Jonathan Pritchett
CSCI-6511: Artificial Intelligence
3/11/2021

Project 2 - CSP Graph Coloring

Introduction

This project implements an algorithm for solving Map Coloring Constraint Satisfaction Problems (CSPs).

Implementation

Algorithm

To solve the CSP, a set ordering of the vertices is first established. A color is then applied to a vertex based on this order. When a vertex is colored, all of its neighbors remove that color from their list of available colors. The algorithm then moves to the next vertex in the ordering. If that vertex has no available colors, the program backtracks to the previous vertex, restores the previous coloring state, and tries the next available color to the vertex. This process continues until either the last vertex in the order is successfully colored, or the first vertex run's out of colors to try, with the first case indicating a successful map coloring, and the last indicating a map that cannot be colored with that few a quantify of colors.

Data Structures

To model each Map Coloring CSP, this program implements a Node class. Each Node maintains the id of the Node, pointers to its neighbor vertices in the map, a numerical

representation of its current color (-1 to start), and a list of available colors for the vertex (initially all colors available for the graph). The colors used in the graph are represented by integers in the range [0,c) where c is the number of colors usable by the graph.

Heuristics

In order to reduce computational complexity, the vertices are placed in a fixed order during pre-processing. This order is determined by a heuristic called Most Constraining Vertex(MCV). The MCV is determined by counting the number of unordered vertices neighboring each vertex, which acts as an estimation of how constraining that vertex is on its neighbors at that point in the ordering.

Additionally, a form of the Least Constraining Value heuristic is used when coloring the vertices. The colors available for the current vertex will be tested in order of least to greatest, and all vertices will have the same ordering of colors (even if some vertices no longer have access to some colors by that point). Effectively, this means that the algorithm will try to use as few colors as possible to color the graph, only introducing a new color when no “lower” color is available.

Backtracking

By far, the most difficult aspect of implementing this algorithm was accounting for backtracking. If a color is found to be invalid for a vertex, not only must that vertex be un-colored, but all of its neighbors must have that color choice re-enabled. However, multiple colored vertices in the graph can impose the same color restriction on a neighboring vertex.

Thus backtracking must ensure that the restrictions for all of the currently colored vertices remain intact. To solve this, the program sets the current vertex's color to unassigned (-1) and resets the available color lists for all vertices while maintaining their assigned colors. Then the coloring for each colored vertex is reapplied in the established order, allowing for the available color lists of neighboring vertices to be updated back to the state before the "bad" coloring was applied. Unfortunately, this process is quite inefficient and makes solving very large graphs (1000+ vertices) impractical with this program. In retrospect, a better solution might have been to have each vertex maintain a queue or map indicating both what color is restricted at each step, and which neighbor vertex caused the restriction. This would allow the neighboring vertex to undo the restrictions its coloring caused while maintaining the restrictions enforced by other vertices.

Running the program

Before running the program, ensure that map_coloring.py is in the same directory as a folder named “data”. This folder should contain the map-coloring-CSP files to be evaluated. These files should be in the format specified in “CSCI6511_P2_CSP.pdf” and may have any filename (although standard filename best practices should be observed for best results). The program may be run by executing “map_coloring.py” with a python 3.x interpreter. The program will read in all text files in the “data” folder and attempt to find solutions to the CSPs. The program will report the results in the following format:

File: <filename>
Vertices: <number of vertices in the graph>
Colors: <list of the colors usable by the graph>
Solvable: <True/False>
Backtraces = <number of times a backtrace was necessary to find solution>
<if the map is colorable, one solution to the csp will be reported>
vertex = 1 : color = 1
...
vertex = i : color = Ci

Note: This program is not designed to handle very large maps. CSP’s of greater than 1000 vertices are not recommended.

Results

This program proved exceptionally good at solving small to medium-sized maps. In 3 of the 5 solvable test files provided (test1.txt, test2.txt, and test5.txt) the program was able to find a coloring without resorting to any backtracking at all. This suggests that the heuristics used by this program, Most Constraining Vertex and Least Restrictive Value, are effective strategies for solving map-coloring constraint satisfaction problems.

Links

Project Github Repo: https://github.com/ithari42/ai_sp2021/tree/master/project2a