Trevor Klinkenberg

CS 461 Intro to AI

Program 1 report

For my approach to the problem I decided on an iterative approach to generate the solution set with a backwards propagation approach in solving the game. This unfortunately had some drastic consequences when generating the 5x5 board, including multiple out of memory issues that resulted in an incomplete generated solution set. If completed the solution file would be multiple gigabytes of data which becomes unwieldy. This approach certainly would not work in any larger approach since the states would only become larger exponentially, and the compute time would end up nearing infinite time effectively.

To create the solution set I started from a win state of all 1’s or “black” for the description purposes. From this state I generated all combinations of single order flips, after this was generated I would take an iterative approach of second order flips based off those single order flips, and so on until I generated the entire tree of possible paths that could possibly be reached. This unfortunately resulted in extreme compute times and a failure in generating the entire tree due to combinatorial explosion. Instead of generating this entire tree I could have improved on this by removing duplicate board states and only generating the bottom leaf nodes, this would still have resulted in 33 million board states, but this is still significantly less then the amount of lines my code would have generated if given enough time and compute resources. This solution file did result in a bubble up effect in making the computer have difficulty in solving certain board states since the entire solution set was missing.

My approach for the game was a straight forward process. Given a specified size by the player the game would generate both the solution file and a randomly generated board of 1’s and 0’s of n by n where n is the size given by the player. The solution file generation is commented out in the submitted program due to the computational needs mentioned above. The randomly generated board is presented to the player in a text based grid of ones and zeroes and the player is asked which coordinate to flip. This will change all states of the board in the row and column selected by the player. The computer will then check if this is a winning state, either all 0’s or 1’s. If not the computer will then ask the player if they would like to be the one to solve the problem. If the player choses yes the computer will then query the generated solution file for the board state and using the line in the solution file will know what move it will need to flip to move towards the solution and how many moves away it is. The benefit of the solution file being generated by an iterative approach means that the first board state the computer will find will be the closest to the win state. This is effectively a breadth first search that the computer is doing even though it is only parsing the file. Once the board file is found the computer will present how many moves it is from the solution and reverse it’s way up the tree as it generates new board states by back tracking from it’s flipped board state.

There is multiple ways this code could have been optimized to reduce compute time however these were not revelations I came across until too late in the generation of the project. Including but not limited to; Removing duplicate board states to reduce the possibility of loops, Generating only the final permutations to save on compute resource and memory allocation, and utilizing the approach of symmetry to reduce the search space in half.

If this board was any larger however these approaches would still not be the most effective, and quite possibly the best approach may be to generate a set of random game boards and including unique states while applying a heuristic of attempting to maintain columns and rows of a single state to get a reasonable approach on the goal. This does not give the program knowledge of how close it is to the goal, but it does save on memory and computational costs.