Sublinear Algorithms for $(\Delta + 1)$ Vertex Coloring

Sepehr Assadi

University of Pennsylvania

Joint work with Yu Chen (Penn) and Sanjeev Khanna (Penn)

Graph Coloring

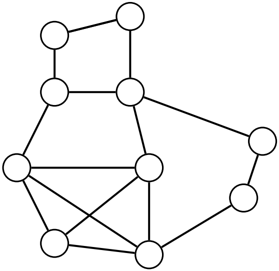
A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.

Graph Coloring

A proper c-coloring of a graph $G(V,E)$:

- assigns a color from the palette $\{1,\ldots,c\}$ to all vertices V of G,
- no monochromatic edges.

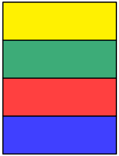


a graph G

Graph Coloring

A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.

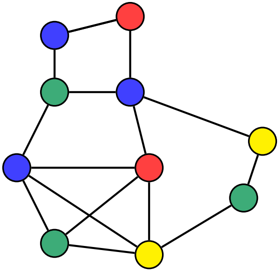


a palette of 4 colors

Graph Coloring

A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.



a proper 4-coloring of G

Graph Coloring

A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.

A central problem in graph theory and computer science.

Graph Coloring

A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.

A central problem in graph theory and computer science.

Numerous applications to scheduling and symmetry breaking.

Graph Coloring

A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.

A central problem in graph theory and computer science.

Numerous applications to scheduling and symmetry breaking.

An important and well-studied case: $(\Delta + 1)$ coloring

Δ: maximum degree n: number of vertices.

Graph Coloring

A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.

A central problem in graph theory and computer science.

Numerous applications to scheduling and symmetry breaking.

An important and well-studied case: $(\Delta + 1)$ coloring

Δ: maximum degree n: number of vertices.

Every graph admits a $(\Delta + 1)$ coloring (tight for cliques and odd cycles).

Graph Coloring

A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.

A central problem in graph theory and computer science.

Numerous applications to scheduling and symmetry breaking.

An important and well-studied case: $(\Delta + 1)$ coloring

Δ: maximum degree n: number of vertices.

Every graph admits a $(\Delta + 1)$ coloring (tight for cliques and odd cycles).

Any partial coloring can be extended to a proper $(\Delta + 1)$ coloring.

Graph Coloring

A proper c-coloring of a graph $G(V, E)$:

- assigns a color from the palette $\{1, \ldots, c\}$ to all vertices V of G,
- no monochromatic edges.

A central problem in graph theory and computer science.

Numerous applications to scheduling and symmetry breaking.

An important and well-studied case: $(\Delta + 1)$ coloring

Δ: maximum degree $\qquad$ n: number of vertices.

Every graph admits a $(\Delta + 1)$ coloring (tight for cliques and odd cycles).

Any partial coloring can be extended to a proper $(\Delta + 1)$ coloring.

Closely related to a plethora of other problems: maximal independent set, maximal matching, $(2\Delta - 1)$ edge coloring, $\cdots$

The Greedy Algorithm for $(\Delta + 1)$ Coloring

On a graph $G(V, E)$:

1. Iterate over vertices of V in arbitrary order,
2. Assign a color to each vertex that does not appear in its neighborhood.

The Greedy Algorithm for $(\Delta + 1)$ Coloring

On a graph $G(V, E)$:

1. Iterate over vertices of V in arbitrary order,
2. Assign a color to each vertex that does not appear in its neighborhood.

maximum degree is $\Delta \implies$ we always find a color for every vertex.

The Greedy Algorithm for $(\Delta + 1)$ Coloring

On a graph $G(V, E)$:

1. Iterate over vertices of V in arbitrary order,
2. Assign a color to each vertex that does not appear in its neighborhood.

maximum degree is $\Delta \implies$ we always find a color for every vertex.

An extremely simple algorithm.

The Greedy Algorithm for $(\Delta + 1)$ Coloring

On a graph $G(V, E)$:

1. Iterate over vertices of V in arbitrary order,
2. Assign a color to each vertex that does not appear in its neighborhood.

maximum degree is $\Delta \implies$ we always find a color for every vertex.

An extremely simple algorithm.

Highly efficient: requires only linear time and space.

The Greedy Algorithm for $(\Delta + 1)$ Coloring

On a graph $G(V, E)$:

1. Iterate over vertices of V in arbitrary order,
2. Assign a color to each vertex that does not appear in its neighborhood.

maximum degree is $\Delta \implies$ we always find a color for every vertex.

An extremely simple algorithm.

Highly efficient: requires only linear time and space.

But is there an even more efficient algorithm?

The Greedy Algorithm for $(\Delta + 1)$ Coloring

On a graph $G(V, E)$:

1. Iterate over vertices of V in arbitrary order,
2. Assign a color to each vertex that does not appear in its neighborhood.

maximum degree is $\Delta \implies$ we always find a color for every vertex.

An extremely simple algorithm.

Highly efficient: requires only linear time and space.

But is there an even more efficient algorithm? **Sublinear Algorithms**

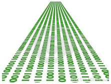
Sublinear Algorithms

1. Sublinear time algorithms:
   - Process the graph faster than even reading the entire input.

Sublinear Algorithms

1. Sublinear time algorithms:
   - Process the graph faster than even reading the entire input.

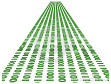
2. Streaming algorithms:
   - Process the graph on the fly with limited memory.



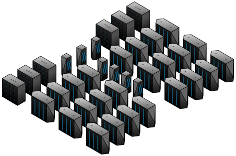
Sublinear Algorithms

1. Sublinear time algorithms:
   - Process the graph faster than even reading the entire input.

2. Streaming algorithms:
   - Process the graph on the fly with limited memory.



3. Massively parallel computation (MPC) algorithms:
   - Process the graph in a distributed fashion with limited communication.



Motivating Question

Can we design sublinear algorithms for $(\Delta + 1)$ coloring problem?

Motivating Question

Can we design sublinear algorithms for $(\Delta + 1)$ coloring problem?

Probably not...

Motivating Question

Can we design sublinear algorithms for $(\Delta + 1)$ coloring problem?

Probably not...

- Similar problems to $(\Delta + 1)$ coloring are provably hard:
  - Maximal independent set: no sublinear space streaming algorithm
  - Maximal matching: no sublinear time algorithm

Motivating Question

Can we design sublinear algorithms for $(\Delta+1)$ coloring problem?

Probably not...

- Similar problems to $(\Delta+1)$ coloring are provably hard:
  - Maximal independent set: no sublinear space streaming algorithm
  - Maximal matching: no sublinear time algorithm
- "Exact" problems are typically hard for sublinear algorithms: one needs "approximation".

Our Results

Our Results

Surprisingly, we present highly efficient sublinear algorithms for $(\Delta + 1)$ coloring in all these models!

Our Results

Surprisingly, we present highly efficient sublinear algorithms for $(\Delta + 1)$ coloring in all these models!

Our algorithms are randomized:

- Output a $(\Delta + 1)$ coloring with high probability,
- Otherwise output FAIL.

Our Results: Sublinear Time Algorithms

The standard query model for dense graphs:

- Degree queries: what is degree of the vertex v?
- Pair queries: is (u, v) an edge?
- Neighbor queries: what is the k-th neighbor of the vertex v?

Our Results: Sublinear Time Algorithms

The standard query model for dense graphs:

- Degree queries: what is degree of the vertex v?
- Pair queries: is (u, v) an edge?
- Neighbor queries: what is the k-th neighbor of the vertex v?

Prior Results:

No sublinear time algorithm for $(\Delta + 1)$ coloring.

Fastest algorithm: the greedy algorithm.

Our Results: Sublinear Time Algorithms

The standard query model for dense graphs:

- Degree queries: what is degree of the vertex v?
- Pair queries: is (u, v) an edge?
- Neighbor queries: what is the k-th neighbor of the vertex v?

Our Result:

An $\widetilde{O}\left(n\sqrt{n}\right)$ time algorithm for $(\Delta + 1)$ coloring.

Our Results: Sublinear Time Algorithms

The standard query model for dense graphs:

- Degree queries: what is degree of the vertex v?
- Pair queries: is (u, v) an edge?
- Neighbor queries: what is the k-th neighbor of the vertex v?

Our Result:

An $\widetilde{O}\left(n\sqrt{n}\right)$ time algorithm for $(\Delta + 1)$ coloring.

- Queries are chosen non-adaptively.

Our Results: Sublinear Time Algorithms

The standard query model for dense graphs:

- Degree queries: what is degree of the vertex v?
- Pair queries: is (u, v) an edge?
- Neighbor queries: what is the k-th neighbor of the vertex v?

Our Result:

An $\widetilde{O}\left(n\sqrt{n}\right)$ time algorithm for $(\Delta + 1)$ coloring.

- Queries are chosen non-adaptively.
- $\Omega(n\sqrt{n})$ query lower bound even for adaptive algorithms.

Our Results: Streaming Algorithms

Semi-streaming algorithms:

- Edges are appearing one by one in a stream.
- Process the stream in one pass and $\widetilde{O}(n)$ space.

Our Results: Streaming Algorithms

Semi-streaming algorithms:

- Edges are appearing one by one in a stream.
- Process the stream in one pass and $\widetilde{O}(n)$ space.

Prior Results:

- No streaming algorithm for $(\Delta + 1)$ coloring with $o(n\Delta)$ space.
- Parallel to our work. Easier problem of $(\Delta + o(\Delta))$: a semi-streaming algorithm by [Bera and Ghosh, 2018].

Our Results: Streaming Algorithms

Semi-streaming algorithms:

- Edges are appearing one by one in a stream.
- Process the stream in one pass and $\widetilde{O}(n)$ space.

Our Result:

A single-pass $\widetilde{O}(n)$ space streaming algorithm for $(\Delta + 1)$ coloring.

Our Results: Streaming Algorithms

Semi-streaming algorithms:

- Edges are appearing one by one in a stream.
- Process the stream in one pass and $\widetilde{O}(n)$ space.

Our Result:

A single-pass $\widetilde{O}(n)$ space streaming algorithm for $(\Delta + 1)$ coloring.

- $\Omega(n)$ space is clearly necessary for this problem.

Our Results: Streaming Algorithms

Semi-streaming algorithms:

- Edges are appearing one by one in a stream.
- Process the stream in one pass and $\widetilde{O}(n)$ space.

Our Result:

A single-pass $\widetilde{O}(n)$ space streaming algorithm for $(\Delta + 1)$ coloring.

- $\Omega(n)$ space is clearly necessary for this problem.
- Our algorithm works even in dynamic graph streams.

Our Results: MPC Algorithms

MPC algorithms with near-linear memory per-machine:

- Edges are partitioned arbitrarily across multiple machines.
- Machines can send and receive $\widetilde{O}(n)$ messages in synchronous rounds.

Our Results: MPC Algorithms

MPC algorithms with near-linear memory per-machine:

- Edges are partitioned arbitrarily across multiple machines.
- Machines can send and receive $\widetilde{O}(n)$ messages in synchronous rounds.

Prior Results:

- An $O(\log\log\Delta \cdot \log^*(n))$ round algorithm with $\widetilde{O}(n)$ memory for $(\Delta+1)$ coloring [Parter, 2018].
- Parallel to our work. the round-complexity improved to $O(\log^*(n))$ rounds [Parter and Su, 2018].
- Easier problem of $(\Delta + o(\Delta))$ coloring: an $O(1)$ round algorithm with $n^{1+\Omega(1)}$ memory [Harvey et al., 2018].

Our Results: MPC Algorithms

MPC algorithms with near-linear memory per-machine:

- Edges are partitioned arbitrarily across multiple machines.
- Machines can send and receive $\widetilde{O}(n)$ messages in synchronous rounds.

Our Result:

An $O(1)$ round $\widetilde{O}(n)$ memory MPC algorithm for $(\Delta + 1)$ coloring.

Our Results: MPC Algorithms

MPC algorithms with near-linear memory per-machine:

- Edges are partitioned arbitrarily across multiple machines.
- Machines can send and receive $\widetilde{O}(n)$ messages in synchronous rounds.

Our Result:

An $O(1)$ round $\widetilde{O}(n)$ memory MPC algorithm for $(\Delta + 1)$ coloring.

- Our algorithm only requires one round assuming public randomness.

Our Results: MPC Algorithms

MPC algorithms with near-linear memory per-machine:

- Edges are partitioned arbitrarily across multiple machines.
- Machines can send and receive $\widetilde{O}(n)$ messages in synchronous rounds.

Our Result:

> An $O(1)$ round $\widetilde{O}(n)$ memory MPC algorithm for $(\Delta+1)$ coloring.

- Our algorithm only requires one round assuming public randomness.
- The first constant round MPC algorithm with $\widetilde{O}(n)$ memory for one of "classic four local distributed graph problems".

Our Main Result

The central tool: a structural result for $(\Delta + 1)$ coloring.

Our Main Result

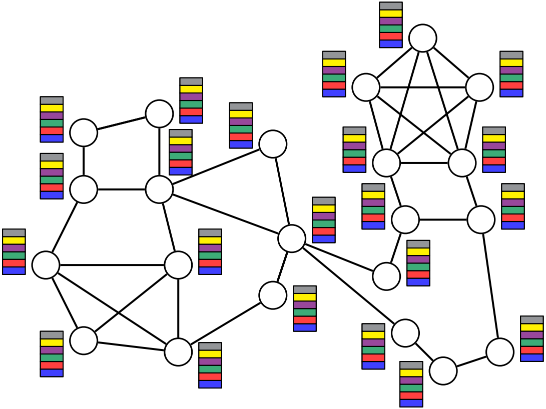
The central tool: a structural result for $(\Delta + 1)$ coloring.

Palette Sparsification Theorem.

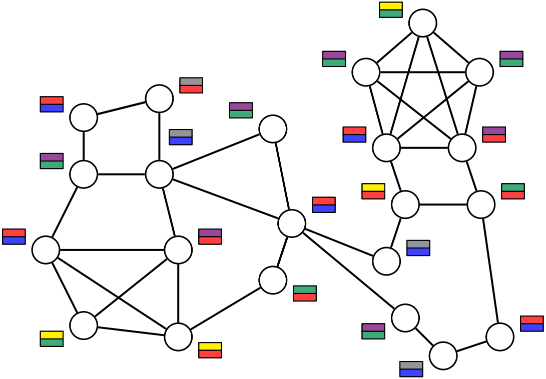
For every vertex v, sample $O(\log n)$ colors $L(v)$ from $\{1, \ldots, \Delta + 1\}$.

W.h.p., G can be colored by coloring any vertex v from the list $L(v)$.

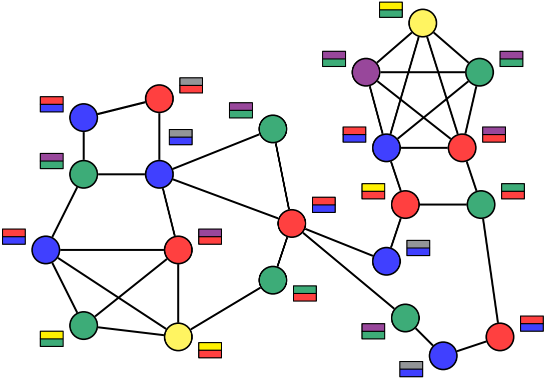
Palette Sparsification: An Illustration



Palette Sparsification: An Illustration



Palette Sparsification: An Illustration



Our Main Result

Why is palette sparsification theorem "useful"?

Our Main Result

Why is palette sparsification theorem "useful"?

- Sample colors L and throw out any edge (u,v) with $L(u) \cap L(v) = \emptyset$.

Our Main Result

Why is palette sparsification theorem "useful"?

- Sample colors L and throw out any edge (u, v) with $L(u) \cap L(v) = \emptyset$.
- Only $O(n \cdot \log^2(n))$ edges remain:

$$n\Delta \cdot O(\log n) \cdot O(\frac{\log n}{\Delta}) = O(n \cdot \log^2 n).$$

Our Main Result

Why is palette sparsification theorem "useful"?

- Sample colors L and throw out any edge (u,v) with $L(u) \cap L(v) = \emptyset$.
- Only $O(n \cdot \log^2(n))$ edges remain:

$$n\Delta \cdot O(\log n) \cdot O(\frac{\log n}{\Delta}) = O(n \cdot \log^2 n).$$

- List-coloring of this new graph $\implies$ $(\Delta + 1)$ coloring of G.

Our Main Result

Why is palette sparsification theorem "useful"?

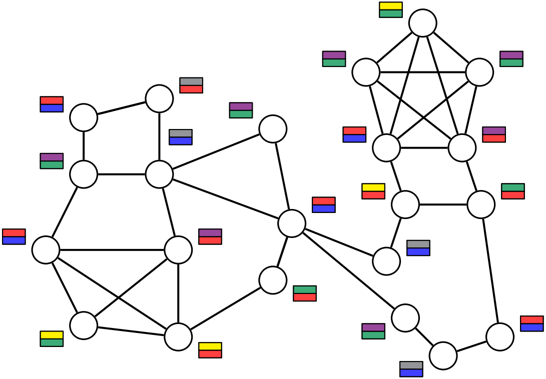
- Sample colors L and throw out any edge (u, v) with $L(u) \cap L(v) = \emptyset$.
- Only $O(n \cdot \log^2(n))$ edges remain:

$$n\Delta \cdot O(\log n) \cdot O(\frac{\log n}{\Delta}) = O(n \cdot \log^2 n).$$

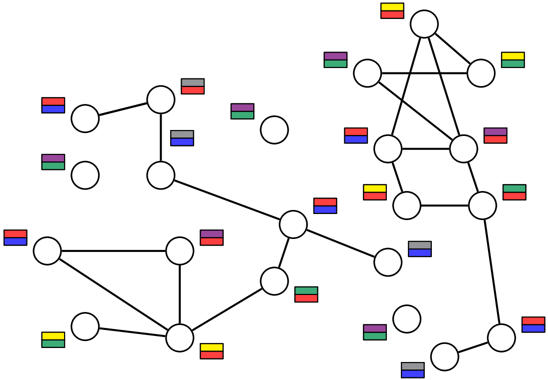
- List-coloring of this new graph $\implies$ $(\Delta + 1)$ coloring of G.

Non-adaptively sparsify a graph with $O(n\Delta)$ edges down to $\widetilde{O}(n)$ edges; still recover a proper $(\Delta + 1)$ coloring!

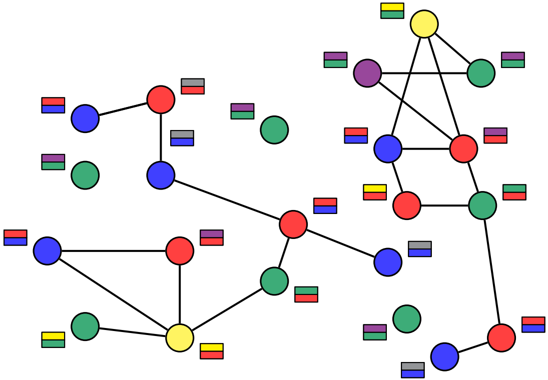
Palette Sparsification: An Illustration



Palette Sparsification: An Illustration



Palette Sparsification: An Illustration



Palette Sparsification Theorem

A Slight Reformulation

Graph coloring as an assignment problem:

A Slight Reformulation

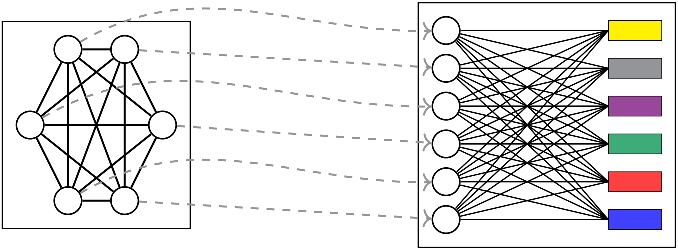
Graph coloring as an assignment problem:

Example. Coloring a 6-clique.

A Slight Reformulation

Graph coloring as an assignment problem:

Example. Coloring a 6-clique.



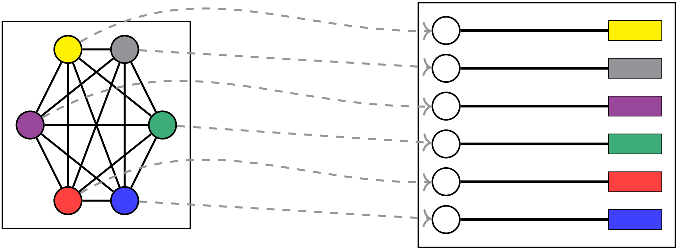
Original Graph

Palette Graph

A Slight Reformulation

Graph coloring as an assignment problem:

Example. Coloring a 6-clique.



Original Graph

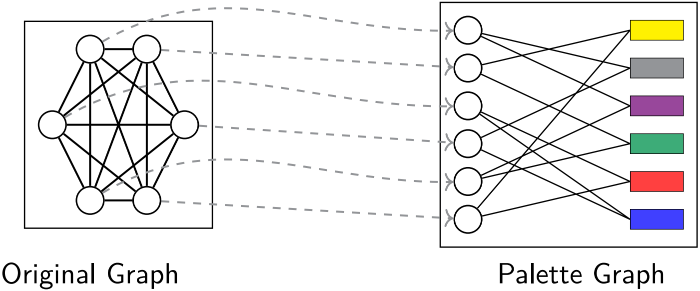
Palette Graph

$(\Delta + 1)$ **Coloring:** Finding a perfect matching in the palette graph.

A Slight Reformulation

Graph coloring as an assignment problem:

Example. Coloring a 6-clique.

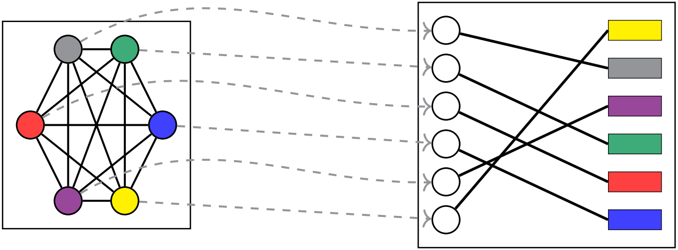


Palette sparsification theorem: Random subgraphs of the palette graph of a clique contain a perfect matching.

A Slight Reformulation

Graph coloring as an assignment problem:

Example. Coloring a 6-clique.



Original Graph

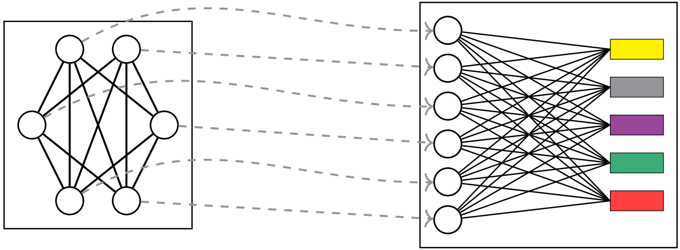
Palette Graph

Palette sparsification theorem: Random subgraphs of the palette graph of a clique contain a perfect matching.

A Slight Reformulation

Graph coloring as an assignment problem:

Another example. Coloring a 6-clique minus a perfect matching.



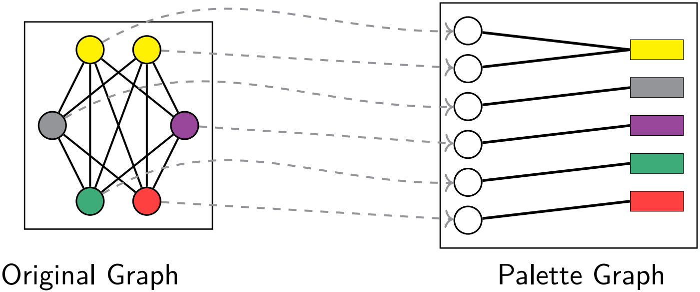
Original Graph

Palette Graph

A Slight Reformulation

Graph coloring as an assignment problem:

Another example. Coloring a 6-clique minus a perfect matching.

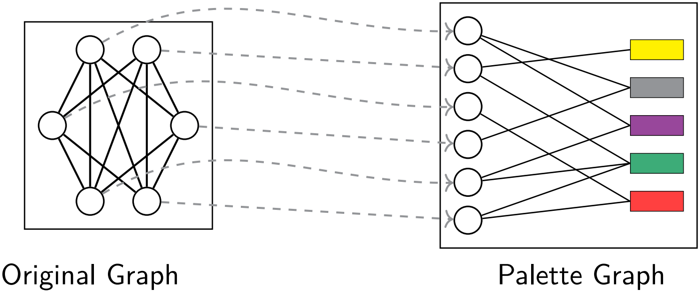


$(\Delta + 1)$ **Coloring:** Finding a "good" subgraph in the palette graph.

A Slight Reformulation

Graph coloring as an assignment problem:

Another example. Coloring a 6-clique minus a perfect matching.

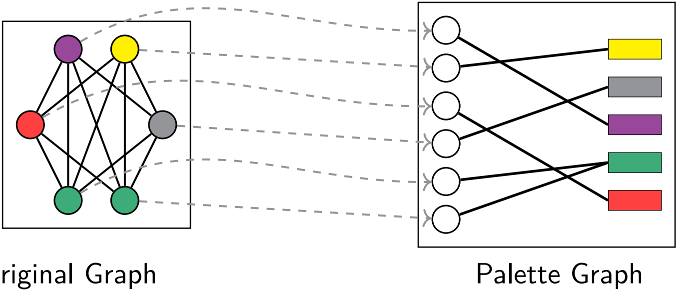


Palette sparsification theorem: Random subgraphs of the palette graph of a clique minus a perfect matching contain a "good" subgraph.

A Slight Reformulation

Graph coloring as an assignment problem:

Another example. Coloring a 6-clique minus a perfect matching.



Palette sparsification theorem: Random subgraphs of the palette graph of a clique minus a perfect matching contain a "good" subgraph.

A Slight Reformulation

General reformulation. Find a subgraph of the palette graph:

A Slight Reformulation

General reformulation. Find a subgraph of the palette graph:

- Degree exactly one for vertices on left.

A Slight Reformulation

General reformulation. Find a subgraph of the palette graph:

- Degree exactly one for vertices on left.
- Neighbors of vertices on right can only be an independent set in the original graph.

A Slight Reformulation

General reformulation. Find a subgraph of the palette graph:

- Degree exactly one for vertices on left.
- Neighbors of vertices on right can only be an independent set in the original graph.

Palette sparsification theorem reduces to a random graph theory question.

A Slight Reformulation

General reformulation. Find a subgraph of the palette graph:

- Degree exactly one for vertices on left.
- Neighbors of vertices on right can only be an independent set in the original graph.

Palette sparsification theorem reduces to a random graph theory question.

The reformulation is quite helpful when graphs are "almost" clique.

A Slight Reformulation

General reformulation. Find a subgraph of the palette graph:

- Degree exactly one for vertices on left.
- Neighbors of vertices on right can only be an independent set in the original graph.

Palette sparsification theorem reduces to a random graph theory question.

The reformulation is quite helpful when graphs are "almost" clique.

But not that helpful for graphs that are "far from" cliques.

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Example. A graph where all vertices have degree $\leq \Delta/2$.

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Example. A graph where all vertices have degree $\leq \Delta/2$.

A simple coloring procedure:

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Example. A graph where all vertices have degree $\leq \Delta/2$.

A simple coloring procedure:

1. Pick a color uniformly at random from $\{1, \ldots, \Delta + 1\}$ for all uncolored vertices.

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Example. A graph where all vertices have degree $\leq \Delta/2$.

A simple coloring procedure:

1. Pick a color uniformly at random from $\{1, \ldots, \Delta + 1\}$ for all uncolored vertices.
2. Assign the color to each vertex if it is not assigned to its neighbors in this iteration or previous ones.

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Example. A graph where all vertices have degree $\leq \Delta/2$.

A simple coloring procedure:

1. Pick a color uniformly at random from $\{1, \ldots, \Delta + 1\}$ for all uncolored vertices.
2. Assign the color to each vertex if it is not assigned to its neighbors in this iteration or previous ones.
3. Repeat until all vertices are colored.

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Example. A graph where all vertices have degree $\leq \Delta/2$.

A simple coloring procedure:

1. Pick a color uniformly at random from $\{1, \ldots, \Delta + 1\}$ for all uncolored vertices.
2. Assign the color to each vertex if it is not assigned to its neighbors in this iteration or previous ones.
3. Repeat until all vertices are colored.

Every vertex has constant probability of being colored in each iteration.

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Example. A graph where all vertices have degree $\leq \Delta/2$.

A simple coloring procedure:

1. Pick a color uniformly at random from $\{1, \ldots, \Delta + 1\}$ for all uncolored vertices.
2. Assign the color to each vertex if it is not assigned to its neighbors in this iteration or previous ones.
3. Repeat until all vertices are colored.

Every vertex has constant probability of being colored in each iteration.

After $O(\log n)$ iterations, all vertices are colored.

Handling Graphs that are Far From Cliques

The other extreme case: low degree graphs.

Example. A graph where all vertices have degree $\leq \Delta/2$.

A simple coloring procedure:

1. Pick a color uniformly at random from $\{1, \ldots, \Delta + 1\}$ for all uncolored vertices.
2. Assign the color to each vertex if it is not assigned to its neighbors in this iteration or previous ones.
3. Repeat until all vertices are colored.

Every vertex has constant probability of being colored in each iteration.

After $O(\log n)$ iterations, all vertices are colored.

This proves the palette sparsification theorem for low degree graphs.

General Proof?

General proof requires interpolating between these two extreme cases:

General Proof?

General proof requires interpolating between these two extreme cases:

Cliques

Low Degree Graphs

Assignment in random graphs

Direct simulation of greedy

Neither approach seems to work for the other extreme case.

General Proof?

General proof requires interpolating between these two extreme cases:

Cliques ⟷ Low Degree Graphs

Assignment in random graphs — Direct simulation of greedy

Neither approach seems to work for the other extreme case.

Our approach: Decompose the graph into dense and sparse regions, then apply the previous ideas to each part.

A Network Decomposition

We exploit and modify the decomposition of Harris, Schneider, and Su [Harris et al., 2016] for distributed $(\Delta + 1)$ coloring.

A Network Decomposition

We exploit and modify the decomposition of Harris, Schneider, and Su [Harris et al., 2016] for distributed $(\Delta + 1)$ coloring.

Extended HSS Decomposition: For any $\varepsilon \in (0, 1)$, any graph $G(V, E)$ can be decomposed into:

A Network Decomposition

We exploit and modify the decomposition of Harris, Schneider, and Su [Harris et al., 2016] for distributed $(\Delta + 1)$ coloring.

Extended HSS Decomposition: For any $\varepsilon \in (0, 1)$, any graph $G(V, E)$ can be decomposed into:

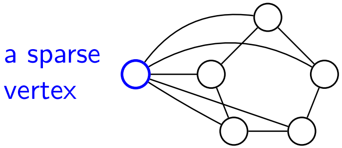
- Sparse vertices: Neighborhood of each sparse vertex is missing at least $\varepsilon \cdot \binom{\Delta}{2}$ edges.

A Network Decomposition

We exploit and modify the decomposition of Harris, Schneider, and Su [Harris et al., 2016] for distributed $(\Delta + 1)$ coloring.

Extended HSS Decomposition: For any $\varepsilon \in (0, 1)$, any graph $G(V, E)$ can be decomposed into:

- Sparse vertices: Neighborhood of each sparse vertex is missing at least $\varepsilon \cdot \binom{\Delta}{2}$ edges.



A Network Decomposition

We exploit and modify the decomposition of Harris, Schneider, and Su [Harris et al., 2016] for distributed $(\Delta + 1)$ coloring.

Extended HSS Decomposition: For any $\varepsilon \in (0, 1)$, any graph $G(V, E)$ can be decomposed into:

- Sparse vertices: Neighborhood of each sparse vertex is missing at least $\varepsilon \cdot \binom{\Delta}{2}$ edges.
- A collection of almost-cliques: Each almost-clique C:

A Network Decomposition

We exploit and modify the decomposition of Harris, Schneider, and Su [Harris et al., 2016] for distributed $(\Delta + 1)$ coloring.

Extended HSS Decomposition: For any $\varepsilon \in (0, 1)$, any graph $G(V, E)$ can be decomposed into:

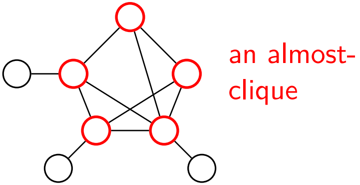
- Sparse vertices: Neighborhood of each sparse vertex is missing at least $\varepsilon \cdot \binom{\Delta}{2}$ edges.
- A collection of almost-cliques: Each almost-clique C:
  - contains $(1 \pm \varepsilon)\,\Delta$ vertices.
  - every vertex in C has $\leq \varepsilon\Delta$ neighbors outside C.
  - every vertex in C has $\leq \varepsilon\Delta$ non-neighbors inside C.

A Network Decomposition

We exploit and modify the decomposition of Harris, Schneider, and Su [Harris et al., 2016] for distributed $(\Delta + 1)$ coloring.

Extended HSS Decomposition: For any $\varepsilon \in (0, 1)$, any graph $G(V, E)$ can be decomposed into:

- Sparse vertices: Neighborhood of each sparse vertex is missing at least $\varepsilon \cdot \binom{\Delta}{2}$ edges.
- A collection of almost-cliques: Each almost-clique C:
  - contains $(1 \pm \varepsilon)\,\Delta$ vertices.
  - every vertex in C has $\leq \varepsilon\Delta$ neighbors outside C.
  - every vertex in C has $\leq \varepsilon\Delta$ non-neighbors inside C.



A Network Decomposition

We exploit and modify the decomposition of Harris, Schneider, and Su [Harris et al., 2016] for distributed $(\Delta + 1)$ coloring.

Extended HSS Decomposition: For any $\varepsilon \in (0, 1)$, any graph $G(V, E)$ can be decomposed into:

- Sparse vertices: Neighborhood of each sparse vertex is missing at least $\varepsilon \cdot \binom{\Delta}{2}$ edges.
- A collection of almost-cliques: Each almost-clique C:
  - **contains** $(1 \pm \varepsilon)\,\Delta$ **vertices.**
  - every vertex in C has $\leq \varepsilon\Delta$ neighbors outside C.
  - every vertex in C has $\leq \varepsilon\Delta$ non-neighbors inside C.

Proof Strategy of Palette Sparsification Theorem

Palette Sparsification Theorem.

For every vertex v, sample $O(\log n)$ colors $L(v)$ from $\{1, \ldots, \Delta + 1\}$.

W.h.p., G can be colored by coloring any vertex v from the list $L(v)$.

Proof Strategy of Palette Sparsification Theorem

Palette Sparsification Theorem.

For every vertex v, sample $O(\log n)$ colors $L(v)$ from $\{1, \ldots, \Delta + 1\}$.

W.h.p., G can be colored by coloring any vertex v from the list $L(v)$.

1. Fix an extended HSS decomposition of the graph for $\varepsilon \approx 0.001$.

Proof Strategy of Palette Sparsification Theorem

Palette Sparsification Theorem.

For every vertex v, sample $O(\log n)$ colors $L(v)$ from $\{1, \ldots, \Delta + 1\}$.

W.h.p., G can be colored by coloring any vertex v from the list $L(v)$.

1. Fix an extended HSS decomposition of the graph for $\varepsilon \approx 0.001$.
2. **Part one:** Use the first half of colors in $L(\cdot)$ to color sparse vertices.

Proof Strategy of Palette Sparsification Theorem

Palette Sparsification Theorem.

For every vertex v, sample $O(\log n)$ colors $L(v)$ from $\{1, \ldots, \Delta + 1\}$.

W.h.p., G can be colored by coloring any vertex v from the list $L(v)$.

1. Fix an extended HSS decomposition of the graph for $\varepsilon \approx 0.001$.
2. **Part one:** Use the first half of colors in $L(\cdot)$ to color sparse vertices.
   - Easy part: The simulation argument does the trick here also!

Proof Strategy of Palette Sparsification Theorem

Palette Sparsification Theorem.

For every vertex v, sample $O(\log n)$ colors $L(v)$ from $\{1, \ldots, \Delta + 1\}$.

W.h.p., G can be colored by coloring any vertex v from the list $L(v)$.

1. Fix an extended HSS decomposition of the graph for $\varepsilon \approx 0.001$.
2. **Part one:** Use the first half of colors in $L(\cdot)$ to color sparse vertices.
   - Easy part: The simulation argument does the trick here also!
3. **Part two:** Iterate over the almost-cliques one by one and color each one using the remaining half of $L(\cdot)$.

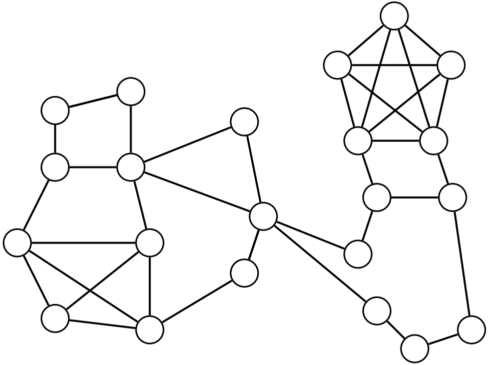
Proof Strategy of Palette Sparsification Theorem

Palette Sparsification Theorem.

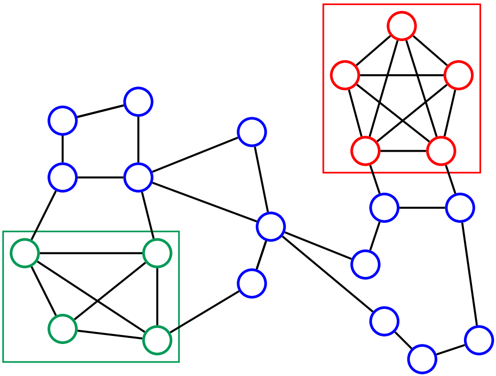
For every vertex v, sample $O(\log n)$ colors $L(v)$ from $\{1, \ldots, \Delta + 1\}$.
W.h.p., G can be colored by coloring any vertex v from the list $L(v)$.

1. Fix an extended HSS decomposition of the graph for $\varepsilon \approx 0.001$.
2. **Part one:** Use the first half of colors in $L(\cdot)$ to color sparse vertices.
   - Easy part: The simulation argument does the trick here also!
3. **Part two:** Iterate over the almost-cliques one by one and color each one using the remaining half of $L(\cdot)$.
   - Hard part: We need a generalization of ideas before in the assignment reformulation for almost-cliques.

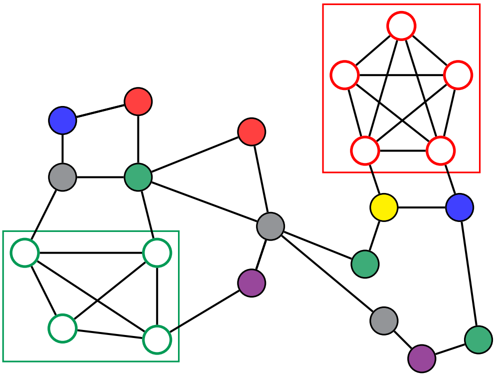
Proof Strategy: An Illustration



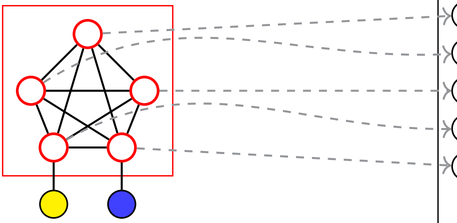
Proof Strategy: An Illustration



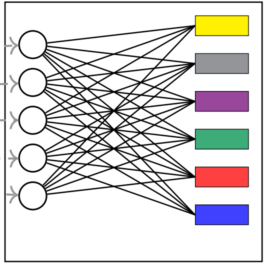
Proof Strategy: An Illustration



Proof Strategy: An Illustration

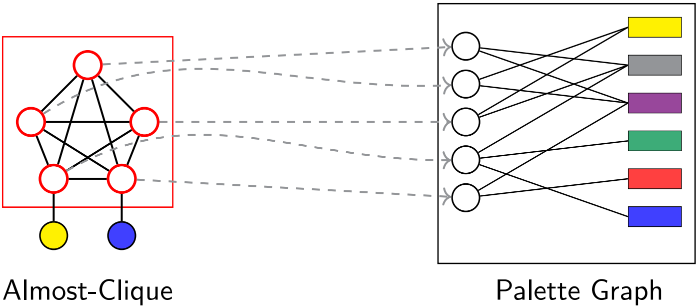


Almost-Clique



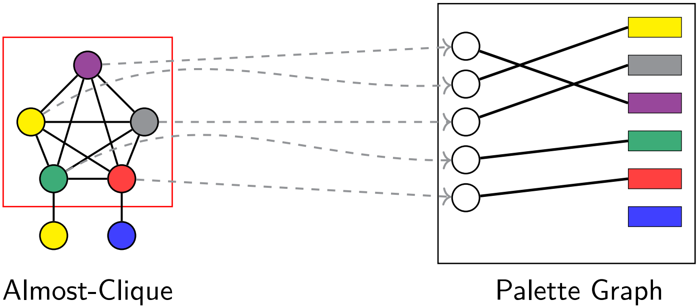
Palette Graph

Proof Strategy: An Illustration



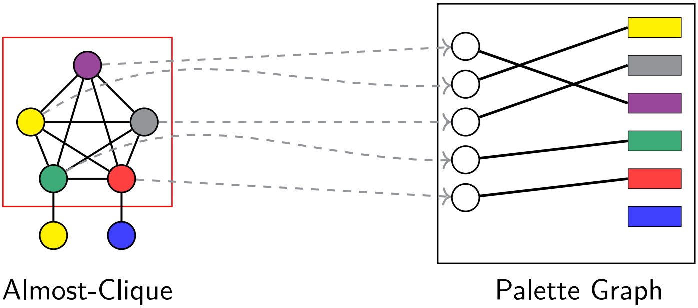
Our main technical result: Random subgraphs of palette graphs for almost-cliques contain a "good" subgraph.

Proof Strategy: An Illustration



Our main technical result: Random subgraphs of palette graphs for almost-cliques contain a "good" subgraph.

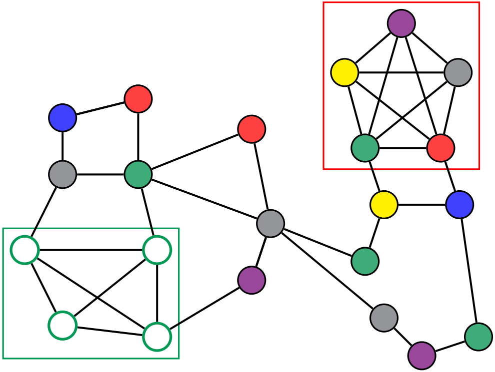
Proof Strategy: An Illustration



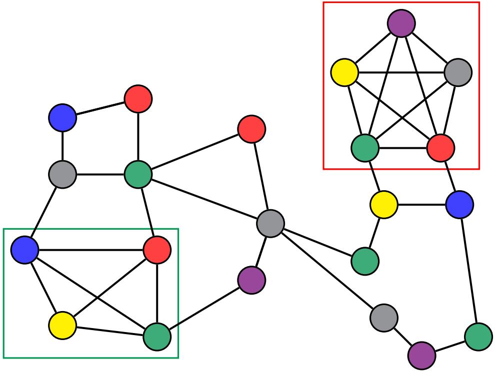
Our main technical result: Random subgraphs of palette graphs for almost-cliques contain a "good" subgraph.

Main challenge: vertices in an almost-clique may have some colored neighbors outside while the almost-clique may have size $> \Delta + 1$.

Proof Strategy: An Illustration



Proof Strategy: An Illustration



Sublinear Algorithms from Palette Sparsification Theorem

The Sublinear Algorithms

All our sublinear algorithms are as follows:

The Sublinear Algorithms

All our sublinear algorithms are as follows:

1. Use palette sparsification to get a sparsified subgraph (conflict-graph).
2. Find a list-coloring of the conflict-graph.

The Sublinear Algorithms

All our sublinear algorithms are as follows:

1. Use palette sparsification to get a sparsified subgraph (conflict-graph).
2. Find a list-coloring of the conflict-graph.

The conflict-graph can be found efficiently in each model:

The Sublinear Algorithms

All our sublinear algorithms are as follows:

1. Use palette sparsification to get a sparsified subgraph (conflict-graph).
2. Find a list-coloring of the conflict-graph.

The conflict-graph can be found efficiently in each model:

- Sublinear time: Find it using $\widetilde{O}(\min\left\{n\Delta, \frac{n^2}{\Delta}\right\})$ queries.
- Streaming: Store its $\widetilde{O}(n)$ edges in the stream.
- MPC: Send its $\widetilde{O}(n)$ edges to a single machine.

The Sublinear Algorithms

All our sublinear algorithms are as follows:

1. Use palette sparsification to get a sparsified subgraph (conflict-graph).
2. Find a list-coloring of the conflict-graph.

The conflict-graph can be found efficiently in each model:

- Sublinear time: Find it using $\widetilde{O}(\min\left\{n\Delta, \frac{n^2}{\Delta}\right\})$ queries.
- Streaming: Store its $\widetilde{O}(n)$ edges in the stream.
- MPC: Send its $\widetilde{O}(n)$ edges to a single machine.

Conflict-graph has all the information needed for list-coloring.

The Sublinear Algorithms

All our sublinear algorithms are as follows:

1. Use palette sparsification to get a sparsified subgraph (conflict-graph).
2. Find a list-coloring of the conflict-graph.

The conflict-graph can be found efficiently in each model:

- Sublinear time: Find it using $\widetilde{O}(\min\left\{n\Delta, \frac{n^2}{\Delta}\right\})$ queries.
- Streaming: Store its $\widetilde{O}(n)$ edges in the stream.
- MPC: Send its $\widetilde{O}(n)$ edges to a single machine.

Conflict-graph has all the information needed for list-coloring.

This gives us our sublinear algorithms modulo a caveat...

The Sublinear Algorithms

Caveat. Palette sparsification theorem is an information-theoretic result not a computational one.

The Sublinear Algorithms

Caveat. Palette sparsification theorem is an information-theoretic result not a computational one.

- Information-theoretically, we only need the conflict-graph.
- But computationally, list-coloring is NP-hard.

The Sublinear Algorithms

Caveat. Palette sparsification theorem is an information-theoretic result not a computational one.

- Information-theoretically, we only need the conflict-graph.
- But computationally, list-coloring is NP-hard.

We further address this issue:

The Sublinear Algorithms

Caveat. Palette sparsification theorem is an information-theoretic result not a computational one.

- Information-theoretically, we only need the conflict-graph.
- But computationally, list-coloring is NP-hard.

We further address this issue:

- Palette sparsification theorem can be made algorithmic assuming we are given an (approximate) decomposition.

The Sublinear Algorithms

Caveat. Palette sparsification theorem is an information-theoretic result not a computational one.

- Information-theoretically, we only need the conflict-graph.
- But computationally, list-coloring is NP-hard.

We further address this issue:

- Palette sparsification theorem can be made algorithmic assuming we are given an (approximate) decomposition.
  - Given the decomposition, we find the list-coloring in $\widetilde{O}(n\sqrt{n})$ time.

The Sublinear Algorithms

Caveat. Palette sparsification theorem is an information-theoretic result not a computational one.

- Information-theoretically, we only need the conflict-graph.
- But computationally, list-coloring is NP-hard.

We further address this issue:

- Palette sparsification theorem can be made algorithmic assuming we are given an (approximate) decomposition.
  - Given the decomposition, we find the list-coloring in $\widetilde{O}(n\sqrt{n})$ time.
- We design sublinear algorithms for finding an approximate decomposition in each model.

Concluding Remarks

Concluding Remarks

We obtained the following sublinear algorithms for $(\Delta + 1)$ coloring:

An $\widetilde{O}(n\sqrt{n})$ time algorithm in the standard query model.

A single-pass $\widetilde{O}(n)$ space algorithm in the streaming model.

An $O(1)$ round $\widetilde{O}(n)$ memory algorithm in the MPC model.

Concluding Remarks

We obtained the following sublinear algorithms for $(\Delta + 1)$ coloring:

An $\widetilde{O}(n\sqrt{n})$ time algorithm in the standard query model.

A single-pass $\widetilde{O}(n)$ space algorithm in the streaming model.

An $O(1)$ round $\widetilde{O}(n)$ memory algorithm in the MPC model.

The central tool: **Palette Sparsification Theorem**.

Concluding Remarks

We obtained the following sublinear algorithms for $(\Delta + 1)$ coloring:

An $\widetilde{O}(n\sqrt{n})$ time algorithm in the standard query model.

A single-pass $\widetilde{O}(n)$ space algorithm in the streaming model.

An $O(1)$ round $\widetilde{O}(n)$ memory algorithm in the MPC model.

The central tool: **Palette Sparsification Theorem**.

Open Problems

- Deterministic sublinear algorithms: streaming $(\Delta + 1)$ coloring?

Concluding Remarks

We obtained the following sublinear algorithms for $(\Delta + 1)$ coloring:

An $\widetilde{O}(n\sqrt{n})$ time algorithm in the standard query model.

A single-pass $\widetilde{O}(n)$ space algorithm in the streaming model.

An $O(1)$ round $\widetilde{O}(n)$ memory algorithm in the MPC model.

The central tool: **Palette Sparsification Theorem**.

Open Problems

- Deterministic sublinear algorithms: streaming $(\Delta + 1)$ coloring?
- Sublinear complexity of related problems: multi-pass streaming/query complexity of maximal independent set?

Concluding Remarks

We obtained the following sublinear algorithms for $(\Delta + 1)$ coloring:

An $\widetilde{O}(n\sqrt{n})$ time algorithm in the standard query model.

A single-pass $\widetilde{O}(n)$ space algorithm in the streaming model.

An $O(1)$ round $\widetilde{O}(n)$ memory algorithm in the MPC model.

The central tool: **Palette Sparsification Theorem**.

Open Problems

- Deterministic sublinear algorithms: streaming $(\Delta + 1)$ coloring?
- Sublinear complexity of related problems: multi-pass streaming/query complexity of maximal independent set?
- Beyond greedy algorithms for sublinear algorithms: Can non-adaptive sparsification help other problems?

Bera, S. K. and Ghosh, P. (2018).
Coloring in graph streams.
CoRR, abs/1807.07640.

Harris, D. G., Schneider, J., and Su, H.-H. (2016).
Distributed $(\Delta + 1)$-coloring in sublogarithmic rounds.
In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 465–478. ACM.

Harvey, N. J. A., Liaw, C., and Liu, P. (2018).
Greedy and local ratio algorithms in the mapreduce model.
In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 43–52.

Parter, M. (2018).
$(\Delta + 1)$ coloring in the congested clique model.
In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 160:1–160:14.

Parter, M. and Su, H. (2018).
Randomized $(\Delta + 1)$-coloring in $O(\log^* \Delta)$ congested clique rounds.
In *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, pages 39:1–39:18.