Rubiks Cube Report

Nikita Mikhaylov, Ryan Johnson, and Vivek Kaushik

May 3, 2019

1 Introduction

A Rubik's cube is any six side cube with either a $2 \times 2, 3 \times 3, 4 \times 4$, number of colored squares per side. It was invented in the mid 1970's by a Hungarian professor Erno Rubik. It was then nominated to be one of the best puzzle toys of the the year, and has now been considered one of the world's best selling puzzle/toy. There are now many worldwide competitions and records that revolve around solving this cube; this includes solving them one-handed, blindfolded, or solving many in a short span of time. The fastest that anyone has ever been recorded solving a Rubik's cube is by Feliks Zemdegs in 4.22 seconds. The original cube was based on 6 sides, with nine stickers per side. There have been various variations by either increasing or decreasing the amount of stickers per side. There are also other variations that have other shape shapes such as a tetrahedron variation. The model that we have decided to explore is the original 3×3 Rubik's cube.

2 Preliminaries

This section focuses on mathematically representing the Rubik's cube and all the possible movements.

2.1 Configurations

Definition. The Rubiks Cube has 6 **sides (facets)**, and we label the sides as F, L, U, B, R, D to stand for front, left, up, rear, right, and down, respectively. We identify each side A with a 3×3 **side matrix** $\mathrm{A} = [\mathrm{A}_{ij}]_{i,j=1}^{3}$, and each entry A_{ij} is referred to as the (i, j)-th **square**.

We have not specified what type of entries the squares are; they can be as generic as desired. In our *SageMath* Notebook, we used two different labelings of the squares. We used numbers from the set $\{1, \ldots, 54\}$ for the facilitation of mathematically defining cube movements, while we used color abbreviations {g, o, w, b, r, y} (standing for the green, orange, white, blue, red, yellow, respectively) for real-life visualization purposes.

Definition. A **numerical configuration** is an ordered tuple of side matrices (F, L, U, B, R, D), where

- Each side S has distinct entries from $\{1, \ldots, 54\}$.
- Any two distinct sides S and T do not have any common entries.

We write $\mathcal{N}$ as the set of all numerical configurations. In particular, we define the **standard numerical configuration** $I_N \in \mathcal{N}$ to be

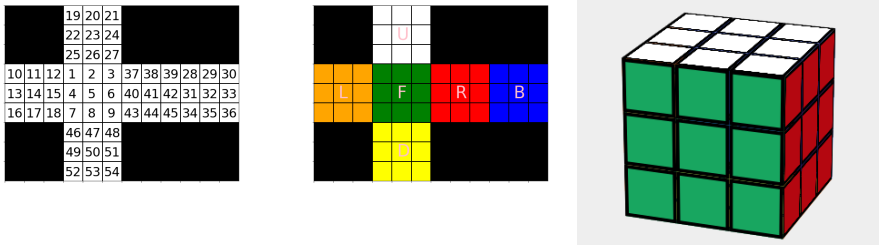
$$I_N = \left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \ldots, \begin{bmatrix} 46 & 47 & 48 \\ 49 & 50 & 51 \\ 52 & 53 & 54 \end{bmatrix} \right).$$

Definition. A **colored configuration** is an ordered tuple of side matrices (F, L, U, B, R, D), where for each side S and all indices i, j, we have $\mathrm{S}_{ij} \in \{\mathrm{g}, \mathrm{o}, \mathrm{w}, \mathrm{b}, \mathrm{r}, \mathrm{y}\}$. The set of all colored configurations is denoted as $\mathcal{C}$. In particular, the **solved configuration** $I_C \in \mathcal{C}$ is the configuration

$$I_C = \left(\begin{bmatrix} g & g & g \\ g & g & g \\ g & g & g \end{bmatrix}, \begin{bmatrix} o & o & o \\ o & o & o \\ o & o & o \end{bmatrix}, \ldots, \begin{bmatrix} y & y & y \\ y & y & y \\ y & y & y \end{bmatrix} \right).$$

Definition. For a configuration $N \in \mathcal{N}$, we write $N(\mathrm{S})$ to refer to the particular side matrix S in N.

We will explain how to visualize a configuration. We set the net diagram of I_N to be **Figure 1a**. One can visualize the cube by imagining folding the sides L, U, D, R, B in **Figure 1a** towards the back of this page and gluing the appropriate edges. To visualize an arbitrary colored configuration C, simply replace the numerical labels of the squares in I_N with the appropriate color abbreviation labels, and repeat the same process. **Figure 1b** shows the net diagram for I_C, and **Figure 1c** shows the resulting real-life cube upon folding of the aforementioned sides.



(a) I_N Net Diagram (b) I_C Net Diagram (c) I_C in 3D

Figure 1: I_N and I_C

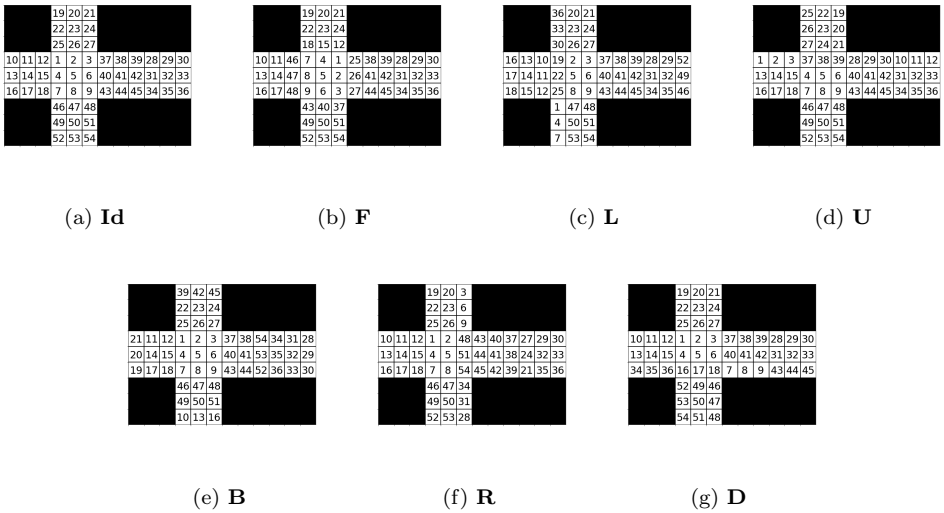
2.2 Rotations

We are now ready to define cube movements.

Definition. Let S be a side of the cube. We define the **clockwise rotation function S** : $\mathcal{N} \to \mathcal{N}$ to be such that $\mathbf{S}(N)$ is the resulting configuration upon rotating the side S clockwise in the given configuration N. Similarly, we define the **counter-clockwise rotation function S'** : $\mathcal{N} \to \mathcal{N}$ to be such that $\mathbf{S'}(N)$ is the resulting configuration upon rotating the side S counter-clockwise in N. When we say clockwise (or counter-clockwise), we mean clockwise (or counter-clockwise) relative to viewing the 3D cube with the side S appearing directly in front.

We define the **identity rotation Id** : $\mathcal{N} \to \mathcal{N}$ to be such that $\mathbf{S}(N) = N$ for all N and define **Id'** = **Id**.

Figure 2 on the next page shows the results of applying the clockwise rotations on I_N.



(a) **Id** (b) **F** (c) **L** (d) **U**

(e) **B** (f) **R** (g) **D**

Figure 2: Net Diagrams of I_N Clockwise Rotations

Using **Figure 2**, it is easy to write out the net diagrams corresponding to all of the counter-clockwise rotations for I_N.

Proposition 2.2.1. For any configuration $N \in \mathcal{N}$, the rotation **S** (or **S'**) changes the values of different 20 squares.

Proof. There are 8 squares of S that are shuffled (all but the middle one). Furthermore, we view N as a cube with side S appearing in front. Note there are 4 sides adjacent to **S**. Observe applying **S** (or **S'**) will result in each of these adjacent sides having exactly 3 squares that change value. So 12 other squares outside of S change value. In all, $12 + 8 = 20$ squares change value upon applying **S** (or **S'**). □

3 Group Theory

We wonder if there is a way to understand these Rubik's Cube rotations better without necessarily resorting to pictures of net diagrams. It turns out we can form a group out of the rotations. Further, we will relate this group to a permutation subgroup of S_{54}.

3.1 Construction of Group

We begin with a proposition that is intuitive and obvious in real life, but is still important nonetheless.

Proposition 3.1.1. For any clockwise rotation **S**, we have

- $\mathbf{S'} = \mathbf{S}^3$
- $\mathbf{S} = \mathbf{S'}^3$
- $\mathbf{S} \circ \mathbf{S'} = \mathbf{S'} \circ \mathbf{S} = \mathrm{Id}$.
- $\mathbf{S}, \mathbf{S'}$ are bijective.

Proof. We prove only the first statement. The proof of the second statement is very similar to that of the first, while the other statements are immediate corollaries.

Let $N \in \mathcal{N}$ be an arbitrary configuration. Assume we view the 3D cube representation of N with side S appearing in front. Let $\mathrm{F}_1, \mathrm{F}_2, \mathrm{F}_3, \mathrm{F}_4$ be respectively the left, top, right, bottom adjacent sides to S.

We first examine what happens to $N(\mathrm{S})$ upon applying **S** or $\mathbf{S}'$ to N. It is easy to check upon applying **S** to N that $N(\mathrm{S})$ transforms into $(N(\mathrm{S}))^T\mathrm{M}$, where

$$\mathrm{M} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

On the other hand, upon applying **S'** to N, it is easy to check $N(\mathrm{S})$ transforms into $\mathrm{M}^T(N(\mathrm{S}))^T\mathrm{M}^2$. Applying $\mathbf{S}^3$ to N, we see that $N(\mathrm{S})$ transforms to $f^3(N(\mathrm{S}))$, where $f(N(\mathrm{S})) = (N(\mathrm{S}))^T\mathrm{M}$. We find

$$\begin{aligned} f^3(N(\mathrm{S})) &= f^2((N(\mathrm{S}))^T\mathrm{M}) \\ &= f(\mathrm{M}^T N(\mathrm{S})\mathrm{M}) \\ &= \mathrm{M}^T(N(\mathrm{S}))^T\mathrm{M}^2. \end{aligned}$$

So $\mathbf{S}^3, \mathbf{S'}$ transform $N(\mathrm{S})$ in the same way.

Now let us examine what happens to each of $N(\mathrm{F}_1), \ldots, N(\mathrm{F}_4)$, upon applying **S** or $\mathbf{S}'$ to N. For each i, let f_1^i, f_2^i, f_3^i be the 3 squares in $N(\mathrm{F}_i)$ that change values upon applying **S** or $\mathbf{S}'$ to N. Assume without loss of generality that when we apply **S** to N, for all $j \in \{1, 2, 3\}$ and i cyclically indexed mod 4, the square in $N(\mathrm{F}_i)$ that had initial value f_j^i before now has value f_j^{i-1}. Then if we apply **S'** to N, the square in $N(\mathrm{F}_i)$ that had initial value f_j^i before now has value f_j^{i+1}. Note for every i, we have $i+1 \equiv i-3 \mod 4$. So when applying **S'** to N, the square in the resulting transformed $N(\mathrm{F}_i)$ with value f_j^{i+1} is precisely f_j^{i-3}, which is what we would get when examining the resulting transformed $N(\mathrm{F}_i)$ upon applying $\mathbf{S}^3$ to N. So $\mathbf{S}^3, \mathbf{S'}$ transform $N(\mathrm{F}_i)$ in the same way for each i.

Finally, the remaining side stays the same upon applying $\mathbf{S}^3$ or **S'** to N. □

Consider $\mathcal{G} = \langle \mathbf{F}, \mathbf{L}, \mathbf{U}, \mathbf{B}, \mathbf{R}, \mathbf{D} \rangle$, which is the set of all possible compositions of one or more clockwise rotation functions.

Theorem 3.1.1. *$\mathcal{G}$ is a nonabelian group under composition.*

Proof. Let $\mathbf{M} = \mathbf{M}_1 \circ \ldots \circ \mathbf{M}_n \in \mathcal{G}$ and $\mathbf{N} = \mathbf{N}_1 \circ \ldots \circ \mathbf{N}_k \in \mathcal{G}$, where each $\mathbf{M}_i, \mathbf{N}_j$ are clockwise rotation functions. We see $\mathbf{M} \circ \mathbf{N} \in \mathcal{G}$ since it is still a composition of clockwise rotation functions. So closure holds. It is well-known that composition of functions is associative, and hence associativity automatically holds. Note by **Proposition 3.1.1**, we have that $\mathbf{S'}, \mathbf{Id} \in \mathcal{G}$ for any clockwise rotation $\mathbf{S}$. Furthermore, $\mathbf{Id}$ is the identity element in $\mathcal{G}$. By induction on n, it can be shown that $\mathbf{M'}_n \circ \ldots \circ \mathbf{M'}_1 \in \mathcal{G}$ is the inverse of $\mathbf{M}$. So $\mathcal{G}$ is a group.

$\mathcal{G}$ is not abelian since, for example, $\mathbf{F} \circ \mathbf{R} \neq \mathbf{R} \circ \mathbf{F}$. □

3.2 Group Isomorphism

Consider the permutations $f, l, u, b, r, d \in S_{54}$ defined

- $f = (1, 7, 9, 3)(2, 4, 8, 6)(12, 46, 43, 27)(15, 47, 40, 26)(18, 48, 37, 25)$
- $l = (1, 19, 36, 46)(4, 22, 33, 49)(7, 25, 30, 52)(10, 16, 18, 12)(11, 13, 17, 15)$
- $u = (1, 37, 28, 10)(2, 38, 29, 11)(3, 39, 30, 12)(19, 25, 27, 21)(20, 22, 26, 24)$
- $b = (10, 21, 45, 52)(13, 20, 42, 53)(16, 19, 39, 54)(28, 34, 36, 30)(29, 31, 35, 33)$
- $r = (3, 48, 34, 21)(6, 51, 31, 24)(9, 54, 28, 27)(37, 43, 45, 39)(38, 40, 44, 42)$
- $d = (7, 16, 34, 43)(8, 17, 35, 44)(9, 18, 36, 45)(46, 52, 54, 48)(47, 49, 53, 51)$

We get an analagous property of **Proposition 3.1.1**

Proposition 3.2.1. For any $s \in \{f, l, u, b, r, d\}$, we have $s^{-1} = s^3$.

Define $\mathcal{S} = \langle f, l, u, b, r, d \rangle$. The following theorem should not be surprising.

Theorem 3.2.1. *$\mathcal{S}$ is a subgroup of S_{54} under composition.*

Proof. Closure follows from construction. Let $m = m_1 \circ \ldots \circ m_p \in \mathcal{S}$ where each $m_i \in \{f, l, u, b, r, d\}$. By induction on p, it is easy to verify its inverse is $m_p^{-1} \circ \ldots \circ m_1^{-1}$. □

The previous proposition and theorem strongly suggest $\mathcal{S}$ has virtually the same properties as $\mathcal{G}$ although they are different groups. The theorem below makes this informal observation precise.

Theorem 3.2.2. *We have that S and G are isomorphic.*

Proof. Suppose that $\phi : \mathcal{S} \to \mathcal{G}$ is defined $\phi(s) = \mathbf{S}$ for any $s \in \{f, l, u, b, r, d\}$

It can be shown for any $s, t \in \{f, l, u, b, r, d\}$, we get that

$$\phi(s \circ t) = \phi(s) \circ \phi(t).$$

If $m = m_1 \circ \ldots \circ m_p$ and $n = m_1 \circ \ldots \circ n_q$ where each $m_i, n_j \in \{f, l, u, b, r, d\}$, then we can show by double induction on p and q that

$$\phi(m \circ n) = \phi(m) \circ \phi(n).$$

Hence ϕ is a homomorphism.

If $\mathbf{M} = \mathbf{M}_1 \circ \cdots \circ \mathbf{M}_p$, where each $\mathbf{M}_i$ is a clockwise rotation, then induction on p shows that

$$\phi(m_1 \circ \ldots \circ m_p) = \mathbf{M}.$$

This proves ϕ is surjective.

It is clear that none of the permutations $\{f, l, u, b, r, d\}$ lie in $\ker(\phi)$. We omit the proof that ϕ is injective, but the idea is to assume the existence of a non identity permutation $m \in \ker(\phi)$, decompose it into a composition of permutations from $\{f, l, u, b, r, d\}$, and induct on the composition length to obtain a contradiction. □

Through *SageMath*, we obtained the order of $\mathcal{S}$ is $2^{27} \times 3^{14} \times 5^3 \times 7^2 \times 11$. This means the order of $\mathcal{G}$ is precisely that same number.

4 Thistlethwaite's Algorithm

Thistlethwaite's algorithm is a method for solving a Rubik's Cube that reduces the cube down into smaller and smaller subgroups until the cube has been solved. This method is most commonly used by computers, but is not typically used in a modern implementation of a Rubik's Cube solver. This is because a better method, known as the two phase algorithm, has been developed which has a maximum move count of 30 moves, compared to a maximum of 52 for Thistlethwaite's algorithm.

This section will cover the specifics of Thistlethwaite's algorithm, and walk through a solution generated by the solver we implemented, so as to provide a better understanding of how our solver works.

4.1 Subgroups

The important subgroups used in Thistlethwaite's algorithm are as follows:

- $G_0 = \langle \mathbf{F}, \mathbf{B}, \mathbf{L}, \mathbf{R}, \mathbf{U}, \mathbf{D} \rangle$
- $G_1 = \langle \mathbf{F}^2, \mathbf{B}^2, \mathbf{L}, \mathbf{R}, \mathbf{U}, \mathbf{D} \rangle$
- $G_2 = \langle \mathbf{F}^2, \mathbf{B}^2, \mathbf{L}^2, \mathbf{R}^2, \mathbf{U}, \mathbf{D} \rangle$
- $G_3 = \langle \mathbf{F}^2, \mathbf{B}^2, \mathbf{L}^2, \mathbf{R}^2, \mathbf{U}^2, \mathbf{D}^2 \rangle$
- $G_4 = \langle \mathbf{Id} \rangle$

With G_4 denoting the solved cube. Recall that each of these groups represent all of the cube states that can be reached by making only the moves assigned to that group. For example, G_1 represents all cube states that can be reached by any combination of the moves $\langle \mathbf{F}^2, \mathbf{B}^2, \mathbf{L}, \mathbf{R}, \mathbf{U}, \mathbf{D} \rangle$. Although the visual difference between a Rubik's cube in G_0 versus G_1 is not immediately obvious, those familiar with a method of solving the Rubik's cube that involves Edge-Orientation (also called EO) will know when a cube is in G_1 because all of its edges will be "good edges," as they are called in some methods. Methods that use this kind of edge orientation include Roux, ZZ, and Petrus. It is not important to understand edge orientation or any of the aforementioned methods for solving the Rubik's cube in order to understand exactly what G_1 or any of the other groups represent. It does, however, give those who do understand EO a better tool with which to visualize the difference between G_0 and G_1.

A group that is easier to discern, but still non-trivial like G_4, would be G_3. Recall that G_3 represents all cube states that can be reached by any arbitrarily long sequence of moves, where each move is a 180 turn of any face. This group can be identified as each the stickers of each side of the Rubik's cube will be one of two colors, determined by the face the stickers are on. In particular, faces will only have stickers colored the color of that face and the color of the opposite face. For example, the blue and the green face will only have stickers that are blue and green. Yellow and white faces will only have yellow and white faces, and so on.

4.2 Group Reduction Explanation

Before going into an example solve, we will describe briefly the process involved in reducing the cube from group to group used by our solver. For ease of reference to specific edges and corners, a specific edge will be denoted by two letters, each indicating a face that it is a part of. For example, E_{ub} indicates the edge that is currently in the up and the back face. On a solved cube, that edge would be the white and red edge. Similarly, C_{ufr} denotes the corner in the up, front, and right face. On a solved cube, that will be the white, green, and red corner.

4.2.1 G_0 to G_1

A scrambled cube starts in G_0. In order to get to G_1, the cube must be gotten into a state where all the edges can be moved into their original location (when the cube is solved) while being oriented correctly (i.e. not "flipped") without turning the front or back face. If the above stipulations do not apply to an edge, it is considered a "bad edge." Otherwise, an edge is a "good edge." Observe that turning the front or back face changes good edges to bad edges, and bad edges to good edges. We can use this to turn four bad edges into four good edges. Also note that any other face turn does not change the orientation (or "goodness"/"badness") of the edge. One last thing to note is that there is always an even number of bad edges on a cube.

Our algorithm changes bad edges into good edges until all edges on the cube are good. If there are at least four bad edges on the cube, then four of them are moved into the front layer, and then the front face is turned. This results in all four of those bad edges to become good. This process is done until there are two or zero bad edges left on the cube. If there are zero bad edges, the process is complete. If there are two left, one bad edge is put in the front layer with three other good edges. The front face is then turned. This results in three bad edges and one good edge. The total number of bad edges is now four, and the algorithm can continue as normal.

4.2.2 G_1 to G_2

As with all of the subgroups, G_2 is well defined in a mathematical sense, but that definition does not provide an intuitive human-recognizable condition by which one can identify a cube in G_2. We will provide one here.

The first condition that the cube must satisfy in order to be a part of G_2 is that it must be a part of G_1. If a cube is not a part of G_1, then it cannot be in G_2. The second condition is that every sticker on the Up and Down faces must have the color of either the Up or Down faces. To illustrate, take a cube where the white face is on the top (Up) and the yellow face is on the bottom (Down). Then, in order for that cube to be in G_2, it must be in G_1 and all the stickers on the top and bottom faces must be either white or yellow. Note that the top and bottom faces need not be only white or only yellow, but instead the top face must only have stickers that are white and/or yellow (as with the bottom face as well). A cube that satisfies these conditions can be said to be in G_2.

Our algorithm gets a cube from G_1 to G_2 via two steps. For the sake of brevity, we will simply provide a brief overview of each step from now on, as the specifics of the steps warrant a paper of its own. The first part involves making white and yellow crosses on the top and bottom faces. A cross need not only be white or only yellow, but can be a combination of both colors. This process is relatively simple and requires no algorithmic like sequences of moves. The second step involves orienting all the corners on the cube so that they are all white or yellow on the top and bottom. This requires one algorithm: $R'DL^2D'R$. As far as we are concerned at this step, this algorithm first rotates the C_{ufl} corner by one and the C_{dfr} corner by two, and then it swaps the corners. Note that when a corner is rotated 3 times, it is returned to its original orientation. Thus, it would make sense that applying this algorithm twice should return the cube to its original position. This is confirmed by the fact that the inverse of this algorithm is itself (i.e. it is the same forwards as it is backwards), therefore doing it twice will return the cube to its original state.

The combination of these two functions is enough to reduce a cube in G_1 to G_2.

4.2.3 G_2 to G_3

Again, here we will provide an intuitive definition for G_3, as it will provide logical motivation for each step.

G_3 is perhaps the easiest of all the groups to identify. Recall the description of the Up and Down faces of a cube in G_2. That is, the Up and Down faces have stickers only of their color and the color opposite. A cube in G_3 will have this condition on all sides. In particular, every side on the cube will be comprised of stickers that are either the same as the color associated with that face, or it will be the same as the color associated with the face opposite the one it is currently a part of.

Our algorithm achieves this through roughly three steps. The first step is not necessary to reduce the cube, as it gets the cube no closer to being in G_3. However, it does make the steps that follow computationally less expensive, as it significantly reduces the number of cases that need to be accounted for, as well as making it easier to identify out-of-orbit corners in the step that follows.

The first step involves putting all the corners in the correct layer. That is to say, all corners that belong in the bottom layer are put in the bottom layer, and all corners that belong in the top layer are put in the top layer. With a cube that has white on the top and yellow on the bottom, this results in all corners with a white sticker being in the top layer, and all corners with a yellow sticker being in the bottom layer. Note, also, that as a result of being in G_2, these corners are all oriented such that the white and yellow stickers face up and down respectively. Again, this step is not necessary, but it does make the next step easier to handle.

The next step involves permuting the corners such that they are all solved. Our algorithm solves both sets of corners (the four in the Up layer and the four in the Down layer) simultaneously. Both layers can be in one of three states, hereby referred to as "headlights", "diagonal", or solved. Note that each layer can be in any of the three states independent of which state the other layer is in. Also note that

these apply to both the top and the bottom layers, but we will refer to the top layer for convenience. A layer is solved when it is some number of turns of the Up face away from having all of its corners in their solved states. A layer has headlights if, when viewing one non-Up/Down face at a time, the stickers in the top right and top left spots are the same color (bottom right and bottom left for the Down layer). Note that a cube can only have one set of headlights at this point. If it has any more, then it must have four headlights, making it solved. If it has any less, then it has none, which makes it the diagonal case. A layer is called diagonal when it has zero headlights. It is called diagonal because if one were to swap two corners within that face that are diagonal from each other, it would result in that layer now being solved.

Because both layers can be any of the three states described above independently, that means there are 3 by 3 many cases to handle for this step, for a total of nine cases. Some cases are trivial, such as the case where both layers are solved. Others are harder to handle, like the case where only the top layer is solved, and the bottom layer is either headlights or diagonal. Thus, our algorithm handles some of the cases directly, in that there is an algorithm to handle that combination specifically. All other cases are handled by changing them into the aforementioned cases. Again, we will not be describing the specific process by which this is done, but the cases that are handled directly are as follows: $(solved, solved), (diagonal, headlights), (headlights, diagonal), (diagonal, diagonal)$. All other cases are handled by changing them into one of these four states.

The last step of this process is to permute the edges. We will be making use of the term "bad edges" again in this section, but with a different definition. At this step, the only pieces that should be disqualifying the cube from being in G_3 should be the edge pieces. In particular, any edge piece that has a sticker that is one 90 degree turn away from being in the correct face. Put another way, any edge that makes it such that the cube is not in G_3. We will now refer to these edges as bad edges.

The process for fixing these bad edges is much like the process for fixing the bad edges from G_0, in that we try to move the bad edges to a certain spot on the cube (the front layer for G_0, (E_ub, E_uf, E_dr, E_dl) for this process), and then we apply a set of moves to make them all good edges (F for G_0, $DR^2L^2U'L^2R^2$ for this process). However, unlike before, there are many edge-cases that need to be handled. This one of the more complicated parts of the whole process, but it can be broken down into what is ultimately a bunch of if statements that cover every possible combination of bad edges. A survey of all the combinations offers very little insight as to how exactly it works, and thus will not be covered here.

4.2.4 G_3 to Solved

Because a solved cube is trivial to identify, and because we have already provided a human understandable description of G_3, no intuition need be provided to explain the goal of this last reduction.

Our algorithm for this last step is once again broken down into three distinct steps. The first involves completely solving the top layer, then completely solving the bottom layer, and finally completely solving the middle layer. The reason for this ordering is because the middle layer is relatively easy to solve, as all its edges are already within itself. That is to say that all edges that belong in the middle layer are in the middle layer. Compare this to the top and bottom layers, where an edge that belongs in the spot E_{uf} can easily be in the spot E_{df}. Therefore, we solve the top and bottom layers first, as they will benefit from the increased flexibility of having edges in the middle to move around freely. For example, the algorithm $U^2F^2U^2F^2U^2F^2$ will swap edges E_{ul} and E_{fl} with edges E_{ur} and E_{fr} respectively. However, because we are not yet concerned with solving or maintaining a solved middle layer, swapping E_{fl} and E_{fr} is inconsequential, and we do not need to worry about it. This gives us a nice algorithm for effectively swapping two edge pieces, an operation which normally is not possible on a Rubik's cube. This freedom is the reason why we solve the middle layer last.

4.3 Example Solve

Now that we have gone over how our our algorithm executes Thistlethwaite's algorithm, we will go through a "walk-through solve" of a Rubik's cube to see it in practice.

4.3.1 Scramble and Solution

The scramble we will be using is as follows:

$$F^2DUR'F'LF'LU'F'R'DU'L'DRLUFR'B'U'F^2BUBDF'D'F'D'L'U^2$$

And the solution generated by our solver is as follows:

$$(LU'L'D'R'B^2L^2FU'RUF'U'F)$$

$$(U'RLDR^2DRD^2U'R'DL^2D'RDU^2F^2R'DL^2D'R)$$

$$(R^2D^2R^2DR^2D'R^2DR^2D'U'B^2L^2B^2L^2B^2L^2D)$$

$$(U^2F^2B^2U^2F^2B^2L^2B^2U^2B^2L^2U^2B^2L^2D^2L^2D^2L^2U^2F^2U^2R^2U^2F^2U^2R^2U^2F^2U^2R^2U^2F^2U^2R^2)$$

Where each section closed in parentheses is the specific solution to get from group to group. Applying all of the moves above in the order shown to a cube with the above scramble will result in a solved cube. We will go over each section individually, having scrambled the cube with white on top and green in front.

4.3.2 G_0 to G_1

Note that the cube already starts with three bad edges in the front layer. Although this is a very convenient situation to take advantage of, you will see that our algorithm does not take advantage of this. Instead, the algorithm finds four bad edges not in the front layer and puts them into the front layer, despite there already being three bad edges in the front layer. This is accomplished through the moves: $LU'L'D'R'B^2L^2$. Now that there are four bad edges in the front layer, it makes them good by turning the front layer once: F. However, now there are only two bad edges left on the cube. Thus, one is moved to the E_{uf} position, and one is moved to the E_{fr} position: $U'R$. Then, one of the bad edges is moved out of the front layer, the front layer is turned, and the bad edge is moved back into the front layer. Now there are four bad edges, all in the front layer. Thus, all that is needed to fix all the edges is a turn of the front face. This all together looks like: $UF'U'F$. And now all the cubes edges are good edges, and we are now in G_1.

4.3.3 G_1 to G_2

In getting to G_2, the algorithm first makes the white and yellow crosses, starting with the one on top. Recall that a cross can be made of both white and yellow stickers. The top cross is easy, as three of the four parts of the cross are completed. To put in the last piece for that cross, the following moves are executed: $U'RL$. Note that the R in the middle seems to serve no purpose. I am not sure where that move is coming from, and it may be a bug in the code. However, ultimately, this does leave us with a solved top cross.

Next is the bottom cross. In this case, two edges need to be replaced in the bottom layer. This is accomplished through the moves: DR^2DR. The cube now has white and yellow top and bottom crosses.

Now that the crosses are finished, the next step is to orient the corners. First, appropriate corners are placed in the C_{ufl} and C_{dfr} spots: D^2U'. Then, an algorithm to orient them correctly is applied: $R'DL^2D'R$. Then, once again, appropriate corners are placed in the appropriate places: DU^2F^2. Finally, that same algorithm is applied, correctly orienting all the corners on the cube: $R'DL^2D'R$.

4.3.4 G_2 to G_3

Recall that the first step to get to G_3 from G_2 is to permute all corners on the cube, done in two steps. First, get all corners into the correct layer, and then permute the corners. The cube is very close to having all corners in the appropriate layers, and the following is executed to get the last of each corner in each layer (note: getting all white corners in the top requires that all yellow corners be on the bottom, so they are solved simultaneously): $R^2D^2R^2DR^2D'R^2DR^2$. Another thing to note is that this sequence of moves simultaneously did steps one and two. This is because our algorithm will simplify a generated sequence of moves such as R^2R^2 to be an identity movement, or in other words, no face turns at all. Thus, in simplifying the generated sequence, the two steps became one step, less understandable to a human, but still useful. What the original sequence likely looked like was as follows: $R^2D'R^2$, which completes step one, leaving us with a headlights case on the top and a diagonal case on the bottom, which has the following solution with the headlights on the left (as they are in this case): $R^2D'R^2DR^2D'R^2DR^2$. Notice, when these two are combined, the result is: $(R^2D'R^2R^2D'R^2DR^2D'R^2DR^2) = (R^2D'D'R^2DR^2D'R^2DR^2) = (R^2D^2R^2DR^2D'R^2DR^2)$ which is the original sequence provided.

The next and final step to get to G_3 is to fix all the bad edges, using the definition of bad edges provided in section 4.2.3. As a reminder, a bad edge in this case is any edge which disqualifies the cube from being in G_3. In this case, there are four bad edges. To fix them, the sequence begins with: $D'U'$ which are setup moves, in particular the D'. Note that setup moves are moves that are done before an algorithm and then undone after so as to slightly change how the algorithm affects the cube. To be clear, after these setup moves, the bad edges in question are located at E_{ub}, E_{ul}, E_{db}, and E_{dl}. For particular insight as to what those setup moves have done, the turn of the Up face aligns the corners with where they should be when they are solved. Then, the Down move moves the bad edges in the bottom layer so that they are underneath the bad edges in the top layer. After this, the following algorithm is applied: $B^2L^2B^2L^2B^2L^2$ which swaps the four bad edges mentioned earlier. This makes them all good edges, and so the last thing to do is undo the setup move with D.

4.3.5 G_3 to Solved

Now it is time to finish solving the cube. To make this section easier to parse and understand, we will be using the non-reduced sequence of moves generated by the solver. See the previous section for an explanation of how sequences can be reduced. The non-reduced sequence of moves that corresponds to the solution provided above is:

$$U^2F^2B^2U^2F^2B^2U^2D^2D^2U^2L^2B^2U^2B^2L^2U^2B^2D^2D^2L^2D^2L^2D^2L^2U^2F^2U^2R^2U^2F^2U^2R^2U^2F^2U^2R^2U^2F^2U^2R^2$$

Recall that the first step our algorithm takes is to completely solve the top layer. Currently, there are two edges that need to be solved in the top layer. The first of these two edges is solved using this algorithm: $U^2F^2B^2U^2F^2B^2U^2D^2$. Then, the last edge for the top layer is solved using this algorithm: $D^2(U^2L^2B^2U^2B^2L^2U^2B^2)D^2$ where the D^2 at the beginning and end of the algorithm are setup moves. After this, the top layer has been solved.

To solve the bottom layer, algorithms that swap around pieces in (but not between) the bottom and middle layers are used. First: $D^2L^2D^2L^2D^2L^2$ swaps the E_{df} and E_{db} edges. After that, the bottom layer has been solved.

The last step is to solve the middle layer. In this case, a single algorithm is applied twice: $U^2F^2U^2R^2U^2F^2U^2R^2$. This algorithm cycles three pieces in the middle layer. Our algorithm knows that if the middle layer is not in one of a few predefined states, then this three-cycle will either solve it, solve it if done twice, or get it into a predefined state. The middle layer was not in any of the predefined states, so the algorithm was applied. After that, the middle layer was still not in any of the predefined states. Thus, after executing the algorithm one last time, the cube finally reached a solved state.

5 Further Research

We wish to analyze $\mathcal{G}$ using representation theory. That is, we want to encode rotations with matrices. Fortunately, we know $\mathcal{G}$ is isomorphic to $\mathcal{S}$, a subgroup of S_{54}, and S_{54} has the standard representation $\rho : S_{54} \to \mathrm{GL}(\mathbb{R}^{54})$ defined by $\rho(\sigma)(v) = (v_{\sigma(1)}, \ \ldots, v_{\sigma(54)})$. Hence, restricting $\mathrm{GL}(\mathbb{R}^{54})$ to those linear transformations that bijectively correspond to permutations in $\mathcal{S}$ will essentially provide the linear transformations that correspond to rotations in $\mathcal{G}$. Do standard representation theory techniques in the case of ρ tell us anything about how to solve the cube ?

References

[1] Nail and Hauke El-Sourani Sascha and Borschbach, *An Evolutionary Approach for Solving the Rubik's Cube Incorporating Exact Methods*, Applications of Evolutionary Computation, 2010, pp. 80–89.

[2] Erik D and Demaine Demaine Martin L and Eisenstat, *Algorithms for solving Rubik's cubes*, European Symposium on Algorithms, 2011, pp. 689–700.

[3] JJ Chen, *Group theory and the Rubik's cube*, 2004.

[4] Tom Davis, *Group Theory via Rubik's Cube* (2006).