CSCI 446 Artificial Intelligence Project 1 Final Report

Roy Smart Nevin Leh Brian Marsh

September 24, 2016

1 Introduction

The *Graph Coloring Problem* (GCP) is the problem of attempting to color a set of interconnected vertices, using a limited set of colors, such that no vertex has the same color as its neighbors. This problem is best visualized as the map coloring problem. As an example of the map coloring problem, consider the problem of assigning one of four colors to every country in the world, such that no two adjacent countries have the same color, as in Figure 1. It can be shown that the map coloring problem reduces to the graph coloring problem[2]

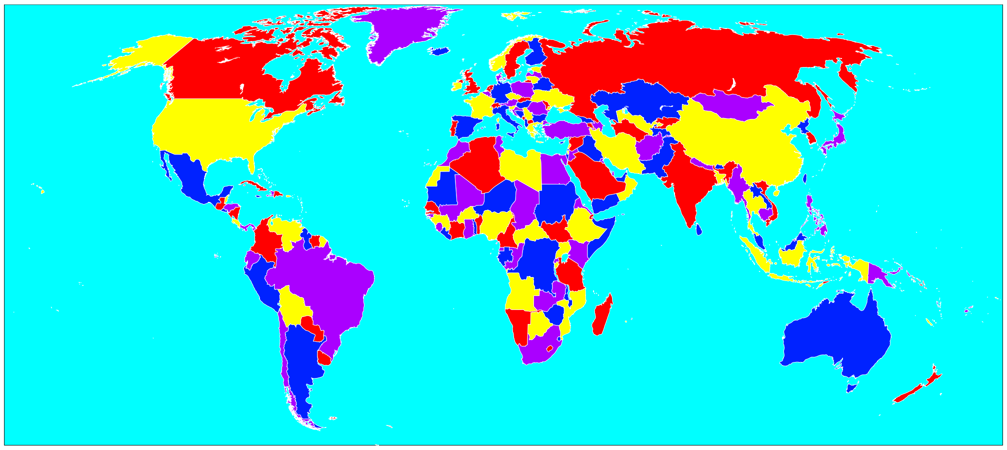


Figure 1: *Map of the world satisfying the map coloring problem [1].*

if we represent the countires as the vertices of the graph, and the borders between countries as the edges of the graph, shown in Figure 2. This configuration produces a *planar* graph, a graph with no edge intersections. For this project, we are asked to solve the GCP problem for planar graphs only. Since the problems are

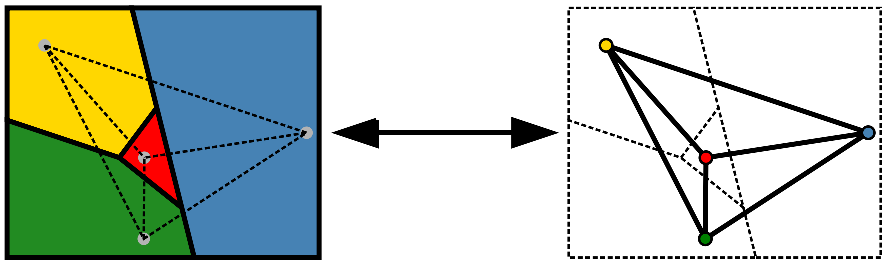


Figure 2: *Diagram describing how the map coloring problem can be transformed into the graph coloring problem [3].*

guaranteed to be planar, we will represent our graphs using polygon maps generated by our program.

To solve the GCP we employed five different algorithms: Minimum Conflicts, Simple Backtracking, Backtracking with Forward Checking, Backtracking with Constraint Propagation (MAC), and Local Search using a Genetic Algorithm. To evaluate these algorithms, we built a problem generating program that can produce a random set of planar graphs. Using the problem generator, we calculated a set of graphs between the sizes {10, 20, 30,...,100} and called the five graph coloring algorithms to solve the GCP. We measured GCP algorithm performance by measuring the number of read/write operations, the time required to find a solution, and the number of function calls for each algorithm. Using these metrics, we predict that the Minimum Conflicts algorithm will be the fastest, based off its performance on the eight-queens problem[2].

The code for this project is implemented in `C++` and depends on the programs `cairo`, `gnuplot`, `ffmpeg`, and LaTeX for graph output, performance output, video output, and documentation respectively.

2 Algorithm Implementation

2.1 Problem Generation

We created a function that could produce many examples of planar graphs from a set of randomly scattered points. For an arbitrary set of points, there is no unique planar graph that can be constructed. To solve this issue, the problem statement has provided a algorithm for calculating an planar graph.

> Select some point X at random and connect X by a straight line to the nearest point Y such that X is not already connected to Y and line crosses no other line. Repeat the previous step until no more connections are possible.

For our implementation of the above algorithm, we started by creating a complete graph (where each vertex is connected to every other vertex). Each vertex was associated with a list of edges, sorted by length. We then selected a point at random and inspected the first unchecked edge E. If E did not cross any of the accepted edges, it is added to the list of accepted edges and marked as checked. This process was repeated until every edge was marked as checked.

The most challenging part of the implementation described above was determining if two line segments intersected. An algorithm to determine if two edges (or line segments) cross has been outlined by LaMothe [4] and described in detail here. Let's start by defining two line segments

$$\begin{aligned} A &= \{\mathbf{a},\ \mathbf{a'}\} \\ B &= \{\mathbf{b},\ \mathbf{b'}\} \end{aligned} \tag{1}$$

where $\mathbf{a}$, $\mathbf{a'}$, $\mathbf{b}$, and $\mathbf{b'}$ are vectors from the origin to the ends of the line segments. Next, compute the direction vectors for each line segment

$$\begin{aligned} \boldsymbol{\alpha} &= \mathbf{a}' - \mathbf{a} \\ \boldsymbol{\beta} &= \mathbf{b}' - \mathbf{b} \end{aligned} \tag{2}$$

Now, the trick to solving this problem is to parameterize the line segment using the direction vector and a parameter t_i in the domain $[0,\ 1]$.

$$\begin{aligned} \mathbf{p} &= \mathbf{a} + \boldsymbol{\alpha} t_a, \quad t_a \in [0,\ 1] \\ \mathbf{q} &= \mathbf{b} + \boldsymbol{\beta} t_b, \quad t_b \in [0,\ 1] \end{aligned} \tag{3}$$

From here, the solution is obvious. To find the point of intersection, we set $\mathbf{p} = \mathbf{q}$ and solve for the values of t_a and t_b. Then, if the solution is outside the domain of t_i, the line segments do not intersect. Thus,

$$\Rightarrow \begin{cases} p_x = q_x \\ p_y = q_y \end{cases} \tag{4}$$

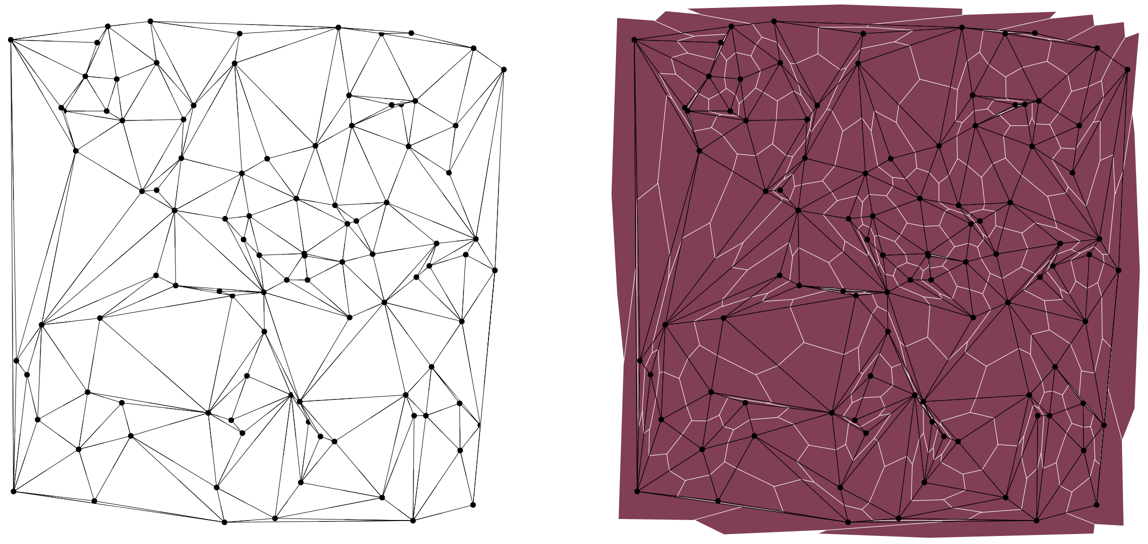
$$\Rightarrow \begin{cases} a_x + \alpha_x t_a = b_x + \beta_x t_b \\ a_y + \alpha_y t_a = b_y + \beta_y t_b. \end{cases} \tag{5}$$

Equation 5 is a system of two equations and two unknowns, solving for t_a and t_b gives

$$\begin{aligned} t_a &= \frac{(-a_y + b_y)\beta_x + (a_x - b_x)\beta_y}{\alpha_y\beta_x - \alpha_x\beta_y} \\ t_b &= \frac{(a_y - b_y)\alpha_x + (-a_x + b_x)\alpha_y}{-\alpha_y\beta_x + \alpha_x\beta_y}. \end{aligned} \tag{6}$$

Finally, we can determine whether A and B intersect if $0 < t_a < 1$ and $0 < t_b < 1$ evaluates to true. This method may be more inefficient than other algorithms, such as the example outlined by Cormen[5] due to the division operation in Equation 6.

Using the expression above to determine line intersections, we were able to implement a problem generator that was reasonably efficient, generating approximately 1×10^3 examples per second. This efficiency was important, because it made testing and debugging the GCP algorithms much faster. Example output from the problem generator is shown in Figure 3a.



(a) *Planar graph with 100 vertices.* **(b)** *Polygon map corresponding to (a).*

Figure 3: *Visualization of the output of the problem generator. First a planar graph was created (a), and then a corresponding polygon map was found (b).*

We have also implemented a way to produce a map of regions that is analogous to the graphs created by the problem generator, shown in Figure 3b. Since the graph coloring problem does not uniquely define the map coloring problem, we had to invent a metric that produces appropriate graphs. To define our metric, consider that if a polygon G_i corresponding to each point P_i were constructed out of the midpoints $M_{i,m}$ of the associated edges $E_{i,m}$ of P_i, then an adjacent polygon G_j would only touch at the corners, that is, our map would be full of gaps. Therefore, to fix the problem we also define the centroid $C_{i,n}$ of each triangle $T_{i,n}$ formed by the two adjacent edges $E_{i,m}$ and $E_{i,m+1}$. We then define the final polygon for P_i using the points $M_{i,m}$ and $C_{i,n}$. This procedure is unnecessary in terms of the assignment, but is helpful for debugging GCP algorithms, demonstrating example output, and for aesthetic pleasure.

2.2 Minimum Conflicts

2.3 Simple Backtracking

To implement simple backtracking we followed the recursive algorithm outlined in Russel and Norvig[2]. The function labeled INFERENCE() is called `has_conflicting_neighbors`

2.4 Backtracking with Forward Checking

2.5 Backtracking with Constraint Propagation

2.6 Genetic Algorithm

3 Experimental Approach

3.1 Minimum Conflicts

3.2 Simple Backtracking

3.3 Backtracking with Forward Checking

3.4 Backtracking with Constraint Propagation

3.5 Genetic Algorithm

3.6 Comparisons

4 Results

4.1 Minimum Conflicts

4.2 Simple Backtracking

4.3 Backtracking with Forward Checking

4.4 Backtracking with Constraint Propagation

4.5 Genetic Algorithm

4.6 Comparisons

5 Summary

References

[1] Wikipedia, the free encyclopedia. Four color theorem, 2016. `https://upload.wikimedia.org/wikipedia/commons/4/4a/World_map_with_four_colours.svg`.

[2] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, Upper Saddle River, New Jersey 07458, 3rd edition, 2010.

[3] Wikipedia, the free encyclopedia. Four color theorem, 2016. `https://en.wikipedia.org/wiki/Four_color_theorem#/media/File:Four_Colour_Planar_Graph.svg`.

[4] André LaMothe. *Tricks of the 3D Game Programming Gurus.* Sams Publishing, 201 West 103rd Street, Indianapolis, Indiana 46290, 2003.

[5] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms.* Massachusetts Institute of Technology, Cambridge, Massachusetts, 3rd edition, 2009.