CSCI 446 Artificial Intelligence Project 1 Design Report

Roy Smart Nevin Leh Brian Marsh

September 14, 2016

1 Introduction

The *Graph Coloring Problem* (GCP) is the problem of attempting to color a set of bordering regions such that no region has the same color as its neighbors using three or four colors. For example consider the problem of coloring a map of the USA (Figure 1), using only four colors and ensuring that no neighboring states share the same color. This is the motivation of the graph coloring problem.

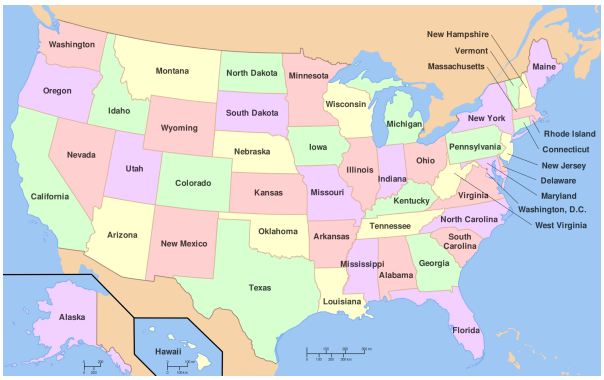

Figure 1: *Map of the United States of America satisfying the graph coloring problem.*

It can be shown that the map coloring problem reduces to the graph coloring problem if we represent the states as the vertices of the graph, and the borders between states as the edges of the graph. This configuration produces a *planar* graph , a graph with no edge intersections. For this project, we will concentrate on solving the graph coloring problem for *maximally planar* graphs (Figure 2), planar graphs for which adding any addition edge would make the graph non-planar.

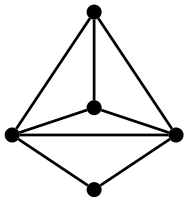

Figure 2: *Simple example of a maximally planar graph.*

We are tasked with solving this problem five different ways: Minimum Conflicts, Simple Backtracking, Backtracking with Forward Checking, Backtracking with Constraint Propagation (MAC), and Local Search using a Genetic Algorithm. To test these algorithms, we will first need to build a problem generating program that can produce a random set of maximally planar graphs. Using the problem generator, we will calculate a set of graphs between the sizes {10, 20, 30,...,100} and then use the five graph coloring algorithms to solve the GCP. We will measure GCP algorithm performance by how many vertex colorings it requires to find a solution.

The solution to this project will be implemented in Python 3.4 using the graphics library to visualize the graphs.

2 Problem Generation

For this project, we will need to create maximally planar graphs (MPGs) from a set of randomly scattered points. For an arbitrary set of points, there is no unique MPG that can be constructed. To solve this issue, the problem statement has provided a prescription for calculating an MPG.

> 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.

To implement this algorithm we will first create a complete graph (graph where each point is connected

to every other). Each vertex will have a list of associated edges, sorted by length. As instructed, we will then select a point at random and inspect the first unchecked edge E. If any edge crosses E it is eliminated from the graph. We then repeat this process until every edge has been checked.

An algorithm to determine if two edges (or line segments) cross has been outlined by LaMothe [1] and described in detail here. Let's start by defining two line segments

$$A = \{\mathbf{a},\ \mathbf{a'}\}$$
$$B = \{\mathbf{b},\ \mathbf{b'}\}$$

where $\mathbf{a}$, $\mathbf{a'}$, $\mathbf{b}$, and $\mathbf{b'}$ are points on the ends of the line segments.

3 Experiment Design

4 Solution Overview

5 Graph Coloring Algorithms

5.1 Minimum Conflicts

5.2 Simple Backtracking

5.3 Backtracking with Forward Checking

5.4 Backtracking with Constraint Propagation

5.5 Local Search using a Genetic Algorithm

References

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