Rubik’s Cube Solver using Thistlethwaite’s Algorithm

S Girish, 19BCE1268

Vellore Institute of Technology, Chennai.

*Abstract****:*— A mechanical puzzle with enormous popularity, the Rubik's Cube has garnered interest due to its distinctive qualities on a global scale. The Rubik's Cube was utilised by numerous academics for scientific study and technological development since it is a well-known and traditional brain-training toy. A 3-D mechanical puzzle called a Rubik's cube has nine stickers in a variety of six solid colours covering each of its six faces. Each face of the puzzle needs to be a solid colour in order to be solved. This essay offers a fundamental knowledge of the Rubik's Cube, group theory's role in solving the puzzle, how to apply Thistlethwaite's algorithm, and how the algorithm works in general. This paper's conclusion and findings are discussed after applying a modified Thistlethwaite's algorithm.**

Keywords — **Rubik’s cube, Thistlethwaite's algorithm, Group theory, Iterative Deepening Depth-First Search (IDDFS).**

1. **INTRODUCTION**

Ernő Rubik, a Hungarian artist and architecture professor, developed the Rubik's cube, a 3D mechanical puzzle. The puzzle, formerly known as the "Magic Cube," was licenced by Rubik and sold by Ideal Toy Corp. in 1980. As a result, more than 350 million Rubik's cubes have been sold globally, making it the most popular, bestselling, and well-known puzzle with a deep underlying logical, mathematical structure.

Each of the six faces of a typical Rubik's Cube is covered by nine stickers, each of which is one of six solid colours (traditionally white, red, blue, orange, green, and yellow). A pivot mechanism allows each face to revolve independently, causing the colours to mingle. To complete the problem, each face must be restored to its original colour. A normal Rubik's cube is 5.7 cm (about 214) on each side.

The puzzle comprises of twenty-six distinct small cubes, also called "cubies" or "cubelets". Each of them has a hidden inside extension that interlocks with the other cubes while allowing them to move to different places. The centre cube of each of the six faces, however, is only a single square façade; all six are attached to the core mechanism. These serve as the framework for the other components to slot into and revolve around. So there are twenty-one components: a single core piece made of three intersecting axes that holds the six centre squares in place while allowing them to spin, and twenty smaller plastic pieces that slot into it to make the finished puzzle.

To solve a Rubik's cube from a scrambled state (Fig. 1) to a desired state (Fig. 2), an algorithm is required, which is a sequence of movements to get closer to solving the problem. For a normal human being, such sequences frequently need more than fifty to hundred movements.

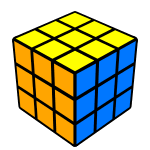
 

Fig.1. Scrambled state Fig.2. Goal state

The most fundamental approach for a computer to represent a Rubik's cube is through the use of graph theory principles, where the Rubik's cube may be represented as a graph. Depth first search and breath first search algorithms can be used to identify a series of edges (cube twists) to solve the cube by attaining a suitable configuration. This is known as a brute force search. The term brute force search is regularly used in computer graph theory and refers to a strategy that has been widely employed to solve issues in areas like as artificial intelligence.

The issue with brute force search is its slowness. On huge issues, algorithms like depth first search or breadth first search are impracticable. The brute force search is only possible for small issue cases and has several major drawbacks that Thistlethwaite's approach overcomes. As a result, instead of the depth first search, this thesis will investigate Thistlethwaite's algorithm.

The major goal of this study is to demonstrate the working of Thistlethwaite’s algorithm, and the explaination of each transition from one group to another using group theory. Along with it, the paper also illustrates an example of solving a random cube state to the solved state using images to understand the transition from one group to the next. The following is a breakdown of the paper's structure:

Section II provides a useful background on key definitions.

Section III gives a detailed literature survey on the area of study.

Section IV brief about the problem statement of the project.

Section V elaborate the methods related to the proposed project.

Section VI delves into the proposed algorithm technique and its implications.

Section VII presents the result of the experiments with an example and

Section VIII concludes with some limitations, final thoughts and future plans

**2.  BACKGROUND  INFORMATION**

This background consists of three different parts. The first part covers the basic terminologies, followed by Rubik’s cube, and finally graph theory.

* 1. Terminology

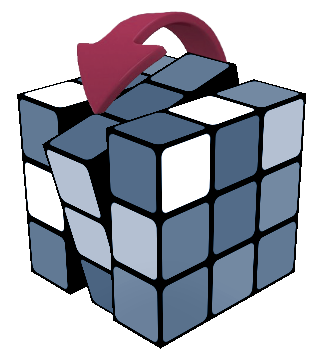
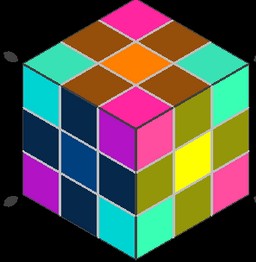
1. Facelet/Cubie: The Rubik’s cube is made of twenty subcubes or cubies that is attached to the centre.
2. Middle layer: Consists of one stage, and is the layer between L and R.
3. Parity of edges and corners: The overall parity must be even according to the laws of the cube.
4. Layer: Contains one side of the cube.
5. Tetrad: is made up of four corners in an orbit.
6. Cost bound: cost of the current iteration.
7. Permutations: Ordering of faces of Rubik’s cube.

Fig. 3. Middle layer Fig.4. Pink shadows

representing a tetrad

* 1. The Rubik’s cube
     1. Cube Mechanics

The regular form 3x3x3 is made up of two unique parts: the centre and the surrounding cubes. The core can be depicted as a central cube with six connected octagons, each of which can spin 90, 180, or 270 degrees in each direction. The exterior cube is coupled to the core and is divided into three types: slides, edges, and corners. There are six side pieces in total; each side piece is connected to one of the octagons of the core and cannot be moved. The Rubik's cube contains twelve edge pieces and eight corners. Each edge piece has two visible faces, while each corner piece has three visible faces.

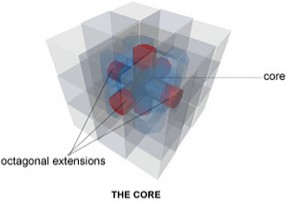


Fig.5. Inside of Rubik’s Cube

Although each particular edge or corner piece can be rotated in any edge location, some permutation sets are not possible. One face of a Rubik's cube is made up of one side piece, four corners, and four edges, with each face identified with a solid colour. Finally, these twenty-six parts combine to make a Rubik's cube (fifty-four coloured stickers) that can spin in all three dimensions. The goal is to return the cube to its original form, where all of the faces on each side are the same colour. Every Cube solver employs an algorithm, which is a series of movements designed to bring the puzzle closer to completion.

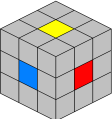
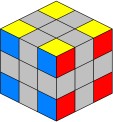
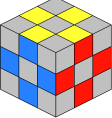


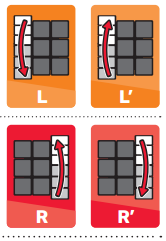
Fig.6. Centre Fig.7. Corner Fig.8. Edge

Pieces Pieces Pieces

Each of the eight corners has 8! permutations and can be placed in three different ways. This indicates that any combination of the corner pieces has 38 possible outcomes. The twelve edges may be combined in twelve different ways, with two alternative orientations for each component. As a result, there are 212 configurations for each permutation of the edge pieces, for a total of 8! x 38 x 12! x 212 states. However, because these permutations are divided into twelve classes, only transformations between states within a single class are conceivable. The total number of states accessible from one state is 8! x 38 x 12! x 212 = 43,252,003,274,489,856,000 (43 quintillion 252 quadrillion 3 trillion 274 billion 489 million 856 thousand!).

* + 1. Definition

By using David Singmaster's notation to refer to the Rubik's cube, which names its six faces as follows: right (R), left (L), up (U), down (D), front (F), and rear (B). Each face may be turned in two distinct directions: the first is a 90-degree clockwise turn, and the second is the opposite, or counter-clockwise, turn.



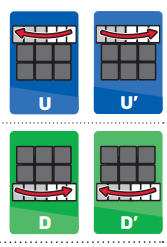


Fig.9. U and D Fig.10. L and R

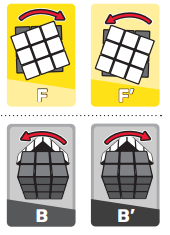


Fig.11. F and B



Fig.12. Double Turn

* + 1. Orientation

An edge cubie might be oriented well (good) or poorly (bad). If a component can be returned to its original location in the correct orientation without twisting the UP or Down faces of the Rubik's cube, it is regarded as having a good orientation. The Rubik's cube's Up and Down turns will cause the Upper or Down cubies' edge orientation to be reversed, while the L or R turn will keep the Left and Right pieces' corner orientation in place. Edges can only have one of two potential orientations, which are denoted by the numbers 1 (good orientation) or 0 (bad orientation).

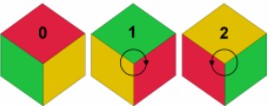


Fig.13. Corner orientation resulting in three

orientation: 0(corrected oriented),

1(clockwise) and 2(counterclockwise).

Since there are three alternative orientations for the corner cubies of the Rubik's cube, things are more difficult. When a corner cubie is correctly orientated, it will be shown by the number 0, and when it is rotated clockwise or anticlockwise, it will be indicated by the numbers 1 or 2, respectively. Be aware that there is still another method for determining each cubie's orientation. 0 can be used to indicate if a corner cubie has a L or R facelet that belongs to the L or R face. If not, the orientation will be decided by the figure. (Fig.13.)

This term will also become more clear when the Thistlethwaite's algorithm is presented in greater detail in subsequent sections.

* 1. Graph Theory to Solve the Rubik’s Cube

In the discipline of computer science, graph theory is extensively investigated, used, and utilised to examine numerous applications. Concepts from graph theory may effectively express and address challenging issues like the Rubik's cube. This theory's main contribution to computer science is the creation of graph algorithms, which may be used to a variety of graph-based issues. The vertices of a Rubik's cube may be thought of as a collection of all conceivable configurations, and the edges can be thought of as the connections between configurations that are one twist off from one another.

It is common knowledge that the shortest path between any two or more graph vertices may be found. Last but not least, this approach can be used successfully to puzzles like figuring out how to solve a Rubik's Cube.

When used to discover a sequence of edges (cube twists) that would solve the puzzle by reaching into a suitable configuration, breadth first search and depth first search are fairly simple to program either repeatedly or recursively.

* + 1. Depth first search algorithm

One of the approaches for searching in tree or graph data structures is depth first search. It is a method that has been extensively used to solve issues in artificial intelligence. This approach has also received widespread recognition as a potent way for traversing mazes and a variety of graph problems, although its qualities have not yet undergone in-depth research.

Consider receiving a graph. The method moves from one vertex of the graph to the next along the edges until a depth cutoff is achieved. The exploration stops when it has travelled as far as it can (run of edges), at which time it turns around and moves on to the most recent node to be extended. Finally, the algorithm will travel along each of G's edges precisely once.

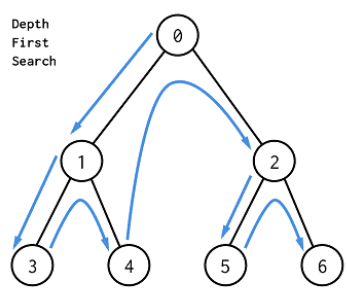


Fig.14. Depth First Search

(0->1->3->4->2->5->6)

There are only a finite number of moves for a given cube layout. When trying to solve a Rubik's cube, it is important to keep in mind that the state space is quite large. As a result, the algorithm can pick an edge that leads to a subgraph that cannot fully expand and does not include a solution state. This causes the algorithm to produce a solution that will never be found since the code may spin the same plane back and forth alternately, making it impossible for the program to advance further.

The fact that depth first techniques cannot identify duplicate nodes means that in a network with numerous pathways leading to a single state, any depth first search may produce significantly more nodes than there are states. The total number of producing nodes may thus be higher than the total number of nodes produced by other methods.

* + 1. Breadth first search Algorithm

Another algorithm for navigating and exploring a tree or a graph is called breadth-first search. It begins from the root of the tree, investigates the nearby nodes first, then proceeds on to the following state until it reaches the desired state. This technique works by utilising a queue to search the search tree one level at a time (First in first out). Since breadth-first search always investigates and extends all of the nodes at a particular depth before extending any nodes at a larger depth, the first solution path discovered by this method will always be the shortest path.

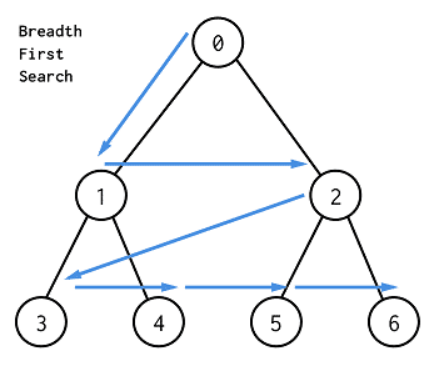
The depth first search algorithm, O(bd), relates the time complexity of this method to the number of nodes produced. The complexity of space is similarly O(bd).

Fig.15. Breadth First Search

(0->1->2->3->4->5->6)

However, there are many problems with this approach:

• Exponentially increasing size - With each increasing depth level of our search tree, the branching factor of 18 means that the number of solutions we are required to look at will increase by a factor of 18. This will explode very quickly and become infeasible to search within a reasonable time.

• Exponential increase in memory consumption - A non-recursive implementation of a breadth first search requires a queue to store the child nodes to be explored. In the worst case, a 20 move solution would require searching 1820 nodes. It is clearly not feasible to maintain a queue of this size. To show the extent of how the number of nodes increase with depth up to 10:

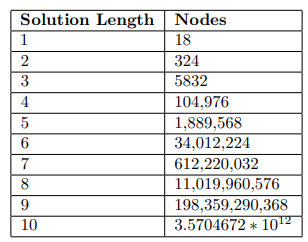


Fig. 16. Nodes generated at each depth

1. **LITERATURE SURVEY**

Ernő Rubik, a Hungarian artist and architecture professor, develop The world has taken notice of the widely used mechanical puzzle known as the Rubik's Cube because of its distinctive qualities. [1] offers a fundamental knowledge of the Rubik's Cube and demonstrates its mechanical art from the viewpoints of its history and evolution, traits, state of study, and, in particular, mechanical engineering design, while also forming an idea for its potential use in mechanisms. The creation and history of the Rubik's Cube are first discussed, and then the unique qualities of the cube itself are examined. The current research on the Rubik's Cube is then evaluated in numerous fields both domestically and internationally. This includes study on scientific metaphors, reduction algorithms, defining applications, and mechanism-related difficulties. The possibilities and uses of the Rubik's Cube in the realm of mechanisms are then examined.

[2] deals with the creation of mechanical structures, the use of image processing, and the creation of algorithms, all of which may be solved by a robot. The most significant function of all is played by image processing, making precision of this process crucial. Because of its straightforward yet accurate range for many hues, the HSV colour space was chosen. Each colour could be clearly distinguished and so identifiable after the colour range was established. The first stage is simply to identify the colours, and the second is to create an algorithm. The appropriate algorithm is produced by referring to a preprogrammed set of potential outcomes and the related replies. Thus, these algorithms provide signals that drive the relevant actuators on the cube faces that require rotation. The right algorithm needs a solid mechanical design to support it, thus a cage-like mechanism was created to make it simple to load and unload the cube. The following five categories have been used to group the solution. Here, the LBL (Layer by Layer) approach is applied. However, this technique is pointless because it requires more steps and time to achieve the desired state.

Heuristic approaches can also be used for solving the Rubik's cube. [3] created two metaheuristic strategies based on GA and SA to solve the Rubik's cube. Using identical random inputs, the suggested methods are compared after being implemented in the Matlab 2009a programme. Although it takes more time to compute to obtain such solutions than SA, the computational results demonstrate that the GA-based technique leads to solutions with fewer moves than SA. When time is taken into consideration, the SA-based strategy solves the cube more quickly than the GA-based approach, but it also gives more motions. There are various well-known traditional methods for solving the cube, and they typically finish the cube layer by layer. Particularly for big size NNN (extended variants) of the Rubik's cube, these algorithms are challenging to learn and retain the order of the movements. Therefore, suggesting heuristic techniques has piqued the interest of academics and mathematicians, however sadly few of them are recorded and published in academic journals.

In order to approach solutions to combinatorial optimization issues like the prediction of protein tertiary structures, the autodidactic iteration technique [4] was created. The function of proteins in an organism can be better understood by the prediction of their tertiary structures in proteins. The protein tertiary structures are represented by the Rubik's cube in the combinatorial optimization puzzle. In contexts with broad state spaces and little rewards, such combinatorial puzzles like the Rubik's cube, the autodidactic iteration process is utilised. The autodidactic iteration method is used in this research to solve a traditional Rubik's cube. It is a reinforcement learning algorithm that can teach itself to solve the Rubik's cube without the aid of a person. Then, using Keras as the deep learning library, a neural network model is trained to unravel a puzzled Rubik's cube. The Rubik's cube is produced using Magic Cube via a graphical user interface, or GUI. By rotating the cube's faces and altering the cube's viewing angle, the Rubik's cube created by the GUI may be interacted with. The neural network model developed based on the model developed to solve a scrambled Rubik's cube is then used in the interactive Rubik's cube. However, training and testing will need a significant amount of time.

This study [5] introduced a brand-new picture encryption technique based on the Rubik's cube's basic idea. The Rubik's cube principle, which simply alters the location of the pixels, is first used to jumble the pixels of the original grayscale image. The bitwise XOR is implemented into the odd rows and columns using two secret, completely random keys. Then, using the secret keys that have been flipped, the bitwise XOR is also performed to even rows and columns. While the number of iterations is not achieved, these actions can be repeated. The suggested encryption technique has been put to the test using numerical simulation to determine its viability and security.

[6] shown that by reducing from the Hamiltonian Cycle issue in square grid graphs, it is NP-complete to solve a n n n Rubik's Cube in the best possible way. This advances the earlier finding that it is NP-complete to solve a n x n x n Rubik's Cube with missing stickers. Additionally, they initially demonstrated this conclusion for the Rubik's Square n n 1 generalisation of the Rubik's Cube case before moving on to a more challenging demonstration for the Rubik's Cube case. Their findings hold true whether the sides are intended to be monochrome or each sticker is intended to go in a certain spot. The proof of the difficult Rubik's cube is also made using the evidence of the simpler case of n x n x 1.

[7] concentrated on using a genetic algorithm to solve this question in a polynomial amount of time. A good and reliable solution to any issue, even NPHard ones, is found via a genetic algorithm. Common algorithmic issues like the Knapsack problem and the Traveling Salesman problem can also be resolved, which is a minimum spanning tree issue. The Rubik's Cube will be solved using a genetic algorithm. A genetic algorithm is employed to identify the best answer to the issue. The Darwinian hypothesis serves as the foundation for genetic algorithms. As seen from the simulation results, the evolutionary algorithm theory allows them to solve this cube's optimal solution in 107 moves, which is fast compared to other machine algorithms. The traditional method requires 227 steps to complete.

[8] provided a learning-based method for using a dexterous hand with many fingers to solve a Rubik's cube. Dexterous in-hand manipulation has shown promise, but it is still difficult to solve complicated problems that need several stages and a variety of internal object structure. This difficulty is addressed in this study using a hierarchical deep reinforcement learning approach that separates planning from manipulation. A model-free cube operator controls all five fingers to carry out each move step by step while a model-based cube solver determines the best move order for restoring the cube. They create a high-fidelity simulator that uses a 24-DoF robot hand to manage a Rubik's Cube, an item with a high-dimensional state space, in order to train our models. Extensive tests on 1400 randomly shuffled Rubik's cubes show that our approach works, with an average success rate of 90.3%.

In order to create deep neural networks with the smallest possible model sizes and computing costs, [9] explores the twisting principles. In contrast to network expanding, we find that for small networks, resolution and depth are more crucial than breadth. In this paper, the authors investigate a little formula for the Rubik's cube model as well as the link between accuracy and the three dimensions (r, d, and w). First, they discover that maintaining the performance of a smaller neural network depends more on resolution and depth than on breadth. As a result of the relatively significant drop in resolution, they then note that the inversed gigantic formula, or the compound scaling approach in EfficientNets, is no longer appropriate for creating portable networks for mobile devices. As a result, they carried out several tests and observations to investigate a very small cube formula. The suggested technique twists the three dimensions based on the observation of frontier models, as opposed to the enormous formula in EfficientNet, which was manually created. We specifically use the small formula—the Gaussian process regression on frontier models—to compute the best resolution and depth for the specified upper limit of FLOPs.

They demonstrate in [10] that the Rubik's Cube likewise has a complex algorithmic structure. They specifically demonstrate that the diameter of the configuration space, or "God's Number," of the nnn and nn 1 Rubik's Cubes is (n2/log n). The lower constraint derives from a counting argument, whereas the higher bound results from efficiently parallelizing common (n2) solution techniques. In the worst situation, the upper bound provides an asymptotically ideal solution for solving a generic Rubik's Cube. They demonstrate how to locate the shortest solution in a nO(1)O(1) Rubik's Cube given a certain starting state. Finally, they demonstrate that when some cubes' locations and colours are disregarded, finding this optimal solution in a nn 1 Rubik's Cube becomes NP-hard (not used in determining whether the cube is solved). This aids in determining the computational challenge of solving this puzzle effectively.

Tracing a scramble back to the goal may also be thought of as a goal-predefined combinatorial problem. It may still be feasible to deduce a reasonable scramble as leading up to a particular state even while the actual scramble cannot be observed. Such a scramble should ideally be as brief as feasible. They suggested a straightforward deep learning strategy in this study [11] to infer a scramble as a backward solution, one move at a time, from a given state. Their technique systematically advances the objective problem farther from its goal state when training a DNN. The DNN learns to anticipate the last move of the scramble up to that moment depending on the outcome state at each stage of the scramble. They continuously invert movements that appear to have led to the present states in an effort to find a solution that would lead them back to their original aim. The fundamental premise is that a DNN will learn a distribution of random scrambling movements that is skewed toward optimality.

A combined value and policy network may be trained using the Autodidactic Iteration (ADI) method, which was developed by [12] to overcome a sparse reward in a model-based environment with a huge state space. ADI uses an iterative supervised learning technique to train the value function. Starting from the desired state and doing random actions results in the creation of the neural network's inputs throughout each iteration. By doing a breadth-first search from each input state and utilising the existing network to estimate the value of each leaf in the tree, the goals aim to predict the ideal value function. The values for each node are recursively backed up using a max operator to produce updated value estimates for the root nodes. Similar to this, the policy network is trained by creating targets from the move that maximises value. To effectively solve the Rubik's Cube, the network is integrated with MCTS after being trained.

[13] showed that models learned just in simulation may be utilised to tackle an unprecedentedly complicated manipulation challenge on a real robot. Two essential elements—a novel method that we call automatic domain randomization (ADR) and a robot platform designed for machine learning—make this feasible. ADR produces a distribution across randomised settings with escalating complexity automatically. Control laws and vision state estimators with ADR training show significantly enhanced sim2real transfer. Memory-augmented models trained on an ADR-generated distribution of settings for control policies exhibit pronounced emergent meta-learning during testing. We can solve a Rubik's cube with a humanoid robot hand using ADR in conjunction with their unique robot platform, which incorporates both control and state estimation issues.

In this study, [14] put forth three solutions—one offline and two online—to the Rubik's cube colour identification problem. To demonstrate the effectiveness of training-based methods, an offline technique called Scatter Balance and Extreme Learning Machine (SB-ELM) is developed. They also highlight the idea of colour drifting, which shows that offline techniques are always useless and incapable of functioning efficiently in situations of constant change. On the other hand, dynamic weight label propagation is suggested for labelling blocks' colours based on the cube's known centre blocks' colours. Another online technique, weak label hierarchic propagation, is also suggested for all unknown colour information, but it just makes use of the weak label of the centre block for colour detection. We finally design a Rubik's cube robot and construct a dataset to illustrate the efficiency and effectiveness of our online methods and to indicate the ineffectiveness of offline method by color drifting in our dataset.

Transformers have been effectively applied in a variety of fields, from computer vision to natural language processing, since their inception. It was only recently suggested to use transformers to Reinforcement Learning by reformulating the issue as a sequence modelling one. The Rubik's cube has a distinct set of difficulties in comparison to other reinforcement learning tasks that are frequently studied. With quintillions of different configurations and only one solved state, the Rubik's cube offers exceedingly little rewards. [15] put out the CubeTR model, which tackles the issue of scarce rewards and attends to longer sequences of activities. Without using any human priors, CubeTR learns how to solve the Rubik's cube from any starting state. After move regularisation, it is anticipated that the lengths of solutions it produces would be substantially similar to those provided by algorithms employed by skilled human solvers. CubeTR sheds light on the applicability of transformers in more pertinent sparse reward contexts as well as the generalizability of learning algorithms to larger dimensional cubes.

1. **PROBLEM STATEMENT**

Some algorithms have a certain desired effect on the cube (for example, swapping two corners) but may also have the side-effect of changing other parts of the cube (such as permuting some edges). Such algorithms are often simpler than the ones without side-effects, and are employed early on in the solution when most of the puzzle has not yet been solved and the side effects are not important. Towards the end of the solution, the more specific (and usually more complicated) algorithms are used instead, to prevent scrambling parts of the puzzle that have already been solved. In any case, remembering the complete steps and solving is a difficult task. Hence, the idea of creating applications was to help solving the cube easily for competing cubists as well as to help the beginners learn it.

The objectives of this work include:

* To use efficient data structures to represent the Rubik’s cube for performing all operations.
* To implement Thistlethwaite’s algorithm to solve any cube in less than 35 moves.

1. **PROPOSED WORK**

Before deep diving into the Thistlethwaite’s algorithm, it is important to understand the use case of IDDFS in this study and its significance in Thistlethwaite’s algorithm.

* 1. **IDDFS (Iterative deepening depth-first search)**

Iterative deepening search (IDS or IDDFS), or more precisely iterative deepening depth-first search (IDS or IDDFS), is a state space/graph search technique in which a depth-limited variant of depth-first search is periodically executed with increasing depth limitations until the target is identified. IDDFS explores the nodes in the search tree in the same order as depth-first search at each iteration, but the cumulative order in which nodes are initially visited is essentially breadth-first. It is optimum like breadth-first search but requires significantly less memory.

IDDFS combines the space-efficiency of depth-first search and the completeness of breadth-first search (when the branching factor is finite). It will locate the solution path with the fewest arcs if a solution is possible.

Iterative deepening visits states more than once, which may appear inefficient, but it turns out to be less expensive since in a tree, the majority of nodes are at the bottom level, making it less important to visit the top levels more than once.

The main benefit of IDDFS in game tree searching is that the earlier searches tend to improve frequently used heuristics, such as the killer heuristic and alpha-beta pruning, so that a more accurate estimate of the score of various nodes at the final depth search can occur. The search also concludes more quickly because it is done in a better order. Alpha-beta pruning, for instance, works best when it looks for the greatest moves first.

The responsiveness of the algorithm is a second benefit. Early iterations execute very rapidly because they employ tiny values for "d". As a result, the algorithm may almost immediately provide early indicators of the outcome, which are further refined as 'd' rises. This feature enables the software to play at any moment using the current best move discovered in the search it has already done when used in an interactive situation, such as in a chess-playing programme. Although the task is recursive at each stage, this may be expressed as each depth of the search yielding a better approximation of the solution. A standard depth-first search is unable to accomplish this since it does not yield intermediate results.

* 1. **THISTLETHWAITE’S ALGORITHM**

This technique is based on group theory, a branch of mathematics that examines the algebraic structures known as groups, as well as intensive computer searches. The following four nested groupings Gi are used by the Thistlethwaite's methods to partition the issue into four separate subproblems.

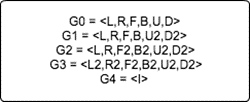


Fig. 17. Nested Group Gi

This algorithm's basic operation is to start with a cube in nested group Gi, and then move some target cubes to their predicted places using only moves from Gi. Continue doing this until G4 is reached, which signifies that the Rubik's cube is fully solved. Every step of the Thistlethwaite’s algorithm, in general, is based on a table that displays the transitions from each element in the current coset space Gi+1 Gi to the following one I = I+1.

**Definition :** ”Given a Group G and a subgroup H < G , a coset of H is the set of Hg = hg : h ∈ H ; thus, H < G partitions G into cosets. The set of all cosets is written h > G.”

The reduced version of the Rubik's cube is described by the coset spaces, which immediately leads to the reduction of the total number of attainable states by employing movements from particular subgroups. In other words, this indicates that fewer movements are allowed each time the software switches to a new group or stage. The calculation of the precise ordering for each group is explained in more detail below.

**Getting from group G0 into group G1**

G0 represents all the states in the Rubik’s cube. This is the number of states reachable from any given state.

Group Positions

G0 4.33 ∗ 1019

All of the edge-orientations are fixed in the initial step. A piece is said to be appropriately orientated along an edge if it can be returned to its initial position without the usage of U and D turns. On each Up and Down rotation, the edges of the Up or Down-face will reverse the edge orientation.

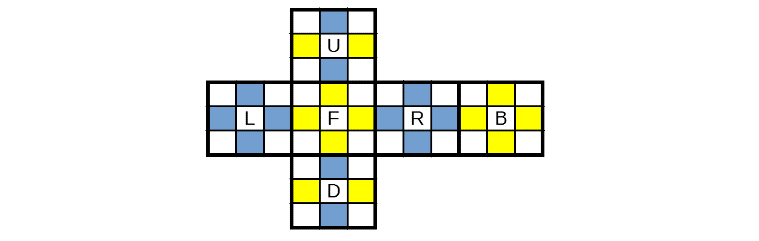


Fig. 18. G1 check

If both places have the same colour (yellow or blue) in the above image, then the edge is good. If they have different colours in the above image, then the edge is bad.

**Getting from group G1 into group G2**

All of the cube states are present in the initial coset space G1/G0. All edge-orientations are fixed in this condition. Due to the fact that the edges are fixed in this stage, the factor for this stage is 211.

Group Positions Factor

G1 2.11 ∗ 1016 211 = 2048

The proper alignment of the edge pieces was accomplished in the earlier step. Due to this, quarter turns of both F and B are no longer allowed. There are two steps in the process of moving the cube from G1 to G2. The middle edges are placed in the middle layer as the initial

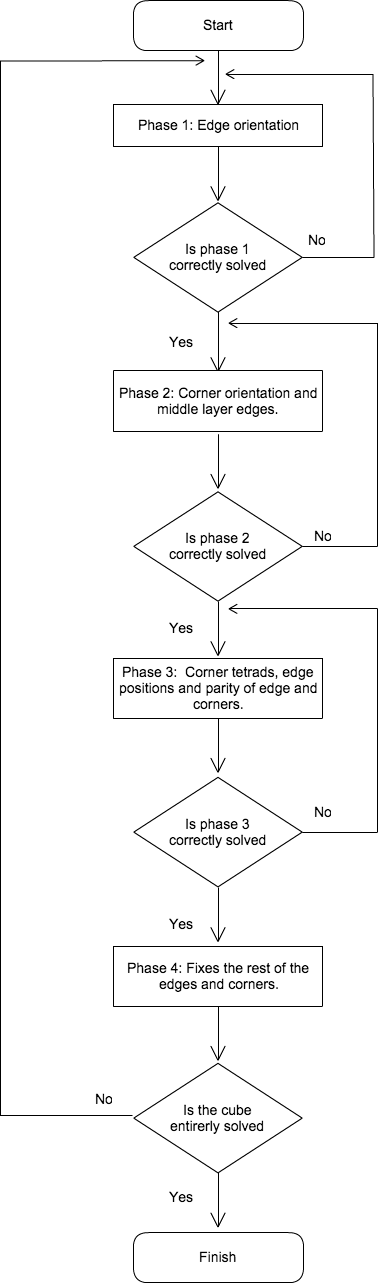


Fig.18. Flowchart of Thistlethwaite's algorithm

step. Second, the angles of the corners are straight.

**Getting from group G2 into group G3**

All edges in the second coset space G2/G1 are fixed. In this level, it is not permitted to turn either the Up or Down faces 90 degrees. The centre edges are placed in the middle layer by a factor in the second stage that corresponds to the fact that all the corners are appropriately orientated.

Group Positions Factor

G2 1.95 ∗ 1010  1,082,565

At this point, both F and B face quarter rotations are prohibited since the corners have been secured into their natural orbits. By doing this, it is certain that the centre edges and corner orientation will never change. The final stage sets the permutation of the corner and edges, inserts the corner into their G3- orbits, and positions all of the edge pieces in their proper positions. The set of places that the corner cubes can reach using only G3 movements are known as G3 orbits.

**Getting form group G3 into group G4**

In the third coset space G3/ G2, using moves from G3 can solve the cube.

Group Positions Factor

G3 6.63 ∗ 105 29400

The cube can be solved at the final level with just double movements. This guarantees that the corners and edges stay in their respective slices. Up until the cube is completely solved, the remaining edges and corners will be put back in their proper positions.

**Final stage**

The solved state is represented by the last stage G4. This indicates that group G3 may be in a potential state that has to be rectified before it can transition to the solved state. This indicates that the permutations of every edge slice and edge corner are now solved.

Group Positions

G4 1

1. **RESULTS**

By using the bitboard data structures to represent the Rubik’s cube, the algorithm is significantly faster than the ones using a 3D array or 54-bit char array. The above method allows twisting which can be done using bitwise operations, and face comparisons which can be done using masks and 64-bit integer comparison.

Sample program execution with input and output, along with faces of Rubik’s cube after each group transition is given below:

* Input : Cube state (all 6 faces up faced).



White Yellow





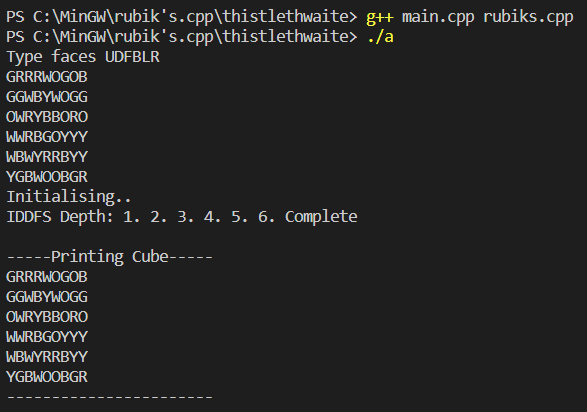
Blue Green

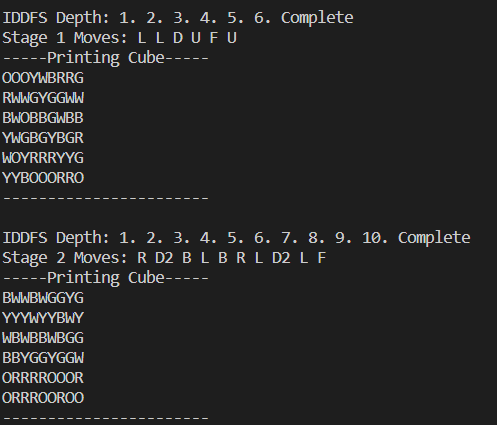


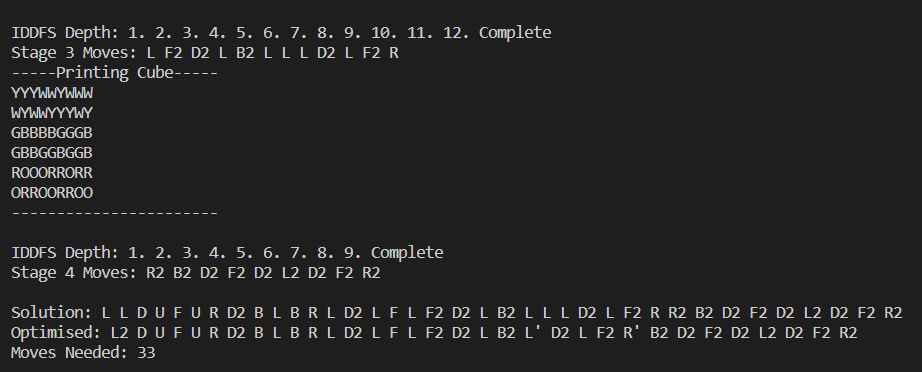


Red Orange

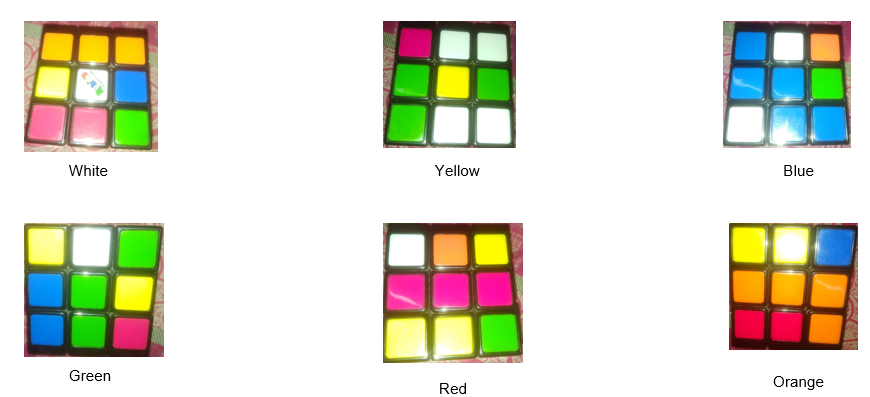
* Output : Executed in VS Code.





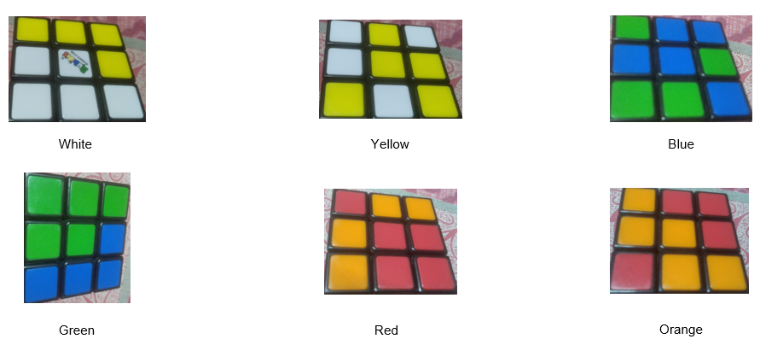


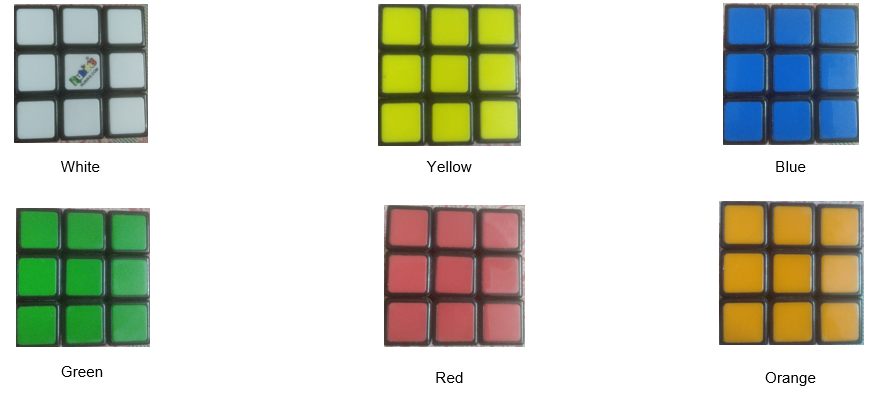
* Cube state after G0 :



* Cube state after G1 :



* Cube state after G2 :
* Cube state after G3 :



1. **CONCLUSION**

This algorithm is effective because we reduce our search space to just searching for moves to transition between each group in a database. The size of each sub problem space is much smaller and therefore more manageable.

Below shows a table of the search space for each group transition:

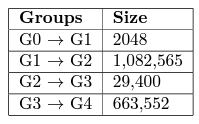


Fig.19. Thistlethwaite’s group transition size

1. **REFERENCES**

[1] Da-Xing Zeng, Ming Li, Juan-Juan Wang, Yu-Lei Hou, Wen-Juan Lu & Zhen Huang (2019). Overview of Rubik’s Cube and Reflections on Its Application in Mechanism. Chinese Journal of Mechanical Engineering.

[2] S. P. Rohith, A. Mohamed Sharif, S. Jayasankar, M. Harikrishnan (2019). Autonomous Rubik’s Cube Solver Bot. International Journal of Scientific Research and Engineering Development – Volume2 Issue 3, May – June 2019.

[3] Shahram Saeidi (2018). Solving the Rubik’s Cube using Simulated Annealing and Genetic Algorithm. MECS, Research Association of Modern Education and Computer Science.

[4] Kevin Yeap Khai Wen, Mohd Nadhir Ab Wahab, Yeap Wei Seng & Wu Chia Chuan (2022). Unraveling the Rubik’s Cube with Autodidactic Iteration Algorithm. Springer Conference Paper.

[5] Khaled Loukhaoukha, Jean-Yves Chouinard, Abdellah Berdai (2012). A Secure Image Encryption Algorithm Based on Rubik’s Cube Principle. Journal of Electrical and Computer Engineering.

[6] E. D. Demaine, S. Eisenstat, and M. Rudoy(2019). Solving the Rubik’s Cube Optimally is NP-complete. 35th Symposium on Theoretical Aspects of Computer Science.

[7] Anil, K. S. (2015), Solution for Rubik’s Cube by Using Genetic Algorithm, International Journal of Engineering Sciences & Research Technology, 636-641.

[8] Tingguang Li , Weitao Xi, Meng Fang, Jia Xu, Max Q.H. Meng (2019), Learning to Solve a Rubik’s Cube with a Dexterous Hand, IEEE International Conference on Robotics and Biomimetics.

[9] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, Tong Zhang, Model Rubik’s Cube: Twisting Resolution, Depth and Width for TinyNets, 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

[10] Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Anna Lubiw, and Andrew Winslow (2015). Algorithms for solving Rubik’s Cubes. In Proceedings of the 19th European Conference on Algorithms. Springer-Verlag

[11] Kyo Takano (2022), Self-Supervision is All You Need for Solving Rubik’s Cube, Cornell University Publication.

[12] Stephen McAleer, Forest Agostinelli, Alexander Shmakov, Pierre Baldi (2018). Solving the Rubik’s Cube Without Human Knowledge. Cornell University Publication

[13] [Ilge Akkaya](https://paperswithcode.com/author/ilge-akkaya), [Marcin Andrychowicz](https://paperswithcode.com/author/marcin-andrychowicz), [Maciek Chociej](https://paperswithcode.com/author/maciek-chociej), [Mateusz Litwin](https://paperswithcode.com/author/mateusz-litwin), [Bob McGrew](https://paperswithcode.com/author/bob-mcgrew), [Arthur Petron](https://paperswithcode.com/author/arthur-petron), [Alex Paino](https://paperswithcode.com/author/alex-paino), [Matthias Plappert](https://paperswithcode.com/author/matthias-plappert), [Glenn Powell](https://paperswithcode.com/author/glenn-powell), [Raphael Ribas](https://paperswithcode.com/author/raphael-ribas), [Jonas Schneider](https://paperswithcode.com/author/jonas-schneider), [Nikolas Tezak](https://paperswithcode.com/author/nikolas-tezak), [Jerry Tworek](https://paperswithcode.com/author/jerry-tworek), [Peter Welinder](https://paperswithcode.com/author/peter-welinder), [Lilian Weng](https://paperswithcode.com/author/lilian-weng), [Qiming Yuan](https://paperswithcode.com/author/qiming-yuan), [Wojciech Zaremba](https://paperswithcode.com/author/wojciech-zaremba), [Lei Zhang](https://paperswithcode.com/author/lei-zhang) (2019), Solving Rubik's Cube with a Robot Hand, Cornell University Publication.

[14] [Shenglan Liu](https://paperswithcode.com/author/shenglan-liu), [Dong Jiang](https://paperswithcode.com/author/dong-jiang), [Lin Feng](https://paperswithcode.com/author/lin-feng), [Feilong Wang](https://paperswithcode.com/author/feilong-wang), [Zhanbo Feng](https://paperswithcode.com/author/zhanbo-feng), [Xiang Liu](https://paperswithcode.com/author/xiang-liu), [Shuai Guo](https://paperswithcode.com/author/shuai-guo), [Bingjun Li](https://paperswithcode.com/author/bingjun-li), [Yuchen Cong](https://paperswithcode.com/author/yuchen-cong) (2019), Color Recognition for Rubik's Cube Robot, IEEE International Conference on Smart Internet of Things (SmartIoT).

# [15]   [Mustafa Ebrahim Chasmai](https://paperswithcode.com/author/mustafa-ebrahim-chasmai) (2021), CubeTR: Learning to Solve The Rubiks Cube Using Transformers, Cornell University Publication.