Design and Analysis of Algorithms for Packing Coloring

theroyakash and B.V. Raghavendra Rao

Indian Institute of Technology Madras

January 10, 2024

Introduction to Packing Coloring

We start with a few definitions. First we define what is graph coloring.

Introduction to Packing Coloring

We start with a few definitions. First we define what is graph coloring.
Let $G = (V, E)$ be an undirected graph.

Introduction to Packing Coloring

We start with a few definitions. First we define what is graph coloring.
Let $G = (V, E)$ be an undirected graph.

Graph Coloring

Definition 1.1: Vertex Coloring

A vertex k coloring of G is a map $f : V \to \{1, \ldots, k\}$. A coloring f is said to be proper, if for every edge $(u, v) \in E$, $f(u) \neq f(v)$. The chromatic number of a graph is the minimum value of k such that G has a proper k colouring.

Graph Packing Coloring

- Given the centrality of graph colouring to graph theory and computer science, there have been several variants of colouring problems on graphs with some additional conditions imposed.

Graph Packing Coloring

- Given the centrality of graph colouring to graph theory and computer science, there have been several variants of colouring problems on graphs with some additional conditions imposed.
- One such variant is the packing coloring problem.

Graph Packing Coloring

- Given the centrality of graph colouring to graph theory and computer science, there have been several variants of colouring problems on graphs with some additional conditions imposed.
- One such variant is the packing coloring problem.
- List coloring, path coloring, repetition free colouring are some of the other prominent examples.

Graph Packing Coloring

We begin with a definition of the graph packing-coloring problem.

Graph Packing Coloring

Definition 1.2: S-Packing Coloring and Packing Coloring

Suppose $S = (a_i)_{i \in [1 \to \infty)}$ is a increasing sequence of integers, then S packing coloring of the graph is partition on the vertex set $V(G)$ into sets $V_1, V_2, V_3 \ldots$ such that for every pair $(x, y) \in V_k$ is at a distance more than a_k. If $a_i = i$ for every $i \in [1 \to \infty)$, then we call the problem packing coloring.

Graph Packing Coloring

Definition 1.3: S-Packing Chromatic Number

If there exists an integer k such that $V(G) = V_1, V_2, V_3 \ldots V_k$, each V_i is a vertex-partition, then this partition is called S-packing, k coloring, and minimum of such k is the S-Packing Chromatic Number.

Difference between Packing Coloring and Graph Coloring

- There are several approximation algorithms for graph coloring, but there aren't any approximation algorithms for packing coloring.

Difference between Packing Coloring and Graph Coloring

- There are several approximation algorithms for graph coloring, but there aren't any approximation algorithms for packing coloring.
- Any graph is three-colorable is a NP-complete problem. So there are reasons to develop an approximation scheme for graph coloring.

Difference between Packing Coloring and Graph Coloring

- There are several approximation algorithms for graph coloring, but there aren't any approximation algorithms for packing coloring.
- Any graph is three-colorable is a NP-complete problem. So there are reasons to develop an approximation scheme for graph coloring.
- We also know from early on that the decision version of the packing coloring is a NP complete problem.

Difference between Packing Coloring and Graph Coloring

- There are several approximation algorithms for graph coloring, but there aren't any approximation algorithms for packing coloring.
- Any graph is three-colorable is a NP-complete problem. So there are reasons to develop an approximation scheme for graph coloring.
- We also know from early on that the decision version of the packing coloring is a NP complete problem.
- The decision version in the form of *A graph G and a positive integer K, does G have a packing-K coloring*, is NP-complete for $k = 4$ even when restricted to *planner* graphs.

Packing Coloring on Trees

Packing coloring is also NP*-Hard for the case of trees (which are acyclic undirected unweighted graphs).*

Packing Coloring on Trees

Packing coloring is also NP-Hard for the case of trees (which are acyclic undirected unweighted graphs).

It is one thing to compute the decision version of the packing coloring problem for which we don't have any efficient algorithm. It is equally or more difficult to have a fast algorithm for getting hold of an valid packing coloring assignment.

Packing Coloring on Trees

In this presentation I'll show an algorithm that gets us a valid packing coloring assignment in polynomial time.

Packing Coloring on Trees

In this presentation I'll show an algorithm that gets us a valid packing coloring assignment in polynomial time. However the algorithm do not get us the minimum number of colors that is the packing chromatic number. It is ***an approximation*** *of the actual packing chromatic number.*

Simple Algorithm for Packing Coloring

We are to find a valid assignment of packing coloring to the vertices of the graph.

Simple Algorithm for Packing Coloring

We are to find a valid assignment of packing coloring to the vertices of the graph. Most straight forward algorithm we can think of is simply assign some color, and backtrack and re-color in case of conflicts.

Simple Greedy Heuristic for Packing Coloring

- *We'll start our algorithm to work specifically on trees first because it is easier to analyze and then we'll extend it to general graphs.*

Simple Greedy Heuristic for Packing Coloring

- *We'll start our algorithm to work specifically on trees first because it is easier to analyze and then we'll extend it to general graphs.*
- *In trees every odd layer we can color with a single color* 1 *as every odd layer nodes are at distance more than* 1*.*

Simple Greedy Heuristic for Packing Coloring

- *We'll start our algorithm to work specifically on trees first because it is easier to analyze and then we'll extend it to general graphs.*
- *In trees every odd layer we can color with a single color* 1 *as every odd layer nodes are at distance more than* 1*.*
- *Hence number of Node remains to be colored is significantly less than the total nodes* (n)*. For example a complete* 3*-ary tree we can color* 75% *of the nodes with color* 1*.*

Simple Greedy Heuristic for Packing Coloring

- *We'll start our algorithm to work specifically on trees first because it is easier to analyze and then we'll extend it to general graphs.*
- *In trees every odd layer we can color with a single color* 1 *as every odd layer nodes are at distance more than* 1.
- *Hence number of Node remains to be colored is significantly less than the total nodes* (n). *For example a complete* 3*-ary tree we can color* 75% *of the nodes with color* 1.
- *We first see how algorithm working on a complete trees. Then we'll look into some of the optimizations we can do to improve the performance of the algorithm.*

Simple Greedy Heuristic for Packing Coloring

Algorithm 1: BASIC GREEDY ALGORITHM FOR ANY TREE

Input: Tree T

Compute Level order traversal of Tree T;
Color Every Odd layer nodes with **COLOR(1)**;
`level` $\leftarrow d - 1$ (d is the last level);
while *`level`* ≥ 0 **do**
    `maximum_permissible_color` $= n$;
    `current_color` $= 2$;
    foreach *Node in this level* **do**
        while *`current_color`* $<$ *`maximum_permissible_color`* **do**
            Travel to every node within distance (`int`) `current_color` and check if there is any node colored with color `current_color`;

Simple Greedy Heuristic for Packing Coloring

Algorithm 2: BASIC GREEDY ALGORITHM FOR ANY TREE

Travel to every node within distance (`int`) `current_color` and check if there is any node colored with color `current_color`;

if *None of the node is colored with color* `current_color` **then**

  Color this node with color `current_color`;

  break from the loop, go to next node in level;

else

  `current_color` $\leftarrow$ `current_color` $+1$;

level $\leftarrow$ level -1;

Output: Output this coloring assignment.

Analysis of the Basic Algorithm

During the analysis we find that there are optimizations we can do to improve the run-time of our algorithm.

Complexity Analysis

Our algorithm for each node $i \in (1, n)$ in the worst case visits all the n node to find a color (from $1 \to n$). Hence worst case time complexity is $O(n^3)$.

Optimizations 1

We observe one simple fact, that for any complete tree, the maximum number of nodes at any level is present at the last level ($= x^d$, x is the number of children and d is the depth of the last level starting root from 0).

Optimizations 1

We are coloring every odd layer with color 1.

Optimizations 1

We are coloring every odd layer with color 1.
Instead of that if we color the last level and then every alternate level with color 1 we'll color much more nodes with color 1 and reduce the total number of colors used. Here is a simple example how this optimization saved thousands of colors.

Optimizations 1

Nodes	Layers	Maximum Colors used	Runtime
265720	12	20633	52m 32s 280ms
265720	12	6890	4m 17s 31ms

Optimizations 1

Nodes	Layers	Maximum Colors used	Runtime
265720	12	20633	52m 32s 280ms
265720	12	6890	4m 17s 31ms

This one simple optimization reduces the runtime by 92%

Optimizations 2

Suppose we are at the moment trying to color node u. Our algorithm for each color $i \in (1, n)$ goes to distance i from the node u and checks if that color exists already or not in all the nodes sitting within distance i from node u?

Optimizations 2

Suppose we are at the moment trying to color node u. Our algorithm for each color $i \in (1, n)$ goes to distance i from the node u and checks if that color exists already or not in all the nodes sitting within distance i from node u?

Lets see this step of the basic algorithm with an example.

Optimizations 2

- Suppose we are currently looking to color some node u with color d.

Optimizations 2

- Suppose we are currently looking to color some node u with color d.
- We went to d distance from node u to find all the colors we find. Suppose we find color $d, d+1, \ldots, d+k$ are present in some of the nodes.

Optimizations 2

- Suppose we are currently looking to color some node u with color d.
- We went to d distance from node u to find all the colors we find. Suppose we find color $d, d+1, \ldots, d+k$ are present in some of the nodes.
- So we should not check this again for node u with color $d + (1 \rightarrow k)$.

Optimizations 2

- Suppose we are currently looking to color some node u with color d.
- We went to d distance from node u to find all the colors we find. Suppose we find color $d, d+1, \ldots, d+k$ are present in some of the nodes.
- So we should not check this again for node u with color $d + (1 \to k)$.
- Hence we implement this modification to improve the runtime.

Optimizations 2

We define a subroutine called `Check(u, d)`*. This subroutine returns a set of colors present within distance d for any node u.*

Optimizations 2

We define a subroutine called **`Check(u, d)`**. *This subroutine returns a set of colors present within distance d for any node u.*

Algorithm 4: `Check(Node u, Color d)`

$\mathcal{C} \leftarrow \phi$;
Visit all nodes within distance d from node u and collect all the colors into $\mathcal{C}$;
return Set $\mathcal{C}$;

Optimizations 2

We define a subroutine called **`Check(u, d)`**. *This subroutine returns a set of colors present within distance d for any node u.*

Algorithm 5: `Check(Node u, Color d)`

$\mathcal{C} \leftarrow \phi$;

Visit all nodes within distance d from node u and collect all the colors into $\mathcal{C}$;

return Set $\mathcal{C}$;

We can call this subroutine from the main coloring BFS call (we are coloring left to right, level by level). We start with the color 2 and then we follow the following coloring strategy.

New coloring strategy

Algorithm 6: `Updated Main Coloring Scheme`

for *Each node from last uncolored level, left to right* **do**
    $d_{\text{prev}} \leftarrow \phi$;
    for *Each Color i from* $2 \rightarrow n$ **do**
        if *Color* $i \in d_{prev}$ **then**
            It is not possible to color this node with color i because we found color i at distance less than i in d_{prev};
            Continue with color $i + 1$;
        else

New coloring strategy

Algorithm 7: `Updated Main Coloring Scheme`

for *Each node from last uncolored level, left to right* **do**
- $d_{\text{prev}} \leftarrow \phi$;
- **for** *Each Color* i *from* $2 \rightarrow n$ **do**
  - **if** *Color* $i \in d_{prev}$ **then**
  - **else**
    - $d_{\text{new}} = \texttt{Check(node, i, }d_{\text{prev}}\texttt{);}$
    - **if** $i \notin d_{new}$ **then**
      - Color this node with color i;
      - Break from this loop and start coloring next uncolored node;
    - **else**
      - $d_{\text{prev}} = d_{\text{new}}$

Optimization 3

- This optimization comes from the observations of the structure of the complete trees.

Optimization 3

- This optimization comes from the observations of the structure of the complete trees.
- Complete x-ary trees has a depth of $\log_x n$ with n many nodes in them.

Optimization 3

- This optimization comes from the observations of the structure of the complete trees.
- Complete x-ary trees has a depth of $\log_x n$ with n many nodes in them.
- With the following optimization our algorithm time complexity will reduce from $O(n^3)$ down to $O(nd^2)$ for x-ary trees with d depth. This is a significant complexity improvement.

Optimization 3

- Suppose j is a color that has been used in the tree for the first time (during our run of the algorithm).

Optimization 3

- Suppose j is a color that has been used in the tree for the first time (during our run of the algorithm).
- If $j >$ the longest path in the tree, then color j can never be used again.

Optimization 3

- Suppose j is a color that has been used in the tree for the first time (during our run of the algorithm).
- If $j >$ the longest path in the tree, then color j can never be used again.
- Any color after j that is $j + 1$ and so on will also not be possible to reuse.

Optimization 3

- Suppose j is a color that has been used in the tree for the first time (during our run of the algorithm).
- If $j >$ the longest path in the tree, then color j can never be used again.
- Any color after j that is $j + 1$ and so on will also not be possible to reuse.
- So there is a upper bound on the number of color that are reusable. This depends on the longest path on the tree.

Optimization 3

- For a complete tree longest path is the diameter of the tree. This is equal to $O(\log_x n)$ for an complete x-ary tree.

Optimization 3

- For a complete tree longest path is the diameter of the tree. This is equal to $O(\log_x n)$ for an complete x-ary tree.
- So in our coloring algorithm we do a simple modification.

Optimization 3

- For a complete tree longest path is the diameter of the tree. This is equal to $O(\log_x n)$ for an complete x-ary tree.
- So in our coloring algorithm we do a simple modification.
- For each of the node we only check for colors from $1 \rightarrow 2 * d + 2$. The value $2d + 2$ is always the upper bound on the color that can be reusable.

Optimization 3

- For a complete tree longest path is the diameter of the tree. This is equal to $O(\log_x n)$ for an complete x-ary tree.
- So in our coloring algorithm we do a simple modification.
- For each of the node we only check for colors from $1 \rightarrow 2 * d + 2$. The value $2d + 2$ is always the upper bound on the color that can be reusable.
- It is loose bound, can be improved to $\epsilon_1 * d \pm \epsilon$ for some fraction ϵ_1 and some integer ϵ. But that would not improve the time complexity of our algorithm.

Experimental Results

We will quickly look at some experimental results on the number of reusable colors and the diameter of the tree.

Experimental Results

Color number	Number of nodes
1	7381
2, 3	738
4, 5	244
6, 7	81
8, 9	27
10, 11	9
12, 13	3
$14 \rightarrow 269$	1

Table: Number of nodes colored with each color for a 9-layer complete tree.

Mathematical Analysis

- We look into some of the properties of the algorithm, how it performs on a three-ary complete tree.

Mathematical Analysis

- We look into some of the properties of the algorithm, how it performs on a three-ary complete tree.
- We'll analyse a upper-bound on the number of colors used in the graph, and a bound on the number of nodes are colored by certain color.

Total color upper bound

We need to prove a upper bound on the number of colors used by our algorithm.

Total color upper bound

To prove this if we could prove how many nodes are colored by each of the colors (or their upper bound) then we can count the total number of colors used by the algorithm.

Theoretical Bounds

First we analyze the number of nodes colored with color 1.

Number of Nodes with Color 1

$$\frac{\text{Total Color 1 Nodes}}{\text{Total Nodes}} = \frac{3^x + 3^{x-2} + 3^{x-4} + \ldots 3^1}{\sum_{i=0}^{i=x} 3^i}$$

If number of layers is odd (x is odd) then the upper part of the fraction stops at 1 *and* 0 *otherwise.*

Number of Nodes with Color 1

$$
\text{Ratio of Color 1 node to total nodes} = \frac{3^x + 3^{x-2} + 3^{x-4} + \dots 3^1}{\sum_{i=0}^{i=x} 3^i}
$$

$$
= \frac{3 \cdot \frac{9^{\frac{x}{2}} - 1}{9-1}}{\frac{3^x - 1}{2}}
$$

$$
= \frac{3}{4} \cdot \frac{3^x - 1}{3^x - 1}
$$

$$
= \frac{3}{4}
$$

Number of Nodes with Color 1

Fact 2.1: Total Nodes with color 1

For complete trees we can color at most $\frac{3}{4}$ many nodes with color 1.

Number of Nodes with Color 2

To count how many nodes are colored with color 2 we'll approach this problem inductively.

Number of nodes with color 2

- First we find out for small size trees (for example level 3 trees) how many nodes are colored with color 2.

Number of nodes with color 2

- First we find out for small size trees (for example level 3 trees) how many nodes are colored with color 2.
- Then we extrapolate this to bigger trees. This analysis holds because bigger complete tree has these smaller complete threes as their children.

Number of nodes with color 2

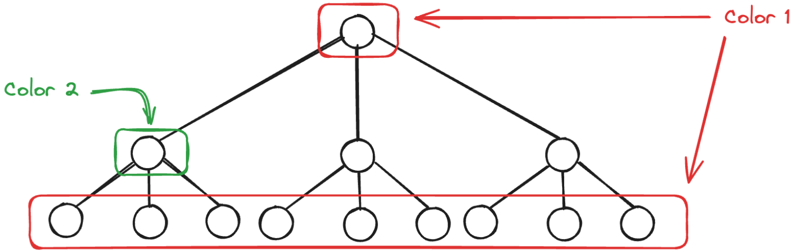

Figure: For a three layer deep tree only one node can be colored with Color 2

Number of nodes with color 2

So for a four layer tree, the total number of nodes colored with color 2 is $3 \times 1 = 3$

Number of nodes with color 2

- Layer 4 tree is a combination of 3 layer 3 trees connected with one extra node.

Number of nodes with color 2

- Layer 4 tree is a combination of 3 layer 3 trees connected with one extra node.
- This extra node can not be colored with color 2. Hence total color 2 needed is three times the total color 2 in the 3 layered tree.

Number of nodes with color 2

- Same thing is true for layer 5 tree also because the extra node (top most node) here will be colored with color 1 as odd layers are colored with color 1.

Number of nodes with color 2

- Same thing is true for layer 5 tree also because the extra node (top most node) here will be colored with color 1 as odd layers are colored with color 1.
- At layer 6 the top most node can be colored with color 2. Hence for layer six the number of node is one more of 3 times the number of nodes colored in layer 5 tree.

Number of nodes with color 2

- Same thing is true for layer 5 tree also because the extra node (top most node) here will be colored with color 1 as odd layers are colored with color 1.
- At layer 6 the top most node can be colored with color 2. Hence for layer six the number of node is one more of 3 times the number of nodes colored in layer 5 tree.
- We see a deviation from this pattern in layer 7 tree.

Layer 6 Trees

- In layer 6 tree we have 3 layer 5 trees connected with one extra node. The top most node is colored with color 2

Layer 6 Trees

- In layer 6 tree we have 3 layer 5 trees connected with one extra node. The top most node is colored with color 2
- Each of the layer 6 tree has 28 nodes colored with color 2.

Layer 6 Trees

- In layer 6 tree we have 3 layer 5 trees connected with one extra node. The top most node is colored with color 2
- Each of the layer 6 tree has 28 nodes colored with color 2.
- However we can not combine these three layer 6 trees with one extra node. Because the top most node in the layer 6 colored with color 2 will violate the distance conditions in the layer 7 tree.

Layer 6 Trees

- In layer 6 tree we have 3 layer 5 trees connected with one extra node. The top most node is colored with color 2
- Each of the layer 6 tree has 28 nodes colored with color 2.
- However we can not combine these three layer 6 trees with one extra node. Because the top most node in the layer 6 colored with color 2 will violate the distance conditions in the layer 7 tree.
- So we can only have one of the children (layer 6) with color 2 and rest are not colored with color 2.

Layer 6 Trees

- In layer 6 tree we have 3 layer 5 trees connected with one extra node. The top most node is colored with color 2
- Each of the layer 6 tree has 28 nodes colored with color 2.
- However we can not combine these three layer 6 trees with one extra node. Because the top most node in the layer 6 colored with color 2 will violate the distance conditions in the layer 7 tree.
- So we can only have one of the children (layer 6) with color 2 and rest are not colored with color 2.
- Hence we need to subtract 2 from three times the total number of nodes colored with color 2 in layer 6 tree to count color 2 nodes in layer 7 trees.

Generalization

Extending this concept to larger and larger trees we can see a pattern.

Generalization

Extending this concept to larger and larger trees we can see a pattern.

$$A = [0, 0, 1, -2]$$
$$f_2(x) = A[x\%4] + 3 \cdot f_2(x-1)$$

Generalization

Extending this concept to larger and larger trees we can see a pattern.

$$A = [0, 0, 1, -2]$$
$$f_2(x) = A[x\%4] + 3 \cdot f_2(x-1)$$

We use $f_2(3) = 1$ as the base-case.

Approximation of number of nodes colored with color 2

Lemma 2.1: Color 2 node count

Simple greedy heuristic for complete tree packing coloring, colors roughly $\frac{1}{10}^{\text{th}}$ many nodes with color 2 as compared to color 1.

Proof

Hypothesis 2.1

Suppose $f_2(x-1) \approx \frac{1}{10} \cdot \frac{3}{4} \cdot \sum_{i=0}^{x-2} 3^i$ is the number of color 2 used by our algorithm for a $x-1$ layer deep tree.

We need to prove $f_2(x) \approx \frac{1}{10} \cdot \frac{3}{4} \cdot \sum_{i=0}^{x-1} 3^i$ and verify this with experimental results.

Proof Continued

Using induction hypothesis and previous results we get,

$$\begin{aligned} f_2(x) &= A[x\%4] + 3 \cdot f_2(x-1) \\ &= A[x\%4] + 3 \cdot \frac{1}{10} \cdot \frac{3}{4} \cdot \sum_{i=0}^{x-2} 3^i \\ &= A[x\%4] + \frac{3}{4} \cdot \frac{1}{10} \cdot \sum_{i=0}^{x-1} 3^i \\ &\approx \frac{1}{10} f_1(x) \end{aligned}$$

Here $A[i] \in \{-2, 0, 1\}$, hence $f_2(x)$ is roughly $\frac{1}{10} \cdot f_1(x)$ where $f_1(x)$ is the number of color 1 used.

Results for other colors

We can also use these similar arguments for other higher colors, with that we can say the following facts

Results for other colors

We can also use these similar arguments for other higher colors, with that we can say the following facts

- For three-ary complete trees number of nodes colored with color $(2,3),(4,5),(6,7),\ldots$ are the same pairwise,

Results for other colors

We can also use these similar arguments for other higher colors, with that we can say the following facts

- For three-ary complete trees number of nodes colored with color $(2,3),(4,5),(6,7),\ldots$ are the same pairwise,
- For three-ary complete trees number of nodes colored with color $(4,5)$, is one-third of the number of nodes colored with $(2,3)$ and so on,

Results for other colors

We can also use these similar arguments for other higher colors, with that we can say the following facts

- For three-ary complete trees number of nodes colored with color $(2,3),(4,5),(6,7),\ldots$ are the same pairwise,
- For three-ary complete trees number of nodes colored with color $(4,5)$, is one-third of the number of nodes colored with $(2,3)$ and so on,
- After a while when some pair of colors $(x, x+1)$ are used ≤ 3 times, all the colors from $x+2$ and so on are used only once.

Total Color upper bound

Now we prove the upper bound on the number of colors used by our algorithm.

Total color upper bound

Theorem 2.1: $n/40$ scheme

Simple greedy heuristic is a $\frac{n}{40}$ approximation algorithm for complete three-ary trees.

Proof

Say we are coloring an x layer deep complete three ary tree with total number of node $= n$ and

$$n = \sum_{i=0}^{x-1} 3^i$$

Proof contd.

- Number of nodes colored with color $1 = \frac{3n}{4}$
- Number of nodes colored with color $2, 3 = \frac{3n}{40}$
- Number of nodes colored with color $4, 5 = \frac{n}{40}$
- Number of nodes colored with color $6, 7 = \frac{n}{120}$
- Number of nodes colored with color $8, 9 = \frac{n}{360}$
- ...

Proof Contd.

We say $j = 1$ at color $2, 3$, from there on j stops at $j = j$ when $\frac{n}{n} = 1$. From there on rest of all the nodes are colored with an non-reusable color.

Proof Contd.

Total number of different colors used $= 1 + 2 \cdot j +$ number of non-reusable colors. We now need to calculate the number of non-reusable colors and the value of j. Value of j when the denominator becomes n is when re-usable colors are finished.

Proof Contd.

$$\text{Total non-reusable color} = n - \left[\frac{3n}{4} + 2 \cdot \left(\frac{3n}{40} + \frac{n}{40} + \frac{n}{40*3} + \frac{n}{40*3^2} \cdots + 1\right)\right] \tag{1}$$

First we calculate $\left(\frac{3n}{40} + \frac{n}{40} + \frac{n}{40*3} + \frac{n}{40*3^2} \cdots + 1\right)$.

$$\begin{aligned} s_n &= \left(\frac{3n}{40} + \frac{n}{40} + \frac{n}{40*3} + \frac{n}{40*3^2} \cdots + 1\right) \\ &= \frac{3n}{40}\left[\frac{1-\left(\frac{1}{3}\right)^{2+\log_3\left(\frac{n}{40}\right)}}{1-\frac{1}{3}}\right] \\ &= \frac{9n}{80} - \frac{1}{2} \end{aligned}$$

Proof Contd.

We put this into the equation (1), then we get

$$
\begin{aligned}
&= n - \left[\frac{3n}{4} + 2 \cdot \left(\frac{9n}{80} - \frac{1}{2}\right)\right] \\
&= \frac{n}{40} + 1
\end{aligned}
$$

Proof Contd.

We put this calculation into the original equation to get the total number of colors used as

$$\text{Total colors used} = 1 + 2 \cdot \left[\log_3\left(\frac{n}{40} + 2\right)\right] + \frac{n}{40} + 1 \tag{2}$$

$$\geq \frac{n}{40} \tag{3}$$

Proof contd.

For very large n expression (2) evaluates to almost $\frac{n}{40}$. This proves our lemma saying, our simple greedy algorithm outputs a valid packing coloring assignment with at-least $\frac{n}{40}$ many colors.

Experimental Results

Here I'll show how the experimental results support our theoritical analysis.

Experimental Results

# of Nodes	# of Layers	X-ary tree	Total Colors	Time
13	3	3	4	0ms
40	4	3	7	0ms
121	5	3	11	2ms
364	6	3	19	6.11029 ms
1093	7	3	40	38ms
3280	8	3	98	63ms
9841	9	3	269	1s 101ms
29524	10	3	781	3s 845ms
88573	11	3	2309	45s 256ms
265720	12	3	6890	4m 17s 31ms
7174453	15	3	185525	9 days 7 hours 29 min 9 sec

Table: Runtime, total color used for a complete three-ary tree.

Total Colors Used

All the experimental results show that the total number of colors used is less than $\frac{n}{40}$ where n is the number of nodes in the tree.

Total Colors Used

All the experimental results show that the total number of colors used is less than $\frac{n}{40}$ where n is the number of nodes in the tree.
This is in line with our theoritical analysis.

Total Colors Used

Color number	Number of nodes
1	5380840
2, 3	538084
4	177391
5	177390
6, 7	59058
8, 9	19684
10, 11	6561

Table: Number of nodes colored with each color for a 15-layer complete tree.

Total Colors Used

Color number	Number of nodes
12, 13	2187
14, 15	729
16, 17	243
18, 19	81
20, 21	27
22, 23	9
24, 25	3
$26 \rightarrow 185525$	1

Table: Number of nodes colored with each color for a 15-layer complete tree.

Experimental Results

These experimental results prove the following things

Experimental Results

These experimental results prove the following things

- Color 2 is used $\frac{1}{10}$th of the total number of nodes colored with color 1.

Experimental Results

These experimental results prove the following things

- Color 2 is used $\frac{1}{10}$th of the total number of nodes colored with color 1.
- Color 1 is used for the $\frac{3}{4}^{\text{th}}$ of the total number of nodes.

Experimental Results

These experimental results prove the following things

- Color 2 is used $\frac{1}{10}$th of the total number of nodes colored with color 1.
- Color 1 is used for the $\frac{3}{4}^{\text{th}}$ of the total number of nodes.
- Pair-wise colors are used for the same amount of nodes in the tree.

Experimental Results

These experimental results prove the following things

- Color 2 is used $\frac{1}{10}$th of the total number of nodes colored with color 1.
- Color 1 is used for the $\frac{3}{4}^{\text{th}}$ of the total number of nodes.
- Pair-wise colors are used for the same amount of nodes in the tree.
- After 26 which is some less than $2 * d = 2 * 15 = 30$, all the colors are used for only once.

Conclusion

This completes our presentation on the approximation algorithm for packing coloring on trees.

Conclusion

This completes our presentation on the approximation algorithm for packing coloring on trees.

In future we'll look for the following things:

Conclusion

This completes our presentation on the approximation algorithm for packing coloring on trees.

In future we'll look for the following things:

- We analysed our algorithm performance on a complete three-ary tree and some trees with randomly delete branch. We need to analyse the performance for any d-degree bounded tree and graphs. To do this one approach we thought of is to design an algorithm that'll find a suitable root such that most of the nodes are equidistant from this root. Suitable root must decrease the number of colors to be used by our algorithm.

Conclusion

This completes our presentation on the approximation algorithm for packing coloring on trees.

In future we'll look for the following things:

- We analysed our algorithm performance on a complete three-ary tree and some trees with randomly delete branch. We need to analyse the performance for any d-degree bounded tree and graphs. To do this one approach we thought of is to design an algorithm that'll find a suitable root such that most of the nodes are equidistant from this root. Suitable root must decrease the number of colors to be used by our algorithm.
- To test the effectiveness of the algorithm we need to come up with a random-graph generation scheme. Through which we can generate graphs at random with certain properties and review our algorithm performance.

Conclusion

This completes our presentation on the approximation algorithm for packing coloring on trees.

In future we'll look for the following things:

- We analysed our algorithm performance on a complete three-ary tree and some trees with randomly delete branch. We need to analyse the performance for any d-degree bounded tree and graphs. To do this one approach we thought of is to design an algorithm that'll find a suitable root such that most of the nodes are equidistant from this root. Suitable root must decrease the number of colors to be used by our algorithm.
- To test the effectiveness of the algorithm we need to come up with a random-graph generation scheme. Through which we can generate graphs at random with certain properties and review our algorithm performance.

Future

- We need to come up with a randomized graph generation scheme that'll generate worst case graphs to color for our algorithm. This will generate the worst case graphs, that'll cost very significant amount of colors to color according to our coloring strategy and some fix for those graphs. We also need to check the performance of our algorithm on randomly chosen graph from a fixed distribution.

Thank You

Thank You