Round-Based Distributed Graph Coloring

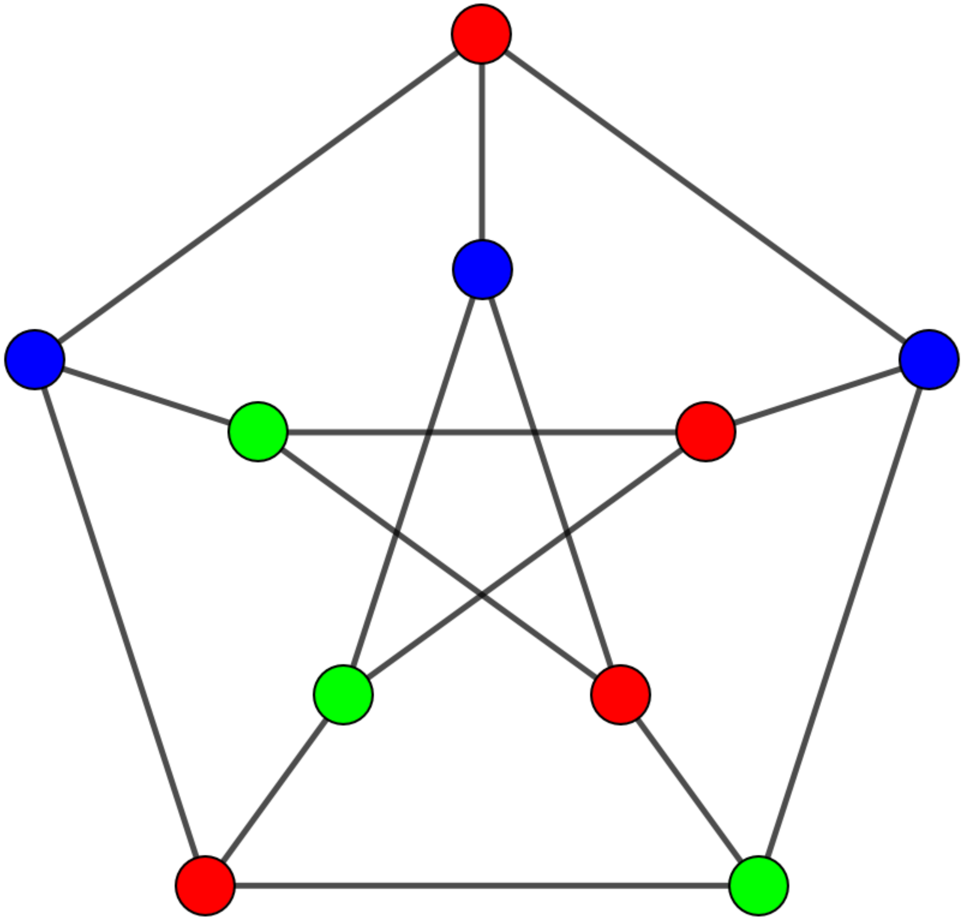
David Worley
School of Electrical Engineering and Computer Science
University of Ottawa, Ottawa, Canada
dworl020@uottawa.ca

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-coloring of a 10-vertex, cubic graph.

Distributed Graph Coloring

Distributed graph coloring is the process of parallelizing graph coloring algorithms to fins 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 used using a separate distribute algorithm that runs as fast or faster

A Lower Bound for Graph Coloring

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

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 algorithm that solves both of these problems, as well as improving state of the art results on (2,r)-ruling sets.

The algorithm is a generalization of many current methods, simplifying their results while achieving the same performance

Maus's Algorithm

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

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

Locally compute:
- polynomial $p_i : \mathbb{F}_q \to \mathbb{F}_q$ with q chosen by (1)
- sequence s_i: $(x \mod k, p_i(x) \mod q)$, $x = 0, \ldots, q-1$

Process s_i in disjoint batches B_j of size k, for $j = 1, \ldots, \lceil \frac{q}{k} \rceil$
- *Try* the colors in batch B_j (in a single round)
- **if** $\exists$ (d-*proper* $c \in B_j$) **then** adopt c, join P_j, and **return**;

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

Colouring a Vertex Within a Round

The algorithm works by having each vertex generate a sequence of colors to try based off its input colors. The colors are then tested in disjoint batches of size *k*

If the sequence are chosen properly (Maus's chosen method was to sample polynomials from a prime field), then it can be shown that each node will be properly colored after R rounds.

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 leaves an algorithm that takes R communication rounds to color all vertices, due to results on polynomial intersections in prime fields.

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 $1 <= k <= 4\Delta$

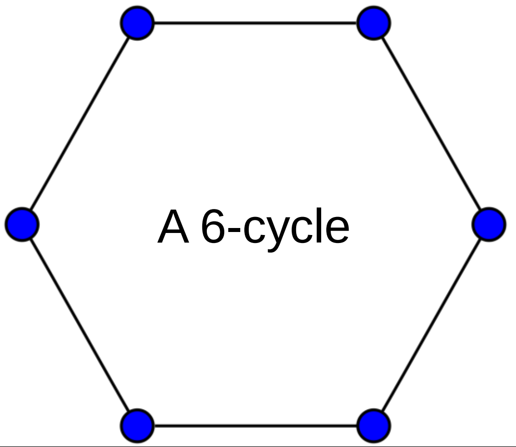
Experimental Results

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 Δ+1 for colorings instead of something smaller/bigger?

3. Is it possible to color an n-cycle in constant time?



Thanks!

Any Questions?