Dynamic graphs & vertex coloring

Daniel Gonçalves, LIRMM, Univ. Montpellier & CNRS

JCRAALMA, 15th January 2024 based on

[BCHN18] *Dynamic algorithms for graph coloring*, by Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai, SODA 2018.

[BCK+19] *Dynamic graph coloring*, by Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot, Algorithmica, 2019.

[HNW20] *Explicit and implicit dynamic coloring of graphs with bounded arboricity*, by Monika Henzinger, Stefan Neumann, and Andreas Wiese, ArXiv 2020.

[CNR23] *Improved Dynamic Colouring of Sparse Graphs*, by Aleksander B.G. Christiansen, Krzysztof Nowicki, and Eva Rotenberg, STOC 2023.

Introduction

$(\Delta + 1)$-coloring with $O(\log n)$ amortized update time

- Warmup
- Algorithm with *Hierarchical Partition*

Coloring with arboricity α

- Limits of explicit colorings
- Implicit & deterministic $2^{O(\alpha)}$-coloring
- Implicit & deterministic $O(\alpha^2)$-coloring

Introduction

$(\Delta + 1)$-coloring with $O(\log n)$ amortized update time

Warmup

Algorithm with *Hierarchical Partition*

Coloring with arboricity α

Limits of explicit colorings

Implicit & deterministic $2^{O(\alpha)}$-coloring

Implicit & deterministic $O(\alpha^2)$-coloring

Dynamic graph

Dynamic graph G:

- A fixed vertex set V, with $n = |V|$.
- A sequence of updates (edge additions/deletions): $(\pm e_i)_{1 \leq i \leq t}$

- Initially, the edge set is empty : $E_0 = \emptyset$.
- If the i^{th} update is a $+e_i$, then $E_i = E_{i-1} \cup \{e_i\}$.
- If the i^{th} update is a $-e_i$, then $E_i = E_{i-1} \setminus \{e_i\}$.

$\Longrightarrow$ a sequence of graphs: $((V, E_i))_{0 \leq i \leq t}$

Dynamic graph

Dynamic graph G:

- A fixed vertex set V, with $n = |V|$.
- A sequence of updates (edge additions/deletions): $(\pm e_i)_{1 \leq i \leq t}$

- Initially, the edge set is empty : $E_0 = \emptyset$.
- If the i^{th} update is a $+e_i$, then $E_i = E_{i-1} \cup \{e_i\}$.
- If the i^{th} update is a $-e_i$, then $E_i = E_{i-1} \setminus \{e_i\}$.

$\Longrightarrow$ a sequence of graphs: $((V, E_i))_{0 \leq i \leq t}$

Combinatorial Problem Π

- We want solutions of Π for the graphs (V, E_i).
- Computing a solution for (V, E_i), should be easier given a solution of (V, E_{i-1}).

Algorithms for dynamic graphs

Updating a solution (for some problem Π) while G evolves

- Given a preset sequence of updates .
- Start with a solution $Sol_0 \in \Pi((V, E_0))$.
- Goal: correct Sol_0 after i updates to obtain Sol_i.

Algorithms for dynamic graphs

Updating a solution (for some problem Π) while G evolves

- Given a preset sequence of updates (oblivious adversary).
- Start with a solution $Sol_0 \in \Pi((V, E_0))$.
- Goal: correct Sol_0 after i updates to obtain Sol_i.

Algorithms for dynamic graphs

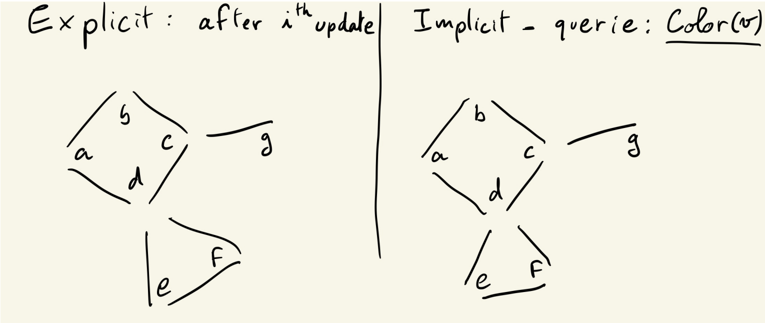
Updating a solution (for some problem Π) while G evolves

- Given a preset sequence of updates (oblivious adversary).
- Start with a solution $Sol_0 \in \Pi((V, E_0))$.
- Goal: correct Sol_0 after i updates to obtain Sol_i.Two ways:
  - **Explicitly**: a current solution is stored in memory.
    Update the solution ($Sol_{i-1} \longrightarrow Sol_i$) so that $Sol_i \in \Pi(V, E_i)$.
  - **Implicitly**: a solution can be retrieved from queries.

Algorithms for dynamic graphs

Updating a solution (for some problem Π) while G evolves

- Given a preset sequence of updates (oblivious adversary).
- Start with a solution $Sol_0 \in \Pi((V, E_0))$.
- Goal: correct Sol_0 after i updates to obtain Sol_i.Two ways:
  - **Explicitly**: a current solution is stored in memory.
    Update the solution $(Sol_{i-1} \longrightarrow Sol_i)$ so that $Sol_i \in \Pi(V, E_i)$.
  - **Implicitly**: a solution can be retrieved from queries.



Time complexity measures

Case of explicit algorithms

- Time complexity **per update**.
  It can be
  Worst-case: maximum complexity for $Sol_{i-1} \longrightarrow Sol_i$,
  or
  Amortized: $Complexity(Sol_0 \longrightarrow Sol_t)\,/\,t$ for a suff. large t.

Time complexity measures

Case of explicit algorithms

- Time complexity **per update**.
  It can be
  Worst-case: maximum complexity for $Sol_{i-1} \longrightarrow Sol_i$,
  or
  Amortized: $Complexity(Sol_0 \longrightarrow Sol_t) \,/\, t$ for a suff. large t.

Case of implicit algorithms

- Twofold complexity: **per update** & **per query**.
  Also, worst-case/amortized options.

Time complexity measures

Case of explicit algorithms

- Time complexity **per update**.
  It can be
  Worst-case: maximum complexity for $Sol_{i-1} \longrightarrow Sol_i$,
  or
  Amortized: $Complexity(Sol_0 \longrightarrow Sol_t) \,/\, t$ for a suff. large t.

Case of implicit algorithms

- Twofold complexity: **per update** & **per query**.
  Also, worst-case/amortized options.

For **randomized algorithms**, the measures can be weakened : e.g. provided in expectation, or with high probability.

Coloring with respect to Δ or the arboricity α

Brooks theorem

$$\chi(G) \leq \Delta + 1$$

Arboricity $\alpha(G)$

- $\alpha(G)$: min k s.t. G decomposes into k forests.

Coloring with respect to Δ or the arboricity α

Brooks theorem

$$\chi(G) \leq \Delta + 1$$

Arboricity $\alpha(G)$

- $\alpha(G)$: min k s.t. G decomposes into k forests.
- Every graph G has at most $\alpha(G) \times (n-1)$ edges.

Coloring with respect to Δ or the arboricity α

Brooks theorem

$$\chi(G) \leq \Delta + 1$$

Arboricity $\alpha(G)$

- $\alpha(G)$: min k s.t. G decomposes into k forests.
- Every graph G has at most $\alpha(G) \times (n-1)$ edges.
- Every graph G is $(2\alpha(G)-1)$-degenerate.

Coloring with respect to Δ or the arboricity α

Brooks theorem

$$\chi(G) \leq \Delta + 1$$

Arboricity $\alpha(G)$

- $\alpha(G)$: min k s.t. G decomposes into k forests.
- Every graph G has at most $\alpha(G) \times (n-1)$ edges.
- Every graph G is $(2\alpha(G)-1)$-degenerate.

$$\chi(G) \leq 2\alpha(G)$$

Colorings in dynamic graphs

Explicit $(\Delta + 1)$-coloring

- $O(\log \Delta)$ expected amortized update time [BCHN18]

Colorings in dynamic graphs

Explicit $(\Delta + 1)$-coloring

- $O(\log \Delta)$ expected amortized update time [BCHN18]

$f(\alpha, n)$-colorings

- Explicit $O(\alpha \log n)$-coloring, with $O(\log^2 n)$ expected amortized update time. [HNW20]

Colorings in dynamic graphs

Explicit $(\Delta + 1)$-coloring

- $O(\log \Delta)$ expected amortized update time [BCHN18]

$f(\alpha, n)$-colorings

- Explicit $O(\alpha \log n)$-coloring, with
  $O(\log^2 n)$ expected amortized update time. [HNW20]
- For explicit $f(\alpha)$-coloring,
  the update time is $\Omega(poly(n))$. [BCK+19]

Colorings in dynamic graphs

Explicit $(\Delta + 1)$-coloring

- $O(\log \Delta)$ expected amortized update time [BCHN18]

$f(\alpha, n)$-colorings

- Explicit $O(\alpha \log n)$-coloring, with
  $O(\log^2 n)$ expected amortized update time. [HNW20]
- For explicit $f(\alpha)$-coloring,
  the update time is $\Omega(poly(n))$. [BCK+19]
- Implicit & deterministic $2^{O(\alpha)}$-coloring, with
  $O(\log^3 n)$ amortized update time, and
  $O(\alpha \log n)$ query time. [HNW20]

Colorings in dynamic graphs

Explicit $(\Delta + 1)$-coloring

- $O(\log \Delta)$ expected amortized update time [BCHN18]

$f(\alpha, n)$-colorings

- Explicit $O(\alpha \log n)$-coloring, with $O(\log^2 n)$ expected amortized update time. [HNW20]
- For explicit $f(\alpha)$-coloring, the update time is $\Omega(poly(n))$. [BCK+19]
- Implicit & deterministic $2^{O(\alpha)}$-coloring, with $O(\log^3 n)$ amortized update time, and $O(\alpha \log n)$ query time. [HNW20]
- Implicit & deterministic $O(\alpha^2)$-coloring, with $O(\log \alpha \log^3 n)$ worst-case update time, and $O(\alpha^5 \log n)$ query time. [CNR23]

Introduction

$(\Delta + 1)$-coloring with $O(\log n)$ amortized update time

- Warmup
- Algorithm with *Hierarchical Partition*

Coloring with arboricity α

- Limits of explicit colorings
- Implicit & deterministic $2^{O(\alpha)}$-coloring
- Implicit & deterministic $O(\alpha^2)$-coloring

Introduction

$(\Delta + 1)$-coloring with $O(\log n)$ amortized update time

Warmup

Algorithm with *Hierarchical Partition*

Coloring with arboricity α

Limits of explicit colorings

Implicit & deterministic $2^{O(\alpha)}$-coloring

Implicit & deterministic $O(\alpha^2)$-coloring

2Δ-coloring with $O(1)$ expected amortized update time

Current coloring stored in a table $c[\cdot]$.

Update algorithm after deleting (u, v)

- Do nothing

Update algorithm after adding (u, v)

- If $c[u] \neq c[v]$: do nothing.
- If $c[u] = c[v]$:
  $c(v) \leftarrow$ pick a color absent from $N(v)$ u.a.r..

2Δ-coloring with $O(1)$ expected amortized update time

Current coloring stored in a table $c[\cdot]$.

Update algorithm after deleting (u, v)

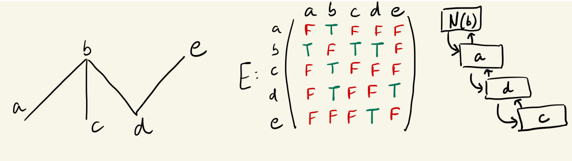
- Do nothing Time $O(1)$

Update algorithm after adding (u, v)

- If $c[u] \neq c[v]$: do nothing. Time $O(1)$
- If $c[u] = c[v]$: This case happen with proba. $\leq 1/\Delta$.
  $c(v) \leftarrow$ pick a color absent from $N(v)$ u.a.r.. Time $O(\Delta)$

2Δ-coloring with $O(1)$ expected amortized update time

Neighborhoods stored with two tables $N[\cdot]$ and $E[\cdot,\cdot]$.



Update algorithm after deleting (u, v)

- Remove u from $N(v)$ & v from $N(u)$ — Time $O(1)$

Update algorithm after adding (u, v)

- Add u in $N(v)$ & v in $N(u)$ — Time $O(1)$
- If $c[u] \neq c[v]$: do nothing. — Time $O(1)$
- If $c[u] = c[v]$: — This case happen with proba. $\leq 1/\Delta$.
  $c(v) \leftarrow$ pick a color absent from $N(v)$ u.a.r.. — Time $O(\Delta)$

$(1+\epsilon)\Delta$-coloring with $O(1/\epsilon)$ exp. amortized update time

Current coloring stored in a table $c[\cdot]$.

Update algorithm after deleting (u, v)

- Do nothing — Time $O(1)$

Update algorithm after adding (u, v)

- If $c[u] \neq c[v]$: do nothing. — Time $O(1)$
- If $c[u] = c[v]$: — This case happen with proba. $\leq 1/\epsilon\Delta$.
  $c(v) \leftarrow$ pick a color absent from $N(v)$ u.a.r.. — Time $O(\Delta)$

When recoloring,
the algorithm picks among at least $(1+\epsilon)\Delta - |N(v)| \geq \epsilon\Delta$ colors.

$(1+\epsilon)\Delta$-coloring with $O(1/\epsilon)$ exp. amortized update time

Current coloring stored in a table $c[\cdot]$.

Update algorithm after deleting (u, v)

- Do nothing — Time $O(1)$

Update algorithm after adding (u, v)

- If $c[u] \neq c[v]$: do nothing. — Time $O(1)$
- If $c[u] = c[v]$: — This case happen with proba. $\leq 1/\epsilon\Delta$.
  $c(v) \leftarrow$ pick a color absent from $N(v)$ u.a.r.. — Time $O(\Delta)$

When recoloring,
the algorithm picks among at least $(1+\epsilon)\Delta - |N(v)| \geq \epsilon\Delta$ colors.

Setting $\epsilon = 1/\Delta$, we have a
$(\Delta+1)$-coloring with $O(\Delta)$ expected amortized update time.

Idea for $(\Delta + 1)$-coloring

When recoloring a vertex v : pick among $O(\Delta)$ colors.

- Maybe few vertices in $\{1, \ldots, \Delta + 1\} \setminus c(N(v))$
- But in that case, many colors of $c(N(v))$ are used only once.

Idea for $(\Delta + 1)$-coloring

When recoloring a vertex v : pick among $O(\Delta)$ colors.

- Maybe few vertices in $\{1, \ldots, \Delta + 1\} \setminus c(N(v))$
- But in that case, many colors of $c(N(v))$ are used only once.

- Pick among the colors used at most once in $N(v)$, there are $\Delta/2$.
- May create a "path of recolorings".
- How to bound the length of this path?

Tool : Hierarchical Partition

Partition of the vertices into levels.

- L levels: $V_1, \ldots, V_L$, with $L = \log_\beta \Delta$ for some $\beta > 20$.
- Level of v: $\ell(v)$

$$N^<(v) = \{u \in N(v) \mid \ell(u) < \ell(v)\}$$
$$N^\leq(v) = \{u \in N(v) \mid \ell(u) \leq \ell(v)\}$$

$$\begin{aligned} &[\text{Large } N^<(v)] : && |N^<(v)| \geq \beta^{\ell(v)} \\ &\left[\text{Small } N^\leq(v)\right] : && C\beta^{\ell(v)} \geq |N^\leq(v)| \end{aligned}$$

$\forall v \in V$ unless $\ell(v) = 0$.

$(\Delta + 1)$-coloring algorithm based on hierarchical partition

$(\Delta + 1)$-coloring : After an update $\pm(u, v)$

1) Insert(u,v) or Delete(u,v)
2) Maintain_HP() assume $\ell(v) \leq \ell(u)$ w.l.o.g.
3) If necessary (i.e. $+(u, v)$ & $c(u) = c(v)$) : $\textsc{Recolor}(v)$

$(\Delta + 1)$-coloring algorithm based on hierarchical partition

$(\Delta + 1)$-coloring : After an update $\pm(u, v)$

1) Insert(u,v) or Delete(u,v)
2) $\mathrm{Maintain\_HP}()$ assume $\ell(v) \leq \ell(u)$ w.l.o.g.
3) If necessary (i.e. $+(u, v)$ & $c(u) = c(v)$) : $\textsc{Recolor}(v)$

$\textsc{Recolor}(v)$

$c(v) \leftarrow$ Pick u.a.r. among colors used at most once in $N(v)$
& not used in $N(v) \setminus N^{<}(v)$

If $\exists w \in N^{<}(v)$ s.t. $c(w) = c(v)$: $\textsc{Recolor}(w)$

$(\Delta + 1)$-coloring algorithm based on hierarchical partition

$(\Delta + 1)$-coloring : After an update $\pm(u, v)$

1) Insert(u,v) or Delete(u,v)
2) $\mathrm{Maintain\_HP}()$ assume $\ell(v) \leq \ell(u)$ w.l.o.g.
3) If necessary (i.e. $+(u, v)$ & $c(u) = c(v)$) : $\textsc{Recolor}(v)$

$\textsc{Recolor}(v)$

$c(v) \leftarrow$ Pick u.a.r. among colors used at most once in $N(v)$
& not used in $N(v) \setminus N^{<}(v)$
If $\exists w \in N^{<}(v)$ s.t. $c(w) = c(v)$: $\textsc{Recolor}(w)$

- Among consecutive recolorings, the level decreases.

Maintaining the Hierarchical Partition (1)

Data structure

- Matrix E
- Doubly chained lists L_i $\qquad \forall i$

For every $v \in V$:

- $\ell(v)$
- $N^{<}(v) = \{u \in N(v) \mid \ell(u) < \ell(v)\}$ & $d^{<}(v) = |N^{<}(v)|$
- $N_i = N(v) \cap L_i$ and $d_i(v) = |N_i(v)|$ $\qquad \forall i$ s.t. $\ell(v) \leq i \leq L$.

For properties [Large $N^{<}(v)$] & $\left[\text{Small } N^{\leq}(v)\right]$

- Q_L & Q_S: Queues with vertices violating these properties

Maintaining the Hierarchical Partition (2)

Insert(u,v)

- $\text{neighb}_u \leftarrow \text{New\_Neighbor}(u)$
- $\text{neighb}_v \leftarrow \text{New\_Neighbor}(v)$
- $E[u,v] \leftarrow (\mathit{TRUE}, \text{neighb}_u, \text{neighb}_v)$

- If $\ell(u) \geq \ell(v)$:
  then: $N_{\ell(u)}(v).\text{add}(\text{neighb}_u)$, $d_{\ell(u)}(v) + +$
  else: $N^{<}(v).\text{add}(\text{neighb}_u)$, $d^{<}(v) + +$
- If $d^{<}(v) + d^{+}_{\ell(v)}(v) > C\beta^{\ell(v)}$ then $Q_S.\text{push}(v)$

- If $\ell(v) \geq \ell(u)$:
  then: $N_{\ell(v)}(u).\text{add}(\text{neighb}_v)$, $d_{\ell(v)}(u) + +$
  else: $N^{<}(u).\text{add}(\text{neighb}_v)$, $d^{<}(u) + +$
- If $d^{<}(u) + d^{+}_{\ell(u)}(u) > C\beta^{\ell(u)}$ then $Q_S.\text{push}(u)$

Maintaining the Hierarchical Partition (3)

Maintain_HP()

If Q_S is not empty, then

- $v \leftarrow Q_S.\text{pop}()$
- $k \leftarrow$ min. level $> \ell(v)$ s.t. $\sum_{i=1}^{k} d_i(v) \leq C\beta^k$
- Move v up to level k, and update data structure

Elseif Q_L is not empty, then

- $v \leftarrow Q_L.\text{pop}()$
- $k \leftarrow$ max. level $< \ell(v)$ s.t. $C\beta^{k-1} \leq \sum_{i=1}^{k-1} d_i(v)$ or level 1
- Move v down to level k, and update data structure

Else return

MAINTAIN_HP()

Maintaining the Hierarchical Partition (3)

Maintain_HP()

If Q_S is not empty, then

- $v \leftarrow Q_S.\text{pop}()$
- $k \leftarrow$ min. level $> \ell(v)$ s.t. $\sum_{i=1}^{k} d_i(v) \leq C\beta^k$
- Move v up to level k, and update data structure

Elseif Q_L is not empty, then

- $v \leftarrow Q_L.\text{pop}()$
- $k \leftarrow$ max. level $< \ell(v)$ s.t. $C\beta^{k-1} \leq \sum_{i=1}^{k-1} d_i(v)$ or level 1
- Move v down to level k, and update data structure

Else return

MAINTAIN_HP()

Property

In both cases we have $\beta^k < C\beta^{k-1} \leq d^{<}(v) \leq d^{\leq}(v) \leq C\beta^k$

Maintaining the Hierarchical Partition (4)

GOAL: $O(\log \Delta) = O(L)$ amortized update time

Budget function

- $Budg(uv) = L - \max(\ell(u), \ell(v))$
- $Budg(v) = \frac{1}{2\beta} \max\left(0, C\beta^{\ell(v)-1} - d^{<}(v)\right)$

Maintaining the Hierarchical Partition (4)

GOAL: $O(\log \Delta) = O(L)$ amortized update time

Budget function

- $Budg(uv) = L - \max(\ell(u), \ell(v))$
- $Budg(v) = \frac{1}{2\beta} \max\left(0, C\beta^{\ell(v)-1} - d^{<}(v)\right)$

When adding an edge uv the budget increase is:
$\Delta Budg(uv) + \Delta Budg(u) + \Delta Budg(v) \leq +L - \frac{1}{2\beta} - \frac{1}{2\beta} < L$

When deleting an edge uv the budget increase is:
$\Delta Budg(uv) + \Delta Budg(u) + \Delta Budg(v) \leq 0 + \frac{1}{2\beta} + \frac{1}{2\beta} < L$

Maintaining the Hierarchical Partition (4)

GOAL: $O(\log \Delta) = O(L)$ amortized update time

Budget function

- $Budg(uv) = L - \max(\ell(u), \ell(v))$
- $Budg(v) = \frac{1}{2\beta} \max\left(0, C\beta^{\ell(v)-1} - d^{<}(v)\right)$

When adding an edge uv the budget increase is:
$\Delta Budg(uv) + \Delta Budg(u) + \Delta Budg(v) \leq +L - \frac{1}{2\beta} - \frac{1}{2\beta} < L$

When deleting an edge uv the budget increase is:
$\Delta Budg(uv) + \Delta Budg(u) + \Delta Budg(v) \leq 0 + \frac{1}{2\beta} + \frac{1}{2\beta} < L$

Remaining to prove

One call to $\textsc{Maintain\_HP}()$ is done in time $O(\beta^{\max(\ell(v),k)})$.
Show that this is $O(-\Delta Budg)$.

Maintaining the Hierarchical Partition (5)

Budget function

- $Budg(uv) = L - \max(\ell(u), \ell(v))$
- $Budg(v) = \frac{1}{2\beta} \max\left(0, C\beta^{\ell(v)-1} - d^{<}(v)\right)$

Maintaining the Hierarchical Partition (5)

Budget function

- $Budg(uv) = L - \max(\ell(u), \ell(v))$
- $Budg(v) = \frac{1}{2\beta} \max\left(0, C\beta^{\ell(v)-1} - d^{<}(v)\right)$

When **moving up** v from level $\ell(v)$ to k, the budget decrease is:

$$\begin{aligned} \Delta Budg(v) + \sum_{u \in N(v)} \Delta Budg(uv) &\geq 0 + \sum_{u \in N(v) \text{ with } \ell(u) < k} \frac{1}{2\beta} \\ &\geq C\beta^{k-1} \frac{1}{2\beta} \\ &\geq O(\beta^{\max(\ell(v),k)}) \end{aligned}$$

Maintaining the Hierarchical Partition (5)

Budget function

- $Budg(uv) = L - \max(\ell(u), \ell(v))$
- $Budg(v) = \frac{1}{2\beta} \max\left(0, C\beta^{\ell(v)-1} - d^{<}(v)\right)$

When **moving up** v from level $\ell(v)$ to k, the budget decrease is:

$$\begin{aligned} \Delta Budg(v) + \sum_{u \in N(v)} \Delta Budg(uv) &\geq 0 + \sum_{u \in N(v) \text{ with } \ell(u) < k} \frac{1}{2\beta} \\ &\geq C\beta^{k-1} \frac{1}{2\beta} \\ &\geq O(\beta^{\max(\ell(v),k)}) \end{aligned}$$

When **moving down** v from level $\ell(v)$ to k: similar ;-)

Introduction

$(\Delta+1)$-coloring with $O(\log n)$ amortized update time

Warmup

Algorithm with *Hierarchical Partition*

Coloring with arboricity α

Limits of explicit colorings

Implicit & deterministic $2^{O(\alpha)}$-coloring

Implicit & deterministic $O(\alpha^2)$-coloring

Introduction

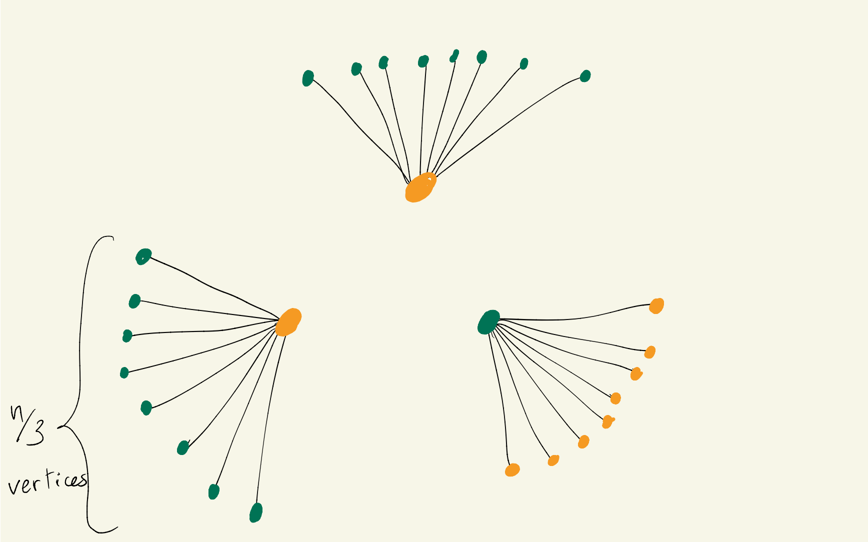
$(\Delta + 1)$-coloring with $O(\log n)$ amortized update time

- Warmup
- Algorithm with *Hierarchical Partition*

Coloring with arboricity α

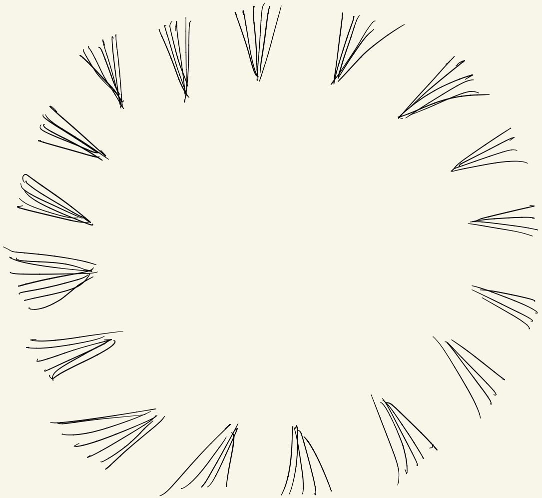
- Limits of explicit colorings
- Implicit & deterministic $2^{O(\alpha)}$-coloring
- Implicit & deterministic $O(\alpha^2)$-coloring

2-colorings of dynamic forests

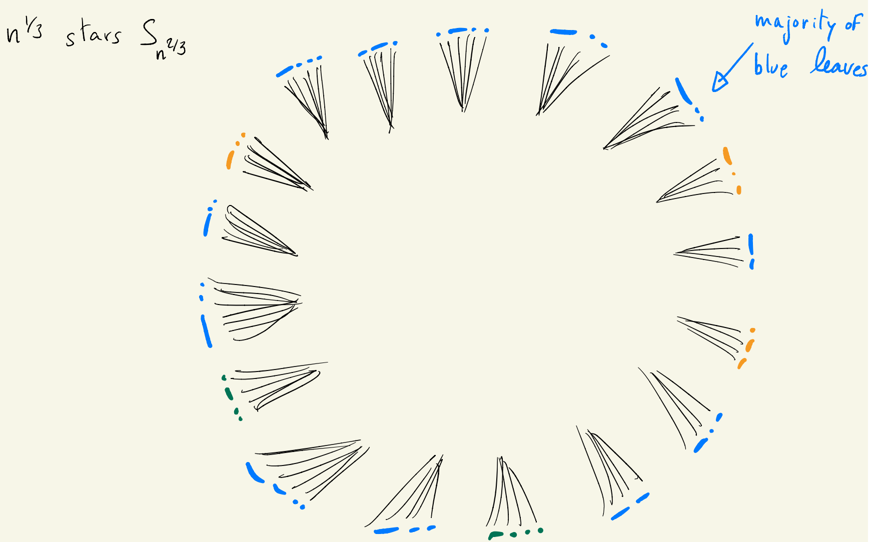


3-colorings of dynamic forests

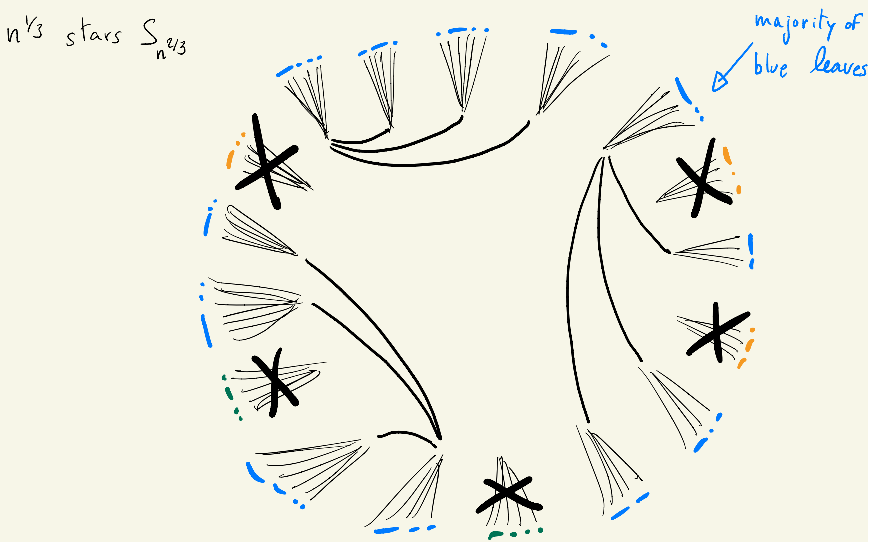
$n^{1/3}$ stars $S_{n^{2/3}}$



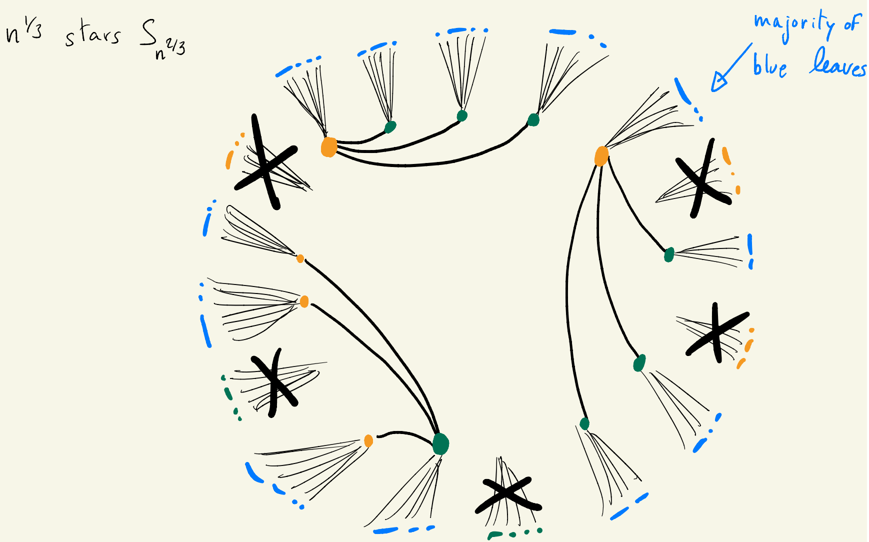
3-colorings of dynamic forests



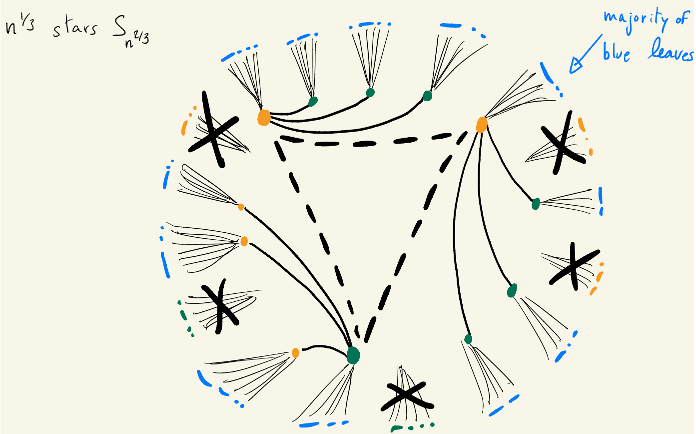
3-colorings of dynamic forests



3-colorings of dynamic forests



3-colorings of dynamic forests



After $O(n^{1/3})$ updates, either

- the thick stars remain 2-colored:

  $O(n^{1/3}) \times O(n^{1/3})$ color changes.
- some thin star has gets a blue root:

  $O(n^{2/3})$ color changes at leaves.

Introduction

$(\Delta + 1)$-coloring with $O(\log n)$ amortized update time

Warmup

Algorithm with *Hierarchical Partition*

Coloring with arboricity α

Limits of explicit colorings

Implicit & deterministic $2^{O(\alpha)}$-coloring

Implicit & deterministic $O(\alpha^2)$-coloring

$O(\alpha)$-forest decomposition

Data structure

- Forests $F_1, \ldots, F_n$
- $\alpha^* = O(\alpha)$ s.t. $F_i = \emptyset \ \ \forall i > \alpha^*$
- $\forall F_i \ \forall T \in F_i$ is a rooted.
- Query : $\textsc{Dist\_to\_root}(v, i)$ $O(\log n)$ time

$O(\alpha)$-forest decomposition

Data structure

- Forests $F_1, \ldots, F_n$
- $\alpha^* = O(\alpha)$ s.t. $F_i = \emptyset \ \ \forall i > \alpha^*$
- $\forall F_i \ \forall T \in F_i$ is a rooted.
- Query : $\textsc{Dist\_to\_root}(v, i)$ $O(\log n)$ time

Query: $\textsc{Color}(v)$

return $(\ldots, \textsc{dist\_to\_root}(v, i) \mod 2, \ldots)$

Reducing to $O(\alpha^4)$ colors

α^*-out orientations

$\forall v \quad d^+(v) \leq \alpha^*$

Reducing to $O(\alpha^4)$ colors

α^*-out orientations

$\forall v \quad d^+(v) \leq \alpha^*$

r-cover free family

There exists a family $\mathcal{S}$ of 2^{α^*} subsets of $\{1, \ldots, \alpha^{*4}\}$ such that:

- $\forall\ S \in \mathcal{S}$
- $\forall\ S_1, \ldots, S_{\alpha^*} \in \mathcal{S}$
- There is a color $c \in S \setminus (\cup S_i)$.

Reducing to $O(\alpha^4)$ colors

α^*-out orientations

$\forall v \quad d^+(v) \leq \alpha^*$

r-cover free family

There exists a family $\mathcal{S}$ of 2^{α^*} subsets of $\{1, \ldots, \alpha^{*4}\}$ such that:

- $\forall\ S \in \mathcal{S}$
- $\forall\ S_1, \ldots, S_{\alpha^*} \in \mathcal{S}$
- There is a color $c \in S \setminus (\cup S_i)$.

2^{α^*}-coloring c_1
$\implies \alpha^{*4}$-coloring c_2

Reducing to $O(\alpha^4)$ colors

α^*-out orientations

$\forall v \quad d^+(v) \leq \alpha^*$

r-cover free family

There exists a family $\mathcal{S}$ of 2^{α^*} subsets of $\{1, \ldots, \alpha^{*4}\}$ such that:

- $\forall\ S \in \mathcal{S}$
- $\forall\ S_1, \ldots, S_{\alpha^*} \in \mathcal{S}$
- There is a color $c \in S \setminus (\cup S_i)$.

2^{α^*}-coloring c_1
$\Longrightarrow \alpha^{*4}$-coloring c_2

$\Longrightarrow \alpha^{*2}$-coloring c_3

Thank you !