Spencer Doyle

COMP560 HW1

Spring 2020

Implementation

My program (Java) takes in the test file from the standard input (should be piped in from an input file to stop the Scanner). The input is effectively translated into a CSP object which stores all of the states, values, and constraints. At first I tried to implement the backtracking search iteratively, but with help from Sean I was able to conceptualize the need for recursion to help us jump back to earlier “nodes” once the search fails. My search uses the MRV state to make choices. After playing around with MCV’s, I decided to use MRV because it seemed to take a lot less time. My function terminates (base case) when there is no color left to choose from for the given state. If there are colors, we continue on a depth first search until there are no more possible colors, or until the problem is solved.

In my local search I begin by assigning random values to all of the states. However, I don’t do this once. I reassign every state until we have a coloring that contains 15 or less constraint violations. This surprisingly only takes about a second to do, but from here it takes exponentially longer. For the next minute, my function goes through a list of violating states, and randomly assigns those. If they violate less constraints the changes are saved, otherwise we go back to the old value. After each traversal of the list if there is no progress, I pick a couple of the more constrained states, and randomly reassign those to help nudge the problem along. Then the process starts over again. Ultimately I was able to get the number of violated states down to 6, although most often it is more. This took a good deal of effort and manipulation of the reassignment strategy to get to.