A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

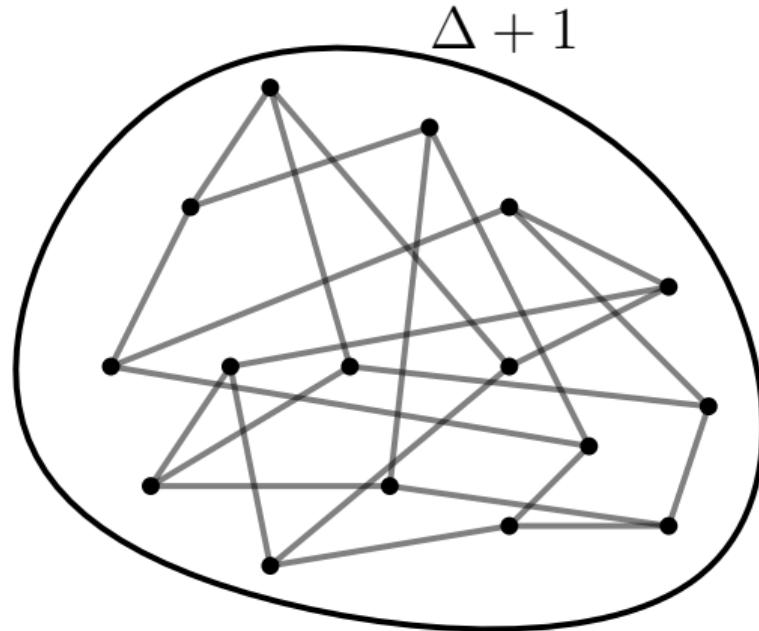
Fix known from earlier work: pseudorandom samples [HN23, HNT22]



Issue 5: can't send multiple colors

A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

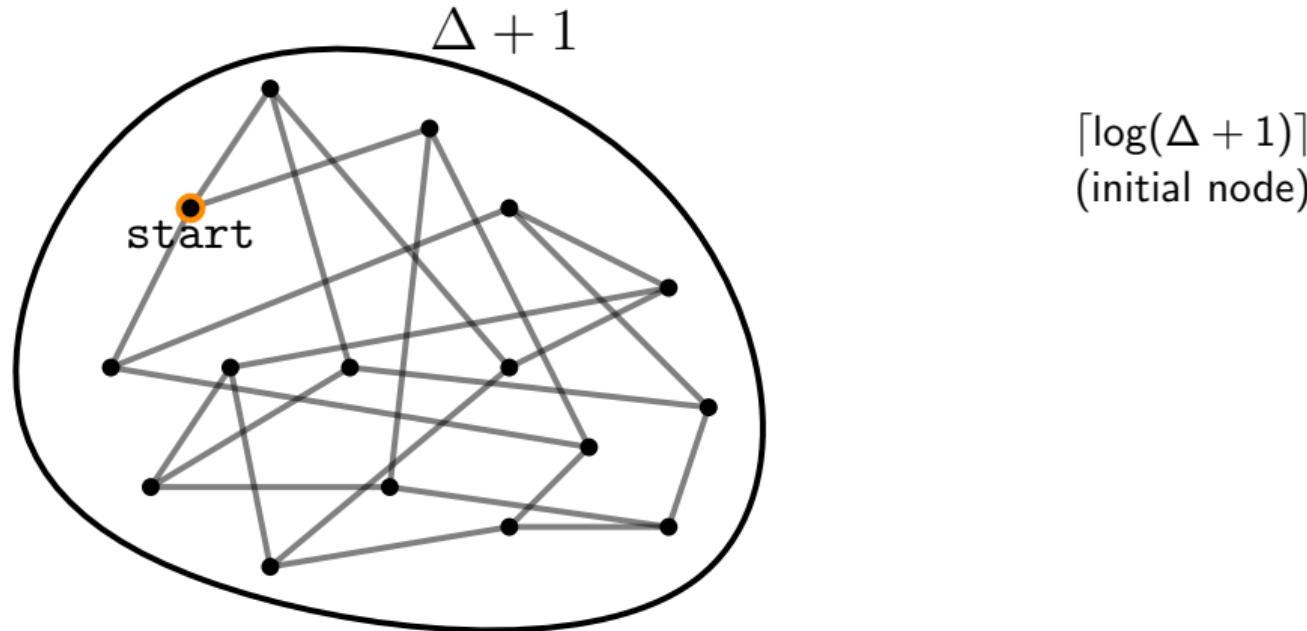
Fix known from earlier work: pseudorandom samples [HN23, HNT22]



Issue 5: can't send multiple colors

A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

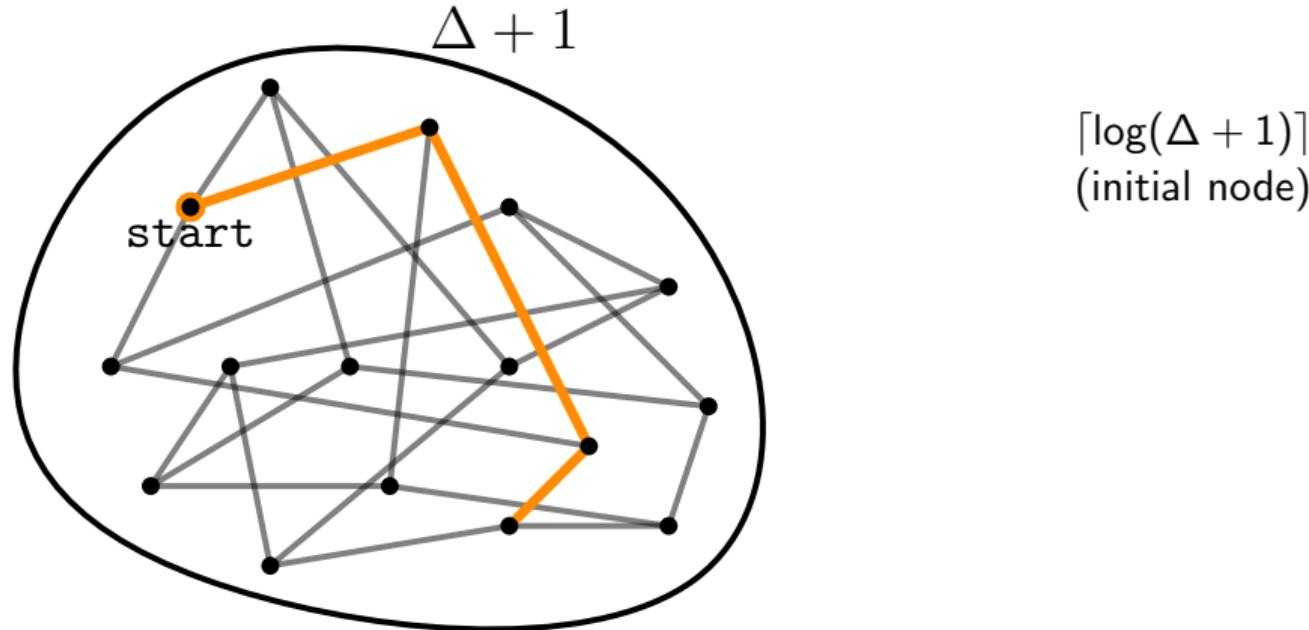
Fix known from earlier work: pseudorandom samples [HN23, HNT22]



Issue 5: can't send multiple colors

A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

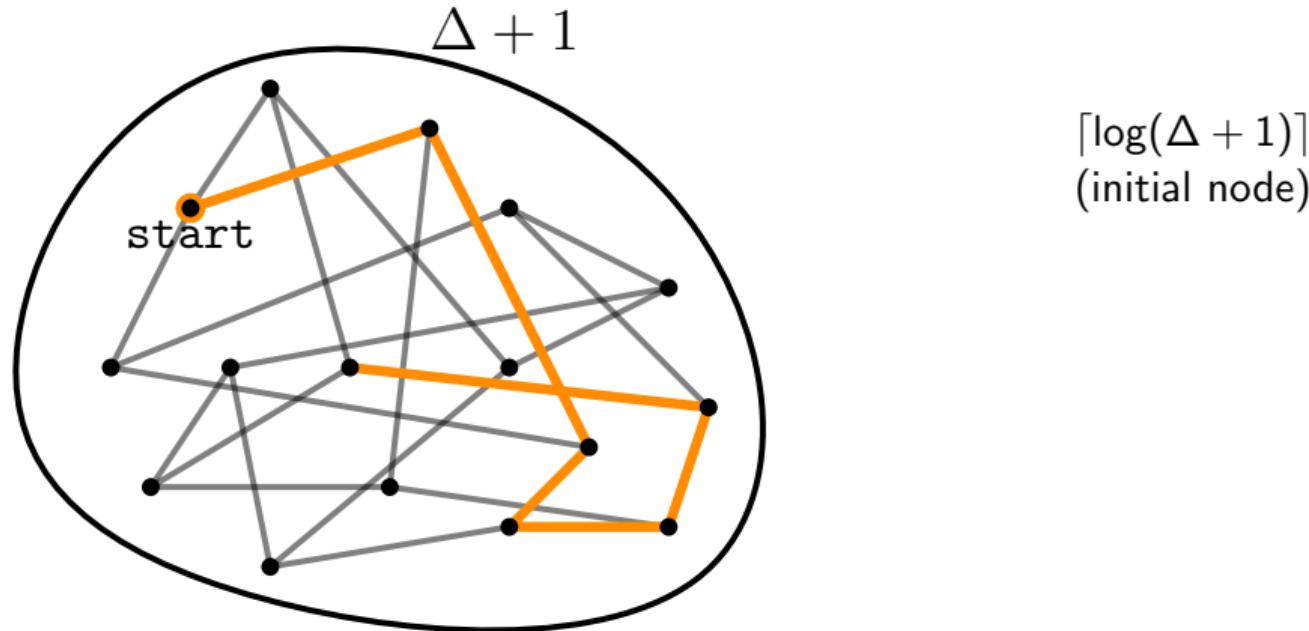
Fix known from earlier work: pseudorandom samples [HN23, HNT22]



Issue 5: can't send multiple colors

A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

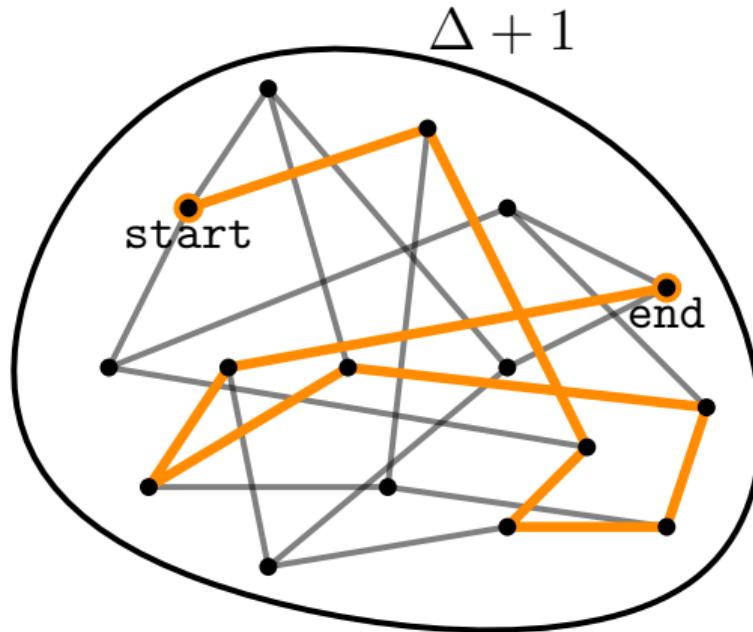
Fix known from earlier work: pseudorandom samples [HN23, HNT22]



Issue 5: can't send multiple colors

A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

Fix known from earlier work: pseudorandom samples [HN23, HNT22]



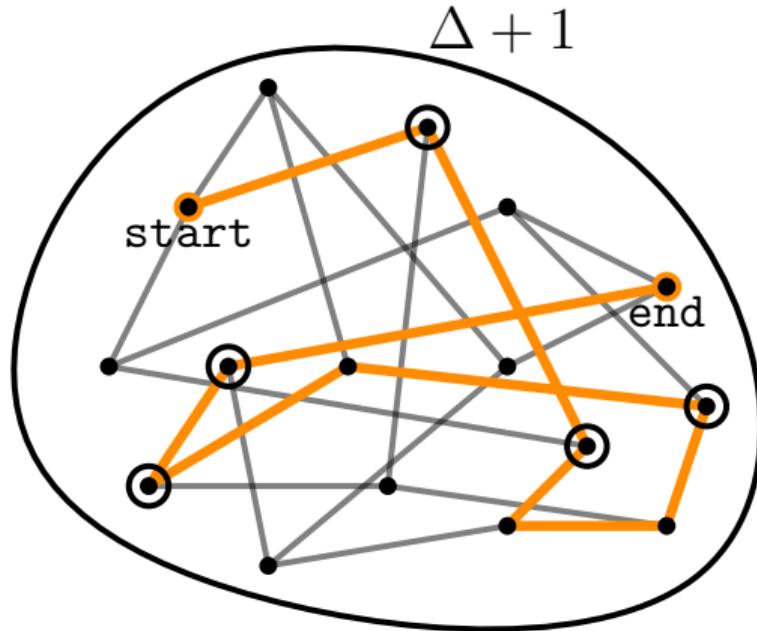
$\lceil \log(\Delta + 1) \rceil$
(initial node)

$+ \Theta(\log n)$
($\Theta(\log n)$ -step random walk)

Issue 5: can't send multiple colors

A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

Fix known from earlier work: pseudorandom samples [HN23, HNT22]



$\lceil \log(\Delta + 1) \rceil$
(initial node)

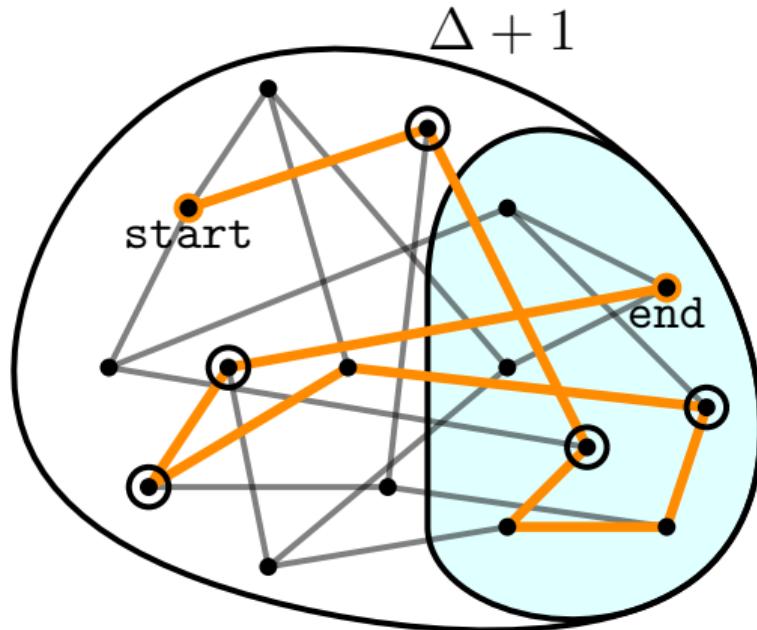
$+ \Theta(\log n)$
($\Theta(\log n)$ -step random walk)

$+ \Theta(\log n)$
(taking a subset)

Issue 5: can't send multiple colors

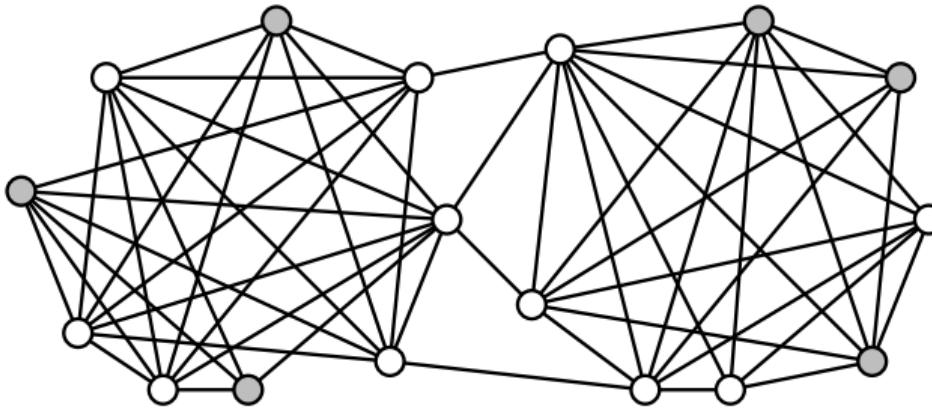
A set of $\Theta(\log n)$ colors between 1 and $\Delta + 1$ takes $\Theta(\log n \cdot \log \Delta)$ bits to represent.

Fix known from earlier work: pseudorandom samples [HN23, HNT22]



$\lceil \log(\Delta + 1) \rceil$
(initial node)
 $+ \Theta(\log n)$
($\Theta(\log n)$ -step random walk)
 $+ \Theta(\log n)$
(taking a subset)

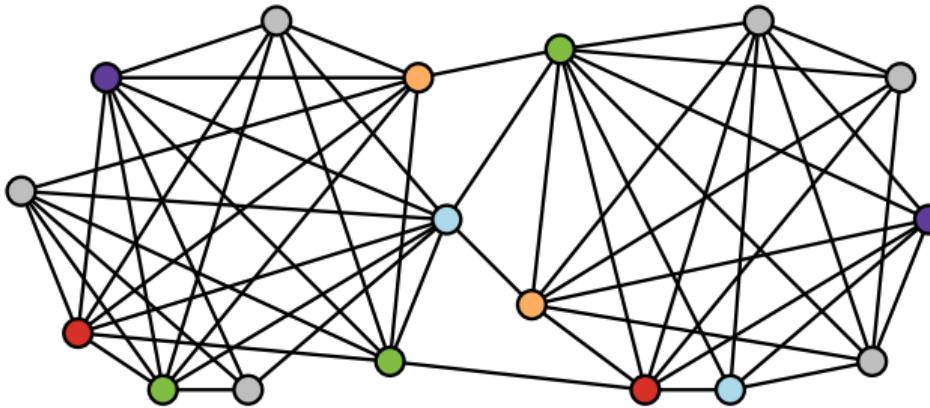
Issue 6: can't color put-aside sets at the end



Two reasons:

- no knowledge of the colors used by their neighbors,
- need to synchronize between themselves, and they can't learn the edges connecting them.

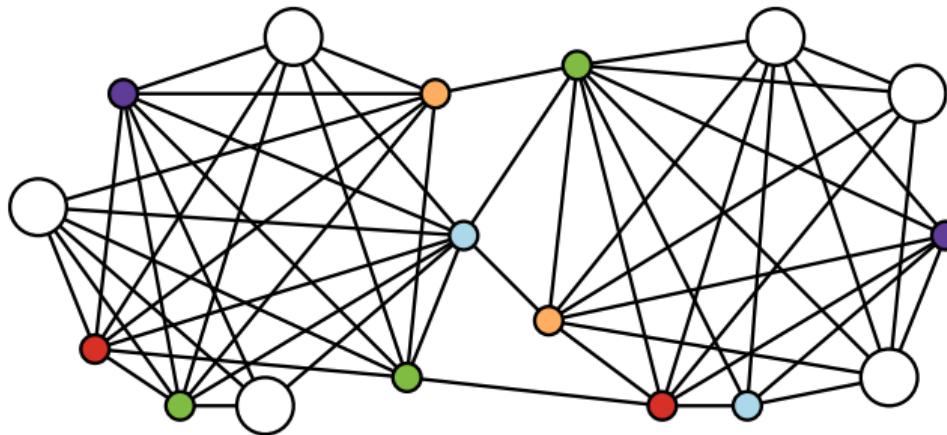
Issue 6: can't color put-aside sets at the end



Two reasons:

- no knowledge of the colors used by their neighbors,
- need to synchronize between themselves, and they can't learn the edges connecting them.

Issue 6: can't color put-aside sets at the end



Two reasons:

- no knowledge of the colors used by their neighbors,
- need to synchronize between themselves, and they can't learn the edges connecting them.

Fingerprints

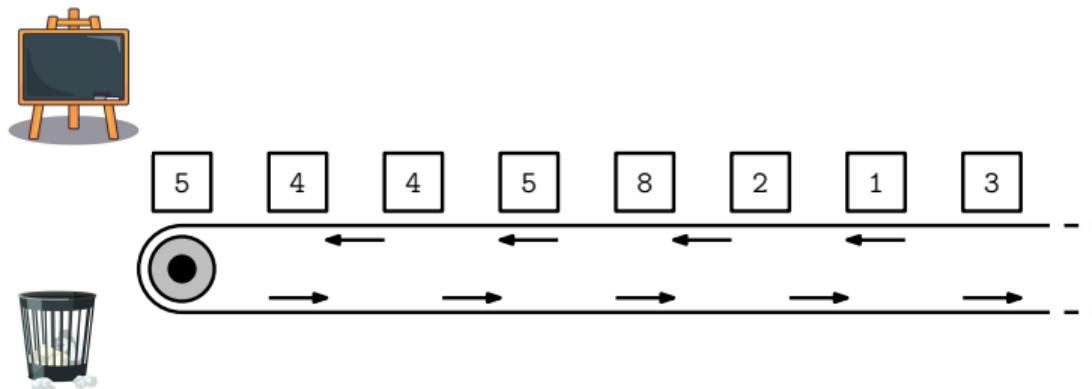
Important for: computing almost-cliques, estimating their density, colorful matching

High-level idea: each cluster v computes a $\Theta(\log n)$ -bit vector f_v , combining two fingerprints f_v and f_w allows to estimate the size of $N(v) \cup N(w)$.

Fingerprints: a detour through streaming

Consider a stream of n numbers between 1 and m .

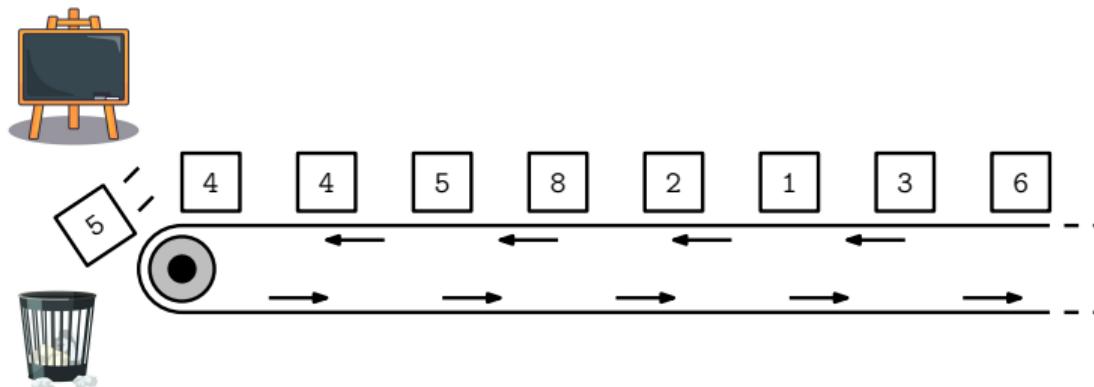
Goal: estimate F_0 , the number of seen distinct elements, using as little memory as possible. (0-th frequency moment)



Fingerprints: a detour through streaming

Consider a stream of n numbers between 1 and m .

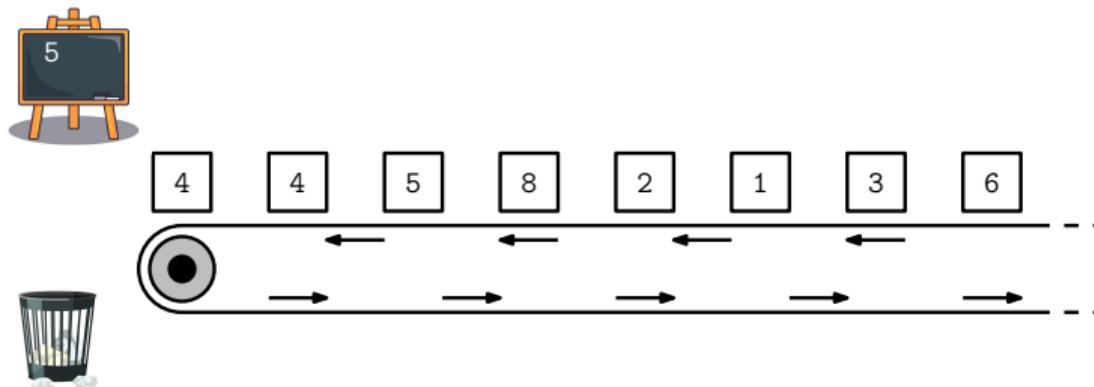
Goal: estimate F_0 , the number of seen distinct elements, using as little memory as possible. (0-th frequency moment)



Fingerprints: a detour through streaming

Consider a stream of n numbers between 1 and m .

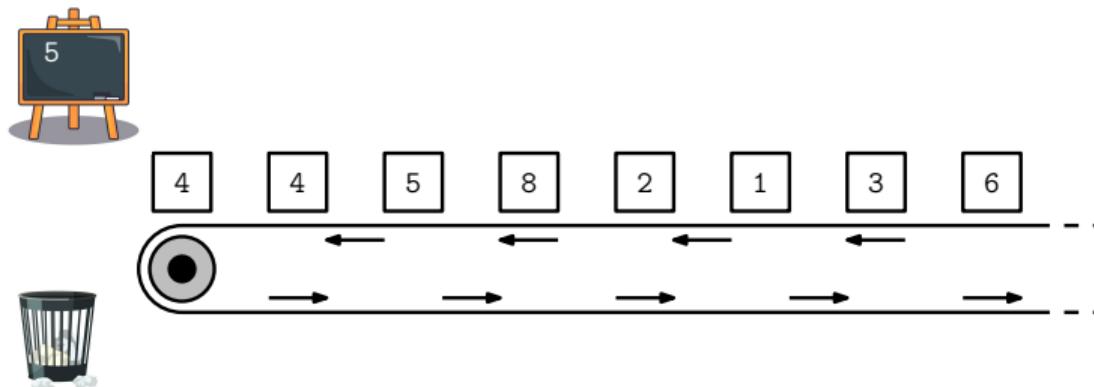
Goal: estimate F_0 , the number of seen distinct elements, using as little memory as possible. (0-th frequency moment)



Fingerprints: a detour through streaming

Consider a stream of n numbers between 1 and m .

Goal: estimate F_0 , the number of seen distinct elements, using as little memory as possible. (0-th frequency moment)

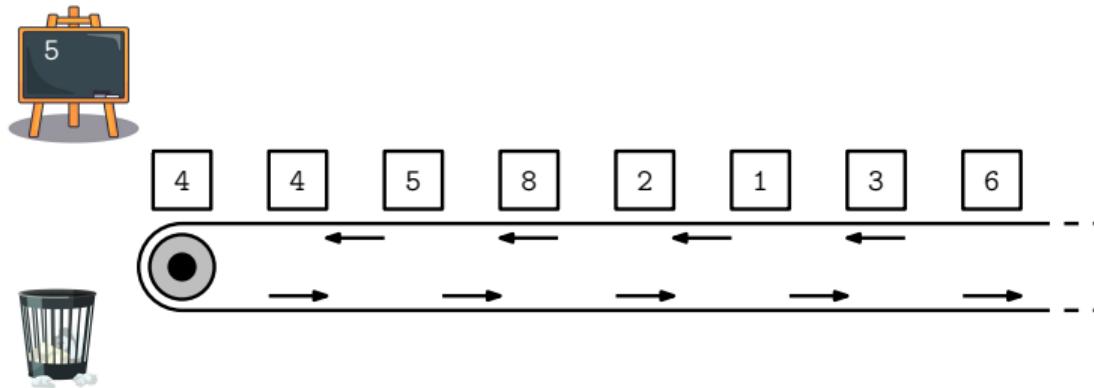


Trivial, exact solutions: $\log \binom{m}{F_0} \leq \min(n \log m, m)$.

Fingerprints: a detour through streaming

Consider a stream of n numbers between 1 and m .

Goal: estimate F_0 , the number of seen distinct elements, using as little memory as possible. (0-th frequency moment)



Trivial, exact solutions: $\log \binom{m}{F_0} \leq \min(n \log m, m)$.

Claim: given access to a large source of randomness, can estimate $(1 \pm \varepsilon)F_0$ using $O(\varepsilon^{-2} \log n)$ bits w.p. $1 - 1/\text{poly}(n)$.

Fingerprints: building block

Consider k independent geometric random variables X_1, \dots, X_k : for each $i \in [k], x \in \mathbb{N}$, $\Pr[X_i \geq x] = 2^{-x}$. Let $Y = \max_i X_i$.

Fingerprints: building block

Consider k independent geometric random variables X_1, \dots, X_k : for each $i \in [k], x \in \mathbb{N}$, $\Pr[X_i \geq x] = 2^{-x}$. Let $Y = \max_i X_i$.

$$\Pr[Y < x] =$$

$$\Pr[Y \geq x] \leq$$

Fingerprints: building block

Consider k independent geometric random variables X_1, \dots, X_k : for each $i \in [k], x \in \mathbb{N}$, $\Pr[X_i \geq x] = 2^{-x}$. Let $Y = \max_i X_i$.

$$\Pr[Y < x] = (1 - 2^{-x})^k \leq \exp(-k \cdot 2^{-x}).$$

$$\Pr[Y \geq x] \leq$$

Fingerprints: building block

Consider k independent geometric random variables X_1, \dots, X_k : for each $i \in [k], x \in \mathbb{N}$, $\Pr[X_i \geq x] = 2^{-x}$. Let $Y = \max_i X_i$.

$$\Pr[Y < x] = (1 - 2^{-x})^k \leq \exp(-k \cdot 2^{-x}).$$

$$\Pr[Y \geq x] \leq \sum_i \Pr[X_i \geq x] \leq k \cdot 2^{-x}.$$

Fingerprints: building block

Consider k independent geometric random variables X_1, \dots, X_k : for each $i \in [k], x \in \mathbb{N}$, $\Pr[X_i \geq x] = 2^{-x}$. Let $Y = \max_i X_i$.

$$\Pr[Y < x] = (1 - 2^{-x})^k \leq \exp(-k \cdot 2^{-x}).$$

$$\Pr[Y \geq x] \leq \sum_i \Pr[X_i \geq x] \leq k \cdot 2^{-x}.$$

Conclusion: $\Pr[|Y - \log k| > 4] < 1/10$.

Fingerprints: construction

For each element $e \in [m]$, use the randomness to build $T \in \Theta(\log n/\varepsilon^2)$ geometric random variables $X_{e,1}, \dots, X_{e,t}$.

Fingerprints: construction

For each element $e \in [m]$, use the randomness to build $T \in \Theta(\log n/\varepsilon^2)$ geometric random variables $X_{e,1}, \dots, X_{e,t}$.

Store in memory t values Y_1, \dots, Y_t , where $Y_i = \max_{e \in S} X_{e,i}$ is taking the maximum of the i^{th} geometric random variables of the elements seen so far in the stream.

Fingerprints: construction

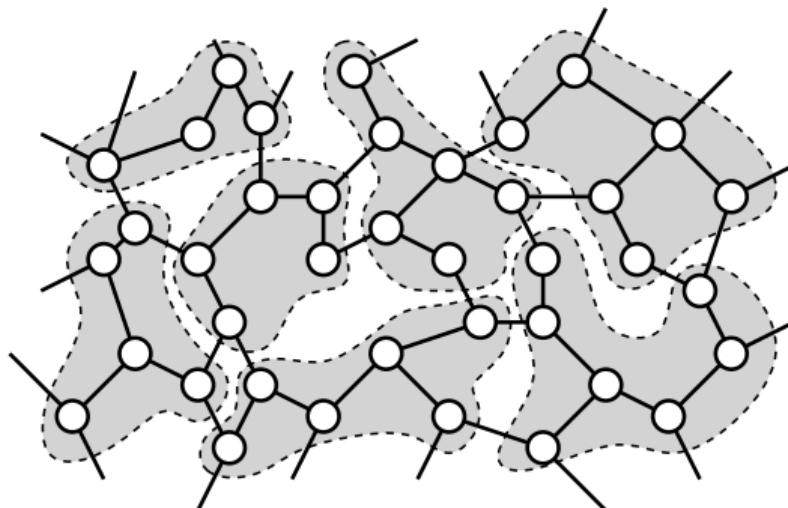
For each element $e \in [m]$, use the randomness to build $T \in \Theta(\log n/\varepsilon^2)$ geometric random variables $X_{e,1}, \dots, X_{e,t}$.

Store in memory t values Y_1, \dots, Y_t , where $Y_i = \max_{e \in S} X_{e,i}$ is taking the maximum of the i^{th} geometric random variables of the elements seen so far in the stream.

With high probability, the Y_i 's are concentrated around $\log|S|$. This gives an estimation of $|S|$, and makes them easy to store.

Fingerprints: distributedly

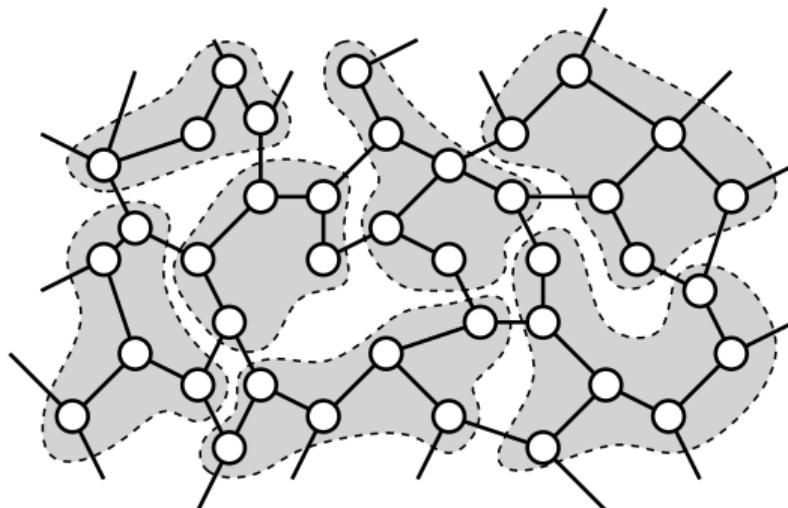
In streaming, this is not enough: still need to replace the large randomness by a hash function [Bla20].



Fingerprints: distributedly

In streaming, this is not enough: still need to replace the large randomness by a hash function [Bla20].

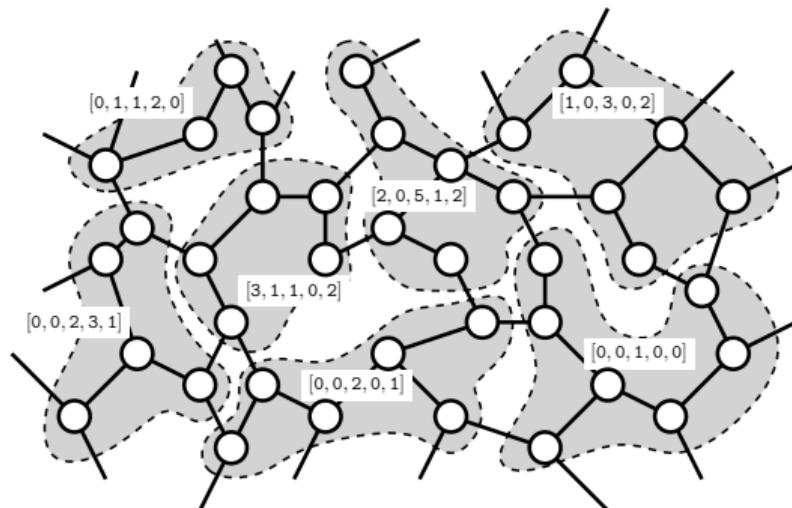
But in **distributed**, we are done!



Fingerprints: distributedly

In streaming, this is not enough: still need to replace the large randomness by a hash function [Bla20].

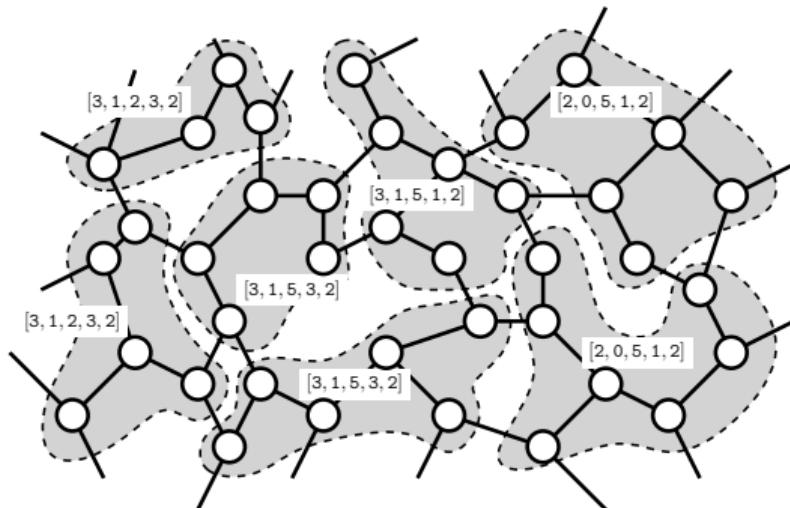
But in **distributed**, we are done!



Fingerprints: distributedly

In streaming, this is not enough: still need to replace the large randomness by a hash function [Bla20].

But in **distributed**, we are done!

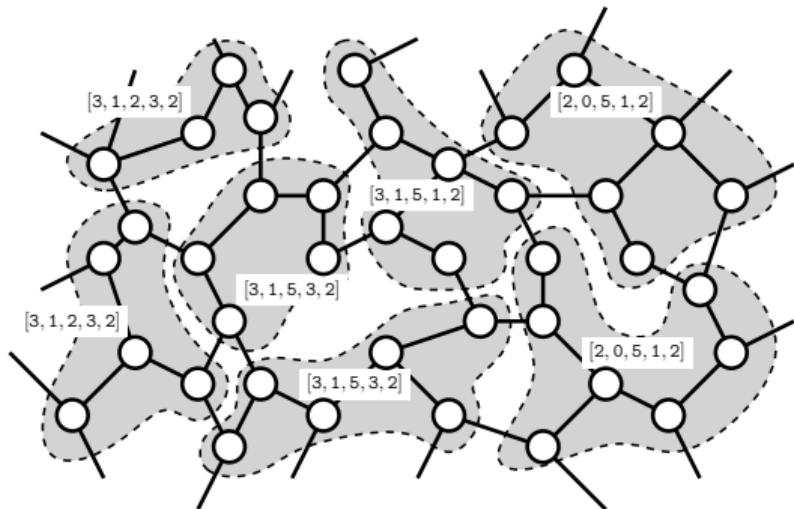


Taking the point-wise max of two fingerprints gives a fingerprint of **the union**.

Fingerprints: distributedly

In streaming, this is not enough: still need to replace the large randomness by a hash function [Bla20].

But in **distributed**, we are done!



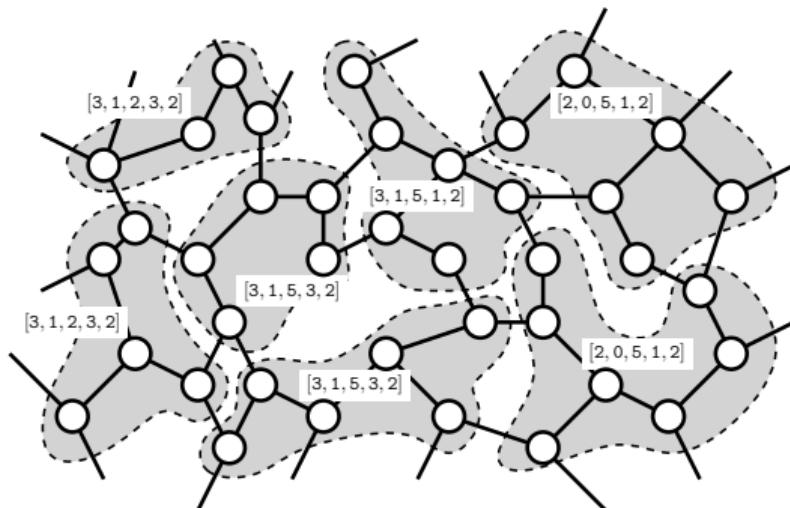
Taking the point-wise max of two fingerprints gives a fingerprint of **the union**.

$|N(u) \cup N(v)| \gg \min(|N(u)|, |N(v)|)$ will be detected using fingerprints at links connecting the clusters u and v .

Fingerprints: distributedly

In streaming, this is not enough: still need to replace the large randomness by a hash function [Bla20].

But in **distributed**, we are done!



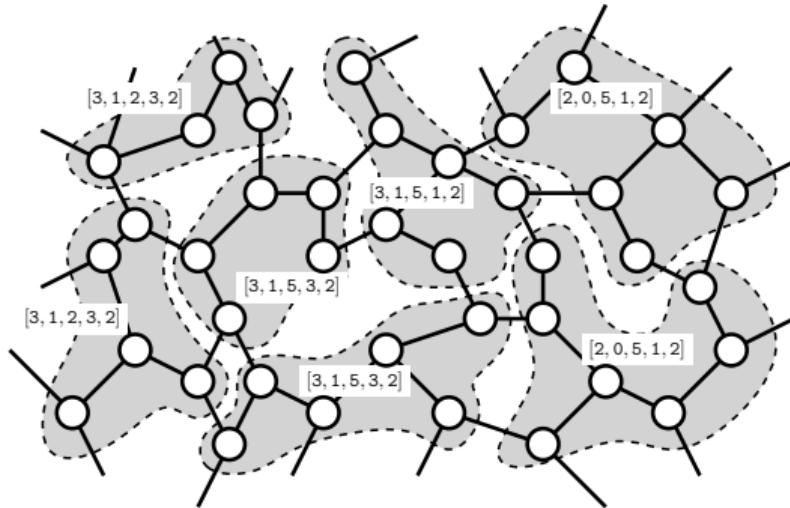
Taking the point-wise max of two fingerprints gives a fingerprint of **the union**.

$|N(u) \cup N(v)| \gg \min(|N(u)|, |N(v)|)$ will be detected using fingerprints at links connecting the clusters u and v .

Unlocks almost-clique decomposition, estimating number of missing edges in almost-cliques, number of external neighbors...

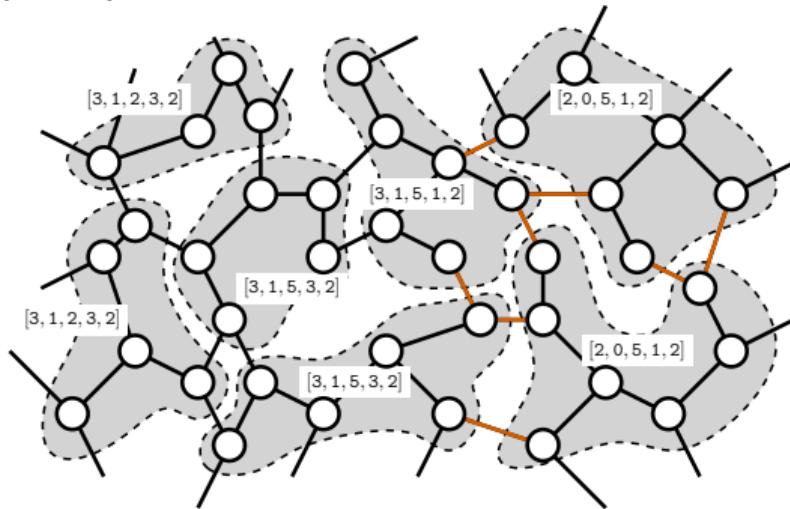
Communicating in almost-cliques

Once we know which links connect cluster in the same almost-cliques, many things open up.



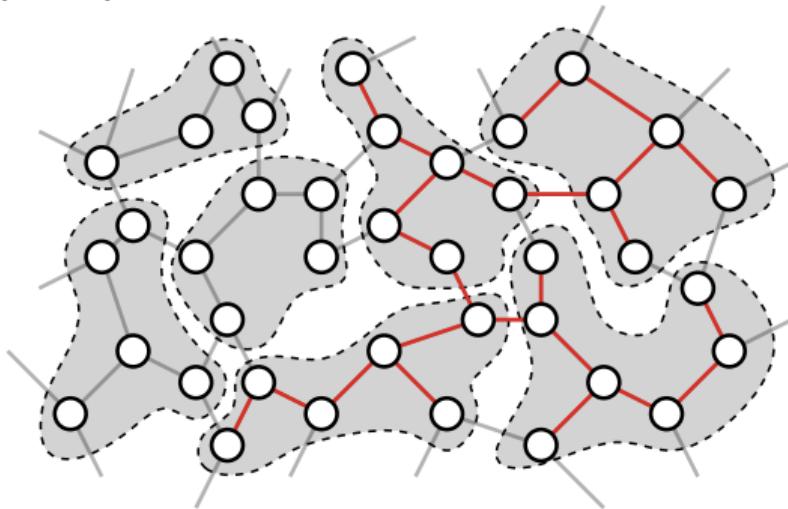
Communicating in almost-cliques

Once we know which links connect cluster in the same almost-cliques, many things open up.



Communicating in almost-cliques

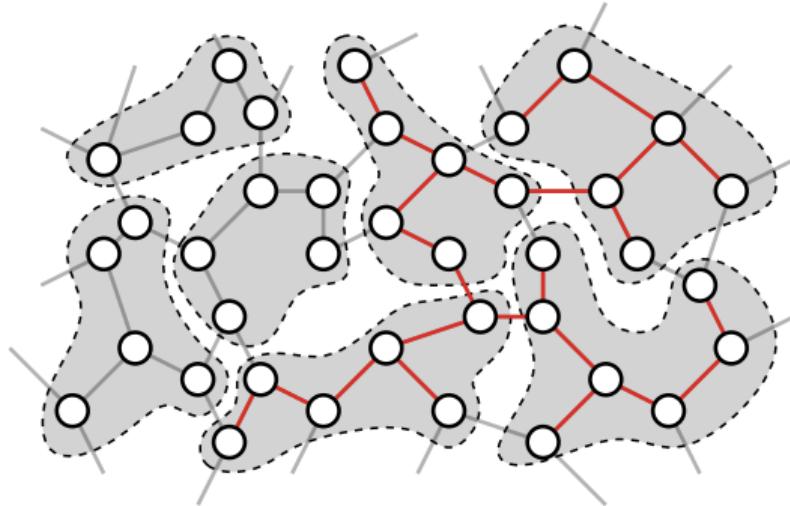
Once we know which links connect cluster in the same almost-cliques, many things open up.



BFS in an almost-clique: allows to count its exact size, number of colored nodes, sum values...

Communicating in almost-cliques

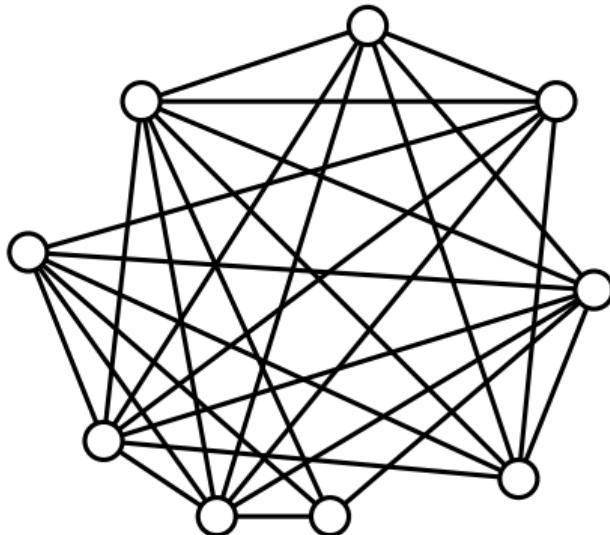
Once we know which links connect cluster in the same almost-cliques, many things open up.



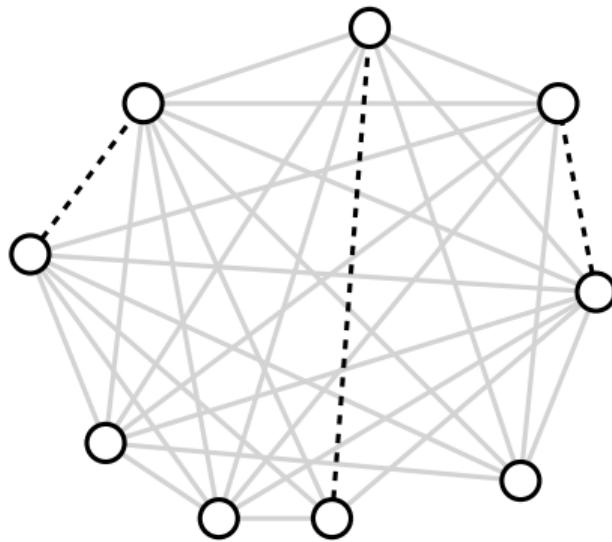
BFS in an almost-clique: allows to count its exact size, number of colored nodes, sum values...

Can partition almost-cliques into $\Theta(\Delta / \log n)$ random groups, each able to broadcast/aggregate $\Theta(\log n)$ bits in $O(1)$ rounds.

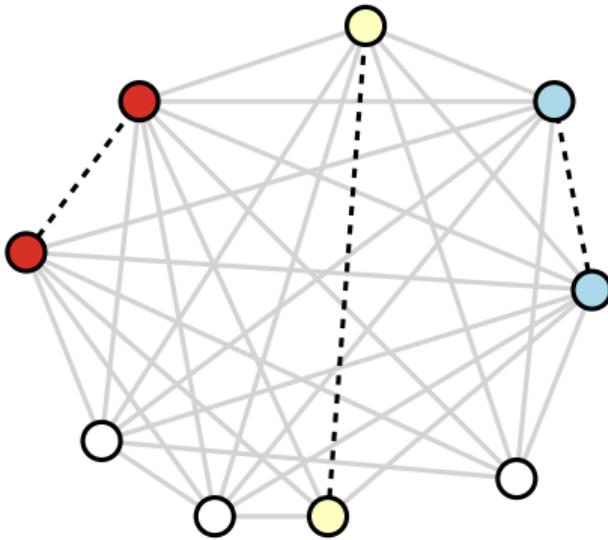
Sampling anti-edges with fingerprints



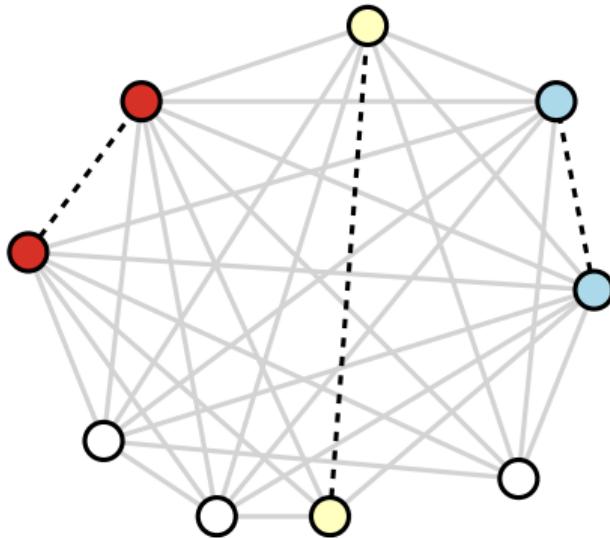
Sampling anti-edges with fingerprints



Sampling anti-edges with fingerprints

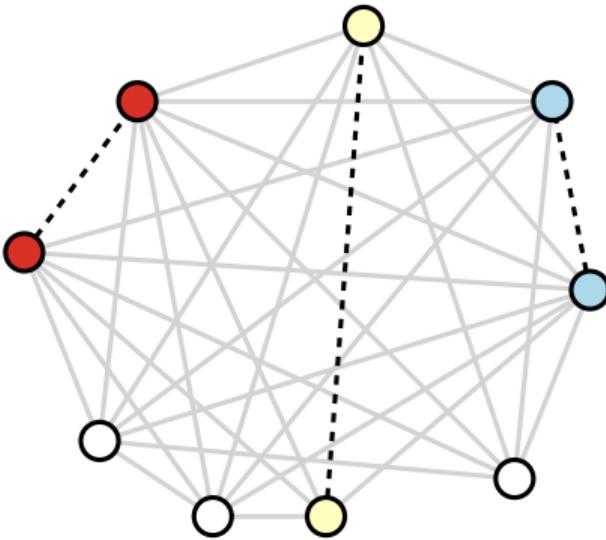


Sampling anti-edges with fingerprints



Idea: $\max \text{ in neighborhood} \neq \max \text{ in almost-clique} \Rightarrow \text{anti-edge found!}$

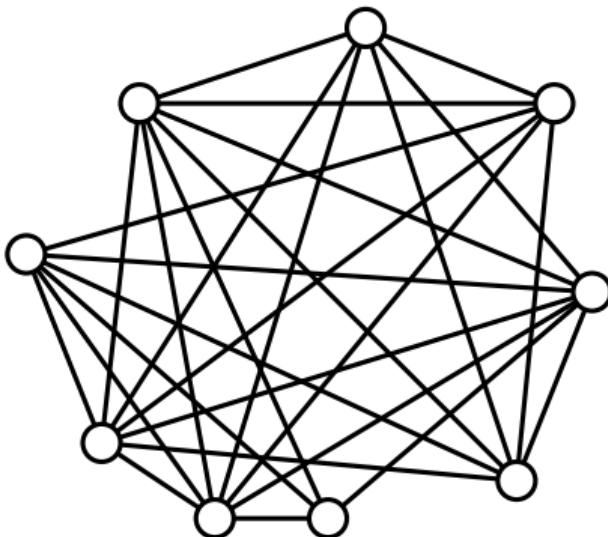
Sampling anti-edges with fingerprints



Idea: max in neighborhood \neq max in almost-clique \Rightarrow anti-edge found!

With $t \in \Theta(\log n)$ geometric random variables, find $\Theta(t)$ anti-edges w.h.p.

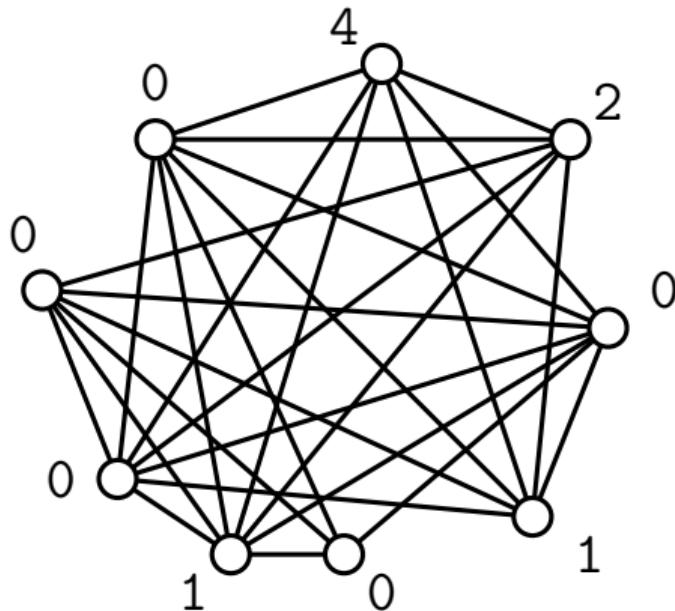
Sampling anti-edges with fingerprints



Idea: max in neighborhood \neq max in almost-clique \Rightarrow anti-edge found!

With $t \in \Theta(\log n)$ geometric random variables, find $\Theta(t)$ anti-edges w.h.p.

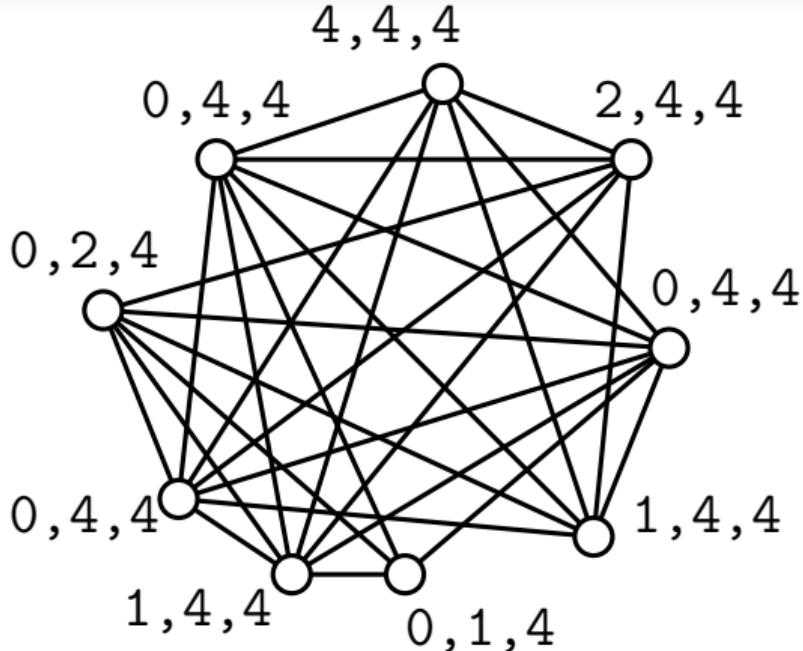
Sampling anti-edges with fingerprints



Idea: max in neighborhood \neq max in almost-clique \Rightarrow anti-edge found!

With $t \in \Theta(\log n)$ geometric random variables, find $\Theta(t)$ anti-edges w.h.p.

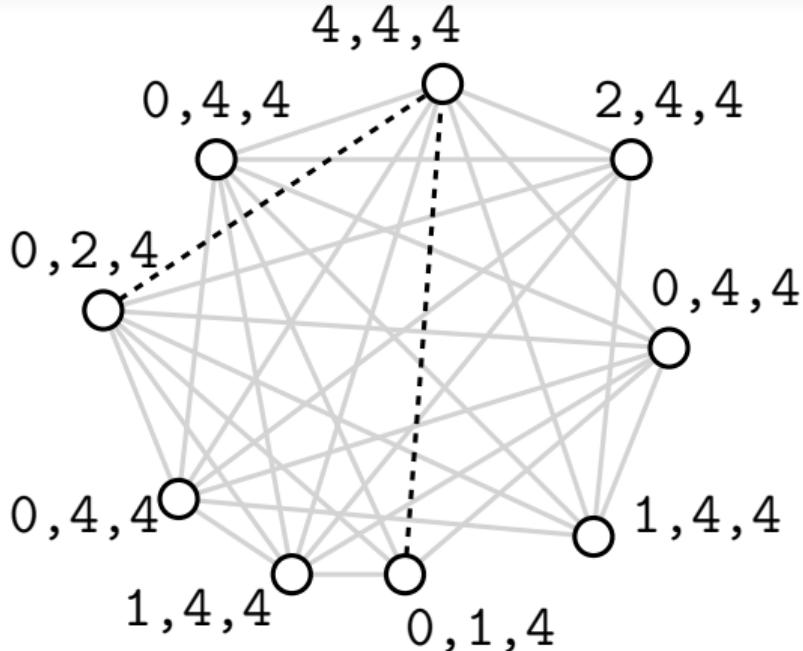
Sampling anti-edges with fingerprints



Idea: max in neighborhood \neq max in almost-clique \Rightarrow anti-edge found!

With $t \in \Theta(\log n)$ geometric random variables, find $\Theta(t)$ anti-edges w.h.p.

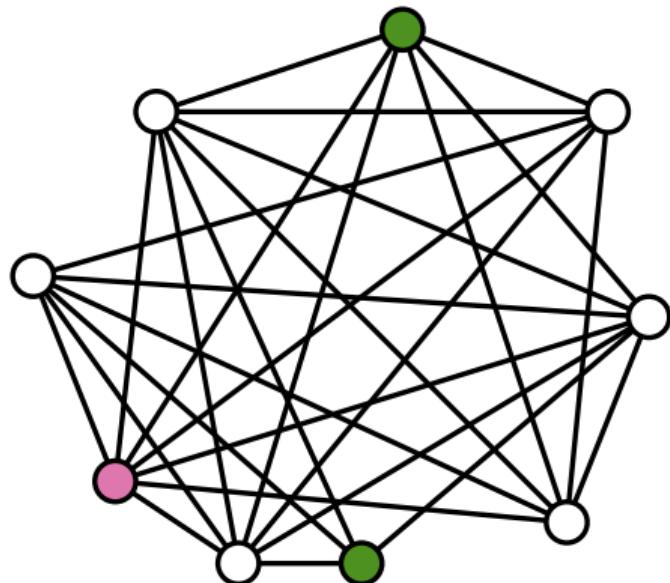
Sampling anti-edges with fingerprints



Idea: max in neighborhood \neq max in almost-clique \Rightarrow anti-edge found!

With $t \in \Theta(\log n)$ geometric random variables, find $\Theta(t)$ anti-edges w.h.p.

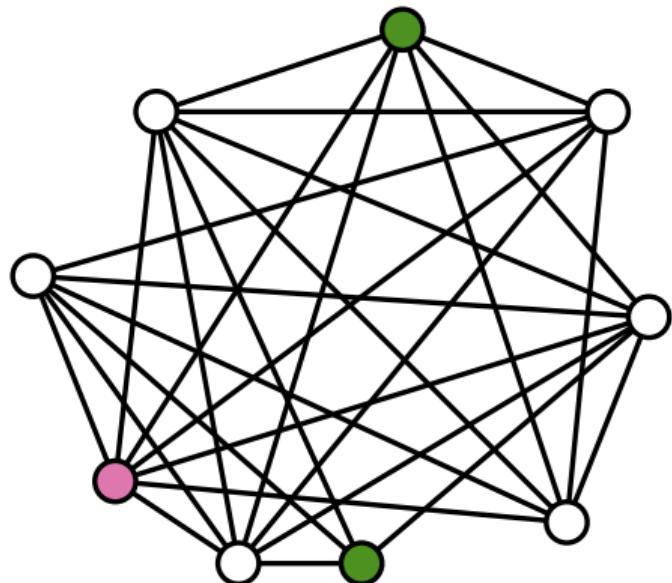
Synchronizing color trials, distributedly



free: 1 2 3 4 5 6 7 8 9

Look at set of colors unused in the almost-clique (the *clique palette*). [FGH⁺23, FHN23]

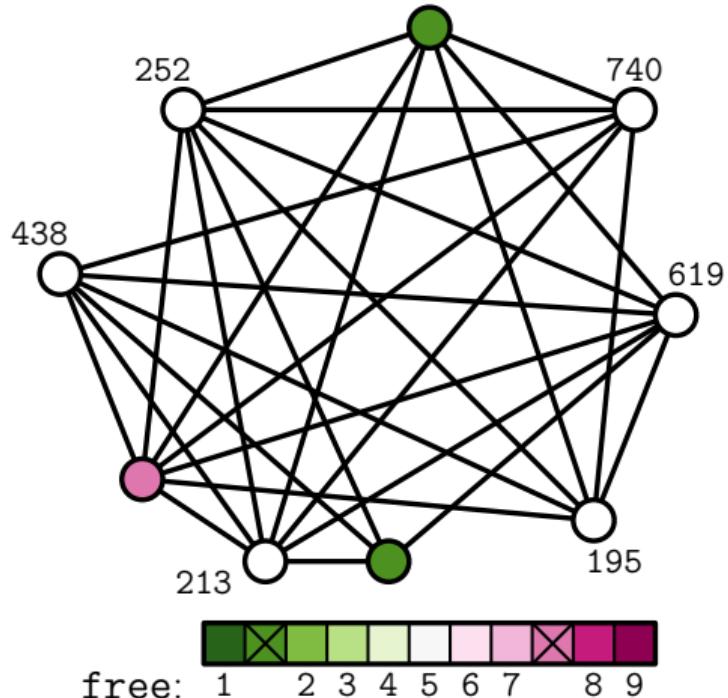
Synchronizing color trials, distributedly



Look at set of colors unused in the almost-clique (the *clique palette*). [FGH⁺23, FHN23]

Uncolored nodes randomly permute themselves, node number i tries free color number i .

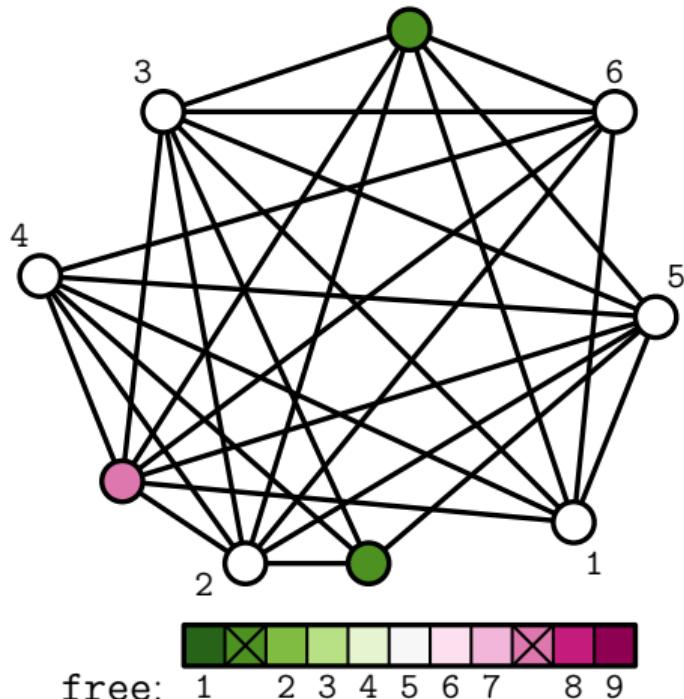
Synchronizing color trials, distributedly



Look at set of colors unused in the almost-clique (the *clique palette*). [FGH⁺23, FHN23]

Uncolored nodes randomly permute themselves, node number i tries free color number i .

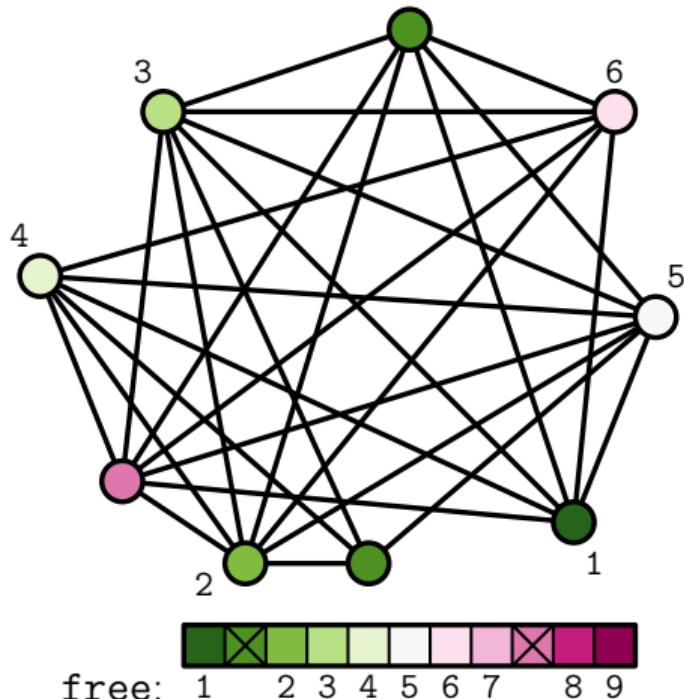
Synchronizing color trials, distributedly



Look at set of colors unused in the almost-clique (the *clique palette*). [FGH⁺23, FHN23]

Uncolored nodes randomly permute themselves, node number i tries free color number i .

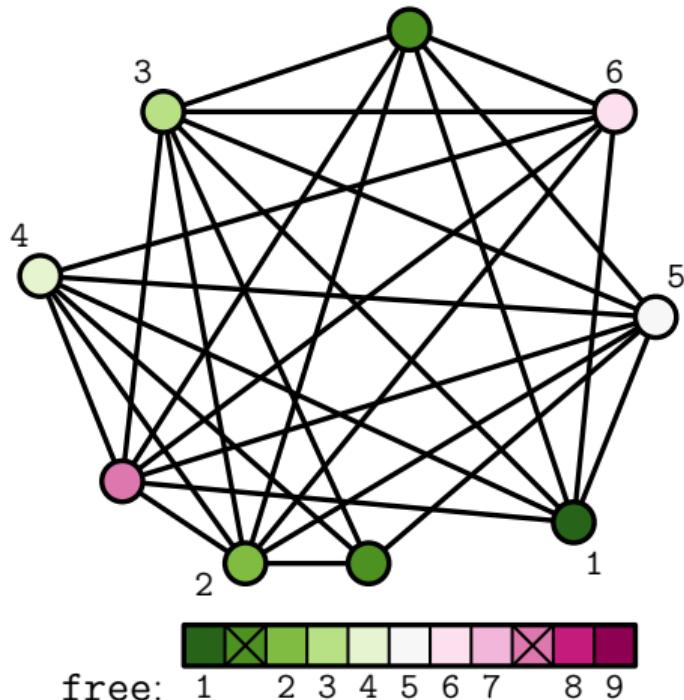
Synchronizing color trials, distributedly



Look at set of colors unused in the almost-clique (the *clique palette*). [FGH⁺23, FHN23]

Uncolored nodes randomly permute themselves, node number i tries free color number i .

Synchronizing color trials, distributedly



Look at set of colors unused in the almost-clique (the *clique palette*). [FGH⁺23, FHN23]

Uncolored nodes randomly permute themselves, node number i tries free color number i .

+ bunch of routing, etc...

What next...



Still unsolved or under the rug

Coloring put-aside sets:

Routing colors to neighbors, finding anti-edges:

Pseudorandom color samples:

Still unsolved or under the rug

Coloring put-aside sets: Instead of a trivial solution based on “learning everything”, we use color swaps inside almost-cliques.

Routing colors to neighbors, finding anti-edges:

Pseudorandom color samples:

Still unsolved or under the rug

Coloring put-aside sets: Instead of a trivial solution based on “learning everything”, we use color swaps inside almost-cliques.

Routing colors to neighbors, finding anti-edges: A lot is done by sampling random “relay clusters”, tasked with forwarding part of the information.

Pseudorandom color samples:

Still unsolved or under the rug

Coloring put-aside sets: Instead of a trivial solution based on “learning everything”, we use color swaps inside almost-cliques.

Routing colors to neighbors, finding anti-edges: A lot is done by sampling random “relay clusters”, tasked with forwarding part of the information.

Intuition: every node in an almost-clique has $\approx \Delta$ neighbors in the almost-clique. If every node picks one of $\Theta(\Delta / \log n)$ random groups, every node has $\Theta(\log n)$ neighbors in each group w.h.p. Divide the work between the groups.

Pseudorandom color samples:

Still unsolved or under the rug

Coloring put-aside sets: Instead of a trivial solution based on “learning everything”, we use color swaps inside almost-cliques.

Routing colors to neighbors, finding anti-edges: A lot is done by sampling random “relay clusters”, tasked with forwarding part of the information.

Intuition: every node in an almost-clique has $\approx \Delta$ neighbors in the almost-clique. If every node picks one of $\Theta(\Delta / \log n)$ random groups, every node has $\Theta(\log n)$ neighbors in each group w.h.p. Divide the work between the groups.

Pseudorandom color samples: Pseudorandom samples are much simpler when everyone agrees on the overall sampling space. We enforce this by reserving some colors for late in the algorithm.

Concluding



Ending remarks

What have we learned?

1. Can perform distance-2 coloring in $O(\log^* n)$ rounds when $\Delta > \log^c n$.
2. In general, we can perform distributed coloring as a series of “broadcast/aggregate” operations.

Ending remarks

What have we learned?

1. Can perform distance-2 coloring in $O(\log^* n)$ rounds when $\Delta > \log^c n$.
2. In general, we can perform distributed coloring as a series of “broadcast/aggregate” operations.

What should be done next?

1. Harder coloring problems remain open: list-coloring, Δ -coloring...

Ending remarks

What have we learned?

1. Can perform distance-2 coloring in $O(\log^* n)$ rounds when $\Delta > \log^c n$.
2. In general, we can perform distributed coloring as a series of “broadcast/aggregate” operations.

What should be done next?

1. Harder coloring problems remain open: list-coloring, Δ -coloring...
2. Forget about coloring with congestion for a while, take the techniques elsewhere.

Ending remarks

What have we learned?

1. Can perform distance-2 coloring in $O(\log^* n)$ rounds when $\Delta > \log^c n$.
2. In general, we can perform distributed coloring as a series of “broadcast/aggregate” operations.

What should be done next?

1. Harder coloring problems remain open: list-coloring, Δ -coloring...
2. Forget about coloring with congestion for a while, take the techniques elsewhere.

Thank you for attending!

-  Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider.
The locality of distributed symmetry breaking.
Journal of the ACM, 63(3):20:1–20:45, 2016.
-  Leonid Barenboim and Uri Goldenberg.
Speedup of distributed algorithms for power graphs in the CONGEST model.
In *the Proceedings of the International Symposium on Distributed Computing (DISC)*, volume 319 of *LIPics*, pages 6:1–6:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
-  Jaroslaw Blasiok.
Optimal streaming and tracking distinct elements with high probability.
ACM Trans. Algorithms, 16(1):3:1–3:28, 2020.
-  Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie.
An exponential separation between randomized and deterministic complexity in the LOCAL model.
SIAM J. Comput., 48(1):122–143, 2019.

-  Yi-Jun Chang, Wenzheng Li, and Seth Pettie.
Distributed $(\Delta + 1)$ -coloring via ultrafast graph shattering.
SIAM Journal of Computing, 49(3):497–539, 2020.
-  Maxime Flin, Mohsen Ghaffari, Magnús M. Halldórsson, Fabian Kuhn, and Alexandre Nolin.
Coloring fast with broadcasts.
In *the Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 455–465. ACM, 2023.
-  Maxime Flin, Magnús M. Halldórsson, and Alexandre Nolin.
Fast coloring despite congested relays.
In *the Proceedings of the International Symposium on Distributed Computing (DISC)*, 2023.
-  Maxime Flin, Magnús M. Halldórsson, and Alexandre Nolin.
Decentralized Distributed Graph Coloring II: Degree+1-Coloring Virtual Graphs.

In the Proceedings of the International Symposium on Distributed Computing (DISC), volume 319, pages 24:1–24:22, 2024.

-  Maxime Flin, Magnús M. Halldórsson, and Alexandre Nolin.
Decentralized distributed graph coloring: Cluster graphs.
In the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), pages 394–405. ACM, 2025.
-  Magnús M. Halldórsson, Fabian Kuhn, and Yannic Maus.
Distance-2 coloring in the CONGEST model.
In PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020, pages 233–242, 2020.
-  Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Alexandre Nolin.
Coloring fast without learning your neighbors' colors.
In 34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference, pages 39:1–39:17, 2020.
-  Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Tigran Tonoyan.

Efficient randomized distributed coloring in CONGEST.

In *the Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 1180–1193. ACM, 2021.

Full version at CoRR abs/2105.04700.

 Magnús M. Halldórsson and Alexandre Nolin.

Superfast coloring in CONGEST via efficient color sampling.

Theor. Comput. Sci., 948:113711, 2023.

 Magnús M. Halldórsson, Alexandre Nolin, and Tigran Tonoyan.

Overcoming congestion in distributed coloring.

In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 26–36. ACM, 2022.

 David G. Harris, Johannes Schneider, and Hsin-Hao Su.

Distributed $(\Delta + 1)$ -coloring in sublogarithmic rounds.

Journal of the ACM, 65:19:1–19:21, 2018.

 Öjvind Johansson.

Simple distributed $\Delta + 1$ -coloring of graphs.

Inf. Process. Lett., 70(5):229–232, 1999.

 M. Luby.

A simple parallel algorithm for the maximal independent set problem.

SIAM Journal on Computing, 15:1036–1053, 1986.

 Moni Naor.

A lower bound on probabilistic algorithms for distributive ring coloring.

SIAM J. Discret. Math., 4(3):409–412, 1991.

 Johannes Schneider and Roger Wattenhofer.

A new technique for distributed symmetry breaking.

In *the Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 257–266. ACM, 2010.