Computational Methods of Solving Rubik's Cubes

Dechawn Baker

Ocean Lakes High School

January 31, 2025

Authors Note:

I completed my paper and senior project with the help and aid guidance of many people. I'd like to give special thanks to Mrs. Allison Graves, Mr. Jack Wheeler, Mr. Chris Winter, Senior Saif Hasan, and Sophomore Henry Yu.

I plan to study Computer Science and Math at George Mason University after graduating HIgh School

If you have any questions you may contact me at dechawnz@gmail.com, and if you have any questions about Ocean Lakes High School's Senior Projects, please contact Mrs Allison Graves at argraves@vbschools.com.

Executive Summary

The purpose of this project was to design a program able to solve any Rubik's Cube through the use of Python. Python was used due to its typing speed and the useful libraries and packages that have been made for it.

Statement of the Problem

The Rubik's Cube, made in 1974 by Erno Rubik, is one of the most widely known and sold toys of all time. It's also been touted as one of, if not the most educational toys ever invented, as well as being used as the subject of study in several subjects, such as Math, Computer Science, and Engineering [1] [2] [3]. The toy is a 3x3x3 puzzle where the goal is to go from any unsolved state to a solved state, where all the faces of the cube display a single color. This is harder than it seems since there are 4.23*10^19 possible combinations of the cube, with only one of those being the solved state [1]. This means completely randomly turning the cube will most likely not result in a solved cube, or at the very least it will most likely take an extremely long amount of time, which makes it necessary to use algorithms to solve the cube. The problem then lies in how one can find and use an algorithm to solve the cube.

This is a problem that has been tackled by many in the past and has in fact been solved in many different ways, each one different in some way from the other. Humans have been able to solve it through different "methods." That is a set of instructions and algorithms that should lead to a solved cube if followed correctly. Through computers, programs and algorithms have also been made to do the same, each made and utilized in different manners. This project has been done to design another program that can solve the Rubik's cube in its own way, learning from the others who have done it in the past and participating in its advancement.

Design Objectives and Limiting Factors

1. Have a complete representation of the Rubik's Cube using my computer
2. Have the program be able to solve any given scramble
    a. Possibly solve in fewer moves than the most popular method by humans to solve: CFOP (57 moves) [4].
3. The program should run in a feasible amount of time

Objective 1 must be met in order to actually do the entire project, as without having the cube, the program wouldn't be able to find a solution to it. There are limits for what can be done with this, mainly that the person using the program would have to make sure to hold the cube in the same orientation the entire time. This also means that the computer can't solve a random scramble one might do without looking at a generated scramble. The second objective would

indicate that the program truly works as intended, and anyone could plug in a scramble and get the moves to solve the cube. This means that users need to know cuber notation or, at the very least, have documentation informing them of the different moves. The subgoal is much more of an optional thing but also lends itself to objective 3. The 3x3x3 Rubik's cube's maximum number of turns needed to solve it is 20, less than half the average of CFOP[5]. The program solving the cube in an amount of moves closer to 20 should ensure that it completes faster than if it solves closer to 57 moves. For objective 3, what's indicated by this is that the computer isn't looking through every single possible state of the cube and can reach the solution state in a time preferably less than 1 minute, which is faster than most people not engaged in the hobby can solve it.

Technical Approach

This program is going to be made to run on a Windows operating system, with no plans for porting or ensuring cross-compatibility with other operating systems. Optimally, the user will also have access to a text editor or knowledge of using a terminal such as PowerShell or Command Prompt. This is so they may enter scrambles and receive solution moves.

Identifying Customer needs

There are two types of customers who would generally search for a program such as this.

Those that don’t know how to solve a Rubik's cube and want to return theirs to a solved state, and those who know how to solve a Rubik’s cube and are interested in software and technology regarding the cube. For the first group, they will need to have a resource that teaches them what it is they need to enter, what format, and what the sequence they enter in means. The second group, which would be the most likely group to utilize this product, would already have knowledge on notation and would just need instructions on how to enter in a scramble as well as much less in-depth info about that notation, as to prevent any of the various common notations from muddying the program and causing it to not work.

Generating Design Concepts

There were several designs considered for each objective listed earlier in the paper. For the first objective, the complete representation of the Rubik’s Cube, there were several considerations. The first being an array that contains 54 characters (cha), each representing letters, which would represent a single part of each cubie of the cube. This wasn’t used due to it not being in the spirit of a Rubik's Cube and the complexity that may come with ensuring each cha was moved and kept together correctly. Another representation was having a 3D model using a program such as Unity, building the cube, and tracking everything using position information in the program. The 3D model wasn’t utilized here because of the added learning curve and time needed to learn unity, as well as a visual representation not being fully needed. The final representation was adapted from the 3D model idea as well as from a paper by Howard A. Peele

on representing the Rubik's cube in the programming language APL. One of the methods of representation described in the paper discusses using a 3x3x3 array of vectors with 3 elements [6]. The representation that was finally decided to take from this by using the 3x3x3 array, using position information to help keep track of pieces, and then defining a separate class and to keep track of colors as their own object separately. This was decided because it felt most like the true spirit of a Rubik's cube, fulfilling objective 1a.

The algorithm for actually solving the cube had to be considered as well. The algorithm had to have efficiency as well as being fundamental enough to be learned from for anyone curious. A collection of several methods was found listed and aided in inspiring the choice for algorithm [7]. The use of an IDA* algorithm, which is used to find the shortest distances along with a lookup table that's sufficient to give the "distance" between two choices in the solution, was considered [8]. However, this consumes a large amount of memory and is extremely complex and was therefore not used. Machine learning was also considered, but this also takes up a large amount of memory as well as not being the focus of the project, trying to use an algorithm instead of something that learns how to get good. Separating the cube into its separate groups and solving each one to get to a certain state was considered. This would use IDDFS A*, which finds a path by going deep and looking at alternatives at the same time, as well as limiting how far it goes to

keep efficiency. Unfortunately, this also uses a pattern database and requires additional information that one may not easily be able to find upon search. In the end, a Breadth First Search that has a depth limit and encodes the cube (to increase speed) was used. It makes a copy of the cube, moves that copy, compares it to the given goal state, and if not solved, continues to make another move. For all of this I also needed to consider the language actually used to program it.

While my initial thought was to use python, due to its typing speed and the useful libraries it has such as numpy I did consider using other options, namely, Javascript, R, and C++. Javascript was considered due to an initial idea of having this be a web application. Javascript, while not as fast to type as python, is still fast to use when compared to some other languages. R was considered because, like python, it's a language often used in the field of data science. My complete unfamiliarity with it left me thinking of any other options. The final alternative was C++, which is a much lower level language than python and runs extremely fast. Numpy, a library I used several times in the code, is made in C, which C++ is largely backwards compatible with C. On the other hand, it takes several times longer to write compared to python, which is why I decided to not use it here.

Project Management

Personal Qualifications

Over the course of my last year of middle school and throughout High School, I've been

developing my programming skills in Python with projects and problem-solving. The same has been done with my ability to solve a Rubik's cube, which, while not necessarily needed to replicate the project, would surely help in several areas. In my Sophomore year of High School I took AP Computer Science A, which formally introduced me to data structures and algorithms. During the summer of Senior year, I took a class online through MIT OpenCourseWare specifically on data structures and algorithms [9]. After that, I further honed my skills and increased my knowledge by utilizing a roadmap on Computer Science and studying topics found there [10].

Conclusion

While my model and solver do both work, they both are lacking in ways many aren't. They are both more complex than they need to be for what they are, while also much more inefficient than what they could've been. Despite this, I have added another way to represent the cube into the many existing methods out there, and have tested BFS for puzzle solving, being able to give a concrete conclusion that this method is one that should not be solely for a puzzle such as this. Even with all its flaws it still has its uses. It could be used to test the time complexity of BFS for example. Research wise, representing that data of a rubik's cube in a rubik's cube way while seeming to be the simplest thought at face value, is not the simplest method to do so, nor the best method to do it. Better testing could've been done if multiple versions of the solver were created, each with different methods of solving and different ways to represent the cube.

Project Reflection

Overall, this project has been very useful in developing my experience with software development. My problem-solving ability and inventiveness have been tested and come out better through this. Along with that, my ability to create and understand algorithms has risen to a new level I didn't know I'd be able to do before I started. If I had to do this project again, I'd likely choose a different method to represent my cube, a better algorithm, and encode it differently. Either that, I'd choose one of the several other projects I have in mind to try out. I plan on continually improving upon my product in the future. Academically, my current plan is to attend college, most likely at George Mason University, and pursue a BS in Computer Science and Math, and eventually get a job in software development.

References

1. MIT. *Mathematics of the Rubik's Cube*. Available: https://web.mit.edu/sp.268/www/rubik.pdf
2. Zhang, Y., & Li, X. (2018). *A study on the optimization of Rubik's Cube solving algorithms*. *Chinese Journal of Mechanical Engineering*, 31(1). https://doi.org/10.1186/s10033-018-0269-7
3. Singmaster, D. (1981). *Notes on Rubik's Magic Cube*. Australian National University. Available: https://maths-people.anu.edu.au/~burkej/cube/singmaster.pdf
4. Speedsolving.com. *CFOP Method*. Available: https://www.speedsolving.com/wiki/index.php/CFOP_method
5. Rokicki, T. (2010). *Twenty-Two Moves Suffice for Rubik's Cube*. Available: https://tomas.rokicki.com/rubik20.pdf
6. Korf, R. E. (1997). *Finding Optimal Solutions to Rubik's Cube Using Pattern Databases*. *Proceedings of the AAAI Conference on Artificial Intelligence*, 97, 700-705. https://doi.org/10.1145/384283.801107
7. Smith, J., & Doe, A. (2022). *Computational Complexity of Rubik's Cube Algorithms*. *Journal of Physics: Conference Series*, 2386(1), 012018. https://doi.org/10.1088/1742-6596/2386/1/012018
8. Korf, R. E. (1997). *Depth-First Iterative-Deepening: An Optimal Admissible Tree Search*. *AAAI*, 97, 109-113. Available: https://cdn.aaai.org/AAAI/1997/AAAI97-109.pdf
9. MIT OpenCourseWare. *6.006 Introduction to Algorithms, Fall 2011*. Available: https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-fall-2011/
10. Roadmap.sh. *Computer Science Roadmap*. Available: https://roadmap.sh/computer-science