Solving the Rubik's Cube

Names: Gauri Wahi, Robin Bandzar

Teach CS ID's: wahigaur, bandzarr

Roles:

Robin: Test Cases (minor) , Input (major), Searches (major), State Space/goal state (major), report (minor)

Gauri: Test Cases (minor), Actions/state change: Cube Interaction (major), report (minor), searches (major)

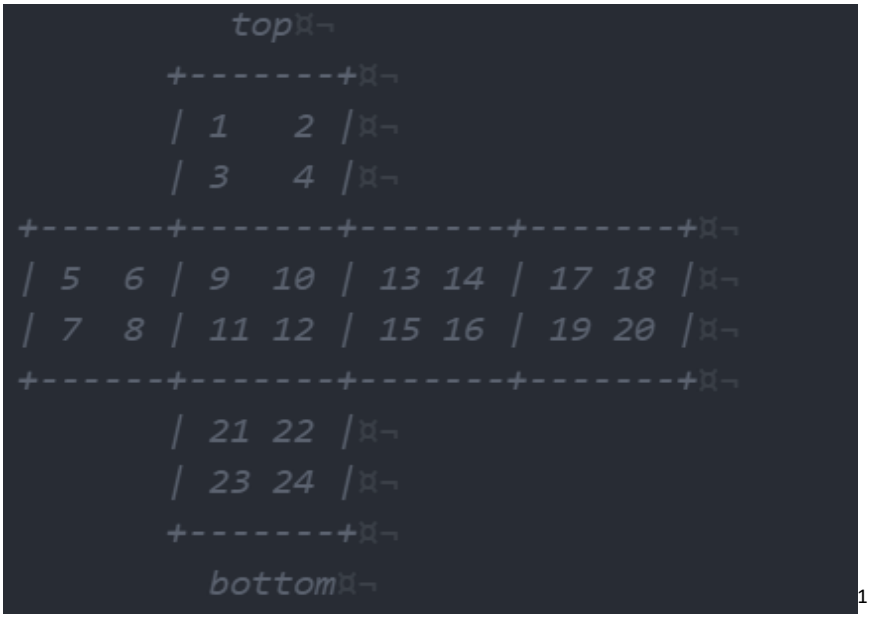
Project Type: Search

Project Motivation/Background

Our project is based on the extensively searched Rubik's cube problem, which we are trying to figure out how to make each face one solid color by rotating. There are many ways to solve this problem and one of the ways is to use a heuristic search. We chose to do the heuristic search using the Manhattan distance to find the original location of each cubie, as well as IDA* to find the goal state.

Methods

We have used an ASCII representation to show each state of the cube until it reaches its goal state. We are mapping each of the faces to a color so we can easily distinguish the final goal state. The turns are being done by visualizing which faces would be affected. E.g., In this 2 x 2 cube shown below, if we are to do a left turn then the top, rear, and bottom faces will be affected as well.



[1]

We are using IDA* as our search method because it will not use as much space as A* will. This is because it will avoid remembering which states we have already visited to find the first solution. Our cost in this scenario is the time it takes to find a solution. We do not want the program to be running for more than 10 minutes, which is another reason why we used IDA* to control the depth of the search tree. One of the problems with our search method is that it is exponential in time complexity, which means the deeper we go the longer it will take. This is because our admissible heuristic will increase exponentially when the depth is greater than 18.[1]

Our heuristic method is using the Manhattan Distance to know which cubies are still misplaced. With the heuristic function, we are analyzing which faces are affected by each turn.

[1] http://symbolaris.com/orbital/Orbital-doc/examples/Algorithms/RubiksCube.java

The method we are using to calculate this is by assigning each corner a coordinate and mapping it to three colors per cubie. This is so we know the unique combination of the eight corners of the cube and where they should be. For our heuristic to be admissible we are going to divide the answer by 4, since we are using a 2 by 2 cube and we are considering the rotation of corners and edges, which are equivalent to one another. E.g., on the face, tiles 9-12 are both edges as well as corners. The equation that we have used is for two points p1 and p2 on the cube:

$$Manhattan\ Distance\ (p1, p2): \mid x1 - x2 \mid + \mid y1 - y2 \mid + \mid z1 - z2 \mid$$ [2]

The turns in a Rubik's Cube are L (left), R (right), U (up), D (down), F (face), and B (bottom). Theoretically, the rotations can be clockwise or counter-clockwise, but to improve the speed of our search we have removed the inverse (counter-clockwise) moves. Instead of using vectors/matrix representations we have decided to look at each moving point for a turn, where each of the faces are represented by a number and a color. Since we are representing a 2 by 2 cube, the maximum number of faces we can have are 24. The number of colors we can have are 6 – one for each side of the cube. We have chosen each face color to be as follows: white, green, yellow, red, and blue. For each face, we are using cycle notation[3], which allows us to know what corner/edge will go where when a face is turned. E.g., if we do a rotation, a corner edge cannot go into the middle, but instead will remain in the corner of the new face. This rule applies to all the tiles in the cube. Each cubie is associated with a coordinate as follows:

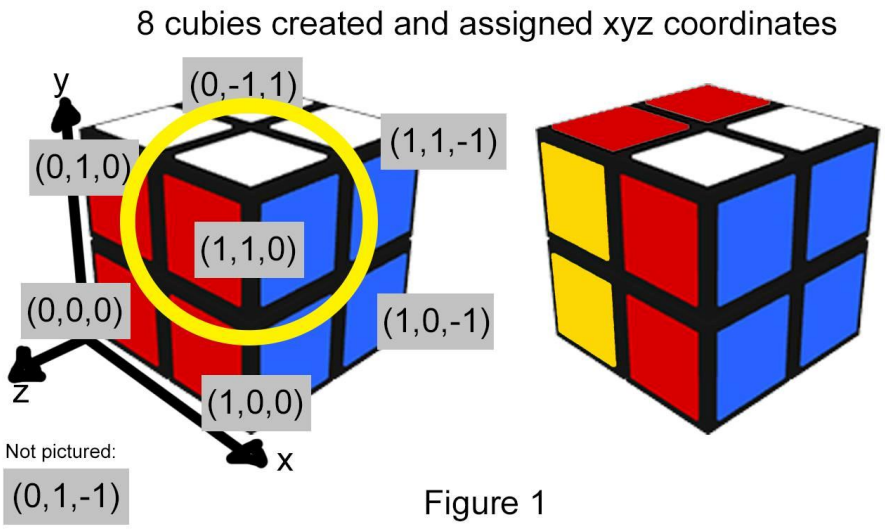


Figure 1

[2] https://heuristicswiki.wikispaces.com/3D+Manhattan+distance

[3] https://books.google.ca/books?id=VePv84nsfWIC&pg=PA57&lpg=PA57&dq=cycle%20notation%20for%202x2%20cube&source=bl&ots=kmE9n02RZ2&sig=HNmtfny-E8Ry-h7td8phTPE0P-4&hl=en&sa=X&ved=0ahUKEwj7lN3q9dLVAhVC74MKHa_HAoQQ6AEIYTAJ#v=onepage&q=cycle%20notation%20for%202x2%20cube&f=false

Evaluations and Results

We have created 5 test cases to compare the amount of time it takes to find a solution using IDA*. It allows it to go to a maximum depth, which we have chosen to be 13. This is because the maximum number of moves a cube can do is 14 for a 2 by 2 Rubik's cube, but since we don't want to use up space complexity and our waiting time, we have limited the amount of the depth in hopes our program will find a solution.[4]

When implementing IDA* we must remember that it goes per depth of the search tree. So, we set the strategy to BFS+A-star as our IDA* algorithm to make sure it was going node per node per depth. Another method we induced was by changing the order of the search as well as improving our hash method, as well as removing the inverses. This helped us reduce our time from 63 seconds to 1.57 seconds for 5 moves. By removing the inverses, that made our search cost go up, but this is because we have avoided pruning the already visited states. This is also because the inverses (Left, Back, etc.) were causing repeated states that we would have already checked. In Figure 1 you can see our current nodes that have been expanded are 22874, compared to the nodes expanded in Figure 2 (768613). As you can see, when searching with the repeated states our program took longer to find a solution, whereas when we removed the inverses it was able to find a solution faster. You can also see from Figures 3 and 4 that when we were considering the repeated states, our search time was increasing exponentially, whereas now our time depends on the first solution that was found and not the "best" solution, which is why our cost is now higher.

```
************************************
PROBLEM 4
Solution Found with cost of 9 in search time of 1.5799999999999992 sec
Nodes expanded = 22874, states generated = 23923, states cycle check pruned = 1049, states cost bound pruned = 0
ACTION was START
()
  wo
  gy
ryrbrwbg
wogoywbo
```

Figure 1: Current Nodes Visited

[4] http://www.popsci.com/science/article/2010-08/gods-number-revealed-20-moves-will-solve-any-rubiks-cube-position

```
PROBLEM 0
Solution Found with cost of 8 in search time of 60.160000000000004 sec
Nodes expanded = 768613, states generated = 929515, states cycle check pruned = 160902, states cost bound pruned = 0
ACTION was START
()
  oy
  gr
byrbwbrw
woygwyoo
  br
  gg
 ==> ACTION was U
```

Figure 2: Previous Nodes Visited

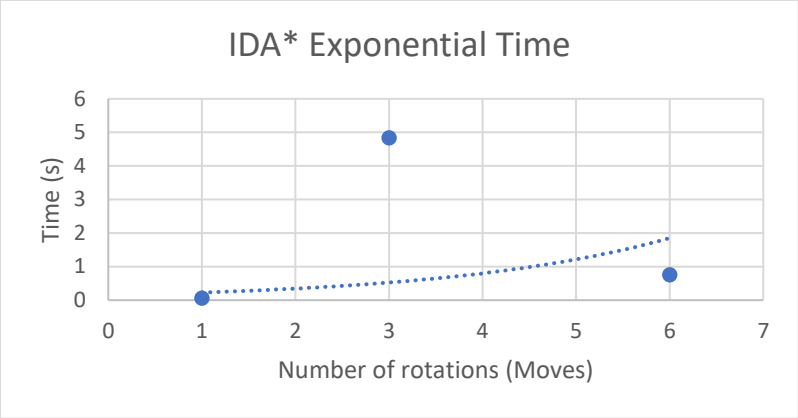


Figure 3: Previous IDA time chart*

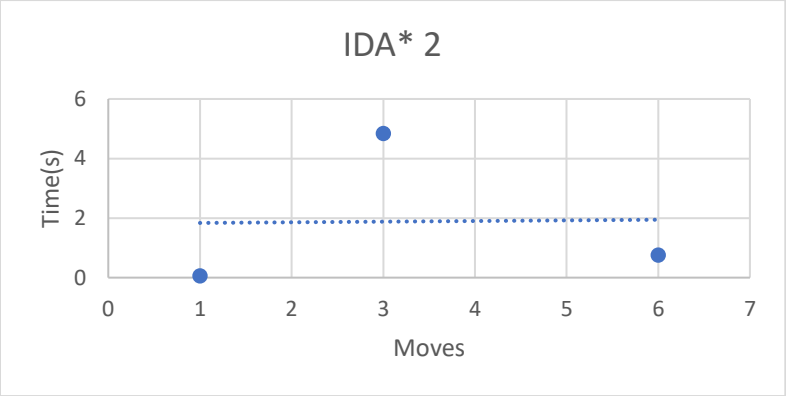


Figure 4: Current IDA time chart*

Another way we sped up our code was by using the conventional FRU method for a Rubik's cube[5].

Limitations/Obstacles

The obstacles we faced in this project were how to represent the cube, as well as making sure we are not making any wrong moves as well as the rotations. Initially, we decided to use

[5] https://rubiks-cube-solver.com/2x2/

vectors and matrices to represent each side of the cube, but unfortunately, that made it difficult to write our heuristic function. Therefore, we chose cycle notation, so we can easily map the colors to each of the tiles as well as make it easier to apply it to our heuristic. Choosing our coordinate system also has a downfall as it cares about the orientation of the cube.

Another one of our limitations is that due to saving too many states when it goes down the tree if there are more than 5 moves our program faces a segmentation fault. This is why we decided to change the order of our search by moving the inverse rotations to the end, as well as choosing a better a hash method.

Conclusion

Our 2 x 2 Rubik's cube can solve to the goal state for the limited number of moves. As the moves increases, the time exponentially increases as well. IDA* search was implemented so we can control the depth of the search tree, so it does not take too long and our program can attempt to find a solution for the given depth. We must improve on our worst-case scenarios for moves greater than 8 moves if possible.

By comparison, we have learned that using IDA* is faster in comparison to A* for a certain number of moves. With the help of our heuristic function, we have created a dictionary so the program knows where the cubies should be located and what is the optimal move. By representing each cubie per corner, we reduced the amount of states we needed to visit from 24 to 8.

By changing our method to start the search and by moving the inverse rotations to the end of the search, as well as choosing a more efficient hashing method helped us decrease our time by a large amount.