CS463G Program 2: A* Search Algorithm

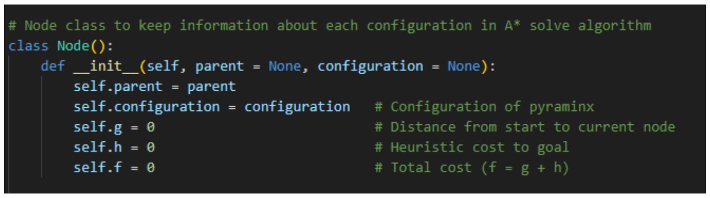
Implementor's Notes
Louis Lin
Dr. Judy Goldsmith, CS463G
9/26/2024

ABSTRACT

This project involves the implementation of A* search algorithm that utilizes an admissible heuristic function. The main objective is to be able to solve a randomized instance of the Pyraminx using this search algorithm. The program allows the user to input *k* values to randomize the puzzle *k* times and the solver attempts to restore the puzzle to its solved state. Using the heuristic, A* search algorithm is used to find the shortest path back to the solved state.

GENERAL APPROACH

Originally, I had used a dict as a data structure to capture the face and colors. For example: *("A : {"red", "red", ... , B: {"green, ...")*, which worked to an extent. However, I had to pivot back to **arrays** due to the complexity and amount of changes I had to make to the rotation functions. So the data structure of the colors consist of colors (0-3), which indicate red, green, blue, and yellow. There were two main variables: *current_colors* and *starting_colors*, which were used to capture the current_state and goal_state which will be described more in the search algorithm. To run A*, I had to make a new class Node in order to store the information of each configuration for the children node.

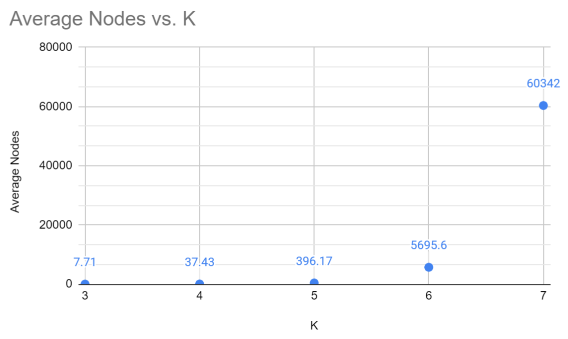


```python
# Node class to keep information about each configuration in A* solve algorithm
class Node():
    def __init__(self, parent = None, configuration = None):
        self.parent = parent
        self.configuration = configuration   # Configuration of pyraminx
        self.g = 0                           # Distance from start to current node
        self.h = 0                           # Heuristic cost to goal
        self.f = 0                           # Total cost (f = g + h)
```

The heuristic employed in the A* algorithm is defined as the maximum number of colors on any Pyraminx face minus 1. This heuristic is admissible, meaning it never overestimates the true cost to reach the goal state. I got the heuristic from the classroom discussion. The heuristic definitely helped speed up the solving of the Pyraminx, but I'm sure there is a much better heuristic but this is just what I ended up settling with.

The implementation of A*, had initial_state and goal_state passed as parameters to pull from any changes that may have occurred from randomize or manual alterations. It does the algorithm using lists and min-heaps and then prints out the path after finding the solution. The program also prints out the time elapsed in hours, minutes, and seconds as well as print out the rotation steps it took to reach the solved state.

Compiling and running the code took quite a while. This was the graph I managed to get. My heuristic was not the best so for me, it took over 12 hours to get a K =7 compiled for over 100,000 nodes.



RUNNING THE CODE

The only file I have is **main.py**. I have it in VS Code to just run when I run the file. It should open up the GUI and all the buttons should be there. Nothing out of the ordinary.

LEARNING OUTCOME

This assignment taught me to find out the best data structure to use BEFORE almost completing it but having to restart completely because your data structure doesn't actually work. That aside, it was really fascinating how the admissible heuristic was able to affect the compile times so much. Originally, I just set Heuristic to always be 1, essentially creating Dijkstra's. It took up to 30 minutes for k = 5 but only takes 7 seconds with an admissible heuristic included. I also learned a lot about GUI manipulation as well as node tracking and their configurations.

Andrew Myers (helped with the GUI set up and data structure from Program 1)
Ruby Harris (asked about general information and heuristics)
Isaiah Huffman (helped with A* and Node creation)
Kristopher Critchfield and Dr. Goldsmith (for the heuristic)
Chat-GPT
Other peers