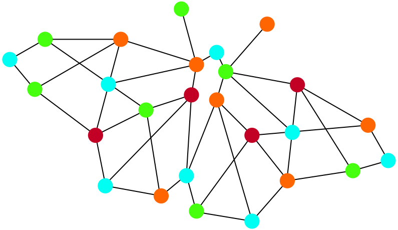
Coloring in Graph Streams via Deterministic and Adversarially Robust Algorithms

Sepehr Assadi (Rutgers) Amit Chakrabarti (Dartmouth) Prantar Ghosh (DIMACS)
Manuel Stoeckl (Dartmouth)*

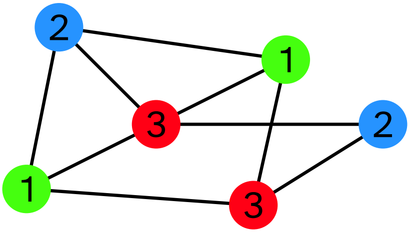
Symposium on Principles of Database Systems 2023



*This work was supported in part by NSF under awards CCF-1907738 and CCF-2006589. S.A's research supported in part by a NSF CAREER Grant CCF-2047061, a Google Research gift, and a Fulcrum award from the Rutgers Research Council.

Δ-based coloring

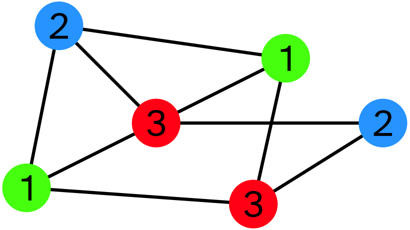
- Graph $G = (V, E)$ with n vertices and max degree Δ
- Vertex coloring: assign a color to each vertex G so that no edge connects two vertices of the same color



- Finding vertex colorings with a *minimum* number of colors is NP-hard
- Δ-*based colorings* use number of colors depending on Δ.
  - Greedy algorithm: $\Delta + 1$ colors
  - Linial's algorithm[Linial92]: $O\left(\Delta^2\right)$ colors
  - Brook's theorem: Δ colors (if possible)

Δ-based coloring

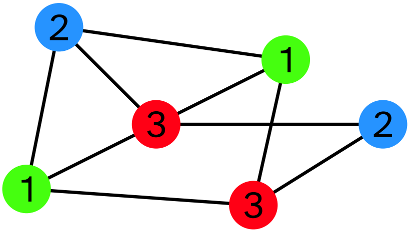
- Graph $G = (V, E)$ with n vertices and max degree Δ
- Vertex coloring: assign a color to each vertex G so that no edge connects two vertices of the same color



- Finding vertex colorings with a *minimum* number of colors is NP-hard
- Δ*-based colorings* use number of colors depending on Δ.
  - Greedy algorithm: $\Delta + 1$ colors
  - Linial's algorithm[Linial92]: $O\left(\Delta^2\right)$ colors
  - Brook's theorem: Δ colors (if possible)

Δ-based coloring

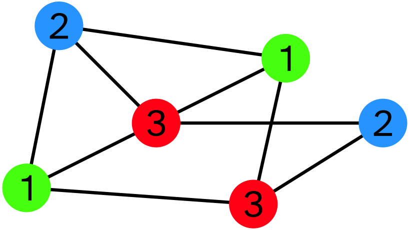
- Graph $G = (V, E)$ with n vertices and max degree Δ
- Vertex coloring: assign a color to each vertex G so that no edge connects two vertices of the same color



- Finding vertex colorings with a *minimum* number of colors is NP-hard
- *Δ-based colorings* use number of colors depending on Δ.
  - Greedy algorithm: $\Delta + 1$ colors
  - Linial's algorithm[Linial92]: $O\left(\Delta^2\right)$ colors
  - Brook's theorem: Δ colors (if possible)

Δ-based coloring

- Graph $G = (V, E)$ with n vertices and max degree Δ
- Vertex coloring: assign a color to each vertex G so that no edge connects two vertices of the same color



- Finding vertex colorings with a *minimum* number of colors is NP-hard
- *Δ-based colorings* use number of colors depending on Δ.
  - Greedy algorithm: $\Delta + 1$ colors
  - Linial's algorithm[Linial92]: $O\left(\Delta^2\right)$ colors
  - Brook's theorem: Δ colors (if possible)

To what extent is randomization necessary for streaming algorithms that compute a Δ-based coloring?

Outline

Deterministic multi-pass $(\Delta + 1)$ coloring

Adversarially robust coloring with $O\left(\Delta^{2.5}\right)$ colors

Deterministic multi-pass $(\Delta + 1)$-coloring on a graph stream

Input:

- A graph $G = (V, E)$ on n vertices with maximum degree Δ, provided as a sequence of edges

Processing:

- Limited working space: only "*semi-streaming*" ($\tilde{O}(n)$, where $\tilde{O}(\cdot)$ hides polylog factors in n and Δ.)[†]
- For each pass, algorithm reads the input edge sequence in order

Output:

- A coloring $\chi : V \to [\Delta + 1]$, so that if $\{u, v\} \in E$, then $\chi(u) \neq \chi(v)$

[†] Storing G takes $\tilde{O}(n\Delta)$ space.

Deterministic multi-pass $(\Delta + 1)$-coloring on a graph stream

Input:

- A graph $G = (V, E)$ on n vertices with maximum degree Δ, provided as a sequence of edges

Processing:

- Limited working space: only "*semi-streaming*" ($\tilde{O}(n)$, where $\tilde{O}(\cdot)$ hides polylog factors in n and Δ.)[†]
- For each pass, algorithm reads the input edge sequence in order

Output:

- A coloring $\chi : V \to [\Delta + 1]$, so that if $\{u, v\} \in E$, then $\chi(u) \neq \chi(v)$

[†] Storing G takes $\tilde{O}(n\Delta)$ space.

Deterministic multi-pass $(\Delta + 1)$-coloring on a graph stream

Input:

- A graph $G = (V, E)$ on n vertices with maximum degree Δ, provided as a sequence of edges

Processing:

- Limited working space: only "*semi-streaming*" ($\tilde{O}(n)$, where $\tilde{O}(\cdot)$ hides polylog factors in n and Δ.)[†]
- For each pass, algorithm reads the input edge sequence in order

Output:

- A coloring $\chi : V \to [\Delta + 1]$, so that if $\{u, v\} \in E$, then $\chi(u) \neq \chi(v)$

[†]Storing G takes $\tilde{O}(n\Delta)$ space.

Selected prior work

- The standard greedy algorithm can $(\Delta + 1)$-color a graph of max degree Δ, but has no semi-streaming implementation
- [AssadiChenKhanna19] Single-pass randomized streaming algorithm for $(\Delta + 1)$ coloring, using semi-streaming space
- [AssadiChenSun22] No 1-pass deterministic semi-streaming algorithms for even coloring a graph with $\mathrm{poly}(\Delta)$ colors
- [AssadiChenSun22] But with $O(\log \Delta)$ passes, can obtain an $O(\Delta)$ coloring

Question

Is there a multi-pass deterministic semi-streaming space algorithm for $\Delta + 1$ coloring?

- [GhaffariKuhn21] Deterministic $(\Delta + 1)$ coloring algorithm in the "LOCAL" and "CONGEST" models of distributed algorithms. [HalldórssonNolinKuhnTonoyan22] "degree + 1" coloring

Selected prior work

- The standard greedy algorithm can $(\Delta + 1)$-color a graph of max degree Δ, but has no semi-streaming implementation
- [AssadiChenKhanna19] Single-pass randomized streaming algorithm for $(\Delta + 1)$ coloring, using semi-streaming space
- [AssadiChenSun22] No 1-pass deterministic semi-streaming algorithms for even coloring a graph with $\mathrm{poly}(\Delta)$ colors
- [AssadiChenSun22] But with $O(\log \Delta)$ passes, can obtain an $O(\Delta)$ coloring

Question

Is there a multi-pass deterministic semi-streaming space algorithm for $\Delta + 1$ coloring?

- [GhaffariKuhn21] Deterministic $(\Delta + 1)$ coloring algorithm in the "LOCAL" and "CONGEST" models of distributed algorithms. [HalldórssonNolinKuhnTonoyan22] "degree + 1" coloring

Selected prior work

- The standard greedy algorithm can $(\Delta + 1)$-color a graph of max degree Δ, but has no semi-streaming implementation
- [AssadiChenKhanna19] Single-pass randomized streaming algorithm for $(\Delta + 1)$ coloring, using semi-streaming space
- [AssadiChenSun22] No 1-pass deterministic semi-streaming algorithms for even coloring a graph with $\mathrm{poly}(\Delta)$ colors
- [AssadiChenSun22] But with $O(\log \Delta)$ passes, can obtain an $O(\Delta)$ coloring

Question

Is there a multi-pass deterministic semi-streaming space algorithm for $\Delta + 1$ coloring?

- [GhaffariKuhn21] Deterministic $(\Delta + 1)$ coloring algorithm in the "LOCAL" and "CONGEST" models of distributed algorithms. [HalldórssonNolinKuhnTonoyan22] "degree + 1" coloring

Selected prior work

- The standard greedy algorithm can $(\Delta + 1)$-color a graph of max degree Δ, but has no semi-streaming implementation
- [AssadiChenKhanna19] Single-pass randomized streaming algorithm for $(\Delta + 1)$ coloring, using semi-streaming space
- [AssadiChenSun22] No 1-pass deterministic semi-streaming algorithms for even coloring a graph with $\mathrm{poly}(\Delta)$ colors
- [AssadiChenSun22] But with $O(\log \Delta)$ passes, can obtain an $O(\Delta)$ coloring

Question

Is there a multi-pass deterministic semi-streaming space algorithm for $\Delta + 1$ coloring?

- [GhaffariKuhn21] Deterministic $(\Delta + 1)$ coloring algorithm in the "LOCAL" and "CONGEST" models of distributed algorithms. [HalldórssonNolinKuhnTonoyan22] "degree + 1" coloring

Our results

Theorem

There is a deterministic streaming algorithm for $(\Delta+1)$-coloring which uses $O(\log \Delta \log \log \Delta)$ passes and $O\left(n (\log n)^2\right)$ space

Theorem

Same bounds hold for (degree+1) list coloring (D1LC), where each vertex $x \in V$ has associated list L_x of permitted colors, where $|L_x| \geq \deg x + 1$.

- Issue: storing color lists would take up to $\tilde{\Theta}(n\Delta)$ space. See paper.

Our results

Theorem

There is a deterministic streaming algorithm for $(\Delta + 1)$-coloring which uses $O(\log \Delta \log \log \Delta)$ passes and $O\left(n (\log n)^2\right)$ space

Theorem

Same bounds hold for (degree+1) list coloring (D1LC), where each vertex $x \in V$ has associated list L_x of permitted colors, where $|L_x| \geq \deg x + 1$.

- Issue: storing color lists would take up to $\tilde{\Theta}(n\Delta)$ space. See paper.

Our results

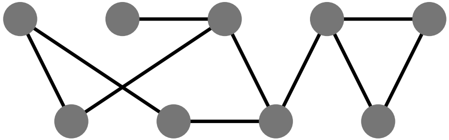
Theorem
There is a deterministic streaming algorithm for $(\Delta + 1)$*-coloring which uses* $O(\log \Delta \log \log \Delta)$ *passes and* $O\left(n (\log n)^2\right)$ *space*

Theorem
Same bounds hold for (degree+1) list coloring (D1LC), where each vertex $x \in V$ *has associated list* L_x *of permitted colors, where* $|L_x| \geq \deg x + 1$.

- Issue: storing color lists would take up to $\tilde{\Theta}(n\Delta)$ space. See paper.

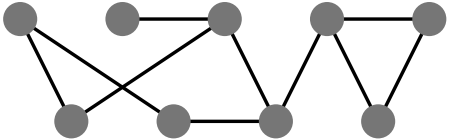
High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



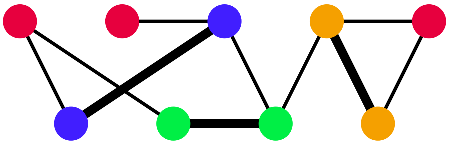
High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



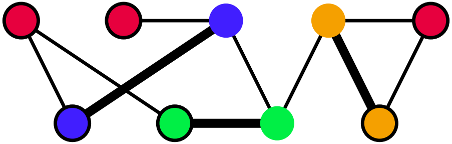
High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



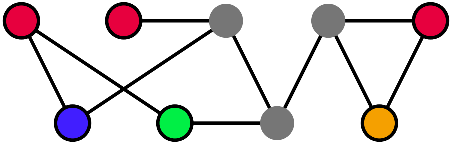
High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



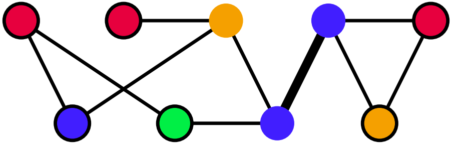
High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



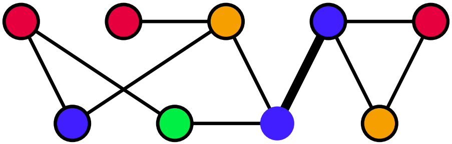
High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



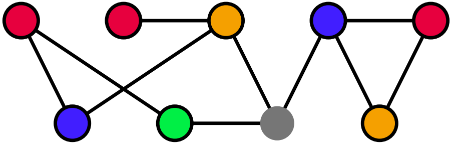
High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



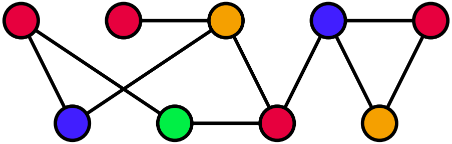
High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



High level description for deterministic $\Delta + 1$ coloring

- Will repeatedly fix colors for more vertices
- Propose colors for all unfixed vertices, with few monochromatic edges $\implies$ can fix a constant fraction of proposed colors
- Example:



Deterministic color proposal

Progressively restrict which colors each vertex may have.

- Assign each uncolored vertex set $[\Delta + 1]$ of all colors
- Repeatedly choose a subset of the current color set for each vertex
  - Have a cost function[‡] bounding the final number of monochromatic edges
  - Pass 1: Compute "slack" values[§], where if x has color set S, then

$$\text{slack}(x, S) = |S| - |\{\{y, x\} \in E : y\text{'s color fixed and in } S\}|$$

  - Pass 2-3: Use hash family to search for good color subset assignment
- After $O(\log \Delta)$ refinements, have a single proposed color for every vertex.

‡Similar: [Kuhn20] and [GhaffariKuhn21]
§Similar: [HalldórssonNolinKuhnTonoyan22]

Deterministic color proposal

Progressively restrict which colors each vertex may have.

- Assign each uncolored vertex set $[\Delta + 1]$ of all colors
- Repeatedly choose a subset of the current color set for each vertex
  - Have a cost function[‡] bounding the final number of monochromatic edges
  - Pass 1: Compute "slack" values[§], where if x has color set S, then

$$\text{slack}(x, S) = |S| - |\{\{y, x\} \in E : y\text{'s color fixed and in } S\}|$$

  - Pass 2-3: Use hash family to search for good color subset assignment
- After $O(\log \Delta)$ refinements, have a single proposed color for every vertex.

[‡]Similar: [Kuhn20] and [GhaffariKuhn21]
[§]Similar: [HalldórssonNolinKuhnTonoyan22]

Deterministic color proposal

Progressively restrict which colors each vertex may have.

- Assign each uncolored vertex set $[\Delta + 1]$ of all colors
- Repeatedly choose a subset of the current color set for each vertex
  - Have a cost function[‡] bounding the final number of monochromatic edges
  - Pass 1: Compute "slack" values[§], where if x has color set S, then

$$\text{slack}(x, S) = |S| - |\{\{y, x\} \in E : y\text{'s color fixed and in } S\}|$$

  - Pass 2-3: Use hash family to search for good color subset assignment
- After $O(\log \Delta)$ refinements, have a single proposed color for every vertex.

[‡] Similar: [Kuhn20] and [GhaffariKuhn21]
[§] Similar: [HalldórssonNolinKuhnTonoyan22]

Deterministic color proposal

Progressively restrict which colors each vertex may have.

- Assign each uncolored vertex set $[\Delta + 1]$ of all colors
- Repeatedly choose a subset of the current color set for each vertex
  - Have a cost function[‡] bounding the final number of monochromatic edges
  - Pass 1: Compute "slack" values[§], where if x has color set S, then

$$\text{slack}(x, S) = |S| - |\{\{y, x\} \in E : y\text{'s color fixed and in } S\}|$$

  - Pass 2-3: Use hash family to search for good color subset assignment
- After $O(\log \Delta)$ refinements, have a single proposed color for every vertex.

[‡] Similar: [Kuhn20] and [GhaffariKuhn21]
[§] Similar: [HalldórssonNolinKuhnTonoyan22]

Deterministic color proposal

Progressively restrict which colors each vertex may have.

- Assign each uncolored vertex set $[\Delta + 1]$ of all colors
- Repeatedly choose a subset of the current color set for each vertex
  - Have a cost function[‡] bounding the final number of monochromatic edges
  - Pass 1: Compute "slack" values[§], where if x has color set S, then

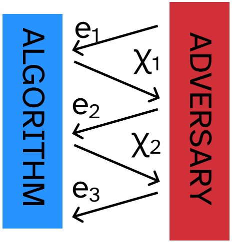
$$\text{slack}(x, S) = |S| - |\{\{y, x\} \in E : y\text{'s color fixed and in } S\}|$$

  - Pass 2-3: Use hash family to search for good color subset assignment
- After $O(\log \Delta)$ refinements, have a single proposed color for every vertex.

‡Similar: [Kuhn20] and [GhaffariKuhn21]
§Similar: [HalldórssonNolinKuhnTonoyan22]

Adversarially robust[¶] streaming algorithms

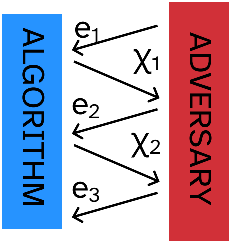
- Two player game between Algorithm and Adversary
- Adversary constructs series of inputs $e_1, e_2, \ldots e_i$, and Algorithm produces outputs $\chi_1, \ldots, \chi_i$ solving task for stream up to this point.
- Adversary's chosen inputs may depend on prior outputs of the Algorithm.
- Algorithm is "adversarially robust" if it has low error rate against *any* Adversary strategy
  - Example: Input generated in real time – outputs may influence future inputs
  - Sub-component of larger algorithm



[¶]See [Ben-EliezerJayaramWoodruffYogev20] for more explanation

Adversarially robust[¶] streaming algorithms

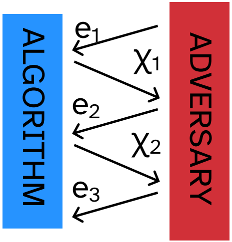
- Two player game between Algorithm and Adversary
- Adversary constructs series of inputs $e_1, e_2, \dots e_i$, and Algorithm produces outputs $\chi_1, \dots, \chi_i$ solving task for stream up to this point.
- Adversary's chosen inputs may depend on prior outputs of the Algorithm.
- Algorithm is "adversarially robust" if it has low error rate against *any* Adversary strategy
  - Example: Input generated in real time – outputs may influence future inputs
  - Sub-component of larger algorithm



[¶] See [Ben-EliezerJayaramWoodruffYogev20] for more explanation

Adversarially robust[¶] streaming algorithms

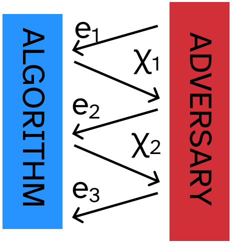
- Two player game between Algorithm and Adversary
- Adversary constructs series of inputs $e_1, e_2, \ldots e_i$, and Algorithm produces outputs $\chi_1, \ldots, \chi_i$ solving task for stream up to this point.
- Adversary's chosen inputs may depend on prior outputs of the Algorithm.
- Algorithm is "adversarially robust" if it has low error rate against *any* Adversary strategy
  - Example: Input generated in real time – outputs may influence future inputs
  - Sub-component of larger algorithm



[¶]See [Ben-EliezerJayaramWoodruffYogev20] for more explanation

Adversarially robust[¶] streaming algorithms

- Two player game between Algorithm and Adversary
- Adversary constructs series of inputs $e_1, e_2, \ldots e_i$, and Algorithm produces outputs $\chi_1, \ldots, \chi_i$ solving task for stream up to this point.
- Adversary's chosen inputs may depend on prior outputs of the Algorithm.
- Algorithm is "adversarially robust" if it has low error rate against *any* Adversary strategy
  - Example: Input generated in real time – outputs may influence future inputs
  - Sub-component of larger algorithm



[¶] See [Ben-EliezerJayaramWoodruffYogev20] for more explanation

Selected prior work

Task: Δ-based graph coloring on stream of graph edges, with known vertex set

- [AssadiChenKhanna19] A randomized streaming algorithm for $(\Delta + 1)$-coloring in semi-streaming $(\tilde{O}(n))$ space.
- [ChakrabartiGhoshStoeckl22] An adversarially robust $O(\Delta^3)$-coloring algorithm using semi-streaming space and access to $\tilde{O}(n\Delta)$ read-only random bits
- [ChakrabartiGhoshStoeckl22] Adversarially robust algorithms for $o(\Delta^2)$-coloring algorithms require $\tilde{\Omega}(n)$ space

Question

Is there an adversarially robust $O(\Delta^2)$-coloring algorithm in semi-streaming space which only needs $\tilde{O}(n)$ random bits?

Selected prior work

Task: Δ-based graph coloring on stream of graph edges, with known vertex set

- [AssadiChenKhanna19] A randomized streaming algorithm for $(\Delta + 1)$-coloring in semi-streaming $(\tilde{O}(n))$ space.
- [ChakrabartiGhoshStoeckl22] An adversarially robust $O(\Delta^3)$-coloring algorithm using semi-streaming space and access to $\tilde{O}(n\Delta)$ read-only random bits
- [ChakrabartiGhoshStoeckl22] Adversarially robust algorithms for $o(\Delta^2)$-coloring algorithms require $\tilde{\Omega}(n)$ space

Question

Is there an adversarially robust $O(\Delta^2)$-coloring algorithm in semi-streaming space which only needs $\tilde{O}(n)$ random bits?

Selected prior work

Task: Δ-based graph coloring on stream of graph edges, with known vertex set

- [AssadiChenKhanna19] A randomized streaming algorithm for $(\Delta + 1)$-coloring in semi-streaming ($\tilde{O}(n)$) space.
- [ChakrabartiGhoshStoeckl22] An adversarially robust $O(\Delta^3)$-coloring algorithm using semi-streaming space and access to $\tilde{O}(n\Delta)$ read-only random bits
- [ChakrabartiGhoshStoeckl22] Adversarially robust algorithms for $o(\Delta^2)$-coloring algorithms require $\tilde{\Omega}(n)$ space

Question

Is there an adversarially robust $O(\Delta^2)$-coloring algorithm in semi-streaming space which only needs $\tilde{O}(n)$ random bits?

Our results

Theorem

There is an adversarially robust streaming algorithm for $O\left(\Delta^{2.5}\right)$-coloring using $\tilde{O}\left(n\right)$ space and $\tilde{O}\left(n\Delta\right)$ random bits.

- Space/color tradeoff: for any $\beta \in [0,1]$, get $\tilde{O}\left(\Delta^{5/2-3\beta/2}\right)$ colors with $\tilde{O}\left(n\Delta^{\beta}\right)$ space and $\tilde{O}\left(n\Delta\right)$ random bits.

Theorem

There is an adversarially robust streaming algorithm for $O\left(\Delta^{3}\right)$ coloring using $\tilde{O}\left(n\right)$ space (and no extra random bits).

Our results

Theorem

There is an adversarially robust streaming algorithm for $O\left(\Delta^{2.5}\right)$-coloring using $\tilde{O}\left(n\right)$ space and $\tilde{O}\left(n\Delta\right)$ random bits.

- Space/color tradeoff: for any $\beta \in [0,1]$, get $\tilde{O}\left(\Delta^{5/2-3\beta/2}\right)$ colors with $\tilde{O}\left(n\Delta^{\beta}\right)$ space and $\tilde{O}\left(n\Delta\right)$ random bits.

Theorem

There is an adversarially robust streaming algorithm for $O\left(\Delta^{3}\right)$ coloring using $\tilde{O}\left(n\right)$ space (and no extra random bits).

Our results

Theorem

There is an adversarially robust streaming algorithm for $O\left(\Delta^{2.5}\right)$-coloring using $\tilde{O}\left(n\right)$ space and $\tilde{O}\left(n\Delta\right)$ random bits.

- Space/color tradeoff: for any $\beta \in [0,1]$, get $\tilde{O}\left(\Delta^{5/2-3\beta/2}\right)$ colors with $\tilde{O}\left(n\Delta^{\beta}\right)$ space and $\tilde{O}\left(n\Delta\right)$ random bits.

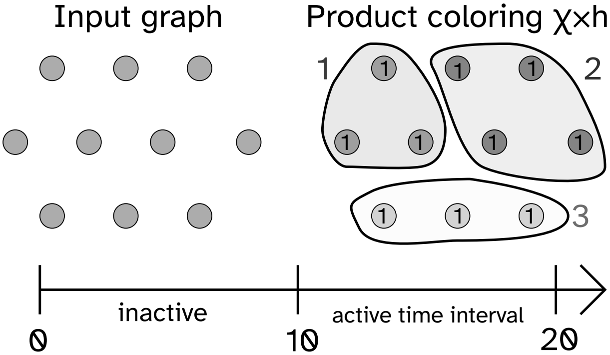
Theorem

There is an adversarially robust streaming algorithm for $O\left(\Delta^{3}\right)$ coloring using $\tilde{O}\left(n\right)$ space (and no extra random bits).

Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

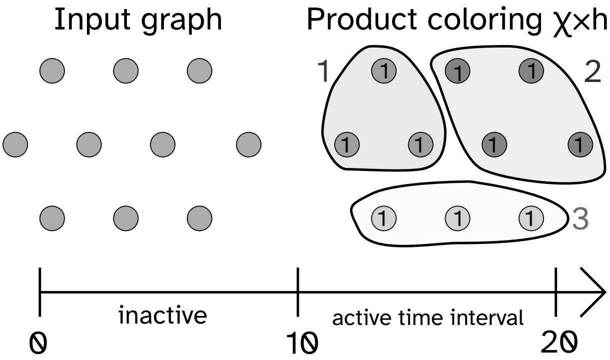
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

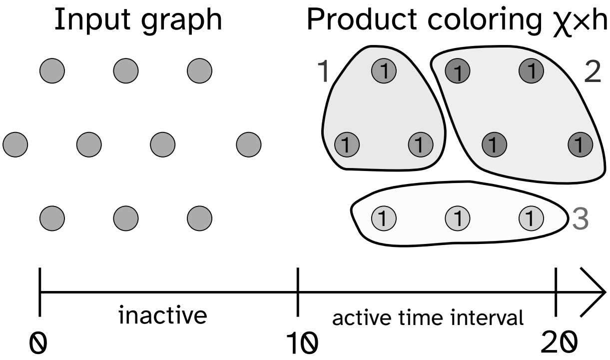
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

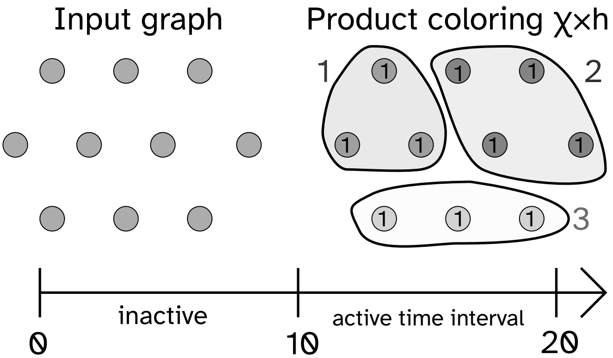
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

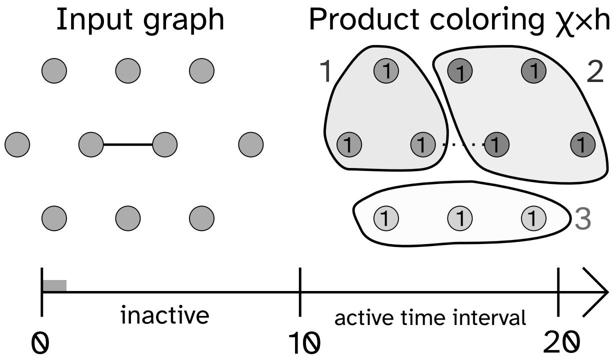
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

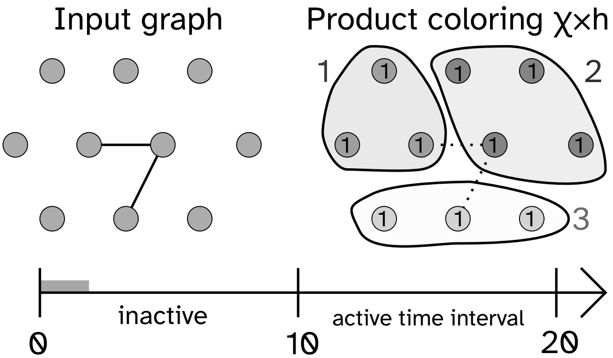
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

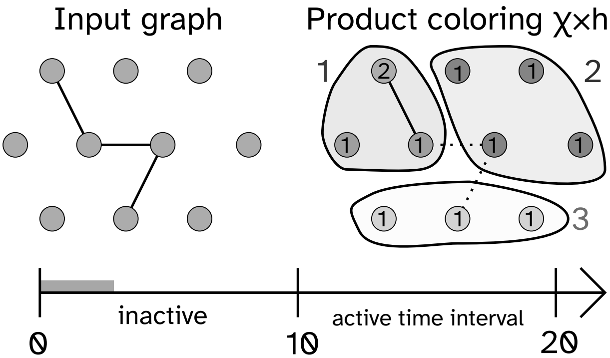
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

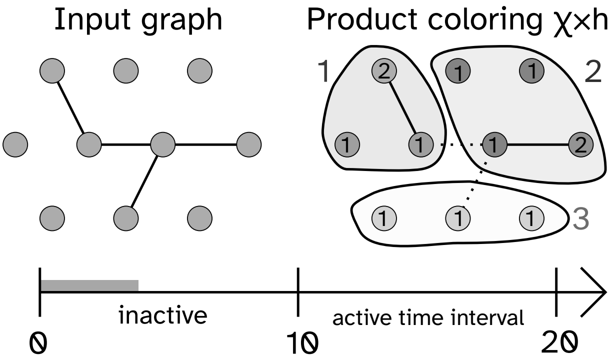
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

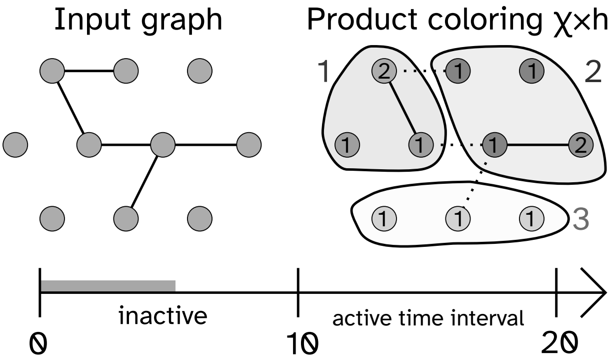
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

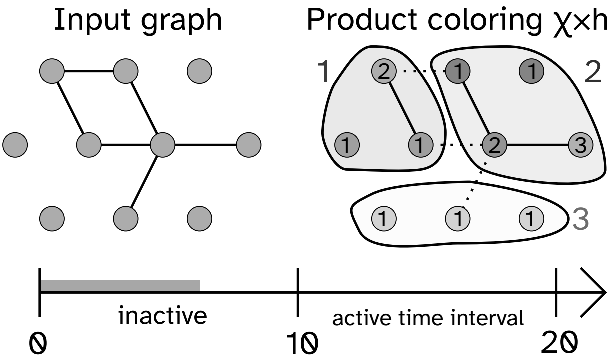
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

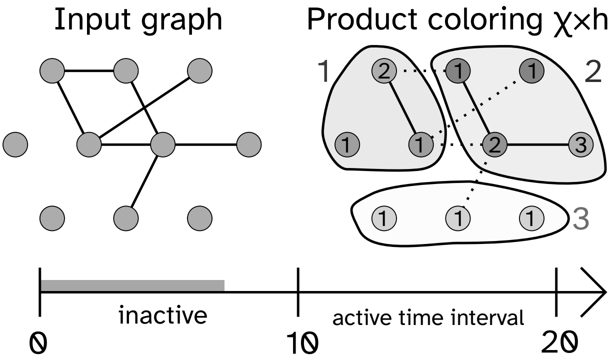
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

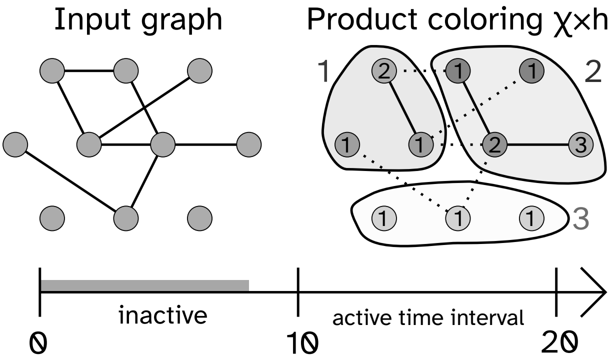
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

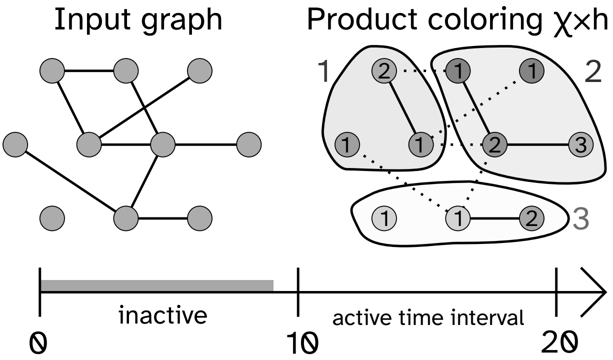
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

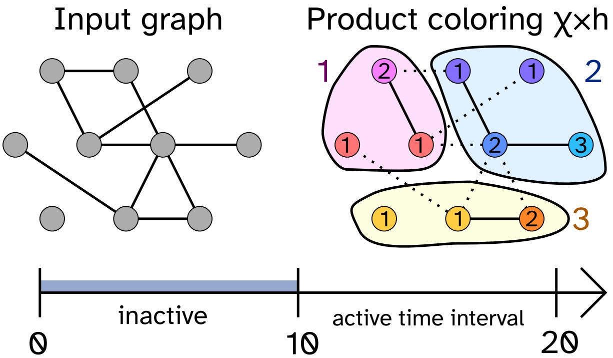
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

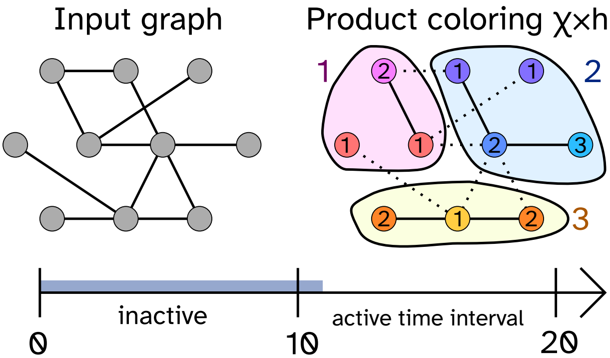
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

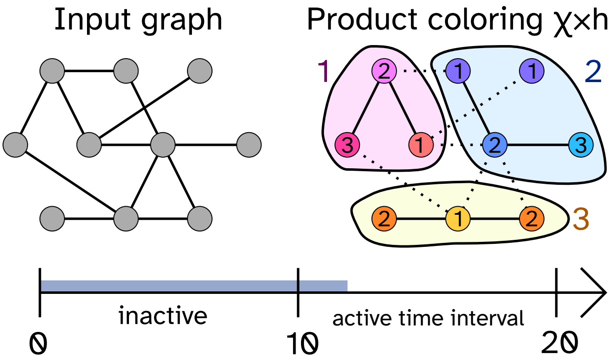
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

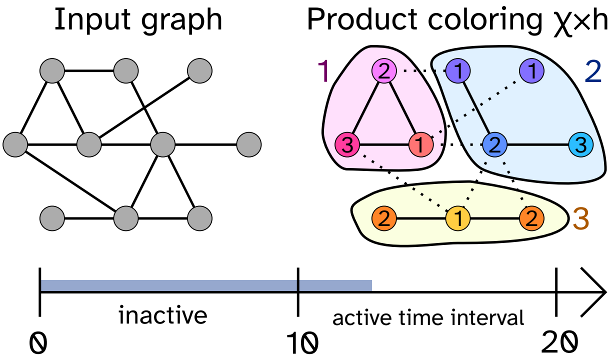
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

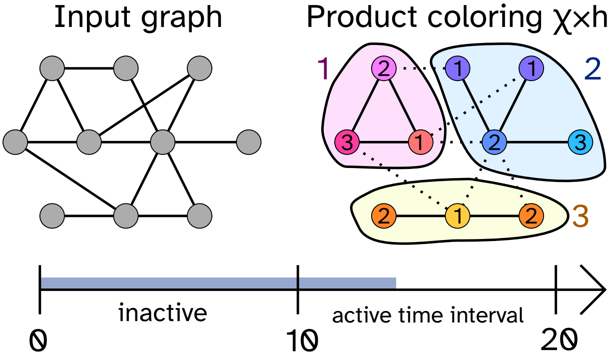
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

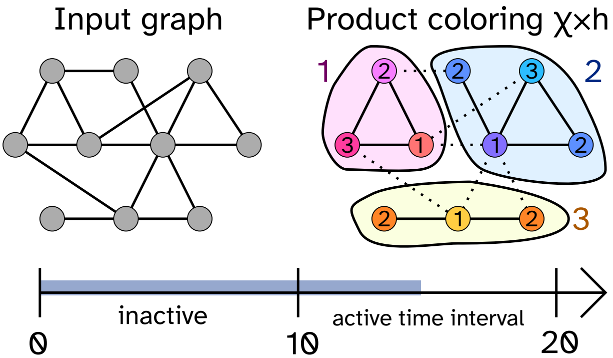
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

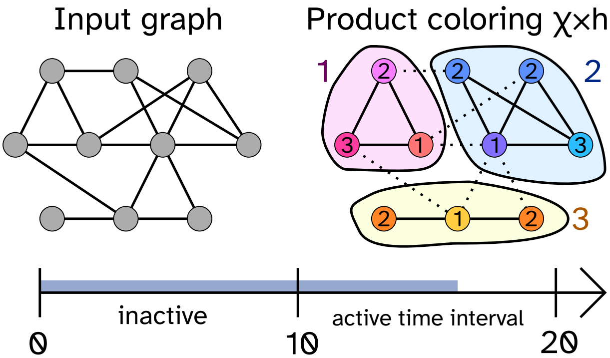
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

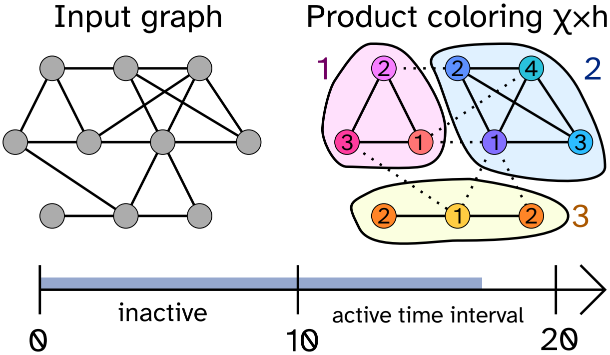
- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



Example: Robust product coloring

- Use random partitions to reduce edges stored / increase colors used
- Adversarial robustness: periodically change partitions to avoid storing many edges

- Random partition $h : V \to [k]$
- Store edge $\{a, b\}$ into set D if $h(a) = h(b)$
- Compute coloring χ of D, and color v with $(h(v), \chi(v))$
- *Before* h is active, $|D|$ is small
- *While* h is active, $|D|$ can grow quickly
- *After* h is active, discard D



High level description of $O\left(\Delta^{2.5}\right)$-coloring algorithm

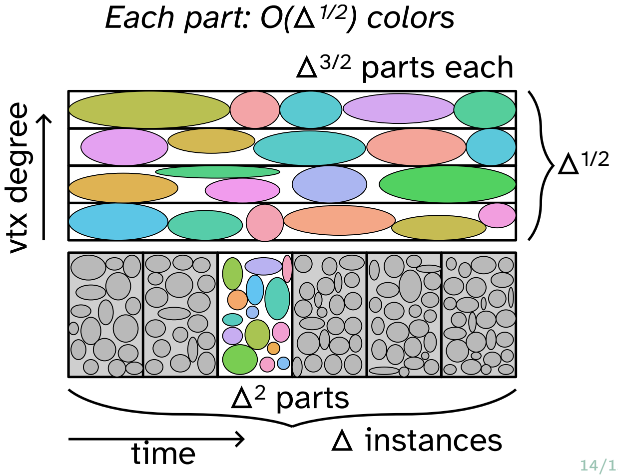
Different forms of product coloring algorithm are efficient for different edge insertion patterns.

"Slow" vertices

- $\leq \sqrt{\Delta}$ incident edges in a batch of n
- Time-linked product coloring instances

"Fast" vertices

- $> \sqrt{\Delta}$ incident edges in a batch of n
- Degree-linked product coloring instances



High level description of $O\left(\Delta^{2.5}\right)$-coloring algorithm

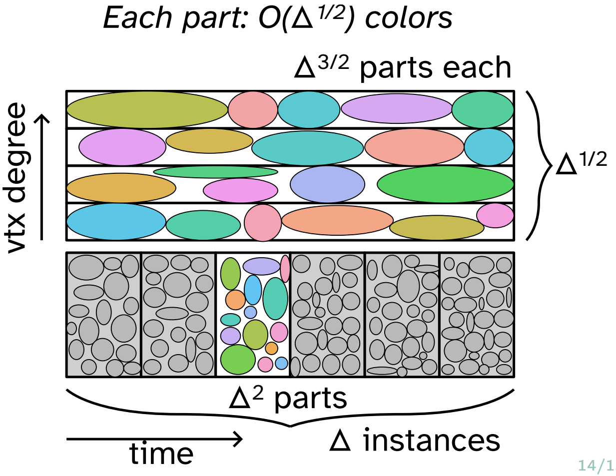
Different forms of product coloring algorithm are efficient for different edge insertion patterns.

"Slow" vertices

- $\leq \sqrt{\Delta}$ incident edges in a batch of n
- Time-linked product coloring instances

"Fast" vertices

- $> \sqrt{\Delta}$ incident edges in a batch of n
- Degree-linked product coloring instances



High level description of $O\left(\Delta^{2.5}\right)$-coloring algorithm

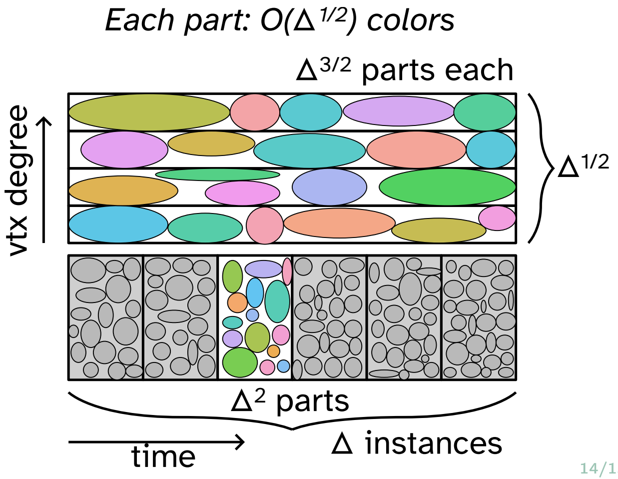
Different forms of product coloring algorithm are efficient for different edge insertion patterns.

"Slow" vertices

- $\leq \sqrt{\Delta}$ incident edges in a batch of n
- Time-linked product coloring instances

"Fast" vertices

- $> \sqrt{\Delta}$ incident edges in a batch of n
- Degree-linked product coloring instances



Summary

- A *multi-pass deterministic* streaming algorithm which outputs a $(\Delta + 1)$-vertex-coloring of an input graph of max degree $\leq \Delta$, using semi-streaming space.
- A single-pass *adversarially robust* streaming algorithm which outputs an $O\left(\Delta^{2.5}\right)$-vertex-coloring of an input graph of max degree $\leq \Delta$, using semi-streaming space (& long random string).
- Open problems:
  - Is there a 2-pass deterministic streaming algorithm in semi-streaming space using $O(\Delta)$ colors?
  - Is there an adversarially robust streaming algorithm in semi-streaming space using $O\left(\Delta^2\right)$ colors, and $\tilde{O}(n)$ bits of randomness?

Summary

- A *multi-pass deterministic* streaming algorithm which outputs a $(\Delta + 1)$-vertex-coloring of an input graph of max degree $\leq \Delta$, using semi-streaming space.
- A single-pass *adversarially robust* streaming algorithm which outputs an $O\left(\Delta^{2.5}\right)$-vertex-coloring of an input graph of max degree $\leq \Delta$, using semi-streaming space (& long random string).
- **Open problems:**
  - Is there a 2-pass deterministic streaming algorithm in semi-streaming space using $O(\Delta)$ colors?
  - Is there an adversarially robust streaming algorithm in semi-streaming space using $O\left(\Delta^2\right)$ colors, and $\tilde{O}(n)$ bits of randomness?