Sublinear Algorithms for Graph Coloring

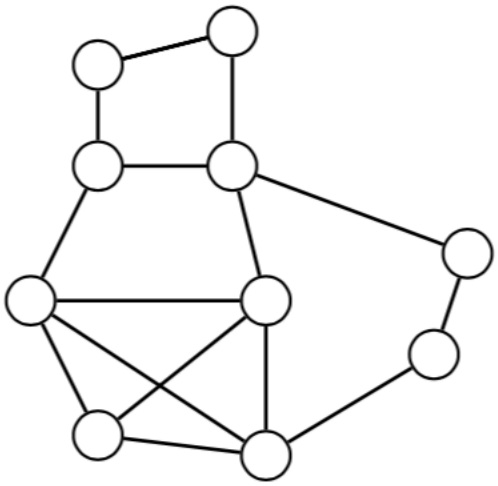
Sanjeev Khanna

University of Pennsylvania

Joint work with Sepehr Assadi (Princeton) and Yu Chen (Penn).

Graph Coloring

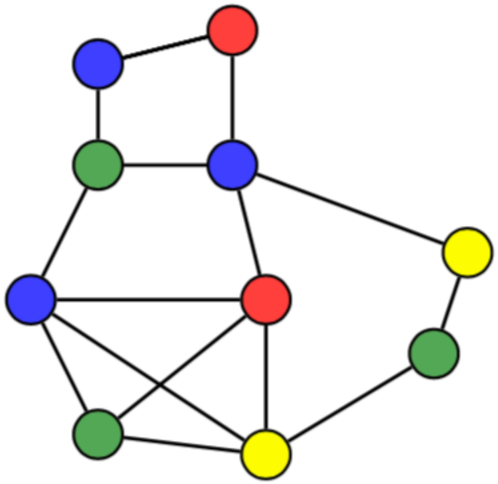
A C-coloring of a graph $G(V, E)$ assigns each vertex a color from the palette $\{1, 2, \ldots, C\}$ such that there are no monochromatic edges.



Graph G

Graph Coloring

A C-coloring of a graph $G(V, E)$ assigns each vertex a color from the palette $\{1, 2, \dots, C\}$ such that there are no monochromatic edges.



Proper 4-coloring of G

Graph Coloring

A C-coloring of a graph $G(V, E)$ assigns each vertex a color from the palette $\{1, 2, \ldots, C\}$ such that there are no monochromatic edges.

- A central problem in graph theory and computer science.
- Applications include scheduling, frequency assignment, register allocations, etc.
  - Vertices represent tasks, edges represent conflicts between tasks.
  - A C-coloring partitions all tasks into C classes such that the tasks inside each class are conflict-free.
  - We wish to find a C-coloring so that C is small as possible.

Graph Coloring

But the task of coloring a graph with a minimum number of colors is a notoriously hard problem.

Theorem [Feige and Kilian '98, Zuckerman '06]: For any $\epsilon > 0$, it is NP-hard to approximate the # of colors needed to within a factor of $n^{1-\epsilon}$.

So in many applications, we instead focus on coloring a graph using a number of colors based on some graph parameter – smaller the parameter, fewer the number of colors needed.

$(\Delta + 1)$-Coloring of Graphs

The most well-studied example of this approach is $(\Delta+1)$-coloring where Δ is the maximum vertex degree.

Every graph admits a $(\Delta+1)$-coloring. There is a text-book greedy algorithm that establishes this:

- Iterate over the vertices in an arbitrary order.
- Assign each vertex a color that is not in its neighborhood.
- Since max degree is Δ, you can never run out of colors.

The $(\Delta+1)$ color bound is tight on cliques and odd cycles.

Linear Resource Algorithms

- This greedy coloring algorithm is extremely simple and can be implemented in linear time and linear space.
- Traditionally, solving a problem in linear time and space have been the gold standard of computational efficiency.
- But as we design algorithms that operate on very large data sets, this is often no longer sufficient.

Sublinear Algorithms

Can a $(\Delta+1)$-coloring be found by a sublinear algorithm?

Sublinear means sublinear in the number of edges. The output of $(\Delta+1)$-coloring is always linear in the number of vertices.

For instance, can a $(\Delta+1)$-coloring be found by an algorithm that examines only a tiny fraction of edges in the graph?

Based on the computational platform, we may want sublinear time, space, or communication algorithms.

Sublinear Time Algorithms

Query Model of Computation:

- Degree queries: What is the degree of a vertex v?
- Pair queries: Is (u, v) an edge?
- Neighbor queries: Who is the k_{th} neighbor of a vertex v?

Goal is to design algorithms that compute by performing only a few queries – much smaller than the size of the graph.

Sublinear Space Algorithms

Streaming Model of Computation

- The graph is presented as a stream of edges.
- The algorithm has limited memory to store information about the edges seen in the stream.
- A natural model when the input is either generated ``on the fly" or is stored on a sequential access device, like a disk.
- The algorithm no longer has random access to the input.

Goal is to design algorithms that use small space -- much smaller than the input size.

Sublinear Communication Algorithms

MPC Model of Computation

- The edges of the graph are partitioned across multiple machines in an arbitrary manner.
- Each machine has small memory – much smaller than the input.
- Computation proceeds in rounds where in each round, a machine can send and receive information to other machines (not exceeding its memory).

Goal is to compute in a small number of rounds.

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

Can a $(\Delta+1)$-coloring be found by a sublinear algorithm?

Computing an exact solution tends to be hard for sublinear algorithms as they typically gain efficiency by settling for a suitable notion of approximate solution.

Theorem: Any streaming algorithm for computing a maximal independent set requires $\Omega(n^2)$ space. Any query algorithm for computing a maximal matching requires $\Omega(n^2)$ time.

Just like $(\Delta+1)$-coloring , a simple greedy strategy gives a maximal independent set and a maximal matching .

Our Results

Surprisingly, one can obtain highly efficient sublinear algorithms for $(\Delta+1)$–coloring in all three models.

All our algorithms are randomized and behave as follows:

- either output a valid $(\Delta+1)$–coloring (w.h.p.), or
- output FAIL.

Our algorithms never output an invalid coloring.

Result 1: Sublinear Space Algorithms

Theorem 1: There is a $\tilde{O}(n)$ space single-pass streaming algorithm for computing a $(\Delta+1)$-coloring.

- $\Omega(n)$ space is needed just to store the solution.
- Best previous bound was $O(n^2)$ space.
- Our algorithm works even for dynamic graph streams where the stream consists of an arbitrary sequence of edge insertions and deletions.
  - Again surprising because for the related maximal matching problem, any algorithm for computing maximal matching in dynamic streams provably requires $\tilde{\Omega}(n^2)$ space.

Result 2: Sublinear Time Algorithms

Theorem 2: There is an $\tilde{O}(n^{3/2})$ time algorithm for computing a $(\Delta+1)$-coloring. Moreover, $\Omega(n^{3/2})$ queries are necessary.

- No algorithm better than the greedy algorithm was known previously.
- The queries performed by our algorithm are chosen non-adaptively.
- In contrast, the $\Omega(n^{3/2})$ lower bound holds even for adaptive algorithms.

Result 3: Sublinear Communication Algorithms

Theorem 3 : There is an $O(1)$ round MPC algorithm for computing a $(\Delta+1)$-coloring where each machine has $\tilde{O}(n)$ memory.

- If we assume public randomness, then our algorithm requires only a single round.
- Prior to our work, the state of the art was
  - $O(\log\log\Delta \log^* n)$ round algorithm with $\tilde{O}(n)$ memory [Parter '18].
  - Parallel to our work, round-complexity improved to $O(\log^* n)$ rounds [Parter and Su '18].
  - For the distinctly easier problem of $(\Delta + o(\Delta))$-coloring, an $O(1)$ round algorithm with $n^{1+\Omega(1)}$ memory [Harvey et al. '18].

Recent Work

Sublinear algorithms for degeneracy-dependent graph coloring [Bera, Chakrabarti, Ghosh'19].

Sublinear algorithms for $(\Delta+1)$-coloring in congested clique model, MPC model, and centralized local computation model [Chang, Fischer, Ghaffari, Uitto, Zheng '19].

How Do We Design These Sublinear Algorithms?

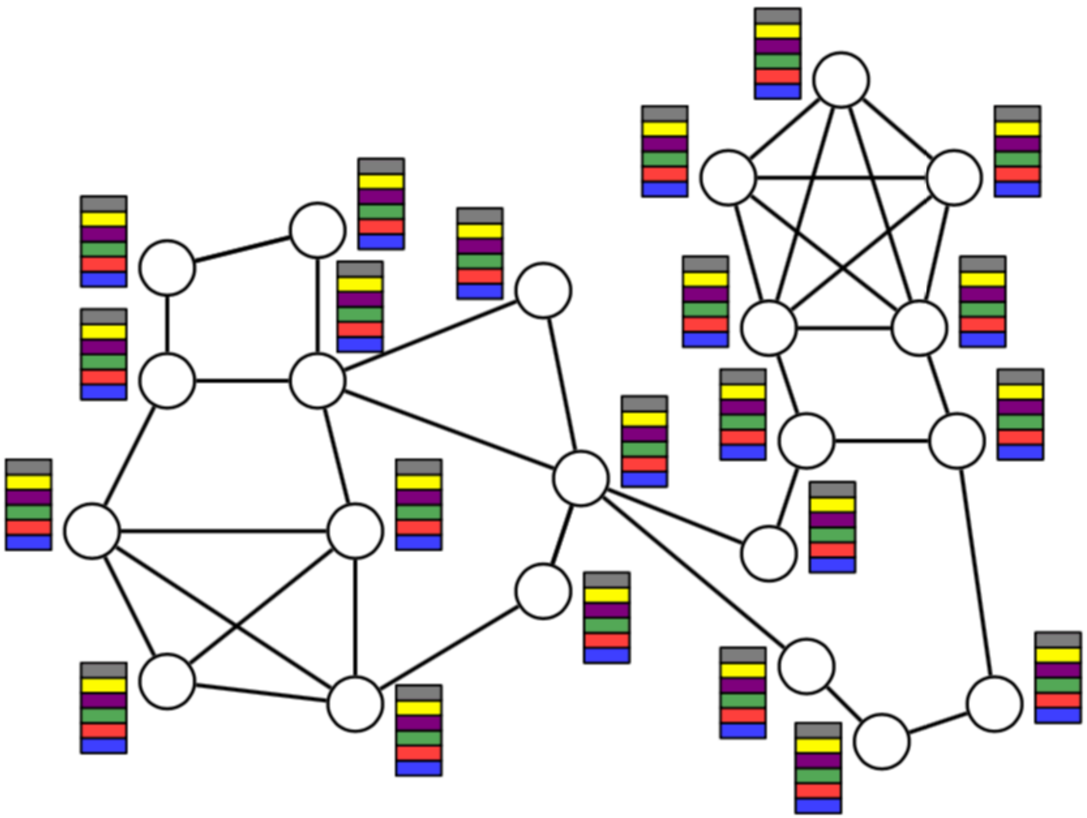
Palette Sparsification Theorem

The theorem below is at the heart of all three results.

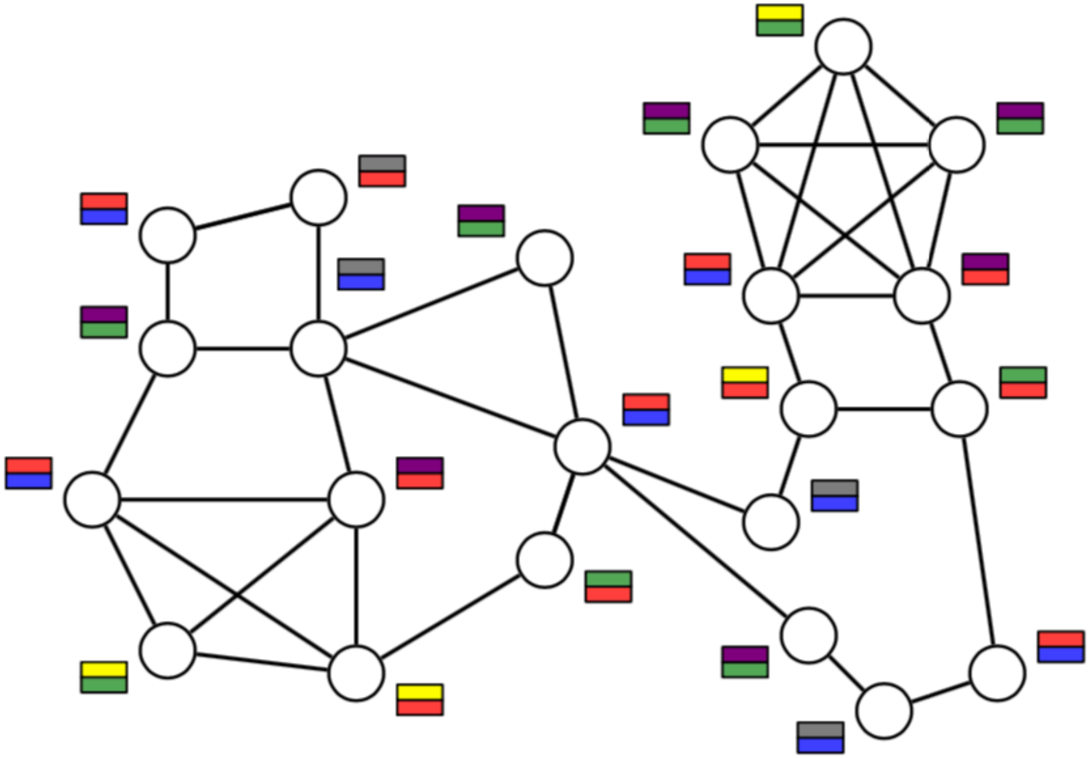
Palette Sparsification Theorem: Suppose each vertex in a graph G independently samples $O(\log n)$ colors uniformly at random from $\{1, 2, \dots, \Delta + 1\}$. Then w.h.p. there is a valid coloring of the graph G such that each vertex is assigned one of its sampled colors.

- A $(\Delta+1)$-coloring can be found using a highly sparsified palette of colors.
- The sparsification is oblivious to the structure of the graph!

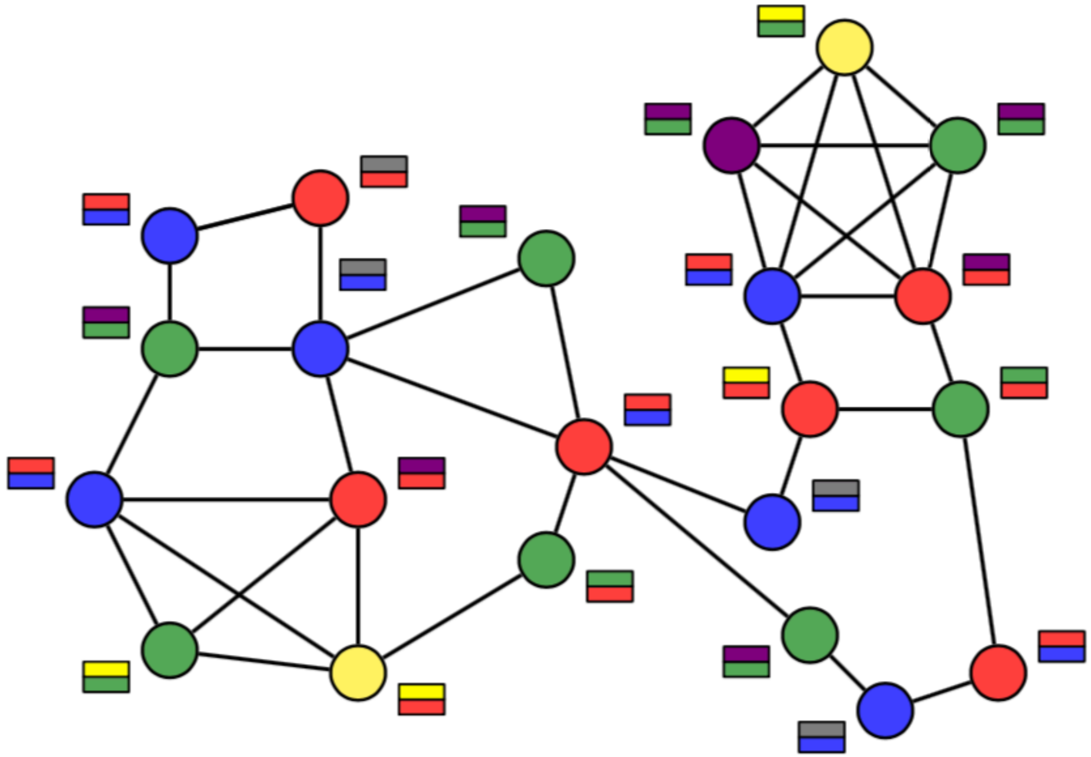
Palette Sparsification Illustrated



Palette Sparsification Illustrated



Palette Sparsification Illustrated



A Meta-Algorithm for $(\Delta + 1)$-Coloring

Input: A graph $G(V, E)$ with max degree Δ.

- At each vertex $v \in V$, sample $\Theta(\log n)$ colors, say $L(v)$, independently and uniformly at random.
- Let $E_{conflict}$ be the set of all edges $(u, v) \in E$ such that $L(u) \cap L(v) \neq \emptyset$.
- Construct the conflict graph $G_{conflict}(V, E_{conflict})$.
- Find a proper list-coloring of $G_{conflict}$ with $L(v)$ being the color list of vertex $v \in V$.

Properties of the Conflict Graph

- By construction, any list-coloring of $G_{conflict}$, if one exists, is a valid coloring of the input graph G.
- By Palette Sparsification theorem, with high probability there exists a list-coloring of $G_{conflict}$.
- So the problem of $(\Delta+1)$-coloring the input graph G can be reduced to the problem of list-coloring the graph $G_{conflict}$.
- Moreover, the process for constructing the graph $G_{conflict}$ is non-adaptive.

But what have we gained?

The Graph $G_{conflict}$ is Very Sparse

- For every edge (u, v) in G, the probability that it appears in $G_{conflict}$ is $\approx O(\log n) \times O\left(\frac{\log n}{\Delta}\right) = O\left(\frac{\log^2 n}{\Delta}\right)$.
- Thus the expected number of edges in $G_{conflict}$ is:

$$n\Delta \times O\left(\frac{\log^2 n}{\Delta}\right) = O(n \log^2 n) \text{ edges.}$$

Palette sparsification theorem thus allows non-adaptive sparsification of a graph with $O(n\Delta)$ edges to a graph with $\tilde{O}(n)$ edges while preserving a $(\Delta+1)$-coloring w.h.p.

Applications to Sublinear Algorithms

A One-Pass $\tilde{O}(n)$ Space Streaming Algorithm

- At the start, each vertex v samples $\Theta(\log n)$ colors independently and uniformly at random – let $L(v)$ be the set of colors sampled by vertex v.

- When an edge (u, v) arrives in the stream, we now determine its membership in the conflict graph by a simple test: if $L(u) \cap L(v) \neq \emptyset,$ add (u, v) to $E_{conflict}$.

- At the end of the stream, we list color the graph $G_{conflict}(V, E_{conflict})$.

A One-Pass $\tilde{O}(n)$ Space Streaming Algorithm

- Total space used by our algorithm is $\tilde{O}(n)$:
  - we need space to store the color lists $L(v)$, which is a total of $\tilde{O}(n)$ space, and
  - the size of $E_{conflict}$ which is also $\tilde{O}(n)$ space.
- The graph $G_{conflict}$ can be colored in $\tilde{O}(n)$ time.
- We can maintain the set $E_{conflict}$ in $\tilde{O}(n)$ space even for dynamic streams where the graph is revealed by an arbitrary sequence of edge insertions and deletions.

An $\tilde{O}(n^{3/2})$ Sublinear Time Algorithm

In the streaming model, the algorithm gets to see each edge once and can decide whether or not it belongs to the conflict graph.

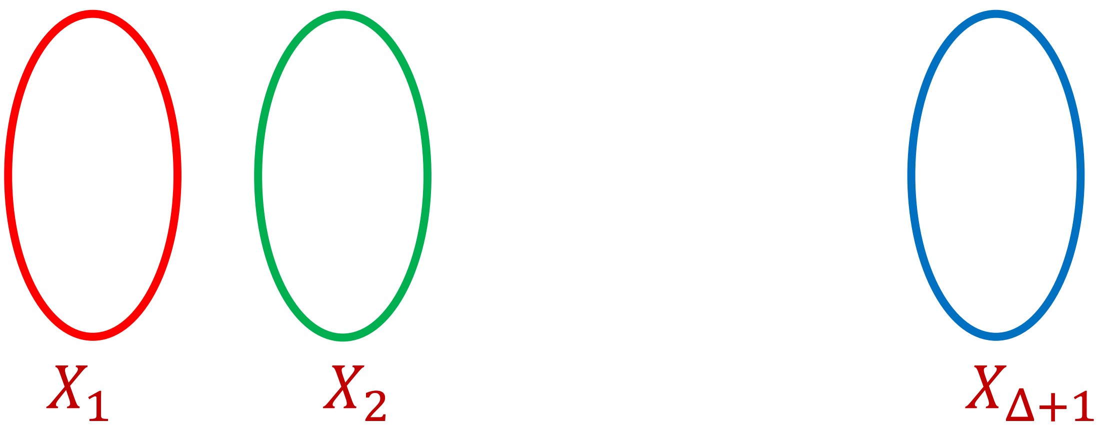
Challenge: How do we identify the edges in the conflict graph without examining each edge in the graph at least once?

We will show that the graph $G_{conflict}$ can be created by performing only $\tilde{O}(n^2/\Delta)$ queries.

An $\tilde{O}(n^{3/2})$ Sublinear Time Algorithm

Claim: The graph $G_{conflict}$ can be created in $\tilde{O}(n^2/\Delta)$ queries.

- It suffices that for each color $c \in [1\,..\,\Delta + 1]$, we find all edges (u, v) such that u and v both sample color c.
- For each color $c \in [1\,..\,\Delta + 1]$, let $X_c = \{v \mid c \in L(v)\}$.



An $\tilde{O}(n^{3/2})$ Sublinear Time Algorithm

Our Plan: Query all pairs of vertices in each set X_c to find all edges in $G_{conflict}$.

- For each color $c \in [1\,..\,\Delta + 1]$, a vertex samples it with probability $\approx \frac{\log n}{\Delta}$.
- So w.h.p. the size of each set X_c is $O(n \log n/\Delta)$.

Total # of queries = $(\Delta + 1) \times [O(n \log n/\Delta)]^2 = \tilde{O}(n^2/\Delta)$.

An $\tilde{O}(n^{3/2})$ Sublinear Time Algorithm

Claim: A $(\Delta+1)$-coloring can be found in $\tilde{O}(n^{3/2})$ time.

- If $\Delta \leq n^{1/2}$, then we can use the standard $O(n\ \Delta)$ time greedy algorithm.
- Otherwise, $\Delta > n^{1/2}$, and we can use the previous claim to create $G_{conflict}$ in $\tilde{O}(n^2/\Delta) = \tilde{O}(n^{3/2})$ time.

Omitted detail: The graph $G_{conflict}$ can also be list-colored in $\tilde{O}(n^{3/2})$ time.

A 1-Round MPC Algorithm

MPC Model of Computation

- The edges of the graph are partitioned across multiple machines in an arbitrary manner.
- Each machine has $\tilde{O}(n)$ memory.
- Computation proceeds in rounds where in each round, a machine can send and receive $\tilde{O}(n)$ bits of information.

Let us assume for simplicity that machines share public randomness. This assumption can be eliminated by adding $O(1)$ additional rounds.

An 1-Round MPC Algorithm

- Each machine checks which edges in its input belong to the conflict graph $G_{conflict}$.

- We designate one machine as special, and all other machines now send the edges in $G_{conflict}$ that are in their input. Total communication to the special machine is $\tilde{O}(n)$.

- The special machine now computes a list-coloring of $G_{conflict}$.

How Do We Prove The Palette Sparsification Theorem?

Proof Idea for Palette Sparsification

Suppose we only have low degree vertices in our graph – a vertex is low degree if its degree is at most $(\Delta/2)$.

- Each uncolored vertex samples a color at random.
- We process uncolored vertices one by one and assign the sampled color to an uncolored vertex if none of its neighbors have same color.
- Repeat the steps above until all vertices are colored.

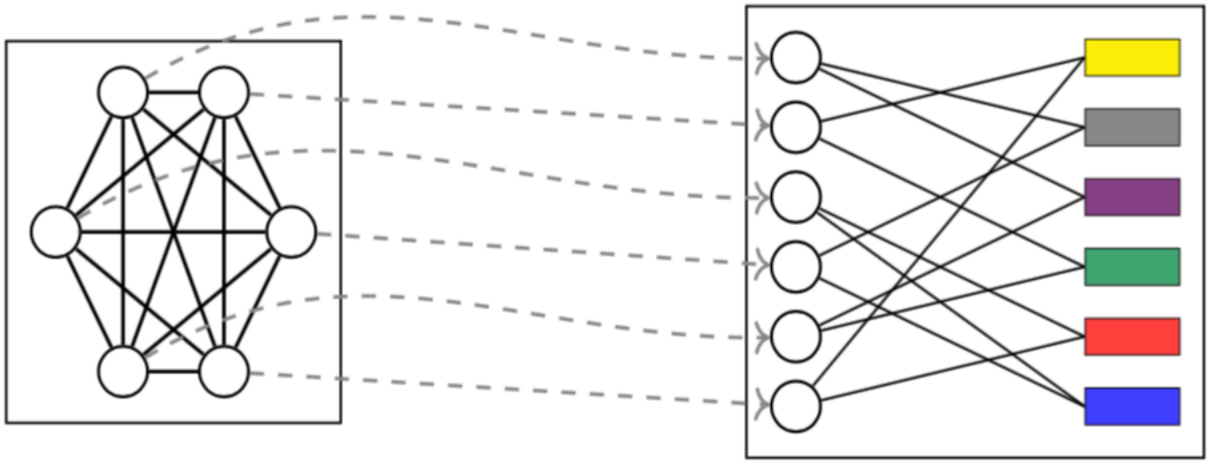
Each vertex has a constant probability of being colored in a single round. So after $O(\log n)$ rounds, w.h.p. all vertices are colored, proving the palette sparsification theorem.

Proof Idea for Palette Sparsification

Now suppose we only have high degree vertices.

- If we were to use the previous approach to color a clique on $(\Delta + 1)$ vertices, then you provably need $\Omega(\Delta)$ rounds, and hence a palette of $\Omega(\Delta)$ colors.
- We need some coordination to find a coloring in this case.
- We will view the $(\Delta+1)$-coloring problem as a matching problem.

Coloring the Clique $K_{\Delta+1}$

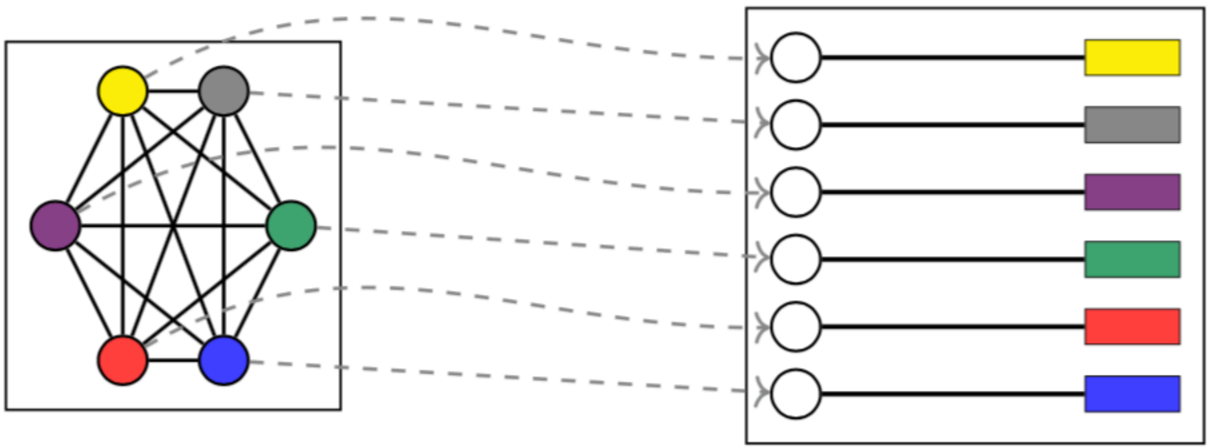


Original Graph

Palette Graph

(Δ+1)-coloring: Find a perfect matching in the palette graph.
Palette Sparsification Theorem: random subgraphs of the complete palette graph contain a perfect matching.

Coloring the Clique $K_{\Delta+1}$

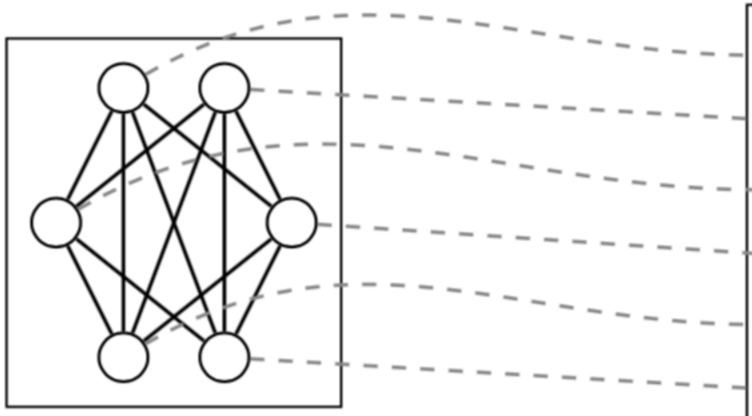


Original Graph Palette Graph

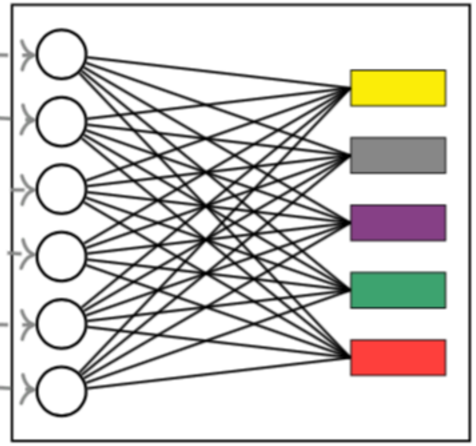
$(\Delta+1)$-coloring: Find a perfect matching in the palette graph.

Palette Sparsification Theorem: random subgraphs of the complete palette graph contain a perfect matching.

Coloring $K_{\Delta+1}$ Minus a Perfect Matching



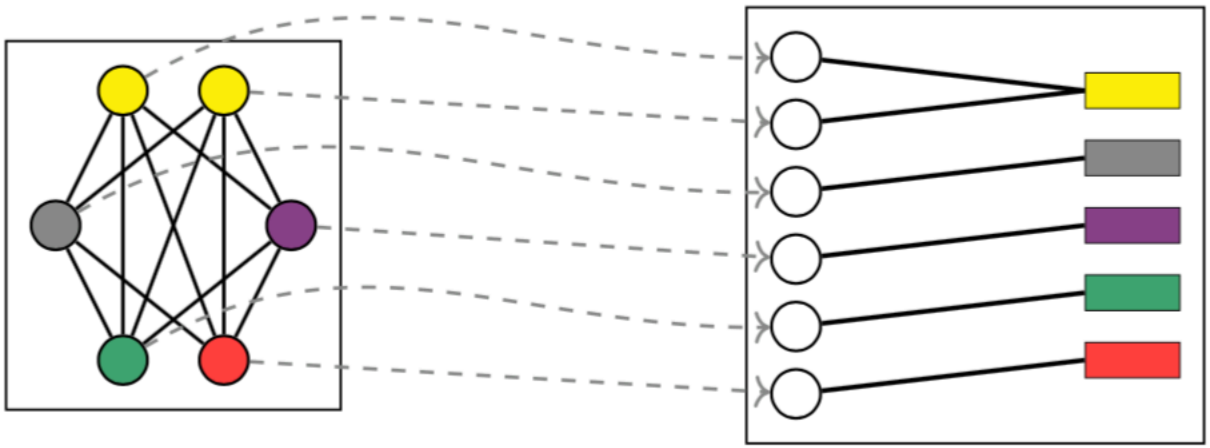
Original Graph



Palette Graph

(Δ+1)-coloring: Find a constrained b-matching in the palette graph.
Palette Sparsification Theorem: random subgraphs of the complete palette graph contain a constrained b-matching.

Coloring $K_{\Delta+1}$ Minus a Perfect Matching



Original Graph

Palette Graph

(Δ+1)-coloring: Find a constrained b-matching in the palette graph.
Palette Sparsification Theorem: random subgraphs of the complete palette graph contain a constrained b-matching.

The General Case

- Handling almost-clique like structures in generality is a key challenge for proving the palette sparsification theorem.
  - The goal is to find a constrained b-matching where every vertex on left is matched to exactly one color on the right, and the set of vertices assigned to any color form an independent set in G.

- Furthermore, we also need to find an approach that can interpolate between these two views on any graph.
  - Simulating a greedy algorithm for low degree graphs.
  - Solving a constrained b-matching problem on almost-cliques.

Our approach: decompose the graph into sparse and dense regions and apply the appropriate view to each region.

A Network Decomposition Theorem

We extend a decomposition result of Harris, Schneider, and Su 2016 for distributed (Δ+1)-coloring.

Theorem 4 [Extended HSS Decomposition]: For any $\epsilon \in (0,1)$, a graph can be decomposed into structures below:

Sparse vertices: neighborhood of each sparse vertex is missing at least $\epsilon\binom{\Delta}{2}$ edges.

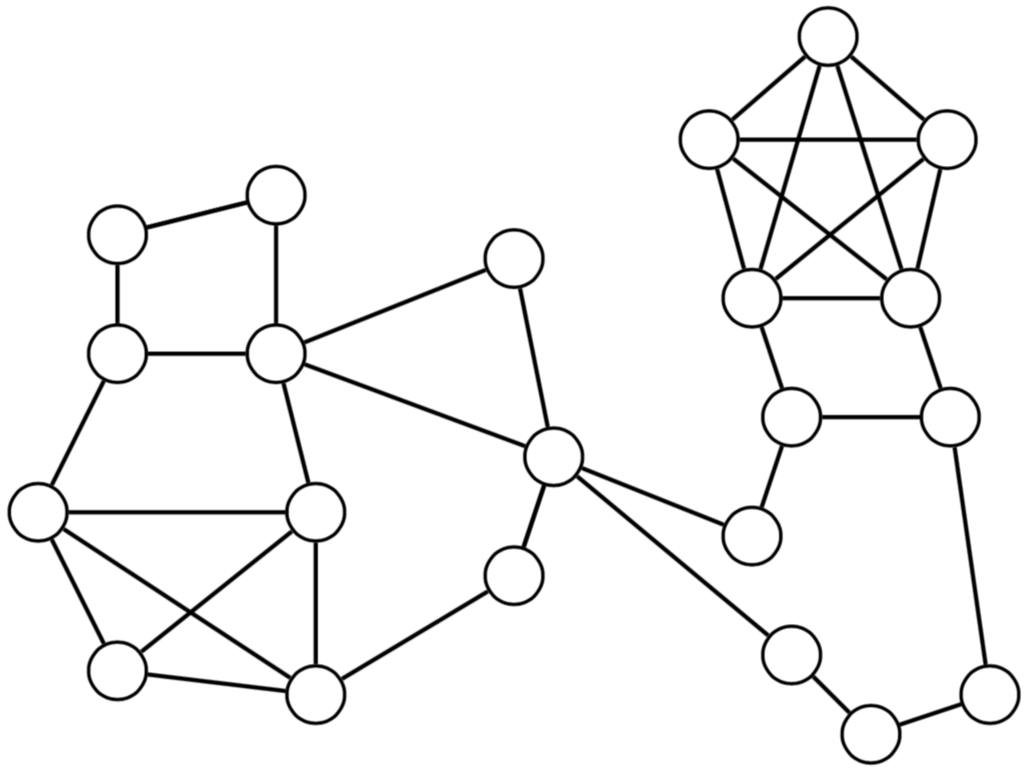
Almost cliques: Each almost clique C contains $(1 \pm \epsilon)\Delta$ vertices s.t.

- Every vertex in C has at most $\epsilon\Delta$ non-neighbors in C.
- Every vertex in C has at most $\epsilon\Delta$ neighbors outside C.

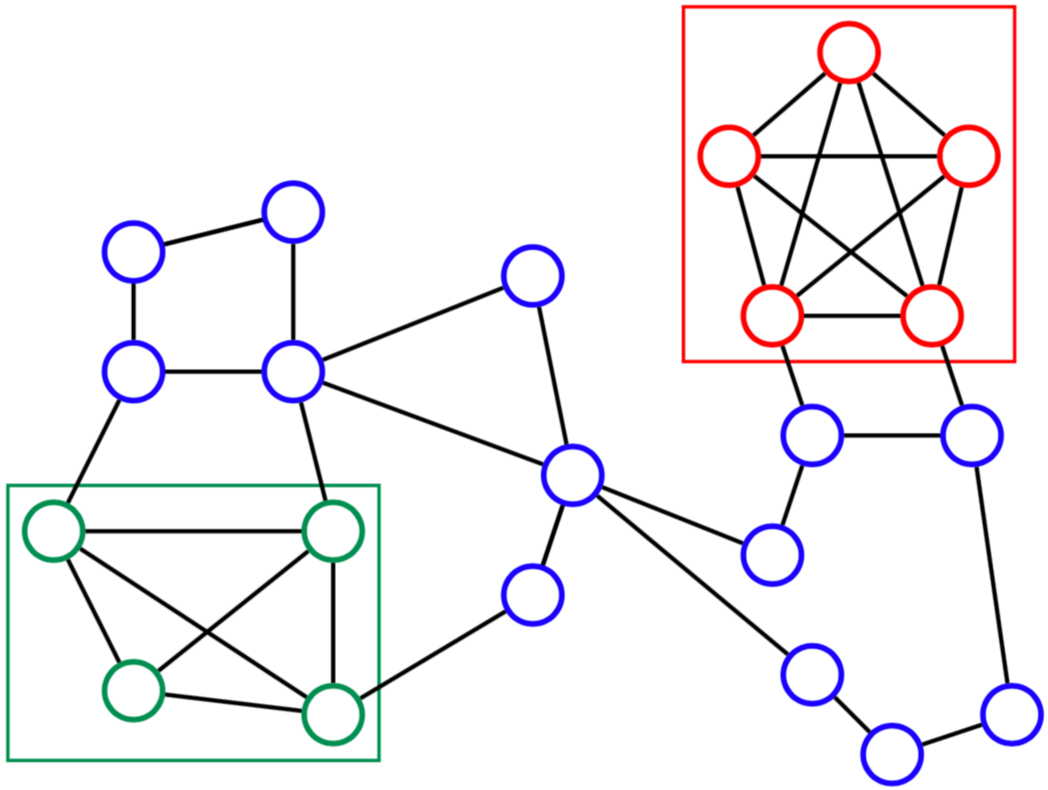
Proving Palette Sparsification Theorem

- Fix an extended HSS decomposition for a small value of ϵ.
- First color the sparse vertices using the greedy strategy.
- Next process almost-cliques one by one and list-color using the constrained b-matching view: the challenging part.
- While we only need an existence argument for proving the palette sparsification theorem, this constructive proof strategy can be turned into an efficient algorithm into each of the three sublinear models considered here.

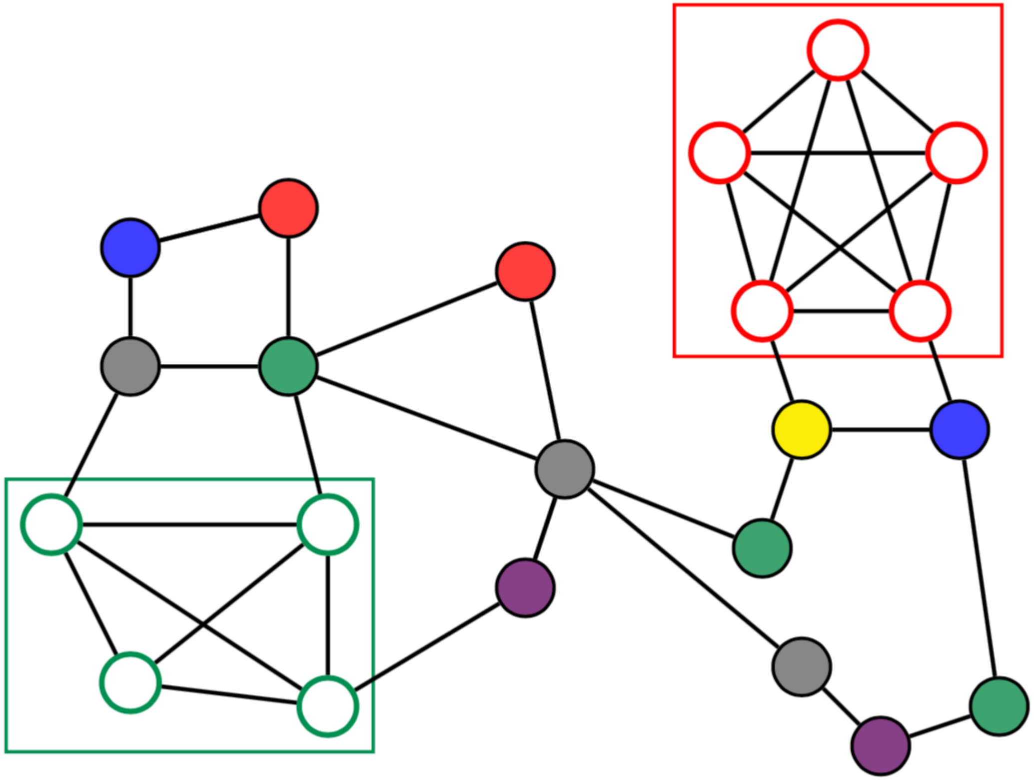
Proving Palette Sparsification Theorem



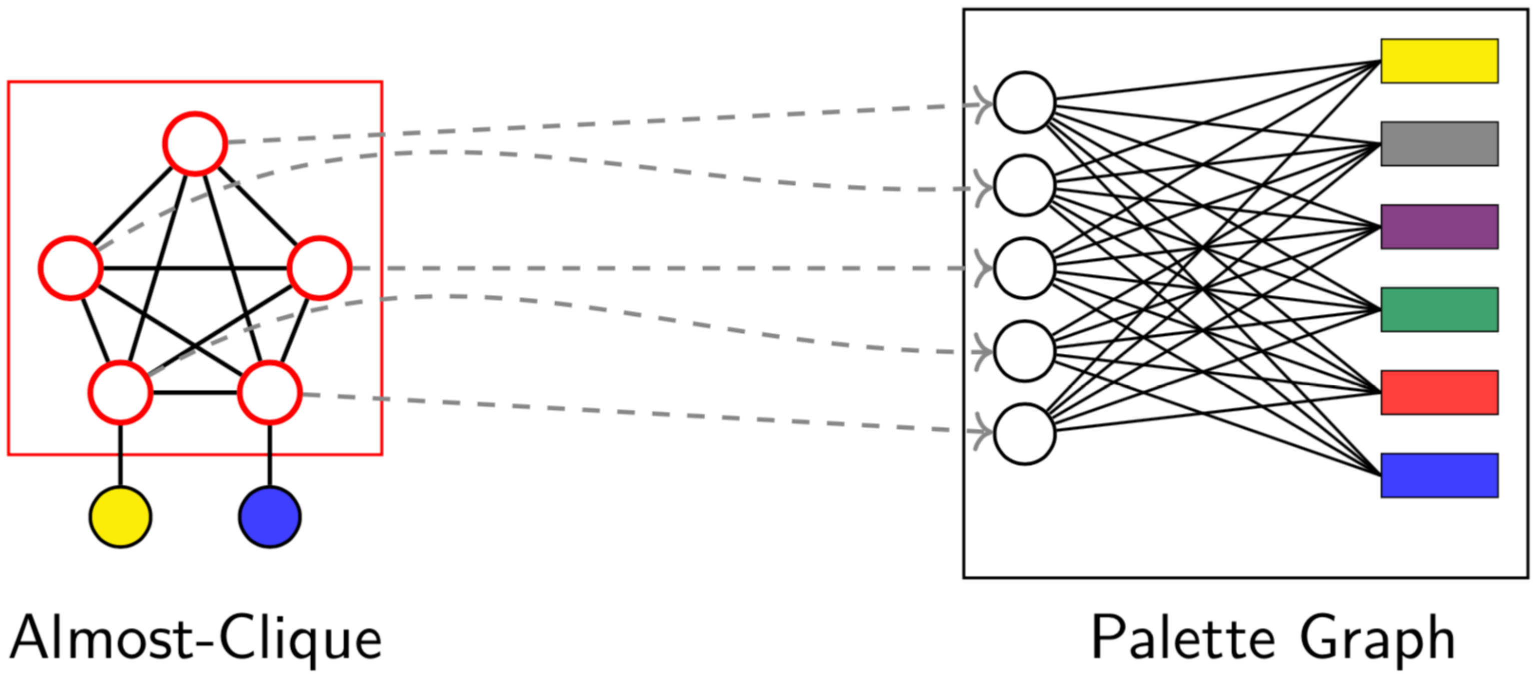
Proving Palette Sparsification Theorem



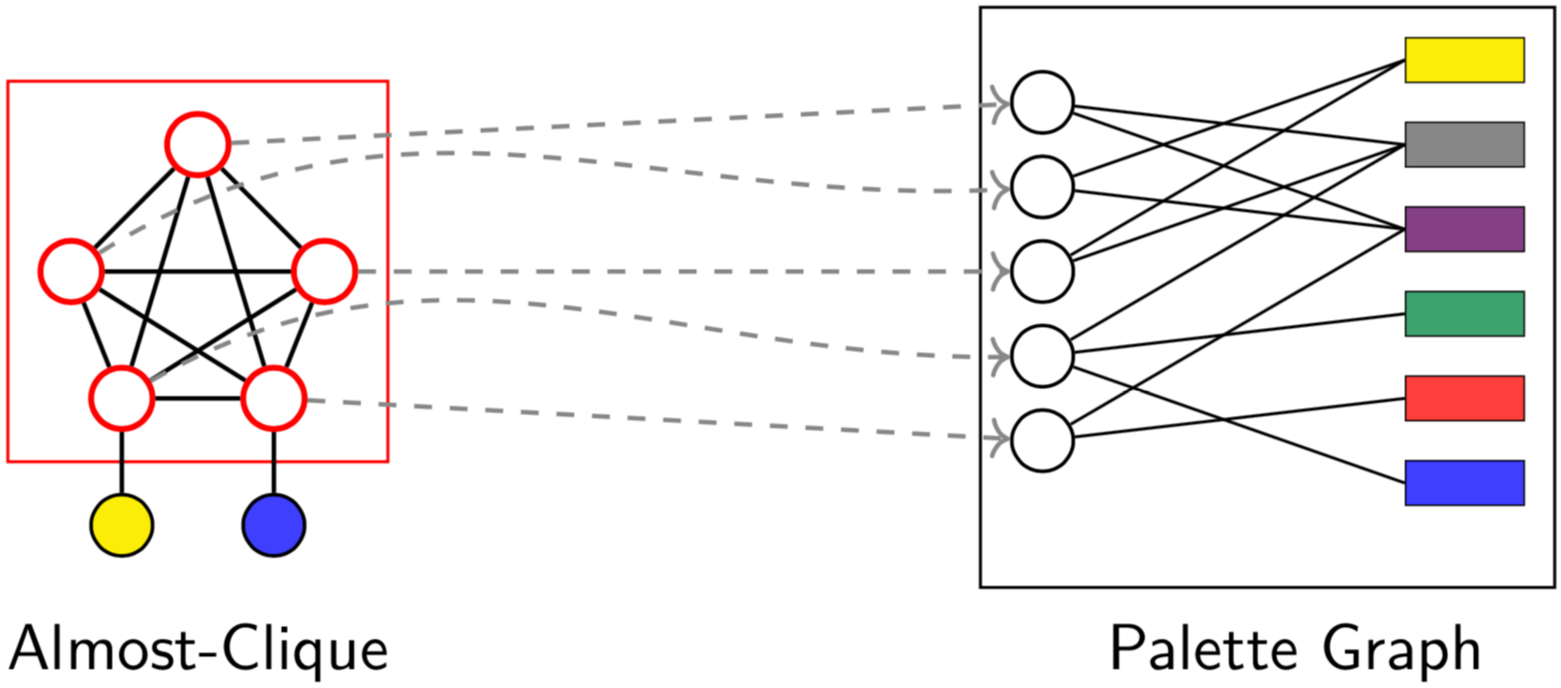
Proving Palette Sparsification Theorem



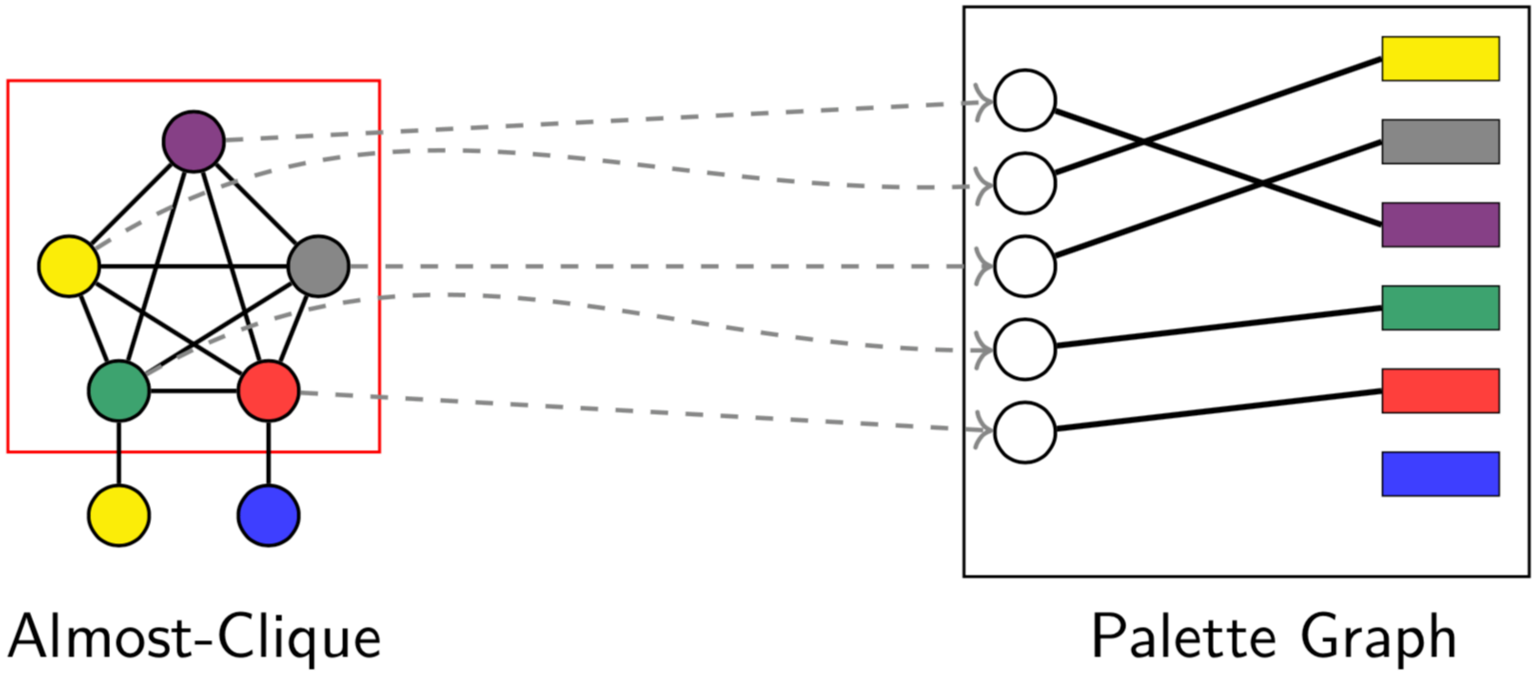
Proving Palette Sparsification Theorem



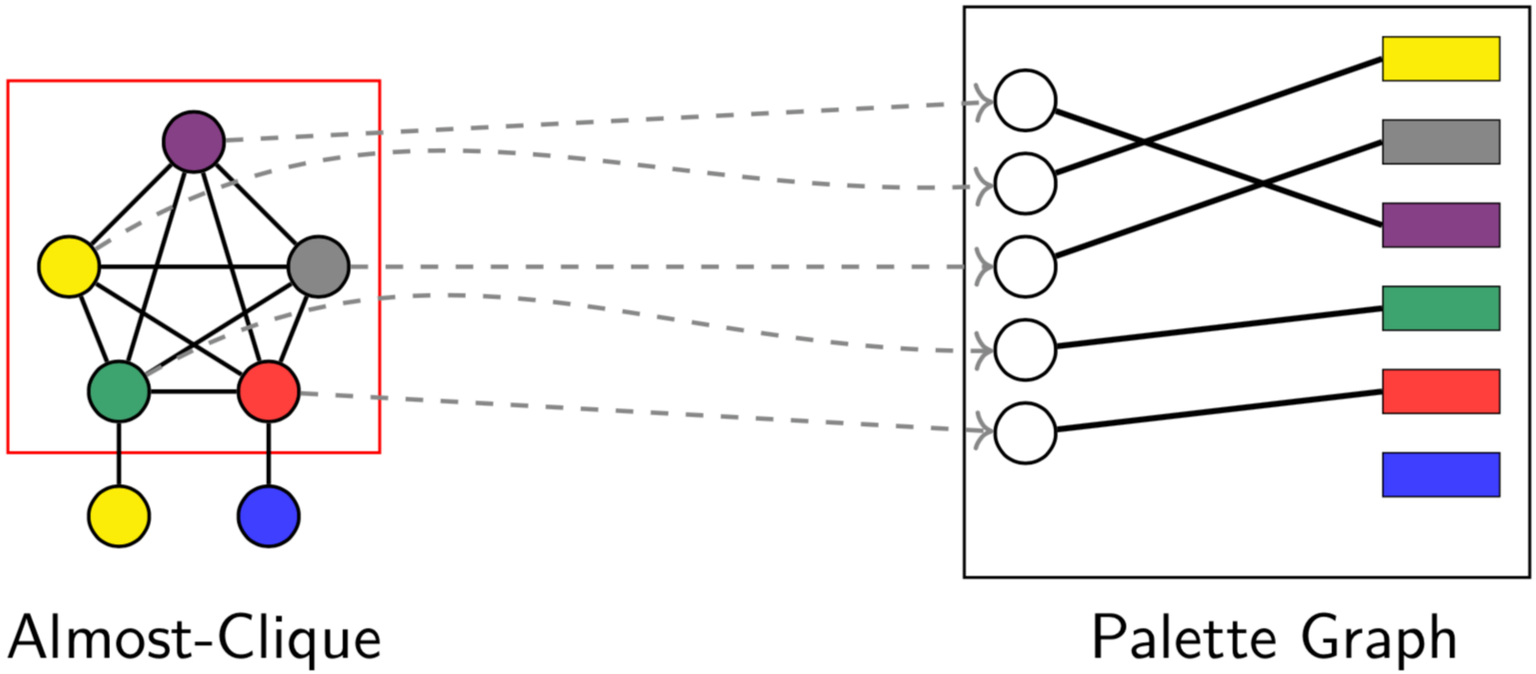
Proving Palette Sparsification Theorem



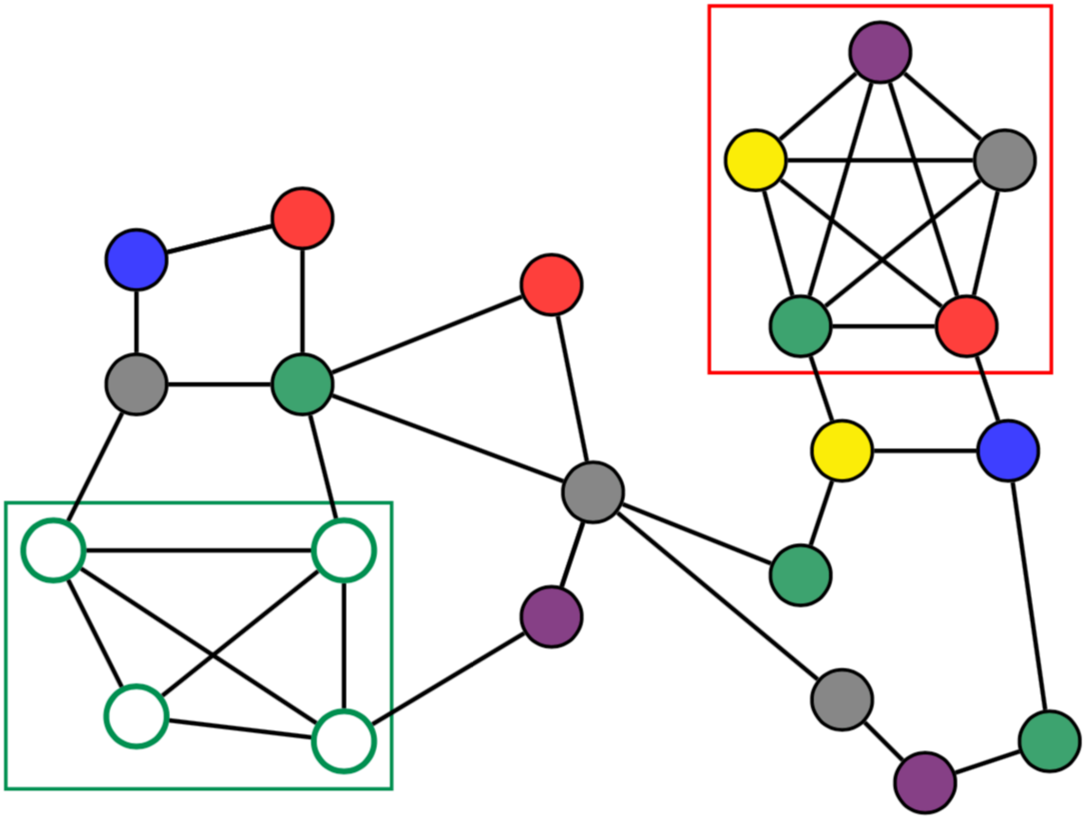
Proving Palette Sparsification Theorem



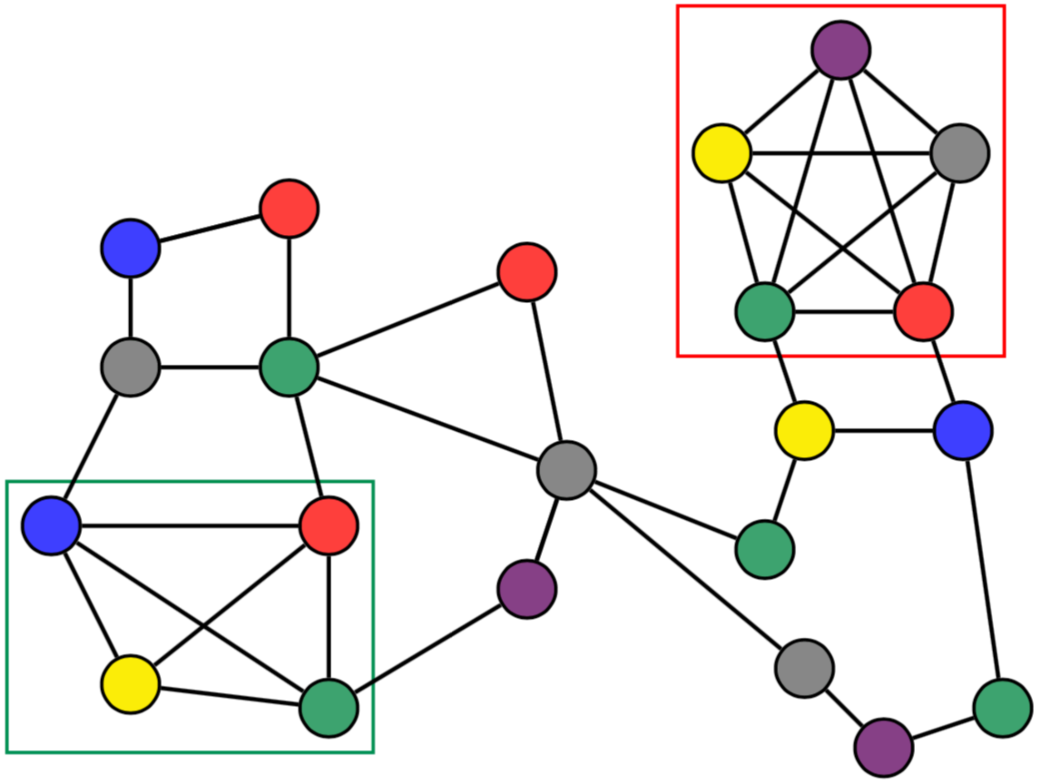
Proving Palette Sparsification Theorem



Proving Palette Sparsification Theorem



Proving Palette Sparsification Theorem



Strengthening the Palette Sparsification Theorem?

Palette Sparsification with Fewer Colors

Could we establish palette sparsification theorem by sampling only O(1) colors at each vertex?

Claim: There exist graphs such that if each vertex samples $o(\log n)$ colors, then almost certainly no feasible coloring exists among sampled colors.

- Suppose G consists of $n/\log n$ copies of the graph $K_{\log n}$.
- Then if each vertex samples $o(\log n)$ colors, then almost certainly some clique fails to sample $\log n$ distinct colors.
- So almost certainly no valid coloring exists among sampled colors.

Palette Sparsification for C-Coloring

Does a similar sparsification result hold for C-colorable graphs for $C \leq \Delta$?

Claim: There exist Δ-colorable graphs such that unless each vertex samples $\Omega(\Delta^{0.5})$ colors, w.h.p. there is no feasible coloring exists among the sampled colors.

- Consider the graph $K_{\Delta+1}$ and remove an edge between any pair u, v of vertices – this is a Δ-colorable graph.
- In any Δ-coloring, u and v must receive the same color.
- But unless each of u and v samples at least $\Omega(\Delta^{0.5})$ colors, this is unlikely.

Concluding Remarks

- We showed a non-adaptive sparsification result for (Δ+1)-coloring.
- Any graph can be sparsified to a graph with $\tilde{O}(n)$ edges such that list-coloring the sparsified graph is equivalent to (Δ+1)-coloring the original graph.
- The sparsification can be used to obtain essentially optimal sublinear algorithms for three well-studied models of sublinear algorithms:
  - Streaming model for sublinear space,
  - Query model for sublinear time, and
  - MPC model for sublinear communication.

Thank you !