Round-Based Distributed Graph Coloring

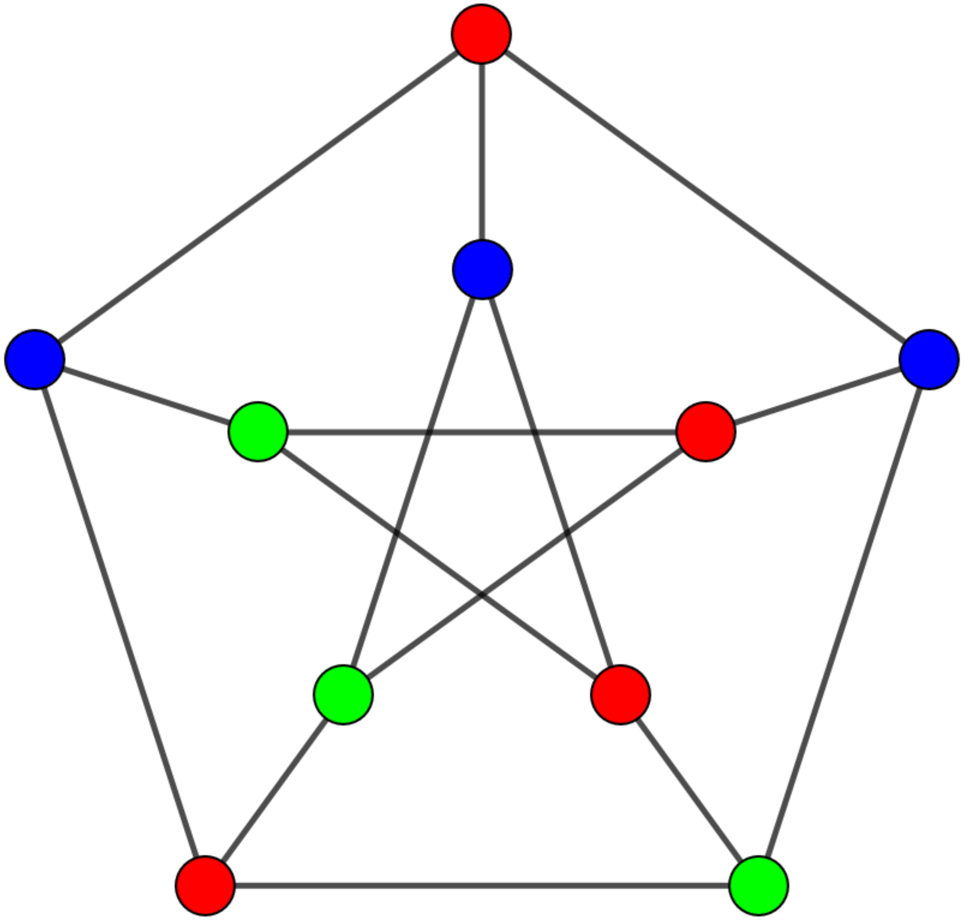
David Worley
School of Electrical Engineering and Computer Science
University of Ottawa, Ottawa, Canada
dworl020@uottawa.ca

What is Graph Coloring?

Graph Coloring is the process of taking a graph and applying a color to each vertex such that no two neighbouring vertices share a color.

The focus of this problem is finding the smallest number of colors possible for certain graph classes, or for finding good, but not optimal, colorings in fast runtime.

An Example Coloring



The image above shows a proper 3-colouring of a 10-vertex, cubic graph.

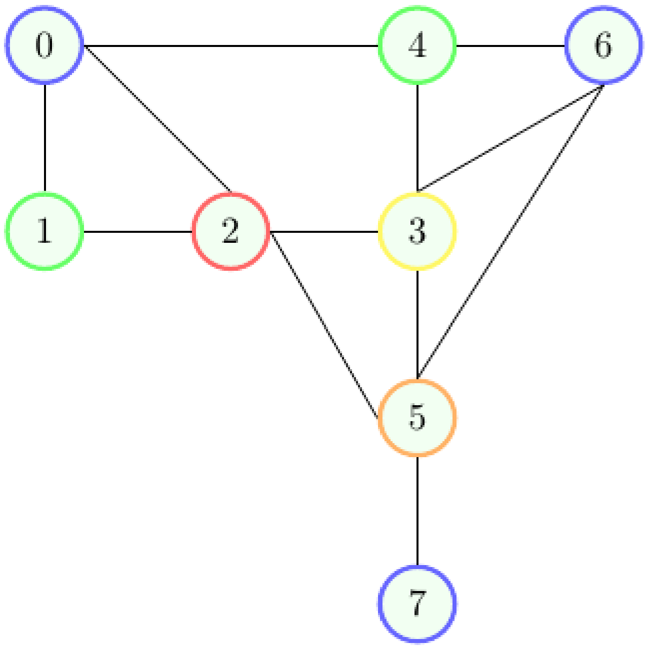
Distributed Graph Coloring

Distributed graph coloring is the process of parallelizing graph coloring algorithms to find proper colorings using as few colors as possible, as fast as possible.

A coloring with a larger *k* can also be useful if we have ways to reduce the number of colors using a separate distributed algorithm that runs as fast or faster

A Lower Bound for Graph Coloring

Let Δ be the maximum degree of the graph, then a Δ +1 coloring can be generated for any graph using a simple greedy sequential coloring algorithm.



A greedy Δ+1 coloring on a graph with 8 vertices and Δ=4

This gives a baseline for a good coloring size on our distributed setting, as we know it is always obtainable.

An Upper Bound by Linial

In 1992, such a bound was established by Linial with an algorithm that generates an $O(\Delta^2)$ coloring in $O(\log^* n)$ time, where $\log^* n$ is the iterated log function.

This gives a suitable upper bound as any graph coloring algorithm running as fast, or slower, than Linial's algorithm can use Linial's algorithm to obtain an $O(\Delta^2)$ coloring.

Algorithm Optimality and Runtime

Linial also proved in his paper that any graph coloring algorithm must use at least $\Omega(\log^{*} n)$ time to color even the simplest graphs.

Since many graph coloring algorithms are round-based, the complexity is expressed with respect to the number of rounds as opposed to runtime with respect to the number of vertices.

Color Reduction

This changes research focus from finding a coloring algorithm to finding a *color reduction* algorithm, that takes in a graph with an input coloring and outputs a graph with a smaller coloring within a certain amount of rounds

In fact, almost all round-based graph coloring algorithms can be considered color reduction algorithms, that take a |V|-coloring as input.

Maus's Paper

Maus introduces a new general coloring algorithm that generalizes many current state of the art algorithms while simplifying the ideas that they use

This generalization is done using multiple parameters, given a graph with an input m-coloring and maximum degree Δ, it uses $R=O(\Delta/k)$ rounds to compute an $O(k\Delta)$ coloring.

Maus's Algorithm

Algorithm 1: for vertex with color i. Parameters d, k, m, Δ.

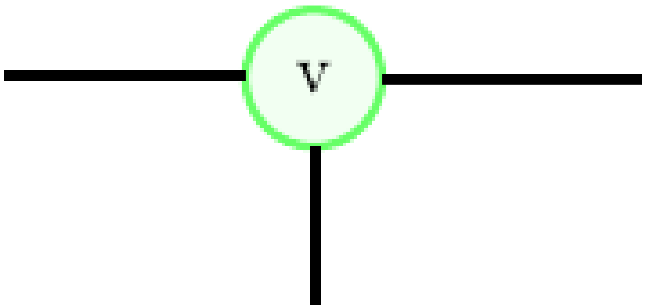
Locally compute:
polynomial $p_i : \mathbb{F}_q \rightarrow \mathbb{F}_q$ with q chosen by (1)
sequence s_i: $\left(x \mod k, p_i(x) \mod q\right), x = 0, \dots, q-1$
Process s_i in disjoint batches B_j of size k, for $j = 1, \dots, \lceil \frac{q}{k} \rceil$
Try the colors in batch B_j (in a single round)
if $\exists \left(d\text{-}proper\ c \in B_j\right)$ **then** adopt c, join P_j, and **return**;

The algorithm generates the above polynomials from a prime field of size q, where q is the smallest prime > $2\Delta \log_{\Delta} m$

The algorithm is also adaptable enough to calculate a d-defective coloring as well, generalizing many relevant results in distributed graph coloring.

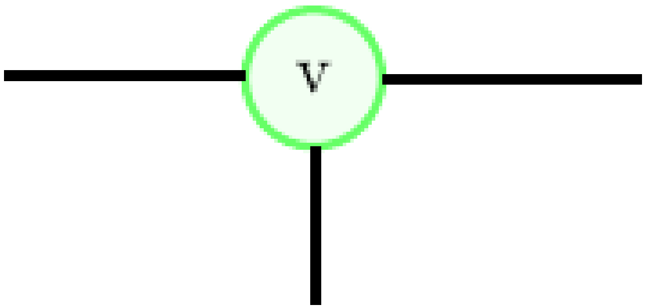
Coloring a Vertex With Maus’s Algorithm

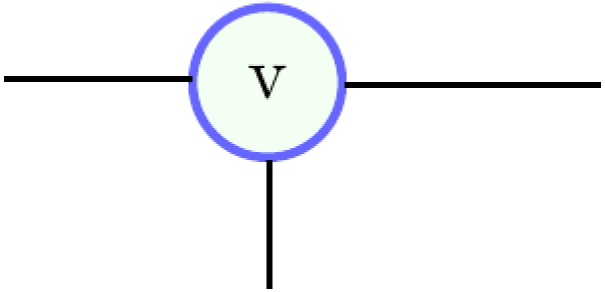
v has input color m so we sample polynomial p_m for *k* values



So take $p_m = a_1x_1 + a_1x_1 + \ldots + a_fx_f$ where $f = \log_{\Delta} m$

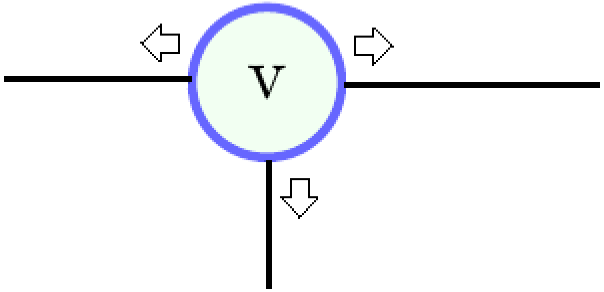
and $(a_0, a_1, \ldots, a_f)$ corresponds to the m^{th} polynomial





$Seq_v = (p_m(0)\%q,\ p_m(1)\%q,\ \ldots,\ p_m(k-1)\%q)$

Temporarily color *v* according to seq_v and test if the neighbour has the same color



If there are conflicts, move on to next color. If there are no conflicts, color this node Permanently so future rounds don’t check it

Algorithm Implementation

The algorithm was implemented in C++ using MPI and involved generating the polynomials for each input color, then running R rounds, with each round picking a color from its sequence, adding it to a disjoint batch, and then testing the batch in 1 communication round.

This process is repeated for the required number of rounds, and the unique properties of polynomials from a prime field guarantee that all vertices will be colored upon completion.

Data and Parameters

A small Python program to generate random graphs was used to generate input colorings for the implementation. These input colorings would randomly generate an edge set, calculate Δ, and generate a coloring, then output the contents for testing.

The coloring initially chosen was to use each vertex's ID as its color for an input $|V|$-coloring and k was chosen such that $1 <= k <= 4\Delta$

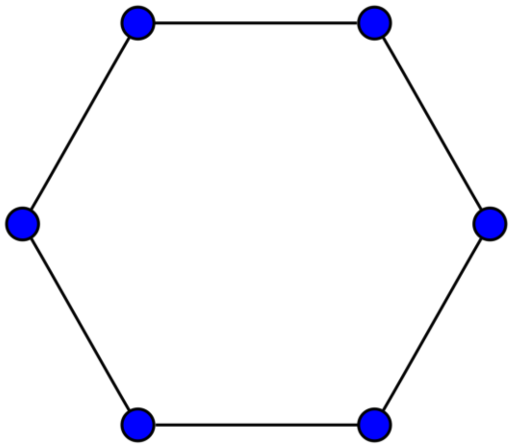
Experimental Results

Number of Vertices	Max Degree (Δ)	Number of Rounds	Colors in Input coloring	Colors in Output Coloring	Time Elapsed (s)
10	6	4	10	8	0.115
100	13	10	100	26	0.692
250	10	16	250	55	1.553
500	12	19	500	76	1.859
1000	13	20	1000	91	53.562*

*The long runtime of the 1000 vertex graph is likely due to the implementation of the graph as an adjacency matrix as opposed to an adjacency list

Questions For Audience

1. Why is research focusing on color reduction algorithms instead of coloring algorithms?
2. Why do we have a lower bound of $\Delta+1$ for colorings instead of something smaller/bigger?
3. What is the time complexity of the fastest algorithm to color a n-cycle? (6-cycle pictured right)



Thanks!

Any Questions?