International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 08 Issue: 06 | June 2021 www.irjet.net p-ISSN: 2395-0072

Introduction to Minimal Thinking Technique using an Extremely Fast Implementation of a Rubik's Cube Solver

Prakhar Gupta

Director of Science, FAIPS, DPS Society, New Delhi, India

---***---

Abstract - *"Minimal Thinking" is a technique developed by me that, as the name suggests, minimizes the thought process behind solving any problem by recognizing the type of problem and using a predetermined result to jump straight to the solution. I have used this technique to develop an extremely fast and efficient Python 3 implementation of a solver for a Rubik's cube puzzle. At each step of solving a puzzle, the solver recognizes the current arrangement and orientation of the individual pieces of the puzzle, and rearranges and reorients the pieces into the final state of the puzzle that the corresponding algorithm would have resulted in, had it been manually executed. This solver is not only faster than every other Rubik's cube solver to date but also a great example of how the technique of "Minimal Thinking" can be used in modern Artificial Intelligence.*

Key Words: Rubik's Cube, Speedcubing, Bi-Directional Breadth First Search, Korf's Algorithm, Kociemba's algorithm, Minimal Thinking technique

1. INTRODUCTION

Modern speedcubers, or individuals who compete in speedcubing (a sport involving solving a variety of combination puzzles, the most famous being the 3x3x3 puzzle or Rubik's Cube, as quickly as possible (src: en.wikipedia.org)) solve the Rubik's cube quickly not by sheer intuition, but with memorized sequences of moves, called algorithms, which they deploy to solve the cube section by section. Knowing which algorithm to use when boils down to pattern recognition: Each algorithm corresponds to a different arrangement of coloured squares on the cube. When a speedcuber spots an arrangement they recognize, they perform the corresponding algorithm, bringing the cube one step closer to solved. Using these algorithms, the most fleet-fingered cubers in the world average between 50 and 60 moves per solve, which they can execute almost without thinking. [1]

An important point to note is that the 3x3x3 Rubik's cube is essentially a 2x2x2 Rubik's cube without the edge and center pieces. Hence anybody who knows how to solve the 3x3x3 cube can also solve the 2x2x2 cube using the exact same algorithms, ignoring those algorithms which are related to permuting and orienting the edge pieces. In particular, solving the 2x2x2 Rubik's cube is just like solving the corners pieces of the 3x3x3 cube and solving a 3x3x3 cube is exactly the same as solving a 2x2x2 cube but with an added complexity of solving the edge pieces. Similarly, solving the 4x4x4 cube is exactly the same as solving the 3x3x3 cube but with an added complexity of completing the centers and pairing up the edges at the beginning of the solve. However, especially after the 4x4x4 Rubik's cube, this difference between the subsequent puzzles of the NxNxN category of twisty puzzles (puzzles like the Rubik's Cube which are manipulated by rotating a section of pieces (src: en.wikipedia.org)) becomes more and more subtle. To a cuber, there is almost no difference between solving a 5x5x5, a 6x6x6 or a 7x7x7 Rubik's cube because solving each one of them involves the general steps of completing the centers, pairing up the edges and then proceeding the complete the puzzle just like a 3x3x3 cube. This general pattern of an added complexity with increasing order of the puzzle can, in fact, be seen not only in the NxNxN puzzle category but in every category of twisty puzzles. For example, a 1x2x3 puzzle can be solved by first correctly positioning its corner pieces as if it were a 1x2x2 puzzle. After this, only a single algorithm is required to correctly orient the pieces in its middle layer, which is absent in a 1x2x2 puzzle.

2. COMMON RUBIK'S CUBE SOLVERS AND THEIR SOLVING METHODS

Rubik's cube solving computer programs are pretty common and have been created by many using many different methods. The general approach of almost every one of these, however, is to use complex graph search algorithms to find an optimum solution that can solve the scrambled cube. Three of the most common and fastest Rubik's cube solving methods are:

1) **Two-way Breadth-first Search method:** Rather than building up a single BFS tree from the scrambled state and searching until the solved state is found, two BFS trees are built - one from the scrambled state and one from the solved state.

2) **Korf's Algorithm:** This algorithm is iterative-deepening-A* (IDA*), with a lower bound heuristic function based on large memory-based lookup tables, or "pattern databases". These tables store the exact number of moves required to solve various subsets of the individual movable cubies. [2]

3) **Kociemba's algorithm:** Kociemba's algorithm identifies a subset of 20 billion positions. Phase one finds a move sequence that takes an arbitrary cube position to some position in the subset, and phase two finds a move sequence that takes this new position to the fully solved state. [3]

3. STATISTICAL ANALYSIS OF THE SOLVERS

Since we have already established the similarity between Rubik's cubes of subsequent orders of the form NxNxN, let us consider the case of a 2x2x2 Rubik's cube for the sake of simplicity and a better understanding of concepts. The 3x3x3 and 2x2x2 cube solvers for each of the above-mentioned solvers are almost identical. In order to compare the speed and efficiency of all these solving methods, I ran the 2x2x2 version of each of the three algorithms on CPython on my PC and compared their mean solve times. To all three I gave the same set of 10 random scrambles and the results are summarized below:

1) **Two-way Breadth-first Search method:**

Source Code: "rubiks_cube_bfs" solver by Mayank Rawat on GitHub (URL: https://github.com/mayank18049/rubiks_cube_bfs)

Mean solve time: 0.5452627182006836 seconds per solve.

2) **Korf's Algorithm:**

Source Code: "PocketCube" solver by Ivan Grudinin on GitHub (URL: https://github.com/kuligram/PocketCube)

Mean solve time: 0.9749078100377863 seconds per solve

The program also took an additional 38.04953694343567 seconds on average to generate the pattern database heuristic before the very first solve.

3) **Kociemba's algorithm:**

Source Code: "Rubiks2x2x2-OptimalSolver" by Herbert Kociemba on GitHub (URL: https://github.com/hkociemba/Rubiks2x2x2-OptimalSolver)

Mean solve time: 0.021625208854675292 seconds per solve

This makes Kociemba's Algorithm the fastest among the three. Kociemba's Algorithm was in fact also used to calculate the solution in the robot that holds the current Guinness World Record for the fastest solve by a robot.

4. Pytwisty PACKAGE

"pytwisty" is an extremely fast and efficient Python 3 implementation of a solver for a number of twisty puzzles including the 1x2x2, 1x2x3, and 2x2x2 Rubik's cube puzzles. This MIT-Licensed package has been developed and owned solely by me. Detailed instructions on the installation and usage of this package can be found at the Python Package Index (PyPI) repository of software for Python (URL: https://pypi.org/project/pytwisty/) and on GitHub (URL: https://github.com/prakharguptafaips/pytwisty) under the project name "pytwisty".

In comparison to the mean solve times of all the above-mentioned solvers, my 2x2x2 Rubik's cube solver produced the following result:

Mean solve time: 6.182333333413226e-05 seconds per solve

This solve time of the order -5 makes this solver 350 times faster than Kociemba's algorithm!

4.1 Working Mechanism of My Solver

Unlike the conventional computerized Rubik's cube solvers that look for an optimal solution from subsets of billions of positions of the cube using complex search techniques, my solver uses the "human approach" to solve the puzzle. Specifically, it runs a combination of a slightly altered version of the layer-by-layer (LBL) method and the CFOP method of the 3x3x3 cube, which is heavily used and relied upon by most of the top speedcubers. I am not the first one, however, to adopt this "human way" of solving a Rubik's cube by a computer. But unexpectedly, all of such solvers are also extremely slow and inefficient. An example for the same is "rubik-cube" package by Paul Glass on PyPI (URL: https://pypi.org/project/rubik-cube/) which uses the Beginner's method and takes about 3 times the time taken by Kociemba's algorithm per solve on CPython.

4.2 Reason Behind the Unexpectedly High Speed and Efficiency of My Solver

One possible reason might be the way I stored and maintained the scrambled state cube throughout the program. Taking the example of the 2x2x2 Rubik's cube, most solvers maintain an array of the colors on all the 24 colored stickers on the cube at all times. In contrast, I have maintained an array of only 8 elements. Without going much into the details, each of these 8 elements maintains the appropriate position and orientation of the corresponding pieces/cubies in the cube at all times during the solve. The logic behind this is that the 2x2x2 Rubik's cube essentially consists of only 8 individual cubies which move about a common center. Each of those 8 pieces comprises 3 stickers that always remain together, no matter what. As a result, I effectively broke down the main problem of solving the entire collection of stickers into two subproblems namely, permuting the locations of the pieces and orienting them. This made my program comparatively easier to implement. However, this is not the main reason behind the extremely high speed and efficiency of my solver. The main reason is the use of the "Minimal Thinking" technique developed by me.

Minimal Thinking technique:

In simplest terms, this is a technique that minimizes the thought process behind solving any problem by recognizing the type of problem and using a predetermined result to jump straight to the solution.

The inspiration for this technique came from one of the most crucial principles of speedcubing, the fact that "each algorithm corresponds to a different arrangement of colored squares on the cube and that when a speedcuber spots an arrangement they recognize, they perform the corresponding algorithm, bringing the cube one step closer to solved." And I successfully used this to create my solver. At each step of solving the puzzle, the solver recognizes the current arrangement and orientation of the individual pieces of the puzzle, and rearranges and reorients the pieces into the final state that the corresponding algorithm would have resulted in, had it been manually executed. This is just like how speedcubers, at each level of solving the Rubik's cube, look at the arrangement of the cube and execute the corresponding algorithm, all without thinking. To better understand this concept, consider this simple math problem:

Calculate the numerical value of $25^2 - 5^2$.

Whenever someone sees this mathematical expression, the very first thing that will come to his mind is the identity $a^2 - b^2 = (a + b) * (a - b)$. And without thinking further, he will immediately rewrite the expression as $(25 + 5) * (25 - 5)$ which will give him 600 as the final answer.

However, someone unaware of this identity would have proceeded as follows:

$25^2 - 5^2$
$= 25^2 - 5^2 + 25 * 5 - 25 * 5$
$= 25 * (25 + 5) - 5 * (25 + 5)$
$= (25 + 5) * (25 - 5)$
$= 600$

While solving a Rubik's cube, each of the algorithms plays precisely the role the identity $a^2 - b^2 = (a + b) * (a - b)$ plays in this math problem. What my solver does is that it first looks at the current arrangement of this cube at every stage and using a few conditional statements, it determines which algorithm is to be used (This is equivalent to us looking at the mathematical problem given in the example, recognizing that it is of the form $a^2 - b^2$ and thus concluding that the identity $a^2 - b^2 = (a + b) * (a - b)$ is to be used). Now just like how we, without thinking anything, directly rearrange the given expression to the form $(25 + 5) * (25 - 5)$, the solver also simply rearranges the cube pieces into the final arrangement that would have resulted had it actually followed each of the steps of the algorithm. It does not really execute that algorithm itself but simply adds the steps of that algorithm to the final solution.

In fact, the main reason why I employed the CFOP method in my solver is because of CFOP's heavy reliance on algorithms, pattern recognition, and muscle memory, as opposed to more intuitive methods such as the Roux or Petrus methods. Therefore, it is also heavily used and relied upon by many speedcubers, including Max Park and Feliks Zemdegs (src: en.wikipedia.org).

7. CONCLUSIONS

The Rubik's cube solver is just one of the several applications that the Minimal Thinking technique can potentially have in modern Artificial Intelligence. This technique can be used to ease the problem-solving part of every program by training the program to use results that have already been established by both human or artificial intelligence and directly jump to the solution without solving the entire problem on its own. The result is a boost in the performance of the program both in terms of memory and speed.

I have thus not only developed the fastest Rubik's cube solving method to date but also a new idea that in my opinion can revolutionize the field of AI to a great extent. Although the "human method" of solving the Rubik's cube comes with the disadvantage of not always providing the optimal solution, the 350x boost in program efficiency effectively outweighs any such potential disadvantages in the long run. With modern robots that can execute each move in as less as 10 milliseconds, executing a handful of extra moves is an almost negligible task. In most real-life problems, an optimal solution may not always prove to be the best solution, especially when it is generated at the cost of an extremely slow and excessively memory-consuming program. Moreover, bringing down this move time to 5 milliseconds is a more achievable task in near future than developing faster and better search algorithms and reprogramming the other existing solvers to execute them. Regardless of how modern AI has advanced to a point where robots tend to have the potential to "think" on their own, even a small human interaction in a completely autonomous program can make a huge difference in the efficiency of the program, as made evident in this example of a Rubik's cube solver.

Another significant takeaway from this particular solver is the fact that such a solver for a puzzle of a particular order can be built by adding on to the solvers of lower orders of the same category of puzzles, without having to start from scratch. This is precisely how I had built my solver for the 1x2x3 Rubik's puzzle from my 1x2x2 puzzle solver by adding a few extra lines of code that correctly orient the middle layer pieces which are absent from a 1x2x2 puzzle.

REFERENCES

[1] R. Gonzalez, "How to Solve a Rubik's Cube in 5 Seconds—or Less," Wired (www.wired.com), 24 May 2019.

[2] R. Korf, "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases," AAAI-97 Proceedings, American Association for Artificial Intelligence (www.aaai.org), 1997, pp. 700-705

[3] T. Rokicki, "Twenty-Five Moves Suffice for Rubik's Cube," Symbolic Computation, Computer Science, Cornell University, arXiv:0803.3435v1 [cs.SC], 24 Mar. 2008, pp. 3-5.