Hongyu Li
Loi Pham
Philip Kim

Assignment 1: Constraint Satisfaction

Backtrack search:

For our algorithm, we used a combination of backtracking search with arc consistency. Using lists of states and domains of colors per state, we were able to track where we were going with the algorithm by seeing neighbors and arc consistency. If given the minimum number of colors needed to color a map, we were always guaranteed a solution.

Given a list of states, we start the backtrack search at the first state, alphabetically. From there, we start iterating through possible colors for our current state and seeing if it's neighbors have any conflicts. Then, we simulate what would happen if we gave that state said color by running it through our AC-3 algorithm, which will report 'False' if there are any empty domains (no possible color options for a state), or 'True', otherwise. The AC-3 algorithm keeps track of a list of domains filled with each possible color a state can assume. Possibilities will be eradicated based on inconsistencies (if a state and neighbor share an already assigned color). If AC-3 returns 'True' then we will proceed to color the state and continue to the next uncolored state. AC-3 algorithms end once the queue of arcs, created from an initial list of every state connection and appended arcs that lead to any changed states (states that just had a color option removed from their domain).

We used a couple of base cases to stop the entire backtracking algorithm: if each domain is left with one color and all states were marked as colored with a valid color, stop the program and report the results.

Local search:

For local search, we implemented a hill climbing algorithm. Every solution that the program produces will be different with different number of iterations and steps; state colored. Sometimes the solution cannot be found and the program will stop if a minute passes. First, in the Hill Climbing function, States are randomly colored to create a map with some solutions. The heuristic_cost function with the states as its argument is called to determine the initial total number of conflicts that the map has for Hill Climbing, saved in the current_conflicts variable, it also records the number of conflicts for each state. With the current_conflicts greater than zero, a

while loop iterates through the search code until the conflicts is zero and no conflict is present; the number of iterations are tracked.

The search code starts with finding the states with conflicts greater or equal to three, then calls the successor_function with state as argument to decrease the total conflicts and state conflict. The successor_function iterates through the state's neighbor and compares the colors. If the color of the state and neighbor are the same, the neighbor's color is changed to a different random color, but it will not be assigned yet. The old and new colors are saved and a new cost is calculated. If the new conflicts are lower than or equal to the old conflicts, then the total conflicts and neighbor's color is updated, steps will be incremented by one. If the new conflict is higher than the old conflicts, then nothing will be updated.

When the total conflicts is zero, the program will print out final total conflicts, the number of iterations and the number of states recolored. If no solution is found, it will print out no solution. It will double check the number of conflicts then print out the states and its colors.

Contributions:

Hongyu Li: wrote local search code; assisted in researching and implementing AC-3 algorithm for backtrack

Loi Pham: wrote parsers for both backtrack and local search; wrote backtrack algorithm, assisted in researching and implementing AC-3 algorithm for backtrack

Philip Kim: optimized local search and implemented state conflict cost; assisted in researching and debugging AC-3 algorithm for backtrack