Different Algorithms to Solve a Rubik's Cube

Different Algorithms to Solve a Rubik's Cube

Juntao Chen

Shanghai United International School (Gubei Campus), Shanghai, 201102

tony_cjt@163.com

Abstract. This essay introduces and compares four methods to solve a Rubik's cube which are Korf's algorithm, Thistlethwaite's algorithm, Kociemba's algorithm and machine learning method. Rubik's cube is a rather complicated problem to solve, investigation on the algorithms to solve a Rubik's cube can provide some valuable enlightenment which can be applied to other complex areas. Through comparing the efficiency of the four methods, this essay indicates that Korf's algorithm is the most efficient algorithm to solve this problem, while Kociemba's algorithm is the second. On the other hand, machine learning method will be the future prevalent algorithm to be investigated. Although it appears to be less efficient in solving this problem, it has more potential than the other three algorithms.

Keywords: Rubik's cube, Korf's algorithm, Thistlethwaite's algorithm, Kociemba's algorithm, machine learning.

1. Introduction

Solving a Rubik's cube has been a really complicated problem, and people have been finding solutions that are fast and efficient to solve this problem. Computers are also used to solve the cube using various algorithms, including Korf's algorithm, Thistlethwaite's algorithm, Kociemba's algorithm and machine learning method. These are the prevalent algorithms to solve this problem.

In this paper, the author will discuss those algorithms and their approach to solve the cube.

Rubik's cube is very complex and has substantial number of possibilities, which makes it extra sophisticate to solve. By applying algorithms to deal with this problem, the potential of using computers and algorithms to deal with problems with this level of complexity can be seen. Therefore, the author can see that it has the potential to solve problems in other areas that are significant to our society.

2. Introduction on the Rubik's cube

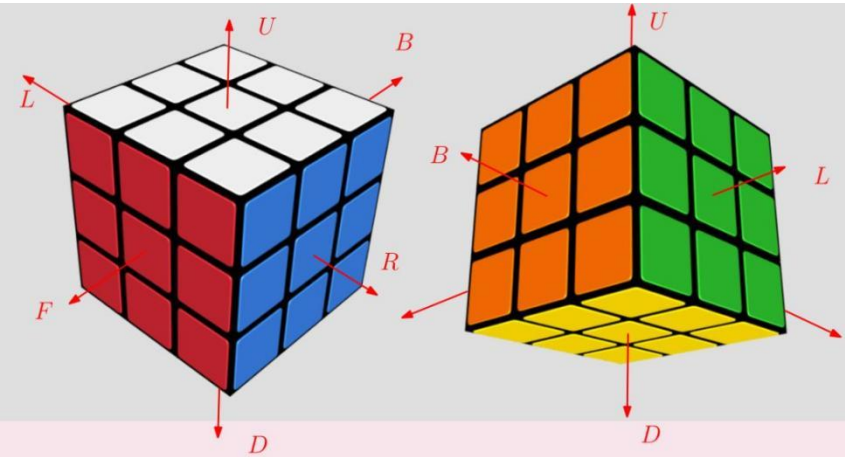

Figure 1. A picture of a 3×3 Rubik's cube from blog.csdn.net. (n.d.) [9].

As shown in Figure 1, it is obvious that the Rubik's cube has 6 faces with different colors. Now, each face can be represented with a letter, which is the following.

F-front (red color)
U-up (white color)
R-right (blue color)
B-back (orange color)
D-down (yellow color)
L-left (green color)

For a single face, there are 9 facelets. The blocks with one color are known as the central blocks, and the blocks with two colors are known as the edge blocks, while the blocks with three colors are known as the corner blocks. Therefore, one single cube has 6 central blocks, 8 corner blocks and 12 edge blocks. In total, one cube consists of 26 blocks. When all the facelets on one face are in the same color, then the cube is called in its solved state, or else the cube is called in its scrambled state. The objective is simply to apply several moves to turn the cube in its scrambled state to its solved state. A possible move to a Rubik's cube is to rotate one face from the six faces. the letters of the faces can be used to represent the moves. Upper case represents rotating 90° clockwise, while lower case represents rotating 90° anticlockwise. For example, F means rotating the front face 90° clockwise and r means rotating the right face 90° anticlockwise.

Next, the author will introduce several algorithms.

3. Korf's algorithm

3.1. Korf's algorithm introduction

Korf's algorithm is using brute search and heuristic search to direct the search into the correct direction, and eventually find a way to the solved state. The main algorithm used to achieve it is using an Iterative-Deepening Depth-First Search (IDDFS) with A^* algorithm (IDA*) [1] along with a huge database to avoid the wrong direction.

The iterative-deepening depth-first search is an algorithm that operates on a tree. Similar to the famous brute search method, which is the Breadth-First Search, iterative-deepening depth-first search starts from the root node, and search through the tree until reaching to the goal node. However, compared with the Depth-First Search, Iterative-Deepening Depth-First Search can operate on a large tree. Considering that the Rubik's cube has a substantial number of states, the tree will be enormous, so this method will be more appropriate.

When applying this method to the tree, it can expand the current node to find the solution. First, the root node is the scrambled state of the cube. Then, applying the set of moves to this state, it will produce several leaf nodes. Afterwards, repeating this process to all the nodes until the solution is obtained. When expanding the nodes, some moves can be simplified and pruned. For example, Bb can be pruned as a move that will cancel its opposite move. There are also situations where the two moves are commutative. For example, RL is the same as LR, but FL is different with LF. Obviously, when the two moves will not interfere with each other (i.e., moves on opposite faces), these moves are commutative and can be pruned.

After applying the algorithm, the state space is still very large. Consequently, A^* algorithm is applied. A^* algorithm is also a search algorithm. It is often used in finding the shortest path from one point to another. This algorithm can estimate the distance and optimizing it through the search in order to achieve its goal. When it is applied to the tree, it will find the distance from the current node to its goal node. The author can simply employ a maximum search distance to it, and when combined with Iterative-Deepening Depth-First Search, it will prune those nodes that exceeds the search distance. This way, the algorithm can find a path that is closer to the solution, rather than a wrong one.

In order to achieve this goal, the heuristic function must be found and applied to the algorithm. There is a way to do it which is to create a database for every state that can estimate the optimum moves. Taking the corner blocks for an example, the database will decide the least number of moves to solve all the corner blocks (i.e., moves required so that all corner blocks will return to their initial position).

Rubik's cube is an educational toy but appeared to be very complex. It also has lots of different forms. Here, only the classic Rubik's cube will be introduced, which is the 3×3 Rubik's cube.

3.2. Building the database

In Korf's algorithm, Korf suggested three databases: 8 corner blocks, 6 out of 12 edge blocks, and the other 6 edge blocks. For the 8 corner blocks, the blocks will have 8 position they can take, so the permutations will be 8!. Then, each corner block has three states, because there are three facelets. According to the law of the Rubik's cube, when the state of 7 blocks are determined, the last will also be determined. Therefore, the total permutations of the corner blocks P_c will be

$$P_c = 8! \times 3^7 = 88179840$$

The magnitude of P_c is capable for computers to iterate. Similarly, for the 6 edge blocks, the blocks will have 12 positions they can take, so the permutations will be P_6^{12}. Then, each edge block has two states. So, the permutations of the edge blocks P_e will be

$$P_e = P_6^{12} \times 2^6 = 42577920$$

Therefore, the databases can be used to reduce the permutations, hence increasing the efficiency.

Now, by using the three databases, the least moves required for each section of the cube can be gained. Assuming the moves to solve all corner blocks is denoted as m_c, the moves to solve all edge blocks are denoted as m_{e_1} and m_{e_2}, then the lower bound of the moves to solve the whole cube will be

$$m = \max(m_c, m_{e_1}, m_{e_2})$$

4. Thistlethwaite's algorithm

4.1. Thistlethwaite's algorithm introduction

Thistlethwaite's algorithm uses group theory in mathematics along with Iterative-Deepening Depth-First Search (IDDFS) which is the same algorithm applied in Korf's algorithm. Unlike Korf's algorithm, Thistlethwaite's algorithm uses decision tree rather than database. The main idea of Thistlethwaite's algorithm is to restrict the moves in several stages in order to reduce the state space, or in other words, to prune the branches that is undesirable.

There are five groups of a cube [2], which are G_0, G_1, G_2, G_3, G_4. G_0 is the initial scrambles state and G_4 is the solved state. Now, the groups are defined to be

$$G_0 = \langle L, R, F, B, U, D \rangle$$

$$G_1 = \langle L, R, F, B, U^2, D^2 \rangle$$

$$G_2 = \langle L, R, F^2, B^2, U^2, D^2 \rangle$$

$$G_3 = \langle L^2, R^2, F^2, B^2, U^2, D^2 \rangle$$

$$G_4 = \{I\}$$

The letters for each group represent the moves that can operate in that specific group. For example, in G_0, which is the scrambles state, any operations can be taken to reach to G_1. However, in G_1, the up and down face can only move in half turns. Then, by applying the moves in each group, the cube can reach to the desired state of the next group, thus moving from G_i to G_{i+1} where $i \in \{0,1,2,3\}$. When the cube is in G_4, it means that it is solved.

It is obvious that for every transition from G_i to G_{i+1}, the branching of the tree will reduce by a factor of 4. This is because during every transition, there are two moves be restricted from quarter turns. For example, from G_0 to G_1, the two moves U and D can only make half turns. Therefore, the moves U, D, u, d are prohibited in this transition. Correspondingly, the branches associate with these moves will be pruned.

4.2. Desired state in each group

Getting from G_0 to G_1:

In this stage, only the corner blocks are dealt with. First a bad corner block is defined. Bad corner block refers to a corner block if quarter moves of U and D are required to take them back into their initial position. Then, by using the moves in G_0, all bad corner blocks are turned into good ones.

Getting from G_1 to G_2:

In G_2, all corner blocks are in their correct orientations. In other words, if the cube has its up face white, then the facelet of all corner blocks will be white or yellow on the up and bottom faces. Moreover, the edge blocks located on FU, FD, BU, BD should also return to their initial position.

Getting from G_2 to G_3:

In G_3, all corner blocks can be moved to their initial positions using only half turns. At the same time, all edge blocks need to be sorted so they are in their initial positions with correct orientations.

Getting from G_3 to G_4:

Now, the solved state of the cube can be reached by only applying the half turns.

5. Kociemba's algorithm

Kociemba's algorithm is developed from Thistlethwaite's algorithm and made a few optimizations. Kociemba's algorithm is also called the two-phase algorithm, it reduces the five groups in Thistlethwaite's algorithm to only three groups that are one initial group G_0, one intermediate group G_1, and one solved group G_2.

The three groups are

$$G_0 = \langle U, D, L, R, F, B \rangle$$

$$G_1 = \langle U, D, L^2, R^2, F^2, B^2 \rangle$$

$$G_2 = \{I\}$$

Kociemba's algorithm uses the similar approach as Korf's algorithm, which uses Iterative-Deepening Depth-First Search (IDDFS) with A* algorithm (IDA*). As mentioned earlier, this algorithm requires a limitation of steps in order for the A* algorithm to work. In this case, when the

cube is in G_0 state, the state space is huge, where the cube can reach to a state space of 18^{12}. However, Kociemba proposed a limited number of moves for the cube to reach from G_0 to G_1 that is at most 12 moves. Similarly, for the cube to reach from G_1 to G_2, it needs at most 18 moves. This then can be applied in the A* algorithm which is capable to solve using a normal computer.

6. Machine learning method

6.1. Machine learning method introduction

In this method, it does not require any pre-knowledge of mathematics and complicated searching algorithms. Machine learning method uses reinforcement learning to optimize the solution of a cube. It can divide into two main steps. The first step is to build the neuron network and train it so that it can make an optimize decision so that the cube can reach to a state closer to the solved state. Then, the model can simply be applied with a searching algorithm known as the Monte-Carlo Tree Search (MCTS) combined with Breadth-First Search (BFS) to find the final solution.

6.2. Building the training set

By using machine learning, a large amount of data is needed to train the artificial intelligence. As mentioned earlier, each possible move on the Rubik's cube using letters can be represented. Therefore, the author can generate a limited length of random string containing elements from the set of moves M, where

$$M = \{F, U, R, B, D, L, f, u, r, b, d, l\} \tag{1}$$

Now, the author generate a large number of scrambled cubes.

6.3. Training

When building the neuron network, two parameters will be included, which are

p-policy: probability distribution of a certain move.

v-value: a scalar used to assess the state.

By using these two parameters, the policy can be used to tell which move should be applied to the current state, whereas the value is used to assess whether the state is good or not. Now, the author will apply auto-didactic iterations (ADI) [3] to train the neuron network. To start with, applying a string with length N to the cube, and using the method discussed earlier to build a training set. Then, for every state in the training set, first applying every move in the set M to the state s. Second, transfering the 12 states to the neuron network to get the values of the 12 substates of s. Then, calculating the value v_i where $i \in [1, N] \cap Z$ using the following formula for the current state.

$$\max_a(R(S(x_i, m)) + v_{x_i}(m)) \text{ where } m \in M \tag{2}$$

In formula (2), $S(s, m)$ is the state after applying the move m to the state s. $R(s)$ will be 1 if s is the aiming state, or it will be 0.

Next, calculate the policy p_i where $i \in [1, N] \cap Z$ using the following formula for the current state.

$$\operatorname{argmax}_a(R(S(x_i, m)) + v_{x_i}(m)) \text{ where } m \in M \tag{3}$$

The only modification in formula (3) is changing the max function into argmax function. Therefore, now p will be 1 only if it is at the position where maximum of the substate is reached, otherwise p will be 0.

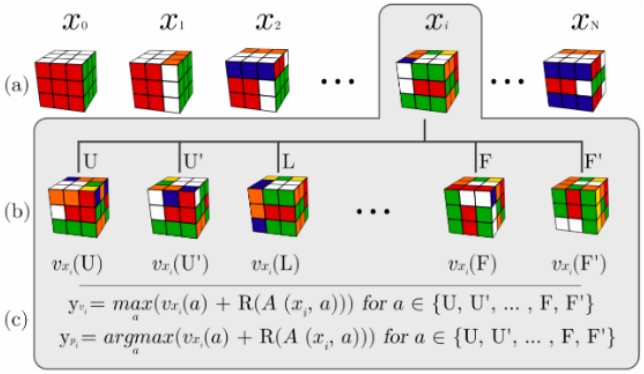

Figure 2. Training process from Lapan, M. (2019) [3].

This whole process can be referred to Figure 2.

6.4. Applying the model

From the previous training, the following method can be applied to solve the cube. First, the model is given a cube in its initial scrambled state. Then, by applying all the moves from M, the author can obtain the first mutated population. Now, by using the trained model, it can estimate the best move using policy p and value v. When the cube is applied with that move, it will get closer to the solved state. Then, repeating the steps until the solution is found.

However, based on the complexity of the Rubik's cube, this method is not working very well. Therefore, other methods should be employed. The method used in the original essay is the Upper Confidence bound applied to trees (UCT), which belongs to Monte-Carlo Tree Search (MCTS).

The method is an operation on trees, where the nodes are the current states of the Rubik's cube, and the edges are the moves. This method is very similar to the famous brute search methods which are Depth-First Search (DFS) and Breadth-First Search (BFS). These two brute search methods tend to go over the whole tree of the Rubik's cube, but based on the magnitude of this tree, it appeared to be less efficient. Consequently, a similar method which is the Monte-Carlo Tree Search is used. It only considered a portion of the tree, which can reduce the search space to increase the efficiency.

The node at the start is the current state, which is also the original scrambled state of the Rubik's cube. Then, for every node, there are two operations. If the node is a leaf node, it will be expanded by applying every possible move to this state. On the other hand, if the node is not a leaf node, or in other words in the middle of the tree, then a pathway should be chosen based on the value v and the policy p of the current node. According to the original essay, the action chosen for a non-leaf node is decided through the following formula.

$$A_t = \underset{a}{\operatorname{argmax}}\left(U_{s_t}(a) + W_{s_t}(a)\right), \text{where } U_{s_t}(a) = cP_{s_t}(a)\frac{\sqrt{\sum_{a'} N_{s_t}(a')}}{1+N_{s_t}(a)} \quad (4)$$

In formula (4), $N(s,a)$ is the total number of times when action a has been chosen in the state s. $P(s,a)$ is the policy of the state s, and $W(s,a)$ is the maximum value among all the states s under the action a.

Then, after applying the Monte-Carlo Tree Search and reaching to the solved state, Breadth-First Search is used to find the solution. Breadth-First Search will find the shortest pathway among all the pathways created by the Monte-Carlo Tree Search, to optimize the final solution and increase the efficiency.

7. Comparison of the efficiency

First Korf's algorithm, Thistlethwaite's algorithm, and Kociemba's algorithm are compared. The following tables correspond to each algorithm's performance when solving the cube [4].

Table 1. Number of moves used by each algorithm.

	Korf's algorithm	**Thistlethwaite's algorithm**	**Kociemba's algorithm**
Test 1	20	45	29
Test 2	20	40	25
Test 3	20	40	29

Table 2. Time taken for each algorithm.

	Korf's algorithm	**Thistlethwaite's algorithm**	**Kociemba's algorithm**
Test 1	2:05	3:56	2:35
Test 2	2:05	3:45	2:26
Test 3	2:05	3:43	2:35

From Table 7.1 and Table 7.2, it is obvious that Korf' s algorithm is the most efficient algorithm in solving the cube. The next efficient algorithm is Kociemba' s algorithm, while Thistlethwaite' s algorithm is the least efficient algorithm.

From the comparison of machine learning method and Kociemba' s algorithm [3], the conclusion is that machine learning method takes significantly longer time than Kociemba' s algorithm. However, in 55% of the cases, machine learning method finds better solution than Kociemba' s algorithm. Though it cannot prove that machine learning method is better than Kociemba' s algorithm.

8. Conclusion

Korf's algorithm uses Iterative-Deepening Depth-First Search (IDDFS) with A* algorithm (IDA*) along with a database to perform heuristic search on a tree.

Thistlethwaite's algorithm and Kociemba's algorithm both uses group theory in mathematics along with Iterative-Deepening Depth-First Search (IDDFS) and A* algorithm (IDA*) to facilitate the search. Moreover, Kociemba's algorithm is an improvement to the Thistlethwaite's algorithm where it reduces the number of groups and increases the efficiency.

The method of machine learning is undeniably easier than the mathematical method and the algorithm method. However, this method still needs to be optimized to completely solve this problem.

Overall, Korf's algorithm is the most efficient algorithm in solving the Rubik's cube, the next one is Kociemba's algorithm. On the other hand, machine learning method is less efficient but it can come up with solutions that are more optimized.

References

[1] Botto, B. (2021). Implementing an Optimal Rubik's Cube Solver using Korf's Algorithm. [online] Medium. Available at: https://medium.com/@benjamin.botto/implementing-an-optimal-rubiks-cube-solver-using-korf-s-algorithm-bf750b332cf9.

[2] www.jaapsch.net. (n.d.). Thistlethwaite's 52-move algorithm. [online] Available at: https://www.jaapsch.net/puzzles/thistle.htm.

[3] Lapan, M. (2019). Reinforcement Learning to solve Rubik's cube (and other complex problems!). [online] Medium. Available at: https://medium.datadriveninvestor.com/reinforcement-learning-to-solve-rubiks-cube-and-other-complex-problems-106424cf26ff [Accessed 18 May 2022].

[4] Marcellienus, J. (n.d.). The most efficient algorithm to solve a Rubik's cube Science Research

project. [online] Available at: http://www.youngscientist.com.au/wp-content/uploads/2015/02/Physics-10-12-Justin-Marcellienus-report.pdf [Accessed 21 May 2022].

[5] De, Moloy. (2015). Group Theory to Solve Rubik Cube. [online] Available at: (PDF) Group Theory to Solve Rubik Cube (researchgate.net)

[6] ruwix.com. (n.d.). Herbert Kociemba's optimal Rubik's Cube solver - Cube Explorer. [online] Available at: https://ruwix.com/the-rubiks-cube/herbert-kociemba-optimal-cube-solver-cube-explorer/ [Accessed 12 Jun. 2022].

[7] Speedsolving.com. (2020). Kociemba's Algorithm - Speedsolving.com Wiki. [online] Available at: https://www.speedsolving.com/wiki/index.php/Kociemba%27s_Algorithm.

[8] Wikipedia Contributors (2019). Optimal solutions for Rubik's Cube. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube.

[9] blog.csdn.net. (n.d.). Rubik's cube and group theory_The blog from algorithm and math-CSDN Blog. [online] Available at: https://blog.csdn.net/FnqTyr45/article/details/81256728 [Accessed 18 May 2022].

[10] blog.csdn.net. (n.d.). New method to solve Rubik's cube with reinforcement learning! The research team from The University of Nottingham proposed fitness function_The blog from the home of science-CSDN Blog. [online] Available at: https://blog.csdn.net/dqcfkyqdxym3f8rb0/article/details/114323997.