Nicholas Gerold PID 730097363

To implement the backtracking search, I used a depth first, starting from the most constrained variable. This method used some filtering, keeping a list of the colors that were still possible to use, removing them when a problem occurred. The backtracking algorithm starts at the most constrained node and assigns it a color. From there, repeated recursive calls are made, giving all adjacent notes one of the colors still in their possible color lists. Once a color is picked for the original node, colors are picked for the children all the way down until all nodes have a color that satisfies the constraints. If a color does not work for one the adjacent nodes, the parent removes the color from the potential list and tries again, until the node has no more potential colors and returns false. If the most constrained node didn't work, the backtrack will try all the other nodes as starting points before deciding that there is no solution.

To implement the local search, I used a hill climbing method with random restarts. The search will start by randomizing the colors for all the states, then tries changing the colors of the states that don't work somewhat randomly, assigning the incorrect states one of the colors that is still in their possible color set. If the newly generated set of colors is better, it replaces the current one, otherwise the colors are randomized again. To prevent the algorithm from being stuck at a local max, all state colors are randomized after 75 attempts of changing invalid colors, potentially dodging issues with local maxima. The algorithm runs for 2 minutes, printing that a solution can't be find if one has not been found yet.

As I worked alone, all work was done by me.