ITCS 6150 Project 3 Report

Map Coloring Problem Formulation

The Map Coloring Problem deals with coloring the divisions of a map such that no two adjacent divisions share a color. The number of colors that it takes to solve this problem for any map is called the map's Chromatic Number. We are solving this problem using different variants of DFS for solving Constraint Satisfaction Problems. We want to support sorting by Heuristics, so we need a heap structure to do efficient sorting. I chose to use Python, as its Dictionary hashmap structure lets me easily load the data on adjacent states into the program, and the heapq module allows heap sorting. The Heuristic functions for choosing the next state to color are "Minimum Remaining Values", in this case favoring states which have the fewest legal color values, and "Degree Heuristic", which favors states which remove the most values from adjacent states. The Heuristic which helps pick the color to color a state is "Least Constraining Value", which favors colors which rule out the fewest values in adjacent states.

We want to keep track of the number of backtracks that each algorithm makes as we propagate through the tree. Since my implementation of DFS uses recursion, it passes the number of backtracks up through the recursion tree and adds each branch's backtracks together. At runtime, the code generates four random coloring order arrangements each for the Australia and America maps, and runs each algorithm on each arrangement. This randomization is more important for the non-Heuristic variants, as the Heuristic variants will reorder the arrangements accordingly.

Program Structure

Helper Functions

The function isAdjacentColor takes in a state, the color being tested, the colored states so far, and the adjacency dictionary. It uses these to determine if the color being tested has already been used on a state adjacent to the state it is testing for. It returns true if the color exists in an adjacent state or false if it does not. This is used to ensure we don't incorrectly color a state.

The function findSingleton takes in the order variable, containing the remaining states in the order in which they are sorted to be colored, and the possibleColors dictionary, which is used to contain the remaining legal color values for each state in the versions of the algorithm which use Forward Checking. It returns the first singleton if one exists or false if one does not, where a singleton is a state with only one possible color choice remaining.

The function findSingletons is very similar to findSingleton except it returns a list of ALL of the singletons. This is important as in the heuristic form of the algorithm with singletons, we need to sort the singletons by the heuristic in order to determine which to color first.

Heuristic Functions

The chooseState function implements the Minimum Remaining Values and Degree Heuristics through the use of a priority queue. It takes in the order list, the adjacency dictionary, and the possibleColors dictionary, and returns a sorted list based on the heuristics, with Minimum Remaining Values taking first precedence, and Degree Heuristic as a tiebreaker.

The chooseColors function implements the Least Constraining Value heuristic by taking in the state, adjacency dictionary, order list, and possibleColors, then determines which color to

choose by iterating through each adjacent state that is still in the order list to count the number of values ruled out by each color, then uses a heap to sort for the least constraining value.

Algorithm Functions

The ColorDepthFirst function takes in the adjacencyLists, and state coloring order lists and readies the data for the recurDFS function. It runs the recurDFS function with an increasing number of colors in the possibleColors array until there is a success, thereby keeping the chromatic number of the map coloring as low as possible. It keeps track of the total number of backtracks through each call of the recursive function, and then returns the colored map and the number of backtracks when the recursion finishes. The ColorDepthFirstForward, ColorDepthFirstForwardSingleton, ColorDepthFirstH, ColorDepthFirstForwardH, and ColorDepthFirstForwardSingletonH functions are all identical except they call their own variant of the recursive function.

The recurDFS function takes in the adjacencyList, the coloring order list, the list of state colors so far, and the list of possibleColors. In this case, the list of possibleColors is never modified, as there is no forward checking, but the list exists regardless for implementation consistency. The function's base case is when the order list is empty. In that case, it knows that every state is colored, and returns the colored states list and a backtrack value of zero for the successful branch. If it has not reached the base case, it picks the first state off the order list (which in this case is the just the order that we give to the ColorDepthFirst function), then iterates through possible colors until it finds one that works and recurs to the next state, having updated the parameters, or returns False and the backtracks value. The updated parameters include the newly colored state in the 'colors' parameter, and removes the state from the 'orders'

parameter. If the recursion fails, it tries the next color, and increments the backtracks value according to the returned backtracks value. With each tried color that fails, it increments the backtracks variable (as we counted a failed color choice as a backtrack in the in-class activity). If the recursion succeeds, it returns the answer up the recursion tree alongside the accumulated backtracks value.

The recurDFSForward function is similar to recurDFS with the addition of a few lines of code to implement Forward Checking. If we are about to try coloring a state without any possible colors, the backtrack value is incremented by one, as backtracking is about to happen, since without any possible colors, the coloring loop never runs. Also, when we successfully color a state, before passing the possibleColors dictionary to the recursion, the function first makes a copy of the possibleColors dictionary and performs forward checking, updating it accordingly and passing the updated version to the recursion.

The recurDFSForwardSingletons function is similar to recurDFSForward, with one addition. When picking the state to color next, instead of picking straight off the order list, it first checks for a singleton using the findSingleton function, and if one exists, it colors it instead of the first state in the order.

The recurDFSH function is similar to recurDFS, but instead of picking the first state in the order, it sorts the order using the chooseState function, and picks off the top of it instead. It also sorts the possibleColors[state] array for the state we are coloring using chooseColors and iterates through the colors in that instead of through the unsorted possibleColors[state] array.

The recurDFSForwardH function makes the same changes to recurDFSForward, requiring no other special changes to do Forward Checking alone.

The recurDFSForwardSingletonH makes the same changes to the function recurDFSForwardSingleton, but it also has to change to the findSingletons function instead of the findSingleton function, as we also want to sort the singletons if there is more than one. We do this using chooseState, just like we do to sort the states order when there are no singletons.

Data Loading Code

The adjacency data is hard-coded into the program so that we don't have to use user-input to input the adjacency lists. In order to ensure that this hard coded data is correct, it counts the number of unique states both in the labels of the dictionary and the adjacency list, so that if any value was misspelled, the number of counted states would appear too high. It loads the Australia adjacency data into AustraliaAdjacentStates and America into AmericaAdjacentStates.

Experiment Code

The experiment code uses the random module to randomly order the list of states for both maps four times, as required by the project requirements. It then runs each algorithm on each random arrangement, printing the returned list of states, time required, and backtracks for each run, and collects average backtracks and average elapsed time for each algorithm. It then prints the average values for each algorithm together at the end as a summary. These averages are stored in global variables like these: avAusDFSback and avAusDFStime.

Experiment Results

Random Arrangements:

Australia:

Order 1:
['New South Wales', 'Tasmania', 'Northern Territory', 'South Australia', 'Queensland', 'Western Australia', 'Victoria']
Order 2:
['Northern Territory', 'Tasmania', 'South Australia', 'Western Australia', 'Queensland', 'Victoria', 'New South Wales']
Order 3:
['Queensland', 'Western Australia', 'Northern Territory', 'Victoria', 'New South Wales', 'South Australia', 'Tasmania']
Order 4:
['Northern Territory', 'Tasmania', 'Queensland', 'New South Wales', 'Victoria', 'South Australia', 'Western Australia']

America:

Order 1:
['Nevada', 'Hawaii', 'New York', 'Maine', 'Nebraska', 'West Virginia', 'South Carolina', 'New Mexico', 'Maryland', 'Wyoming', 'Connecticut', 'Washington', 'Colorado', 'Ohio', 'Arizona', 'Vermont', 'Idaho', 'Missouri', 'Illinois', 'Tennessee', 'North Carolina', 'Kansas', 'California', 'Massachusetts', 'Utah', 'Texas', 'Michigan', 'Alaska', 'Oklahoma', 'Pennsylvania', 'Alabama', 'South Dakota', 'Arkansas', 'Montana', 'Georgia', 'Minnesota', 'Oregon', 'Virginia', 'Louisiana', 'New Hampshire', 'Wisconsin', 'Indiana', 'Delaware', 'Iowa', 'New Jersey', 'North Dakota', 'Mississippi', 'Kentucky', 'Florida', 'Rhode Island']
Order 2:
['Hawaii', 'Arizona', 'Alaska', 'Mississippi', 'Maryland', 'Tennessee', 'Idaho', 'North Dakota', 'Wyoming', 'New Hampshire', 'Georgia', 'Louisiana', 'Arkansas', 'Indiana', 'Pennsylvania', 'Delaware', 'Virginia', 'South Dakota', 'Missouri', 'Nevada', 'Illinois', 'Texas', 'North Carolina', 'Nebraska', 'Washington', 'Kentucky', 'Utah', 'Vermont', 'Oklahoma', 'Montana', 'Iowa', 'Massachusetts', 'Wisconsin', 'Florida', 'Colorado', 'New Jersey', 'California', 'Connecticut', 'West Virginia', 'Maine', 'South Carolina', 'Michigan', 'Ohio', 'Minnesota', 'Rhode Island', 'Kansas', 'Alabama', 'New Mexico', 'Oregon', 'New York']
Order 3:
['Pennsylvania', 'Louisiana', 'Virginia', 'Ohio', 'Washington', 'Idaho', 'Oklahoma', 'West Virginia', 'New Hampshire', 'Missouri', 'Wisconsin', 'Kentucky', 'Massachusetts', 'Iowa', 'Arkansas', 'Connecticut', 'Colorado', 'New York', 'Texas', 'Tennessee', 'Indiana', 'Michigan', 'Hawaii', 'Alabama', 'Kansas', 'Wyoming', 'Maryland', 'Alaska', 'Mississippi', 'North Carolina', 'Delaware', 'Nebraska', 'California', 'Nevada', 'Montana', 'Vermont', 'Minnesota', 'South Carolina', 'Georgia', 'Rhode Island', 'South Dakota', 'Arizona', 'Oregon', 'North Dakota', 'Maine', 'Illinois', 'Florida', 'New Jersey', 'New Mexico', 'Utah']

Order 4:
['Georgia', 'Iowa', 'Maine', 'Alaska', 'West Virginia', 'Nevada', 'Kansas', 'Arizona', 'New Mexico', 'California', 'Arkansas', 'Colorado', 'Ohio', 'Vermont', 'Rhode Island', 'Pennsylvania', 'Utah', 'Mississippi', 'Delaware', 'Hawaii', 'Nebraska', 'Idaho', 'New York', 'Minnesota', 'New Hampshire', 'North Dakota', 'Alabama', 'Oklahoma', 'Louisiana', 'Maryland', 'Connecticut', 'Indiana', 'Illinois', 'Tennessee', 'North Carolina', 'Washington', 'Texas', 'Oregon', 'South Carolina', 'Florida', 'Kentucky', 'Missouri', 'Wyoming', 'Wisconsin', 'Montana', 'South Dakota', 'Virginia', 'Massachusetts', 'New Jersey', 'Michigan']

Results Tables:

Australia

Algorithm	Order #	Backtracks	Time
DFS	1	28	9.87999992503319e-05 seconds
DFS	2	22	7.639999967068434e-05 seconds
DFS	3	21	7.779999941703863e-05 seconds
DFS	4	26	8.380000144825317e-05 seconds
DFS w/ FC	1	9	0.0002598 seconds
DFS w/ FC	2	6	0.0002082 seconds
DFS w/ FC	3	7	0.0002129 seconds
DFS w/ FC	4	5	0.0001716 seconds
DFS w/ FC and S	1	5	0.0001216 seconds
DFS w/ FC and S	2	5	0.0001112 seconds
DFS w/ FC and S	3	3	0.0001097 seconds
DFS w/ FC and S	4	5	0.0001172 seconds
DFS w/H	1	17	0.0001756 seconds
DFS w/H	2	17	0.0001576 seconds
DFS w/H	3	17	0.0001512 seconds

DFS w/H	4	17	0.0001659 seconds
DFS w/ FC and H	1	0	0.0002821 seconds
DFS w/ FC and H	2	0	0.0002062 seconds
DFS w/ FC and H	3	0	0.0001532 seconds
DFS w/ FC and H	4	0	0.0001510 seconds
DFS w/ FC, S, and H	1	3	0.0002112 seconds
DFS w/ FC, S, and H	2	3	0.0001813 seconds
DFS w/ FC, S, and H	3	3	0.0001741 seconds
DFS w/ FC, S, and H	4	3	0.0002949 seconds

America

Algorithm	Order #	Backtracks	Time
DFS	1	20306062	64.38911 seconds
DFS	2	15451015	48.47034 seconds
DFS	3	81526889	180.42782 seconds
DFS	4	8205376	19.71657 seconds
DFS w/ FC	1	2694153	149.42721 seconds
DFS w/ FC	2	1631801	114.36733 seconds
DFS w/ FC	3	9462360	390.43853 seconds
DFS w/ FC	4	1130436	43.71341 seconds
DFS w/ FC and S	1	13973	3.91351 seconds
DFS w/ FC and S	2	42377	2.06163 seconds
DFS w/ FC and S	3	1767	0.33828 seconds
DFS w/ FC and S	4	42091	3.88574 seconds

DFS w/H	1	35973000	278.83919 seconds
DFS w/H	2	35973000	274.09978 seconds
DFS w/H	3	35973000	288.48511 seconds
DFS w/H	4	35973000	268.48153 seconds
DFS w/ FC and H	1	0	0.0116136 seconds
DFS w/ FC and H	2	0	0.011344 seconds
DFS w/ FC and H	3	0	0.0112367 seconds
DFS w/ FC and H	4	0	0.0135572 seconds
DFS w/ FC, S, and H	1	9	0.011943 seconds
DFS w/ FC, S, and H	2	9	0.0105498 seconds
DFS w/ FC, S, and H	3	9	0.0102544 seconds
DFS w/ FC, S, and H	4	9	0.0141367 seconds

Averages:

```
Final Results:
---------------------------
DFS:
        Australia:
                Average Backtracks: 24.25
                Average Time: 8.419999994657701e-05 seconds
        United States:
                Average Backtracks: 31372335.5
                Average Time: 78.25096134999694 seconds

DFS w/ Forward Checking:
        Australia:
                Average Backtracks: 6.75
                Average Time: 0.00021312500030035153 seconds
        United States:
                Average Backtracks: 3729687.5
                Average Time: 174.4866234250003 seconds

DFS w/ Forward Checking and Singletons:
        Australia:
                Average Backtracks: 4.5
                Average Time: 0.00011494999853312038 seconds
        United States:
                Average Backtracks: 25052.0
                Average Time: 2.5497934500017436 seconds

DFS w/ Heuristics :
        Australia:
                Average Backtracks: 17.0
                Average Time: 0.00016257500101346523 seconds
        United States:
                Average Backtracks: 35973000.0
                Average Time: 277.47640332500305 seconds

DFS w/ Forward Checking and Heuristics:
        Australia:
                Average Backtracks: 0.0
                Average Time: 0.0001981250024982728 seconds
        United States:
                Average Backtracks: 0.0
                Average Time: 0.011937875002331566 seconds

DFS w/ Forward Checking and Singletons and Heuristics:
        Australia:
                Average Backtracks: 3.0
                Average Time: 0.00021537500288104638 seconds
        United States:
                Average Backtracks: 9.0
                Average Time: 0.011720975002390333 seconds
```

Analysis:

The most dramatic improvements are shown in the case of the America map. The fact is, the Australia map is simple enough that while improving on DFS does technically yield improvements, they are barely measurable in the timing category. Implementing BOTH Forward Checking and Singletons improved the algorithm dramatically in both time and number of backtracks, but Forward Checking alone actually made the times worse, but improved the number of backtracks. I suspect this is because of the time it takes to copy and update the arrays storing the possible colors for each recursion. Heuristics alone yielded backtrack improvements only in the Australia map, and took more time than DFS for both maps. Heuristics and Forward checking together resulted in zero backtracks for both maps and dramatic time improvements. Adding singletons to the heuristics and forward checking gave better results than the singleton version without heuristics, but increased backtracks over the version without singletons, and yielded times within margin of error compared to without singletons.

A good takeaway from this is, forward checking and singletons without heuristics improved the algorithm significantly, meaning that it may be viable to use those methods without heuristics depending on the problem. Also, singletons don't seem to yield measurable improvements when you do use heuristics.

All of the algorithm versions successfully found solutions with the correct chromatic numbers for both maps of 3 for Australia and 4 for America, so I found it unnecessary to include chromatic numbers in the tables.