(Degee+1)-List Coloring in Congested Clique and MPC

AMG Workshop
DISC, 2023

Sam Coy
University of Warwick

Artur Czumaj
University of Warwick

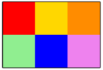
Peter Davies
University of Durham

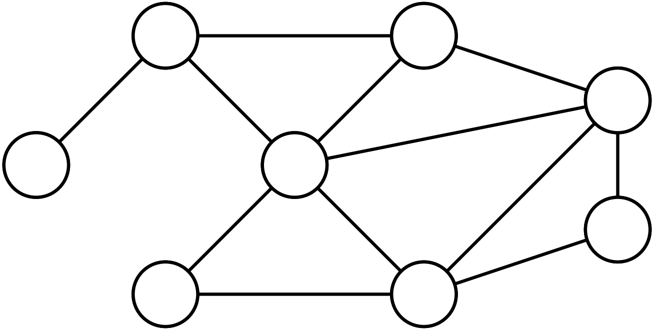
Gopinath Mishra
National University of Singapore

Contents

$(\Delta + 1)$-coloring (Δ1C)

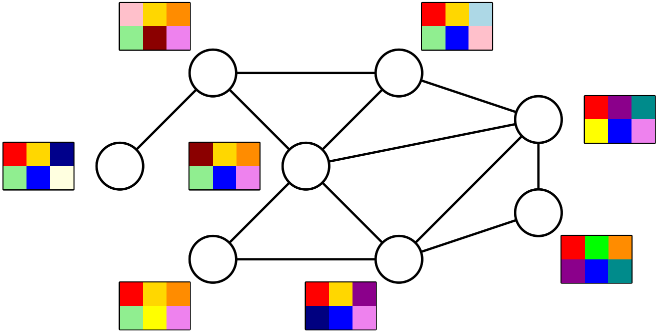
$n = \#$ of nodes $\quad\quad \Delta =$ maximum degree

 $\}\Delta + 1$



Each node has the same palette of $\Delta + 1$ colors, i.e., $\{1, \ldots, \Delta + 1\}$.

Generalizing $(\Delta + 1)$-coloring



$(\Delta + 1)$-list coloring (Δ1LC)

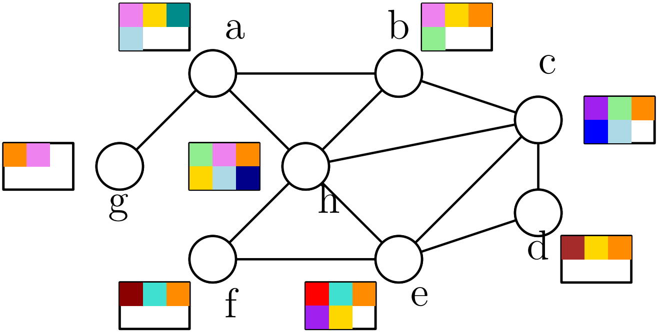
Each node has an arbitrary palette of $(\Delta + 1)$ colors.

D1LC

D1LC: (deg +1)-list coloring

Each node v has a *palette* $\psi(v)$ of colors of size $(d(v)+1)$; the goal is to find a valid coloring. [a]

[a] $d(v)$ denotes the degree of v.



Note: $(\Delta + 1)$-coloring is a special case of D1LC!

Abbreviations

- $\Delta 1C$: $(\Delta + 1)$-coloring.
- $\Delta 1LC$: $(\Delta + 1)$-list coloring.
- D1LC : $(\deg + 1)$-list coloring.

Slack

Slack: number of "spare" colors a node has

$$s(v) = |\psi(v)| - d(v)$$

In D1LC each node may start with exactly 1 slack.

Distributed and parallel coloring

- Greedy algorithms in the centralized setting are in general hard to parallelize;
- Coloring is harder in distributed and parallel settings.

Complexity of Δ1C Coloring

C-Clique : Congested Clique, S-MPC : Sublinear-MPC, R : Randomized, and D : Deterministic.

Model	R/D	Δ1C	Reference	D1LC	Reference
Local	R	$\widetilde{O}(\log^2 \log n)$	[CLP20]		
Local	D	$\widetilde{O}(\log^2 n)$	[GG23]		
Congest	R	$\widetilde{O}(\log^3 \log n)$	[HKMT21]		
Congest	D	$\widetilde{O}(\log^3 n)$	[GK21]		
C-Clique	R	$O(1)$	[CFGUZ19]		
C-Clique	D	$O(1)$	[CDP21]		
S-MPC	R	$O(\log \log \log n)$	[CFGUZ19]		
S-MPC	D	$O(\log \log \log n)$	[CDP21]		

Complexity of Δ1C vs. D1LC

C-Clique : Congested Clique, S-MPC : Sublinear-MPC, R : Randomized, and D : Deterministic.

Model	R/D	Δ1C	Reference	D1LC	Reference
Local	R	$\widetilde{O}(\log^2 \log n)$	[CLP20]	$\widetilde{O}(\log^2 \log n)$	[HKNT22]
Local	D	$\widetilde{O}(\log^2 n)$	[GG23]	$\widetilde{O}(\log^2 n)$	[GG23]
Congest	R	$\widetilde{O}(\log^3 \log n)$	[HKMT21]	$\widetilde{O}(\log^3 \log n)$	[HNT22]
Congest	D	$\widetilde{O}(\log^3 n)$	[GK21]	$\widetilde{O}(\log^3 n)$	[GK21]
C-Clique	R	$O(1)$	[CFGUZ19]		
C-Clique	D	$O(1)$	[CDP21]		
S-MPC	R	$O(\log \log \log n)$	[CFGUZ19]		
S-MPC	D	$O(\log \log \log n)$	[CDP21]		

Complexity of Δ1C vs D1LC

C-Clique : Congested Clique, S-MPC : Sublinear-MPC, R : Randomized, and D : Deterministic.

Model	R/D	Δ1C	Reference	D1LC	Reference
Local	R	$\widetilde{O}(\log^2 \log n)$	[CLP20]	$\widetilde{O}(\log^2 \log n)$	[HKNT22]
Local	D	$\widetilde{O}(\log^2 n)$	[GG23]	$\widetilde{O}(\log^2 n)$	[GG23]
Congest	R	$\widetilde{O}(\log^3 \log n)$	[HKMT21]	$\widetilde{O}(\log^3 \log n)$	[HNT22]
Congest	D	$\widetilde{O}(\log^3 n)$	[GK21]	$\widetilde{O}(\log^3 n)$	[GK21]
C-Clique	R	$O(1)$	[CFGUZ19]	$O(1)$	[CCD**M**23b]
C-Clique	D	$O(1)$	[CDP21]	$O(1)$	[CCD**M**23b]
S-MPC	R	$O(\log \log \log n)$	[CFGUZ19]	$O(\log \log \log n)$	[CCD**M**23a]
S-MPC	D	$O(\log \log \log n)$	[CDP21]	$O(\log \log \log n)$	[CCD**M**23a]

Rest of the talk

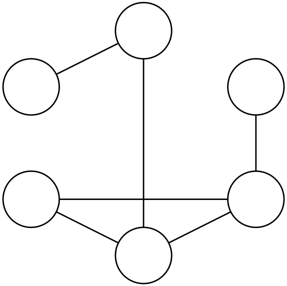
- D1LC in Congested Clique;
- D1LC in Sublinear MPC.

Contents

Contents

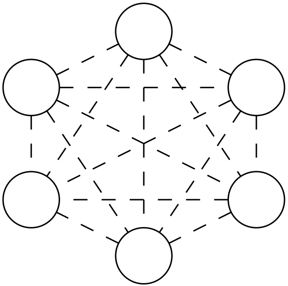
Congested Clique Model [LPP05]

- Input is a graph
- Each node is a computer
- Nodes have unique $O(\log n)$ bit IDs
- Nodes start with a list of neighbors
- Want to compute something about the input graph



Congested Clique Model [LPP05]

- Computation in rounds:
  - Nodes do local computation
  - Nodes exchange messages
- Aim to minimize #rounds
- Each round, each node can send an $O(1)$ word message to each other node
  - Node/color IDs are 1 word



Congested Clique Model [LPP05]

Lenzen's Routing [Len13]

If each node wants to send and receive $O(n)$ messages in total, these messages can all be routed in $O(1)$ rounds.

If we have a coloring instance of size $O(n)$...

...we can send it to a single node and color it locally!

Congested Clique Model [LPP05]

Lenzen's Routing [Len13]

If each node wants to send and receive $O(n)$ messages in total, these messages can all be routed in $O(1)$ rounds.

If we have a coloring instance of size $O(n)$...
...we can send it to a single node and color it locally!

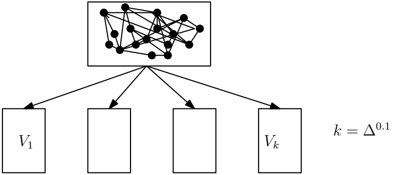
Contents

$O(1)$ round algorithm for Δ1C in Congested Clique

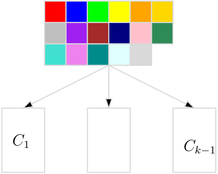
- Randomized algorithm by Chang et al. [CFGUZ19];
- Deterministic algorithm by Czumaj et al. [CDP20].

$O(1)$ round algorithm for Δ1C in C-Clique [CDP20]

Nodes are partitioned into buckets randomly.

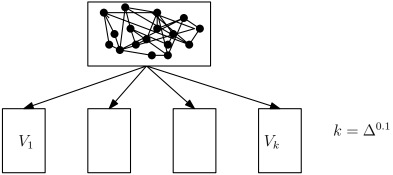


Colors are partitioned into buckets randomly.

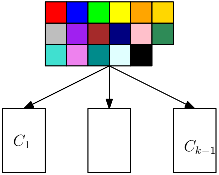


$O(1)$ round algorithm for Δ1C in C-Clique [CDP20]

Nodes are partitioned into buckets randomly.

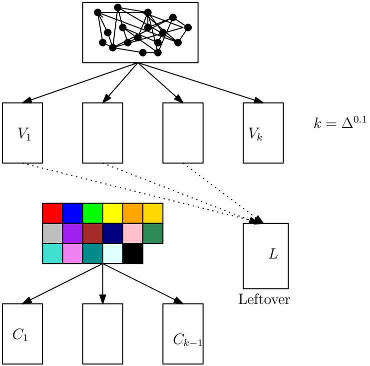


Colors are partitioned into buckets randomly.



$O(1)$ round algorithm for Δ1C in C-Clique by [CDP20]

- G_i: subgraph induced by V_i, $i \in [k-1]$;
- Consider coloring G_i restricting palettes to C_i, $i \in [k-1]$;
- $L \subset \cup_{i=1}^{k-1} V_i$: leftover vertices that possibly can't be colored;
- Color G_i, $i \in [k-1]$, in parallel;
- Color $H = (V_k, E_k)$ recursively;
- Color the vertices in L.



$O(1)$ round algorithm for Δ1C in C-Clique by [CDP20]

The main crux of the analysis

- The size of the subgraphs induced by each $G_i, i \in [k-1]$ and L is $O(n)$;
- After $O(1)$ recursive calls, we have an $O(n)$ size instance.

Recall Lenzen's routing!

Difficulty in D1LC

- In Δ1C/Δ1LC, the low degree vertices are easy case;
- In D1LC, it is difficult as random partitioning based on Δ won't work;
- Using too few buckets, the size of the induced subgraphs will be too large;
- Using too many buckets, we can't guarantee on the colorability of the reduced instances.

Difficulty in D1LC

- In Δ1C/Δ1LC, the low degree vertices are easy case;
- In D1LC, it is difficult as random partitioning based on Δ won't work;
- Using too few buckets, the size of the induced subgraphs will be too large;
- Using too many buckets, we can't guarantee on the colorability of the reduced instances.

Difficulty in D1LC

- In Δ1C/Δ1LC, the low degree vertices are easy case;
- In D1LC, it is difficult as random partitioning based on Δ won't work;
- Using too few buckets, the size of the induced subgraphs will be too large;
- Using too many buckets, we can't guarantee on the colorability of the reduced instances.

Contents

An $O(\log\log\Delta)$ algorithm

Observation

The $O(1)$ Δ1C algorithm by [CDP20] can be adapted to D1LC when the degree of each vertex lies in $[\Delta^{\varepsilon}, \Delta]$.

$O(\log\log\Delta)$ algorithm

Color the graph in $O(\log\log\Delta)$ phases each of $O(1)$ rounds:

- Phase 1: consider coloring vertices with degree range $[\Delta^{\varepsilon}, \Delta]$.
- Phase 2: consider coloring with degree range $[\Delta^{\varepsilon^2}, \Delta^{\varepsilon}]$.
- So on...

An $O(\log\log\Delta)$ algorithm

Observation

The $O(1)$ Δ1C algorithm by [CDP20] can be adapted to D1LC when the degree of each vertex lies in $[\Delta^{\varepsilon}, \Delta]$.

$O(\log\log\Delta)$ algorithm

Color the graph in $O(\log\log\Delta)$ phases each of $O(1)$ rounds:

- Phase 1: consider coloring vertices with degree range $[\Delta^{\varepsilon}, \Delta]$.
- Phase 2: consider coloring with degree range $[\Delta^{\varepsilon^2}, \Delta^{\varepsilon}]$.
- So on...

Our approach [CCD**M**23b]

Note that we have just one slack for each vertex in D1LC.

- We give an $O(1)$ algorithm BucketColor when each vertex v has relatively more colors than its high order neighbors, i.e.,

$$d^+(v) \le p(v) - \frac{1}{4}d(v)^{0.9}.$$ [1]

// In the worst case, $d^+(v) = p(v) - 1$.

- Then extend it to D1LC by generating slack in $O(1)$ rounds to satisfy the need of BucketColor.

[1] $d(v) =$ degree of v,
$d^+(v) =$ the number of higher order neighbors of v,
$p(v) =$ palette size of v.

Our approach [CCD**M**23b]

Note that we have just one slack for each vertex in D1LC.

- We give an $O(1)$ algorithm BucketColor when each vertex v has relatively more colors than its high order neighbors, i.e.,
$$d^+(v) \le p(v) - \frac{1}{4} d(v)^{0.9}.\ ^1$$
// In the worst case, $d^+(v) = p(v) - 1$.
- Then extend it to D1LC by generating slack in $O(1)$ rounds to satisfy the need of BucketColor.

[1] $d(v) =$ degree of v,
$d^+(v) =$ the number of higher order neighbors of v,
$p(v) =$ palette size of v.

Our approach [CCD**M**23b]

Note that we have just one slack for each vertex in D1LC.

- We give an $O(1)$ algorithm BucketColor when each vertex v has relatively more colors than its high order neighbors, i.e.,
$$d^+(v) \le p(v) - \frac{1}{4}d(v)^{0.9}.$$ [1]
// In the worst case, $d^+(v) = p(v) - 1$.
- Then extend it to D1LC by generating slack in $O(1)$ rounds to satisfy the need of BucketColor.

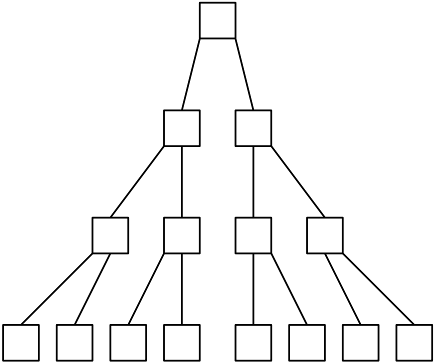
[1] $d(v)$ = degree of v,
$d^+(v)$ = the number of higher order neighbors of v,
$p(v)$= palette size of v.

Algorithm BucketColor

Hierarchical bucketing

A tree of buckets:

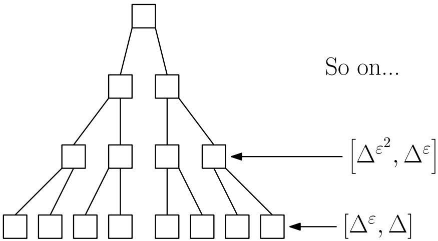
- $O(\log \log \Delta)$ levels;
- Determined by random strings;
- The length of the string is a function of the level of the bucket;
- Ancestor relationship based on substring.



Algorithm BucketColor

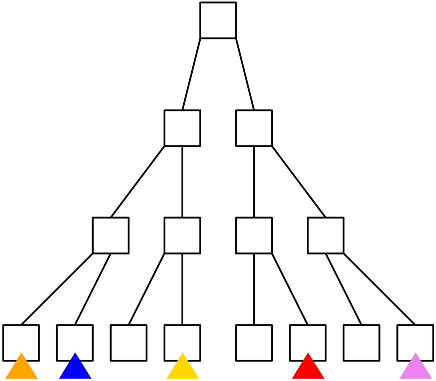
Hierarchical bucketing

- Vertices of degree $[\Delta^{\varepsilon}, \Delta]$ put into one of the leaf bucket randomly;
- Vertices of degree $[\Delta^{\varepsilon^2}, \Delta^{\varepsilon}]$ put into one of buckets above the leaves randomly;
- So on..



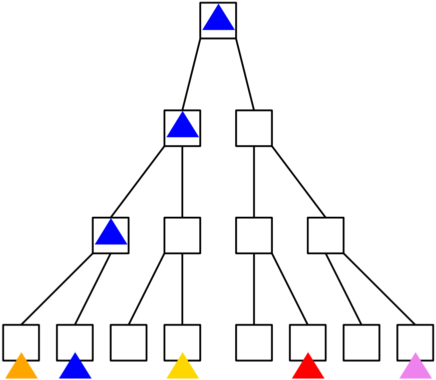
Algorithm BucketColor

- Assign colors randomly to leaves of bucket tree.



Algorithm BucketColor

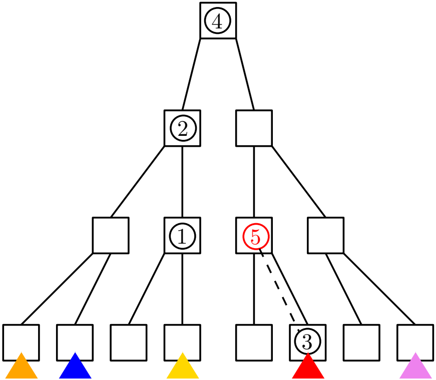
- Color present in a leaf bucket can be considered to be present in any bucket in the leaf to the root path.



Algorithm BucketColor

We have an instance where palettes of the vertices are sparsified:

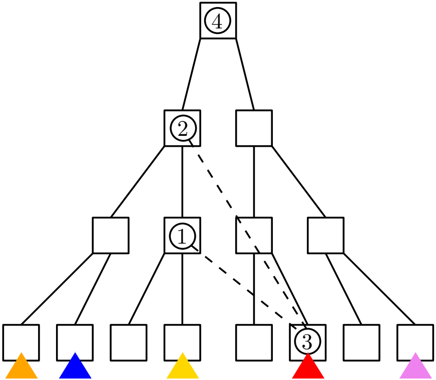
- each vertex would like to use colors in descendant buckets only.



Algorithm BucketColor

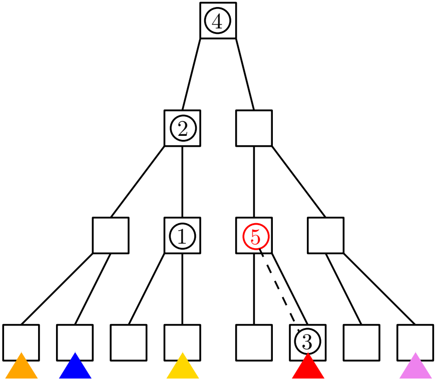
We have a sparsified instance

- where edges between unrelated nodes don't matter.



Algorithm BucketColor

- Would like to use colors in descendant buckets only;
- Should have more descendant colors than descendant neighbors;



Colorability guarantee

This is possible when each node v has relatively more color than the number of high order neighbors :
assumption of BUCKETCOLOR.

Claim

If $d^+(v) \leq p(v) - \frac{1}{4}d(v)^{0.9}$, then with probability $1 - 1/d(v)^2$, v has relatively more colors in its descendant buckets than that of the number of neighbors.

Colorability guarantee

This is possible when each node v has relatively more color than the number of high order neighbors :
assumption of BUCKETCOLOR.

Claim

If $d^+(v) \leq p(v) - \frac{1}{4}d(v)^{0.9}$, then with probability $1 - 1/d(v)^2$, v has relatively more colors in its descendant buckets than that of the number of neighbors.

After putting nodes and colors into buckets

Informally,

Algorithm BUCKETCOLOR removes some bad nodes, such that

- The size of the relevant information corresponding to each bucket is $O(n)$ — can be gathered onto a network node;
- The subgraph induced by the bad node is of $O(n)$ size — can be colored later in $O(1)$ rounds;
- Every node has more colors in the descendant buckets than that of its neighbors.

After putting nodes and colors into buckets

Informally,

Algorithm BUCKETCOLOR removes some bad nodes, such that

- The size of the relevant information corresponding to each bucket is $O(n)$ — can be gathered onto a network node;
- The subgraph induced by the bad node is of $O(n)$ size — can be colored later in $O(1)$ rounds;
- Every node has more colors in the descendant buckets than that of its neighbors.

After putting nodes and colors into buckets

Informally,

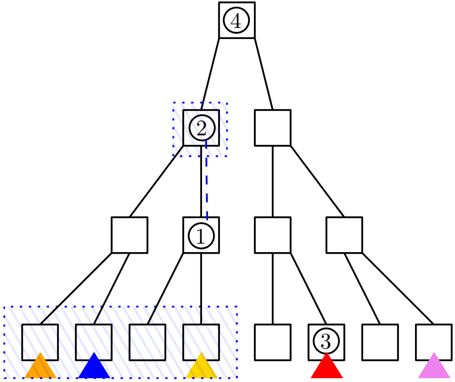
Algorithm BUCKETCOLOR removes some bad nodes, such that

- The size of the relevant information corresponding to each bucket is $O(n)$ — can be gathered onto a network node;
- The subgraph induced by the bad node is of $O(n)$ size — can be colored later in $O(1)$ rounds;
- Every node has more colors in the descendant buckets than that of its neighbors.

Relevant information of a bucket

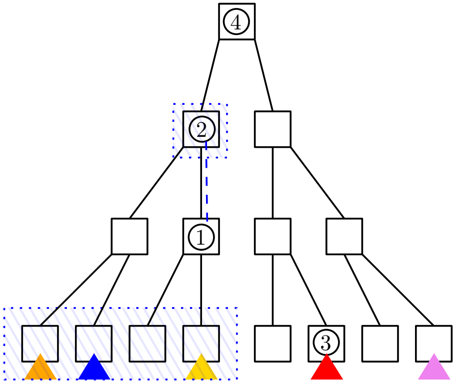
It refers to a valid coloring instance:

- Graph with the set of edges with one node in the bucket and the other in some descendant bucket;
- Color palette of a node is the set of colors present in the descendant buckets;



Dependency between buckets

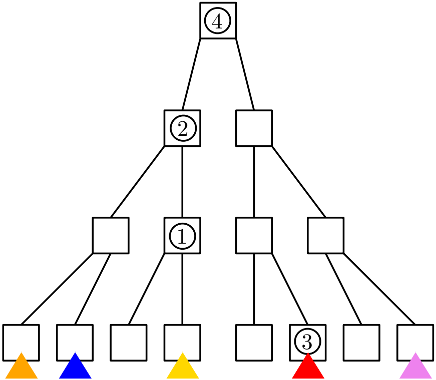
- The valid coloring instances w.r.t two buckets are not necessarily independent;
- Particularly, consider buckets with ancestor-descendant relationship.



Find good child buckets in parallel

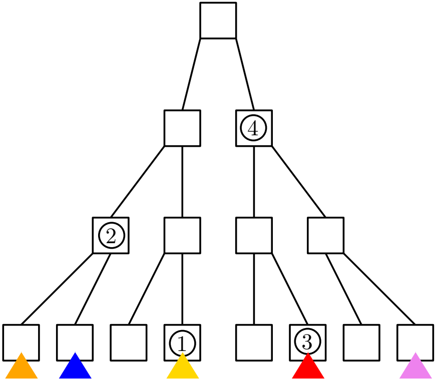
However,

- Each node can find a good child bucket . . .
- . . . such that all nodes have more descendant colors than descendant neighbors.



Find good child buckets in parallel

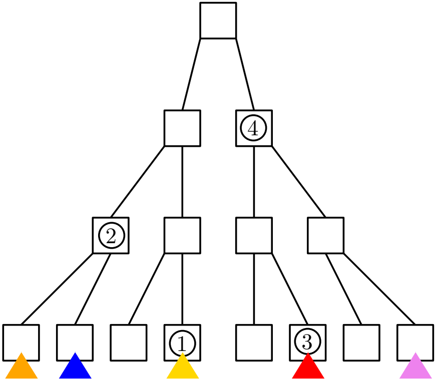
- All nodes can find good child buckets in parallel.



Repeated moving nodes to good child buckets

$O(1)$ steps, in parallel:

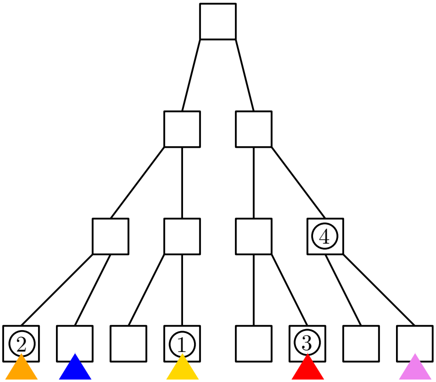
- Move all nodes to a child bucket
- . . . such that all nodes have more descendant colors than descendant neighbors.



Repeated moving nodes to good child buckets

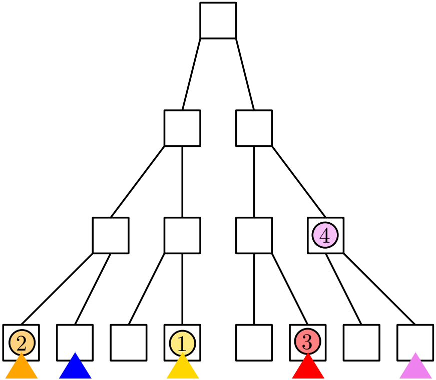
After $O(1)$ steps:

- All nodes have exactly 1 descendant color;
- All nodes have zero descendant neighbors.



Coloring nodes after $O(1)$ steps

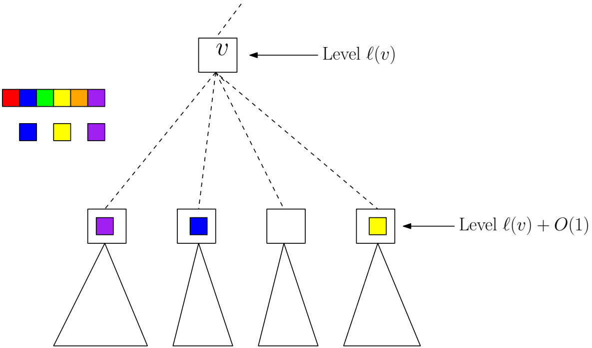
Assign all nodes their remaining color!



Why $O(1)$ rounds are enough?

Claim

W.p. $1 - 1/d(v)^2$, none of the v's descendants buckets of level $\ell(v) + O(1)$ has more than one $v's$ palette color.



Algorithm BucketColor

BucketColor

It can color the graph in $O(1)$ rounds if each v has relatively more color than the number of higher order neighbors, i.e.,

$$d^+(v) \leq p(v) - \frac{1}{4}d(v)^{0.9}.$$

How to generate the required slack for BucketColor?

Algorithm BucketColor

BucketColor

It can color the graph in $O(1)$ rounds if each v has relatively more color than the number of higher order neighbors, i.e.,

$$d^+(v) \leq p(v) - \frac{1}{4} d(v)^{0.9}.$$

How to generate the required slack for BucketColor?

How to generate the required slack for BucketColor?

- By using ColorTrial.

  - Nodes nominate themselves with probability $1/4$
  - Nominated nodes pick a random color from their palettes
  - …and permanently color themselves if no neighbor picked the same color

- Delaying coloring of node v w.p. $d(v)^{-0.1}$.

How to generate the required slack for BucketColor?

- By using ColorTrial.

  - Nodes nominate themselves with probability $1/4$
  - Nominated nodes pick a random color from their palettes
  - ...and permanently color themselves if no neighbor picked the same color

- Delaying coloring of node v w.p. $d(v)^{-0.1}$.

Algorithm

Overall structure of COLOR(G):

- Generate some slack with COLORTRIAL;
- $S \leftarrow$ Nodes delay themselves w.p. $d(v)^{-0.1}$;
- Run BUCKETCOLOR to color $G \setminus S$;
- Call COLOR($G[S]$) recusrsively.

Algorithm

Overall structure of COLOR(G):

- Generate some slack with COLORTRIAL;
- $S \leftarrow$ Nodes delay themselves w.p. $d(v)^{-0.1}$;
- Run BUCKETCOLOR to color $G \setminus S$;
- Call COLOR($G[S]$) recusrsively.

Algorithm

Overall structure of COLOR(G):

- Generate some slack with COLORTRIAL;
- $S \leftarrow$ Nodes delay themselves w.p. $d(v)^{-0.1}$;
- Run BUCKETCOLOR to color $G \setminus S$;
- Call COLOR($G[S]$) recusrsively.

These steps gives us
"slack relative to high-degree neighbors" for BUCKETCOLOR.

Algorithm [CCDM23b]

Overall structure of $\textsc{Color}(G)$:

- Generate some slack with $\textsc{ColorTrial}$;
- $S \leftarrow$ Nodes delay themselves w.p. $d(v)^{-0.1}$;
- Run $\textsc{BucketColor}$ to color $G \setminus S$;
- Call $\textsc{Color}(G[S])$ recursively.

Note

- The size of the remaining graph in $O(n)$ after $O(1)$ level of recursion.
- Some nodes may fail. But that can be handled suitably.

Algorithm [CCDM23b]

Overall structure of $\textsc{Color}(G)$:

- Generate some slack with $\textsc{ColorTrial}$;
- $S \leftarrow$ Nodes delay themselves w.p. $d(v)^{-0.1}$;
- Run $\textsc{BucketColor}$ to color $G \setminus S$;
- Call $\textsc{Color}(G[S])$ recursively.

Note

- The size of the remaining graph in $O(n)$ after $O(1)$ level of recursion.
- Some nodes may fail. But that can be handled suitably.

Derandomization

- Several randomized subroutines in our algorithm;
- We can derandomize them all with the *method of conditional expectations*.

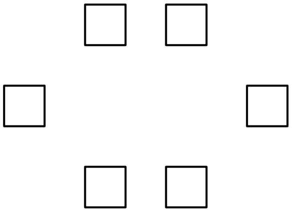
Contents

- Introduction
- D1LC in Congested Clique
  - The model
  - Overview of $O(1)$ round algorithm for $(\Delta + 1)$-coloring
  - $O(1)$ round algorithm for $(\deg +1)$-list coloring
- D1LC in Sublinear MPC
  - The model
  - The main crux of the $(\Delta + 1)$-coloring algorithm
  - Our approach for $(\deg +1)$-list coloring
- Open Questions

Contents

Massively Parallel Computation (MPC) [KSV10]

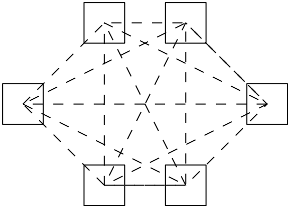
- Edges of the graph is divided among machines [a]
- Each machine has local space S



[a] Unlike, the distributed models, the machines does not necessarily correspond to nodes in the input graph.

Massively Parallel Computation (MPC)

- Communication is done via all-to-all mode over rounds
- In each round,
  - Machines can do some local computation
  - Send/receive at most S words
- Optimization parameters:
  - Primary: # rounds
  - Secondary: total space used by all machines (ideally $O(m+n)$)



Different MPC based on local space

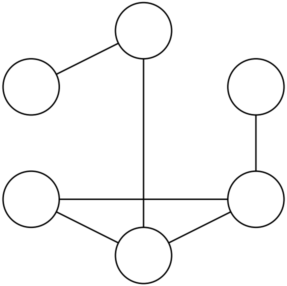
Superlinear MPC: $S = \Omega(n^{1+\delta})$, where $\delta \in (0, 1)$

Linear MPC: $S = \Theta(n)$

Sub-linear MPC: $S = \mathcal{O}(n^{\delta})$, where $\delta \in (0, 1)$

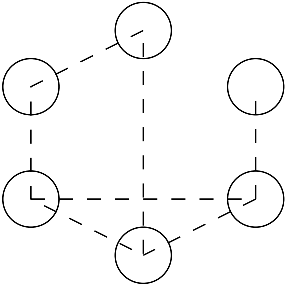
Local model [Lin92]

- Input is a graph
- Each node is a computer
- Nodes have unique $O(\log n)$ bit IDs
- Nodes start with a list of neighbors
- Want to compute something about the input graph



Local model [Lin92]

- Computation in rounds:
  - Nodes do local computation
  - Nodes exchange messages
- Aim to minimize #rounds
- In each round, each node can send message of any size to its neighbors only



Contents

- Introduction
- D1LC in Congested Clique
  - The model
  - Overview of $O(1)$ round algorithm for $(\Delta + 1)$-coloring
  - $O(1)$ round algorithm for $(\deg +1)$-list coloring
- D1LC in Sublinear MPC
  - The model
  - The main crux of the $(\Delta + 1)$-coloring algorithm
  - Our approach for $(\deg +1)$-list coloring
- Open Questions

$O(\log\log\log n)$ algorithm for Δ1LC coloring in Sublinear-MPC

- Randomized algorithm first given by Chang et al. [CFGUZ19];
- Deterministic algorithm by Czumaj et al. [CDP21]

The main idea

- Consider the local algorithm for Δ1LC by Chang et al. [CLP20];
- Simulate and/or derandomize the local procedures in [CLP20] one by one in sublinear MPC.

$O(\log\log\log n)$ algorithm for Δ1LC coloring in Sublinear-MPC

- Randomized algorithm first given by Chang et al. [CFGUZ19];
- Deterministic algorithm by Czumaj et al. [CDP21]

The main idea

- Consider the local algorithm for Δ1LC by Chang et al. [CLP20];
- Simulate and/or derandomize the local procedures in [CLP20] one by one in sublinear MPC.

$O(\log\log\log n)$ deterministic algorithm for $D1LC$ in Sublinear MPC

A possible approach:

Consider the randomized algorithm for D1LC in local model by Halldórsson et al. [HKMT22]

- Simulation and derandomization of each local procedure of [HKMT22].

Contents

Our main contribution [CCD**M**23a]

- We define a generic local procedure (for coloring kind of problems);
- We show that those can be simulated deterministically in sublinear MPC;
- All the local procedures in the local algorithm by Halldórsson et al. [HKNT22] fits this framework.

Our main contribution [CCD**M**23a]

- We define a generic local procedure (for coloring kind of problems);
- We show that those can be simulated deterministically in sublinear MPC;
- All the local procedures in the local algorithm by Halldórsson et al. [HKNT22] fits this framework.

Our main contribution [CCD**M**23a]

- We define a generic local procedure (for coloring kind of problems);
- We show that those can be simulated deterministically in sublinear MPC;
- All the local procedures in the local algorithm by Halldórsson et al. [HKNT22] fits this framework.

(τ, Δ)-normal distributed procedure

- τ rounds of Local;
- Each node has
  - $O\left(\Delta^{O(\tau)}\right)$ words of input information;
  - produces $O\left(\Delta^{O(\tau)}\right)$ words of output information;
  - uses input from its τ-hop neighborhood and $\Delta^{O(\tau)}$ random bits;
  - performs $O(\Delta^{O(\tau)})$ computation.
- Strong success property: can be decided for each node based on the output information from its τ-hop neighborhood. Each node succeeds with probability $1 - 1/2n$.
- Weak success property: some vertices can be deferred. It doesn't cause the property to fail.

Implication in D1LC in sublinear MPC

Main theorem

If $\Delta \leq n^{\delta/c}$, then a series of k number of (τ, Δ)-normalized distributed procedure can be implemented in $O(k\tau + \log^* n)$ rounds of MPC.

Informally

Each randomized procedure in [HKNT22] is a series of $O(\log^* n)$ number of $(O(1), \Delta)$-normal distributed procedure, i.e., $O(\log^* n)$ rounds of MPC is enough.

Final result

Combining the above with degree reduction and efficient algorithm for low degee graphs, we get $O(\log \log \log n)$ algorithm.

Implication in D1LC in sublinear MPC

Main theorem

If $\Delta \leq n^{\delta/c}$, then a series of k number of (τ, Δ)-normalized distributed procedure can be implemented in $O(k\tau + \log^* n)$ rounds of MPC.

Informally

Each randomized procedure in [HKNT22] is a series of $O(\log^* n)$ number of $(O(1), \Delta)$-normal distributed procedure, i.e., $O(\log^* n)$ rounds of MPC is enough.

Final result

Combining the above with degree reduction and efficient algorithm for low degee graphs, we get $O(\log \log \log n)$ algorithm.

Implication in D1LC in sublinear MPC

Main theorem

If $\Delta \leq n^{\delta/c}$, then a series of k number of (τ, Δ)-normalized distributed procedure can be implemented in $O(k\tau + \log^* n)$ rounds of MPC.

Informally

Each randomized procedure in [HKNT22] is a series of $O(\log^* n)$ number of $(O(1), \Delta)$-normal distributed procedure, i.e., $O(\log^* n)$ rounds of MPC is enough.

Final result

Combining the above with degree reduction and efficient algorithm for low degee graphs, we get $O(\log \log \log n)$ algorithm.

Contents

Summary: complexity of Δ1C and D1LC

C-Clique : Congested Clique, S-MPC : Sublinear-MPC, R : Randomized, and D : Deterministic.

Model	R/D	Δ1C	Reference	D1LC	Reference
Local	R	$\widetilde{O}(\log^2 \log n)$	[CLP20]	$\widetilde{O}(\log^2 \log n)$	[HKNT22]
Local	D	$\widetilde{O}(\log^2 n)$	[GG23]	$\widetilde{O}(\log^2 n)$	[GG23]
Congest	R	$\widetilde{O}(\log^3 \log n)$	[HKMT21]	$\widetilde{O}(\log^3 \log n)$	[HNT22]
Congest	D	$\widetilde{O}(\log^3 n)$	[GK21]	$\widetilde{O}(\log^3 n)$	[GK21]
C-Clique	R	$O(1)$	[CFGUZ19]	$O(1)$	[CCD**M**23b]
C-Clique	D	$O(1)$	[CDP21]	$O(1)$	[CCD**M**23b]
S-MPC	R	$O(\log \log \log n)$	[CFGUZ19]	$O(\log \log \log n)$	[CCD**M**23a]
S-MPC	D	$O(\log \log \log n)$	[CDP21]	$O(\log \log \log n)$	[CCD**M**23a]

Open questions

What about more constrained coloring than D1LC?

- Recently Δ-coloring has been considered [FMH23];
- What about going beyond Δ-coloring like $(\Delta - 1)$-coloring, $(\Delta - 2)$-coloring,..., $(\Delta - \Omega(\sqrt{\Delta}))$-coloring?

Thanks!

Open questions

What about more constrained coloring than D1LC?

- Recently Δ-coloring has been considered [FMH23];
- What about going beyond Δ-coloring like $(\Delta - 1)$-coloring, $(\Delta - 2)$-coloring,..., $(\Delta - \Omega(\sqrt{\Delta}))$-coloring?

Thanks!

Open questions

What about more constrained coloring than D1LC?

- Recently Δ-coloring has been considered [FMH23];
- What about going beyond Δ-coloring like $(\Delta - 1)$-coloring, $(\Delta - 2)$-coloring,..., $(\Delta - \Omega(\sqrt{\Delta}))$-coloring?

Thanks!