Dean Cochran
CSC 372
Artificial Intelligence
Prof. Allen
Feb 2021

Programming Assignment #2 Write-Up

This program assignment takes the model 2x2 Rubik's Cube which was capable of being manipulated by the user in real time, and uses implemented A.I. to solve a Rubik's cube. The A.I. takes a scrabbled cube and uses the Iterative Deepening A* (IDA*) algorithm to find the most optimal solution to solve a cube. For reference A* is a best-first search algorithm, which means that it is an algorithm that uses both "past knowledge", gathered while exploring the search space, denoted by g(n), and an admissible heuristic function, denoted by h(n), which estimates the distance to the goal node, for each node n. There are other best-first search algorithms, which differ only in the definition of their "evaluation function", denoted by f(n). For example, the evaluation function of A* is f(n)=g(n)+h(n), where h is admissible. IDA* combines A* and Iterative deepening depth first search to always find the most optimal solution, as opposed to A*, it has a higher space complexity.

Our algorithm is given a scabbled cube instance which then it determines the next possible moves by evaluating penalty of turning a cube face in a certain way. For our A.I. solver we only require the computer to be able to turn 3 sides. This allows us to fix one corner piece of the 2x2 Rubik's cube and solve for the 3 sides that the corner piece colors show (This is true since solving three sides of a cube also solves the entire cube). Therefore, there are 3 ways to turn 3 different faces, clockwise, counterclockwise, or a double clockwise turn, which significantly lowers the number of computations that the algorithm needs to search through. The purpose of this specification is to create a space for the solver algorithm to efficiently be able to identify the most optimal solution to a scrabbled Rubik's cube. The algorithm utilizes this evaluation function by implementing a heuristic function which can influence the algorithm as to what face should be turned.

Our heuristic function utilized a common metric for evaluating distance called Manhattan Distance. Essentially, we can compute the number of face turns needed to move each corner piece into the correct position, by counting the amount of its stickers that are already on the correct face (relative to the fixed corner piece). The sum is then divided by 4 since there are 4 pieces that are changed with every turn. By doing this it makes our heuristic admissible, which is necessary for IDA*, and allows our solver to prevent itself from overestimating the number of turns that will need to be required to solve the cube.

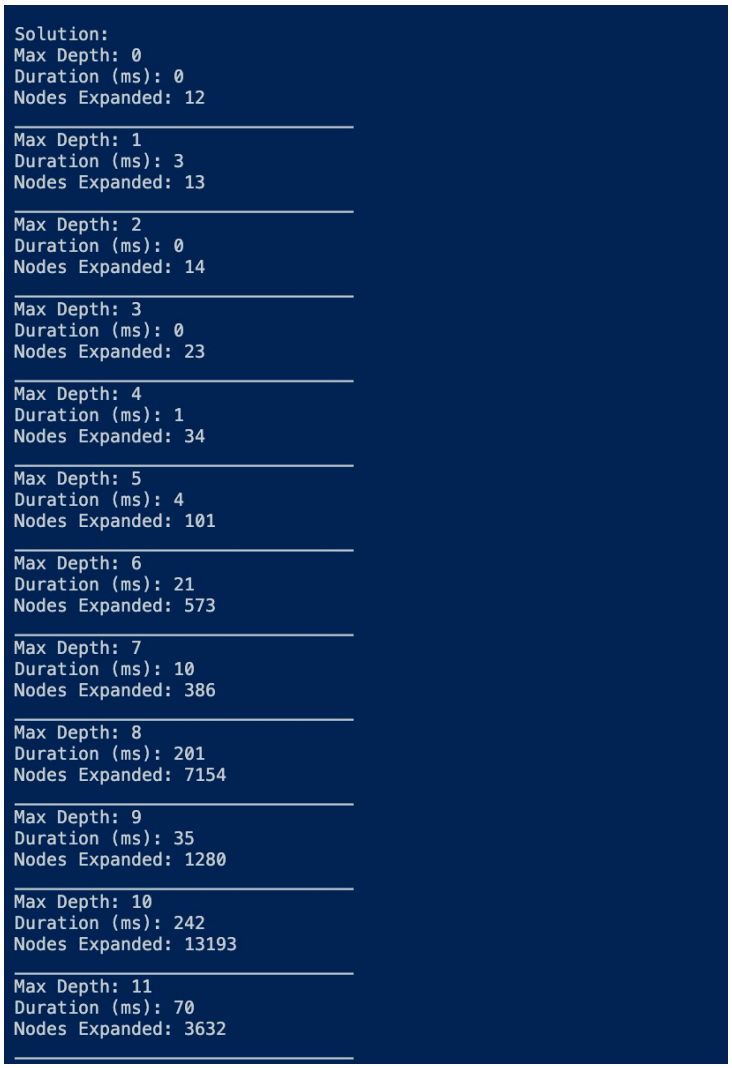
To work the code please start by downloading the zipped A2.zip file and open it in any directory. Now, in your terminal change your current directory to this new directory and compile the code by entering the command,

javac java,

then to run the code enter,

java A2,

You should have to wait as the code takes a few minutes to complete and see something like this.



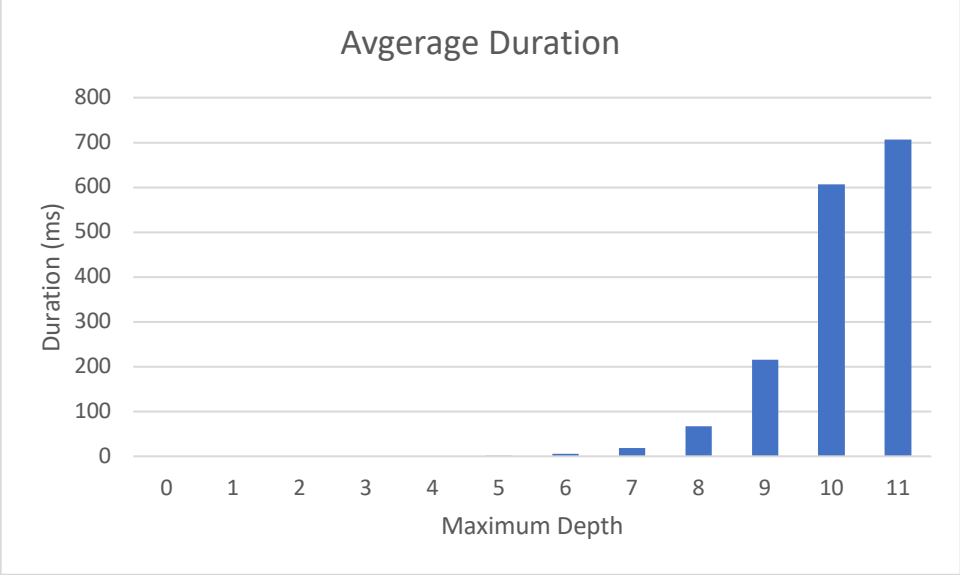
What you see above is the programs output to the user. Essentially, for each Max depth setting in the range of zero to eleven (since any cube can be solved in a maximum of 11 moves), we generate 10 cubes. Therefore, we will eventually generate 110 in total. For each cube, we record the number of possible moves attempted by the computer (show as the number of Nodes expanded), and the CPU time (wall time) it takes for the program to solve a given cube. Additionally, the start state and solution sequence are stored. The output above shows the average turns attempted, and CPU time of all of the 10 cubes given that the cube was randomly turned according to the Max depth setting.

For the assignment the implementation was done in java. The newest and most notable addition to this Rubik's cube creation was the Solver and Node class. To create usable classes that were capable of storing and processing the correct information we were required to import a couple of packages.

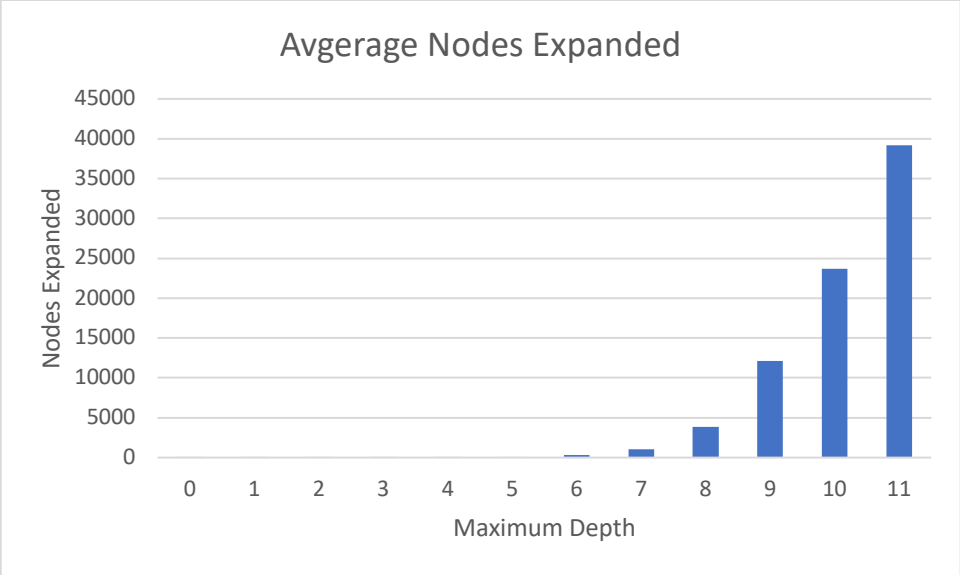
```
import java.util.HashMap;
import java.util.ArrayList;
```

These allowed us to add the necessary functionality to compute complex algorithmic procedures which would enable us to solve a scabbled cube.

The results from out assignment were optimistic in theory. From the program we compute the average turns attempted (nodes expanded), and the average CPU for the program to compute a cube per randomized state of turns equal to the maximum depth setting. Our results are shown below.



This plot allows us to identify that as the cube became more scrabbled. The computer took longer to solver the cube. Intuitively this makes sense that in order to find the most optimal solution when presented with exponentially many solutions the run time might be high.



Secondly, this plot shows us the average amount of turns attempted to solve as the cube became more scrabbled. Again, this intuitively makes sense that the more a cube is scabbled, the more turns a computer will need to compute in order to estimate the most optimal solution for solving any random cube.

From this assignment I learned how to implement a heuristic function to enable a computer to compute an optimal decision based off an evaluation function. This in turn helped me learn the power of heuristic functions and their applicability to A.I. Additionally, this project helped me understand how to code distance metrics and use them to compute evaluation scores.

There was no use of outside resources being used to implement this project besides the Richard Korf document which helped us create this IDA* algorithm capable of finding the most optimal solution.