2017

Complexities of Bi-Colored Rubik's Cubes

Taylor Pieper

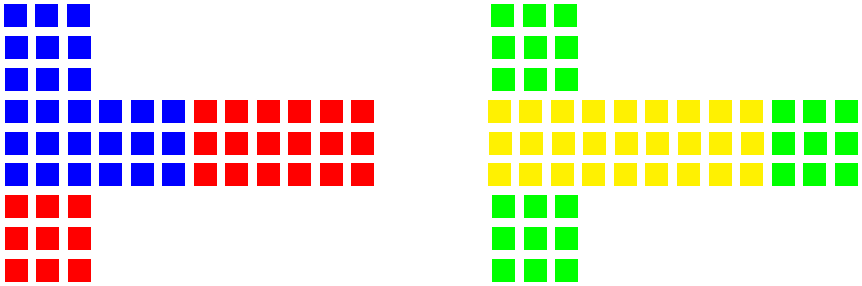
Complexities of Bi-Colored Rubik's Cubes

Taylor Pieper

Contents

Complexities of Bi-colored Rubik's Cube

Abstract



Which of two bi-colored cubes is the simpler puzzle? The differences in the coloring of the cubes creates different symmetries that dramatically reduce the number of states each cube can reach. Which of the symmetries is most reductive? The answer to these questions can be achieved by discovering and comparing the "'God's Number"' for these cubes. By analyzing the bi-colored cubes as subgroups of the original cube, the answer can be found by comparing the number of solvable states, as well as the minimum number of moves it takes to solve every state of the cubes. By simple discrete analysis, we found that the Green and Yellow (GY) cube has fewer states than the Red and Blue (RB) cube. Then, we created a computer code that worked from the solved state of each cube, and figured out many moves it takes to get to the state in order to find a God's Number for each cube. Whichever has the lower number could be argued as the simpler puzzle.

History of the Rubik's Cube

Released in 1974 by Erno Rubik, the Rubik's Cube captivated the minds of mathematicians due to its group structuring. God's number was sought after, and was reduced and reduced until it was proved by Tomas Rokicki and colleagues that there existed cube states that needed 20 moves in order to be solved. [2] Not only are mathematicians enamored with the cube, but the public is as well.There are now competitions to see who can solve cubes the fastest. Some cube experts can solve a cube with their eyes closed, or even while juggling!.In order to learn how to solve a cube, many people memorize sets of moves, and can apply those sets over and over again to solve the cube. While one objective is to find the distinct sets of moves for our unique cubes, it is important to fully understand the original Rubik's Cube fully first. We began by understanding the number of solvable states the cube could reach.

We begin counting the possible states of the Original Rubik's cube by first thinking as if we had dismantled the cube, and are counting the ways we could reconstruct it again. We started by determining the number of placements the

corner pieces could take. Since there are 8 corner pieces, we count the placement of the corner cubies as $8! = 40,320$. We then must account for the orientations of the corners. Since each of the eight corner pieces can be oriented in three distinct ways within each placement, we count the orientation of the corner pieces as $3^8 = 6561$. Now accounting for the 12 edge pieces, we must start by counting the number of ways we can place them on the cube. We can count the number of placements of these edge pieces by taking the number of possible orientations, $12! = 479,001,600$.

Taking into consideration the orientation of these pieces, Since each cubie can only be situated in two ways within a certain spot, we have $2^{12} = 4,096$ orientations of the edge pieces. Combining all of the previous arguments, we find that there are $8! \cdot 3^8 \cdot 12! \cdot 2^{12} = 519,024,039,293,878,272,000$ possible constructions of the Rubik's Cube.

God's Number

God's Number is the maximum number of moves required to optimally solve any of the 43,252,003,274,489,856,000 combinations of the original cube. The term "God's Number" was coined because in order for a cube to be solved optimally in 20 moves, the cube would need to be handed to an all knowing being, which would know how to solve the cube optimally.

However, God's Number assumes the user is completing the cube in the half turn metric, in which any turn of a face is considered to be one move. God's number varies depending on the turn metric being used. For example, it is 26 in the quarter turn metric, which states that any 90^o turn of a face is considered to be one move. While the previous two metrics are the most studied, there is another. The half-slice metric is the final metric that counts any turn of the middle slices count as one move. In the quarter turn metric, where any turn of a face counts as a valid move, the moves are named after faces, So F denotes a 90° clockwise turn of the front face, F^2 denotes a 180° turn, and F' denotes the inverse of F, or a 90° counterclockwise turn of the front face. The other face turns are noted as R, L, U, D, B for right, left, up, down, and back, respectively.

It took 29 years for mathematicians to discover God's number. In July 1981, Morwen Thistlethwaite proved there is an upper bound of at most 52 moves to solve every cube state, with a lower bound of 18 moves. In January 1995, Michael Reid proved the "superflip" position, which is where the corners are correct but the edges placed are flipped, requires exactly 20 moves to optimally solve, which raised the lower bound of God's Number to 20. Then finally in July 2010, Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge proved that God's Number for the Cube is exactly 20. [2]

Our particular problem is both pedagogical and practical. It helps us further understand the cube by understanding its group structuring. It also allows us to ask the question, "'Is there an intermediate version of this puzzle?"' If

there is an intermediate version, how simple is it in practice, and will it still make a challenging, but easier, puzzle to tackle? To evaluate this question, we looked at the number of solvable states each bi-colored cube can reach, as well as determinig the "'God's Number"' for them as well.

Group Theory and the Rubik's Cube

From the beginning, the Rubik's Cube was studied for its group structure. A group consists of a set G of elements and a binary operation (called the group operation) that together satisfy the four fundamental properties of closure, associativity, the identity property, and the inverse property. The operation with respect to which a group is defined is often called the "group operation," and a set is said to be a group "under" this operation. We will now prove that the Rubik's Cube is a group.

Let the rotations of the cube's faces, denoted as R, L, F, U, B, and D, be the elements of G. We will now think of a Rubik's Cube as a group $(G, *)$. Two moves are considered the same if they result in the same cube state, Therefore, F' and $(F^2 * F)$ are the same move. The group operation is defined by if M_1 and M_2 are moves, then $M_1 * M_2$ means perform the move M_1 first, followed by the move M_2. To prove this is a group, we must show that $(G, *)$ is closed, associative, has an identity element, and has inverses.

First, $(G, *)$ is closed because any combination of moves is also a move, as it produces a valid cube state. Next, the identity is the move that does nothing. In this way, $M_1 * e = M_1$, which says to do move M_1 and then do nothing, is the same as simply doing M_1. Therefore, there exists a right identity and since $e * M_1 = M_1$, $(G, *)$ has a left identity as well.

Next, $(G, *)$ has inverses because to undo any move M, one can reverse those steps to get a move M^{-1}. Then $M * M^{-1} = e$, therefore M^{-1} is the inverse of M. Therefore, every element in $(G, *)$ has a right inverse. In the same way, if you perform the "undo move for M" on a cube, and then perform M, you will also end up with the identity, and therefore every element of $(G, *)$ has a left inverse.

Finally, we must prove $*$ is associative. Let C be a cube state. Then we can write $M(C)$ as the resulting state after performing the move M on C. If M_1 and M_2 are two moves, then $M_1 * M_2$ is the move where we do M_1 first, denoted as $M_1(C)$ and then M_2, written as $M_2(M_1(C))$. Therefore, $(M_1 * M_2)(C) = M_2(M_1(C))$. To show $*$ is associative, we must show that $(M_1 * M_2) * M_3 = M_1 * (M_2 * M_3)$ for any moves M_1, M_2, and M_3. We know from our above calculation that $[(M_1 * M_2) * M_3](C) = M_3 * [(M_1 * M_2)](C) = M_3 * (M_2 * (M_1(C)))$. Therefore, $(M_1 * M_2) * M_3 = M_1 * (M_2 * M_3)$. Thus $*$ is associative.

We have now proven that $(G, *)$, and therefore the Rubik's Cube is in fact a group.

Since the cube is a group, it can be described by using group properties. We have defined a few that are regularly used below.

- Generators-We say that S, a subset of G, generates G if $G = \langle S \rangle$; that is, every element of G can be written as a finite product under the group operation of elements of S and their inverses. In this case, the subset that generates G is the set of all face turns.
- Cyclic- A group is cyclic if there $\exists M \in G$ such that $G = \langle M \rangle$. Let G be a finite group and $M \in G$, then $M^{-1} = M^n$ for some $n \in Z^+$. You can see this in the Rubik's cube by noting how turning the Front face counter clockwise once is the same as turning it clockwise three times. Therefore, $F^{-1} = F^3$.
- Face turns are bijections, which is a function that is both one to one and onto. Face turns are one to one because no two cubies can be in the same cubicle, and they are onto because after any turn, every cubicle is filled.
- Disjoint cycle decomposition. Two cycles are disjoint if they do not permute any of the same cubies. Since each face move will permute a maximum of 8 cubies, some moves will be disjoint, moving completely seperate cubies. This then also means that two disjoint moves, M_1 and M_2, when composed together are commutative, so $M_1 * M_2 = M_2 * M_1$.
- Homomorphisms- a homomorphism from G to H is a map $\phi : G \to H$ such that $\phi(a * b) = \phi(a) * \star(b), \forall a, b \in G$. One example of a homomorphism in our cubes is the turn, M on the front face, f, and the back face, b. Then $M(f) \star M(b) = M(f * b)$
- The kernel of a homomorphism is the preimage of a move, so $\phi(M) =$ the permutation of the cube before the move M was applied.

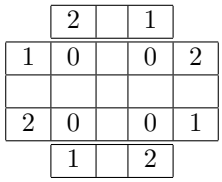
There is some vocabulary that is necessary for speaking of the cube and its properties. First, a cubie is one of the 26 blocks of a Rubik's Cube. Then there is a cubicle, which refers to where a cubie is situated on the cube. It is the space in which a cubie is located. A move of a face takes cubies and transfers them to different cubicles.

The faces of the Rubik's cube are denoted as R, L, U, D, B for right, left, up, down, and back, respectively, similarly to the face moves. Now when speaking about a specific edge cubie, it is noted in two parts, the first being which face the side is on, followed by B, D U R, for what edge of that face the cubie is on. The cubie noted as FR is on the front face, on the right edge. Corner cubies are denoted in three parts, the first being the face they are on, second being either U, D, B, F, to indicate which slice the cubie is on, and finally either L, or R, to indicate which corner of the slice the cubie is on.

The number of valid configurations is determined by four things. Those things are:

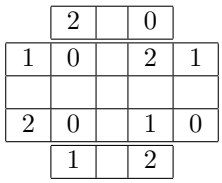
- positions of corner cubies
- positions of edge cubies
- orientations of corner cubies
- orientations of edge cubies

The positions of corner cubies can be described as an element of S_8, which is a permutation of the 8 cubies. The positioning of the edge cubies can be describes as an element of S_12 as there are 12 edge cubies. We must create a way of keeping track of how a cubie's orientation changes after a permutation. I will use the technique Janet Chen uses in her paper, *Group Theory and the Rubik's Cube* [1]. Each corner cubie has three possible orientations, and they can be numbered 0,1, and 2. The solved state has every corner face labeled 0 either on the U or D face. Then, each cubie is labeled clockwise with 1, followed by a 2. On the Down Face, this looks like:

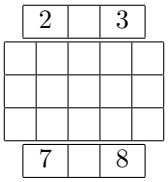


Each corner cubicle is also labeled with a number 1-8. Therefore we now have a notation to identify where a cubie begins in the solved state, and where it ends up after a permutation is applied to the cube. This notation will be written as $(x_1, x_2, x_3, ..., x_8)$ with x_i denoting the number of the cubie (0, 1, or 2) that is in the i^{th} cubicle. In this way, the elements x_i are elements of $\mathbb{Z}/3\mathbb{Z}$.

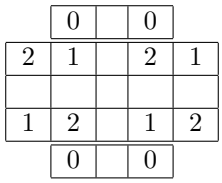
We will now look at what it looks like, with this way of tracking the cube's permutations, when the move R is applied to the cube. The cubicle numbers on this face are



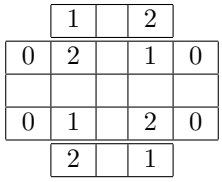
The labeling of the corner cubies on the right face looks like



Then the cubie faces in the cubicles are labeled as



After rotating the Right Face by 90 degrees, the cubies are labeled



The cubies on the left face are unmoved by R, so $x_1 = 0, x_4 = 0, x_5 = 0$,and $x_6 = 0$. Now, we can see from our diagrams that $x_2 = 1, x_3 = 2, x_7 = 2$, and $x_8 = 1$. We can think of each x_i as counting the number of clockwise twists the cubie i is away from having its 0 face on the numbered face of the cubicle. So,$x = (0, 1, 2, 0, 0, 0, 2, 1)$. Each x_i is an element of $\mathbb{Z}/3\mathbb{Z}$, and x is an element of $(\mathbb{Z}/3\mathbb{Z})^8$. It is also important to note that $\sum x_i = 6 = 0 mod 3$.

We can do something similar with the edge pieces, by labeling the cubicles 1-12 and labeling the edges of each cubie 0,1. Our edge cubies will be denoted as y_i, where $1 \leq i \leq 12$. So a configuration $y = (y_1, y_2, ..., y_{12})$ is an element $\tau \in (\mathbb{Z}/2\mathbb{Z})^{12}$.

Therefore, any configuration of the Rubik's cube can be described as (σ, τ, x, y).

Theorrem 11.1 A configuration (σ, τ, x, y) is valid iff $\sum x_i = 0(mod 3), \sum y_i = 0(mod 2)$, and sign of σ is equal to the sign of τ.

We will follow Janet Chen's proof of this theorem.
In order to prove this theorem, we will show that if (σ, τ, x, y) is valid, then $sgn\sigma = sgn\tau$, $\sum x_i = 0(mod 3)$, and $\sum y_i = 0(mod 2)$. In the process, we will prove some other, more general facts that will be useful.

Recall that $\mathbb{G}$ acts on the set of configurations of the Rubik's Cube. The valid congurations form a single orbit of this action. So, it makes sense that

statements we make about valid congurations can be generalized to other orbits.
Lemma 11.2. If (σ, τ, x, y) and (σ', τ', x', y') are in the same orbit, then $(sgn\sigma)(sgn\tau) = (sgn\sigma')(sgn\tau')$.
Proof. It suffices to show that, if $(\sigma', \tau', x', y') = (\sigma, \tau, x, y) \cdot M$ where M is one of the 6 basic moves, then $(sgn\sigma)(sgn\tau) = (sgn\sigma')(sgn\tau')$. $\sigma' = \sigma\phi_{corner}(M)$ and $\tau' = \tau\phi_{edge}(M)$. Therefore, $(sgn\sigma')(sgn\tau') = (sgn\sigma)(sgn\phi_{corner}(M))(sgn\tau)(sgn\phi_{edge}(M))$. If M is one of the 6 basic moves, then $\phi_{corner}(M)$ and $\phi_{edge}(M)$ are both 4-cycles, so they both have sign 1. Thus, $(sgn\sigma')(sgn\tau') = (sgn\sigma)(sgn\tau)$.
Corollary 11.3. If (σ, τ, x, y) is a valid conguration, then $sgn\sigma = sgn\tau$. Proof. This is a direct consequence of Lemma 11.2 since any valid conguration is in the orbit of the start conguration (1,1,0,0).

Lemma 11.4. If (σ', τ', x', y') is in the same orbit as $(\sigma, \tau, x, y))$, then $\sum x_i' \sum x_i (mod 3)$. and $\sum y_i' = \sum y_i (mod 2)$

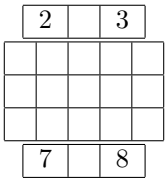
Proof. In light of Proposition 10.12, it suffices to show that, if (σ', τ', x', y') $=(\sigma, \tau, x, y)\dot{M}$ where M is one of the 6 basic moves, then$\sum x_i' \sum x_i (mod 3)$. and $\sum y_i' = \sum y_i (mod 2)$. Here is a table showing what x0 and y0 are if (0,0,x0,y0) = (,,x,y)·M and M is one of the 6 basic moves. In each case, it is easy to check $\sum x_i' \sum x_i (mod 3)$. and $\sum y_i' = \sum y_i (mod 2)$

M	x' and y'
D	$(x_1, x_2, x_3, x_4, x_8, x_5, x_6, x_7)$
	$(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_{10}, y_{11}, y_{12}, y_9)$
U	$(x_2, x_3, x_4, x_1, x_5, x_6, x_7, x_8)$
	$(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_{10}, y_{11}, y_{12}, y_9)$
R	$(x_1, x_7 + 1, x_2 + 2, x_4, x_5, x_6, x_8 + 2, x_3 + 1)$
	$(y_1, y_7, y_3, y_4, y_5, y_2, y_{10}, y_8, y_9, y_6, y_{11}, y_{12})$
L	$(x_4 + 2, x_2, x_3, x_5 + 1, x_6 + 2, x_1 + 1, x_7, x_8)$
	$(y_1, y_2, y_3, y_5, y_{12}, y_6, y_7, y_4, y_9, y_{10}, y_{11}, y_8)$
F	$(x_6 + 1, x_1 + 2, x_3, x_4, x_5, x_7 + 2, x_2 + 1, x_8)$
	$(y_1, y_2, y_8 + 1, y_4, y_5, y_6, y_3 + 1, y_{11} + 1, y_9, y_{10}, y_7 + 1, y_{12})$
B	$(x_1, x_2, x_8 + 1, x_3 + 2, x_4 + 1, x_6, x_7, x_5 + 2)$
	$(y_6 + 1, y_2, y_3, y_4, y_1 + 1, y_9 + 1, y_7, y_8, y_5 + 1, y_{10}, y_{11}, y_{12})$

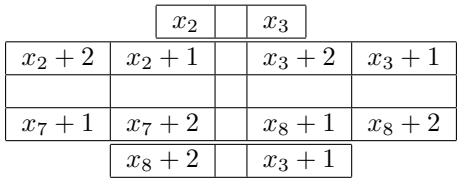
As an example, we'll see how to nd x0 when M is the move R. The cubicles of the right hand face look like this:

	U	U	U	
F	R	R	R	B
F	R	R	R	B
F	R	R	R	B
	D	D	D	

The cubicles are labeled like this:



Therefore, if the Rubik's cube is in the configuration $\sigma, \tau, x, y)$, the cubies on the right face are labeled like this:



Thus, $x' = (x_1, x_7{+}1, x_2{+}2, x_4, x_5, x_6, x_8{+}2, x_3{+}1)$. So,$\sum x'_i = \sum x_i + 6(mod3)$.

Corollary 11.5. If (σ, τ, x, y) is a valid conguration, then$\sum x_i = 0(mod3) and \sum y_i = 0(mod2)$.

Proof. This is a direct consequence of Lemma 11.4 since any valid conguration is in the orbit of the start conguration (1,1,0,0). Thus, we have proved one directon of Theorem 11.1. Now, we will prove the converse. Suppose $sgn\sigma = sgn\tau, \sum x_i = 0(mod3) and \sum y_i = 0(mod2)$. We want to show that there is a series of moves which, when applied to (σ, τ, x, y), gives the start conguration; that is, if the Rubik's cube is in the conguration (σ, τ, x, y), it can be solved. The idea of the proof is basically to write down the steps required to solve the Rubik's cube. Thus, we will prove these four facts:

- If (σ, τ, x, y) is a conguration such that $sgn\sigma = sgn\tau, \sum x_i = 0(mod3)$ and $\sum y_i = 0(mod2)$, then there is a move $M \in \mathbb{G}$ such that $(\sigma, \tau, x, y) \cdot M$ has the form $(1, \tau', x', y')$ with $sgn\tau' = 1, \sum x_i = 0(mod3) and \sum y_i = 0(mod2)$. That is, we can put all the corner cubies in the right positions.
- If (σ, τ, x, y) is a conguration with $sgn\tau = 1, \sum x_i = 0(mod3) and \sum y_i = 0(mod2)$, then there is a move $M \in \mathbb{G}$ such that $(1, \tau', x', y') \cdot M$ has the form $(1, \tau', 0, y')$ with $sgn\tau' = 1$ and $\sum y'_i = 0(mod2)$. That is, we can put all the corner cubies in the right orientations (and positions).
- If $(1, \tau, 0, y)$ is a conguration with $sgn\tau = 1$ and$\sum y_i = 0(mod2)$, then there is a move $M \in \mathbb{G}$ such that $(1, \tau, 0, y) \cdot M$ has the form (1,1,0,y') with $\sum y'_i = 0(mod2)$. That is, we can put all the edge cubies in the right positions (without disturbing the corner cubies).
- If (1,1,0,y) is a conguration with $\sum y_i = 0(mod2)$, then there is a move $M \in \mathbb{G}$ such that $(1, 1, 0, y) \cdot M = (1, 1, 0, 0)$. That is, we can solve the cube!

Before proving these, let's point out a useful fact. Suppose that (σ, τ, x, y) satises $sgn\sigma = sgn\tau, \sum x_i 0(mod 3), and \sum y_i 0(mod 2)$. Then, Lemma 11.2 and 11.4 show that, for any (σ', τ', x', y') in the same orbit as $(\sigma, \tau, x, y), sgn\sigma' = sgn\tau', \sum x_i' 0(mod 3), and \sum y_i' 0(mod 2)$. Thus, for example, in the rst statement above, if we can prove that there is a move $M \in \mathbb{G}$ such that $(\sigma, \tau, x, y) \cdot M$ has the form $(1, \tau', x', y')$, it is automatic that $sgn\tau' = 1, \sum x_i' 0(mod 3), and \sum y_i' 0(mod 2)$. Therefore, to nishthe proof of Theorem 11.1, it suffices to prove the following four propositions.

Proposition 11.6. If (σ, τ, x, y) is a conguration such that $sgn\sigma = sgn\tau$, $\sum x_i 0(mod 3), and \sum y_i 0(mod 2)$, then the orbit of (σ, τ, x, y) contains some conguration of the form $(1, \tau', x', y')$.

Proposition 11.7. If $(1, \tau, x, y)$ is a conguration with $sgn\tau = 1, \sum x_i 0(mod 3)$, and $\sum y_i 0(mod 2)$, then the orbit of $(1, \tau, x, y)$ contains some conguration of the form $(1, \tau', 0, y')$.

Proposition 11.8. If $(1, \tau, 0, y)$ is a conguration with $sgn\tau = 1$ and $\sum y_i 0(mod 2)$, then the orbit of $(1, \tau, 0, y)$ contains some conguration of the form $(1, 1, 0, y)$.

Proposition 11.9. If $(1, 1, 0, y)$ is a conguration with $\sum y_i 0(mod 2)$, then the orbit of $(1, 1, 0, y)$ contains the start conguration $(1, 1, 0, 0)$. We will prove these in order. So, we want to rst show that we can put all the corner cubies in the right positions.

We will prove these in order. So, we want to rst show that we can put all the corner cubies in the right positions.

Lemma 11.10. The homomorphism $\phi_{corner} : G \to S_8$ is onto. Proof. S8 is generated by the set S of 2-cycles in S_8. It suces to show that $S \subset im\phi_{corner}$. After all, if $S \subset im\phi_{corner}$, then $S_8 = \langle S \rangle \subset im\phi_{corner}$. $im\phi_{corner}$ is a group, so $\langle \phi_{corner} \rangle = im\phi_{corner}$.

So, we want to show that every 2-cycle in S_8 is in the image of ϕ_{corner}. One such move that switches just 2 corner cubies, and leaves the other corner cubies fixed is $M_0 = ([D, R]F)^3$, which has disjoint cycle decomposition (dbr urb)(dr uf)(br rf)(df lf). Then, $\phi_{corner}(M_0) = (dbrurb)$. So, we at least know that (dbr urb) lies in the image of ϕ_{corner}.

Let C_1 and C_2 be any pair of corner cubies. There exists a move $Min\mathbb{G}$ which sends dbr to C_1 and urb to C_2. Let $\sigma = \phi_{corner}(M)$. Then, $\sigma(dbr) = C_1$ and $\sigma(urb) = C_2$. Since ϕ_{corner} is a homomorphism,
$\phi_{corner}(M^{-1}M_0M) = \phi_{corner}(M)^{-1}\phi_{corner}(M_0)\phi_{corner}(M)$
$= \sigma^{-1}(dbrurb)\sigma$
$= (\sigma(dbr)\sigma(urb))$
$= (C_1C_2)$
Therefore, $(C_1C_2) \in im\phi_{corner}$, which nishes the proof.

Proof of Proposition 11.6. By Lemma 11.10, there exists a move $M \in \mathbb{G}$ such that $\phi_{corner}(M) = \sigma^{-1}$. $(\sigma, \tau, x, y) \cdot M = (1, \tau', x', y')$ for some $\tau' \in S_1 2, x' \in (\mathbb{Z}/3\mathbb{Z})^8$, and $y' \in (\mathbb{Z}/2\mathbb{Z})^1 2$.
Next, we will prove Proposition 11.7. The basic idea for orienting all of the corner cubies correctly was to use moves which change the orientations of just 2 cubies. First, we must show that such moves exist.
Lemma 11.11. If C_1 and C_2 are any two corner cubies, there is a move $M \in \mathbb{G}$ which changes the orientations (but not positions) of C_1 and C_2 and which does not affect the other corner cubies at all. Moreover, there is such a move M which rotates C_1 clockwise and rotates C_2 counterclockwise.

Proof. As in the proof of Lemma 11.10, the point is to rst nd a single move M' which changes the orientations of 2 cubies and then conjugate M' to nd other moves that change the orientations of 2 cubies.

One possibility is $M' = (DR^{-1})^3(D^{-1}R)^3$, which has disjoint cycle decomposition (dfr rdf frd)(drb rbd bdr)(df dr fr ur br db dl). Then, $\phi_{corner}(M') = 1$ and $\psi_{corner}(M')$ = (dbr rdb brd)(drf rfd fdr). So, if C_1 = dbr and C_2 = drf, the lemma is true.

Now, we will conjugate this move. There exists $M \in \mathbb{G}$ which sends dbr to C_1 and drf to C_2. Let $M' = M^1 M' M$. We see that M' changes the orientations of C_1 and C_2 and does not affect the other corner cubies. Specically, M' rotates C_1 clockwise and rotates C_2 counterclockwise.

Proof of Proposition 11.7. Suppose that the Rubik's cube is in a conguration where at least two corner cubies C_1 and C_2 have the wrong orientation. By Lemma 11.11, there is a move which rotates C_1 clockwise, rotates C_2 counterclockwise, and does not affect the other corner cubies. By applying this move once or twice, we can ensure that C_1 has the correct orientation. Since this move does not aect any corner cubies besides C_1 and C_2, the Rubik's cube now has one fewer corner cubie with an incorrect orientation. Doing this repeatedly, we end up with a conguration $(1, \tau', x', y')$ where there is at most one corner cubie with the incorrect orientation. That is, at least 7 of the x_i' are 0. By Lemma 11.4, $\sum x_i' \sum x_i 0 (mod 3), so it must be the case that the last x_i'$ is also 0, so the conguration of the Rubik's cube is $(1, \tau', 0, y')$.

Next, we want to prove Proposition 11.9; that is, we want to x the positions of the edge cubies. The idea of the proof is very similar to the one we used to prove Proposition 11.7. Recall that, in that case, we rst proved that $\phi_{corner} : \mathbb{G} \to S_8$ is onto. In this case, we only want to use moves that don't aect the corner cubies, since we have already done a lot of work to get the corner cubies in the right positions and orientations. Therefore, instead of looking directly at ϕ_{edge}, we will look at the restriction of $_{edge}$ to $ker\psi_{corner}$.
Lemma 11.12. The image of $\phi_{edge}|ker\psi_{corner} : ker\psi_{corner} \to S_1 2$ contains

A12.
Proof: A_12 is generated by the set of 3-cycles in A_12. By the same argument as in the proof of Lemma 11.10, it suces to show that every 3-cycle is in the image of $\phi_{edge}|ker\psi_{corner}$. As in the proof of Lemma 11.10, the strategy is to use conjugates of a single move to prove this.
One move that does not affect any corner cubies but cycles 3 edge cubies is $M_0 = LR^1U^2L^1RB^2$, which has disjoint cycle decomposition (ub uf db). Then, $M_0 \in ker\psi_{corner}$, and $\phi_{edge}(M_0) =$ (ub uf db). If C_1, C_2, and C_3 are any 3 corner cubies, there is a move M of the Rubik's cube which sends ub to C_1, uf to C_2, and db to C_3. Then, $M' = M^1M_0M$ has disjoint cycle decomposition $(C_1C_2C_3)$, so $M' \in ker\psi_{corner}$ and $\phi_{edge}(M') = (C_1C_2C_3)$. Therefore, $(C_1C_2C_3) \in im\phi_{edge}|ker\psi_{corner}$, which completes the proof.

Remark 11.13. In fact, the image of $\phi_{edge}|ker\psi_{corner} : ker\psi_{corner} \to S_12$ is exactly A_12, which you can prove using Corollary 11.3.
Now, Proposition 11.8 follows directly from Lemma 11.12. (The proof is exactly the same idea as the proof of Proposition 11.6.)
Finally, we must prove Proposition 11.9. This is quite similar to Proposition 11.7; rst, we need an analog of Lemma 11.11.
Lemma 11.14. If C_1 and C_2 are any two edge cubies, there is a move $M \in \mathbb{G}$ which changes the orientations (but not positions) of C_1 and C_2 and which does not affect the other cubies at all.

Proof. One move that switches the orientations of 2 edge cubies without affecting other cubies is
$(LR^1FLR^1DLR^1BLR^1ULR^1F^1LR^1D^1LR^1B^1LR^1U^1$
(this move is described more easily as $(M_RU)^4(MRU^1)^4$). Call this move M_0; it has disjoint cycle decomposition (fu uf)(bu ub). G acts transitively on the set of ordered triples (C_1, C_2, C_3) where C_1, C_2, and C_3 are different edge cubies. In particular, if C_1 and C_2 are any two different edge cubies, there exists $M \in \mathbb{G}$ sending uf to C_1 and ub to C_2. MM_0M^1 changes the orientations of C_1 and C_2 and does not affect the other cubies at all.
Now, the argument we used to prove Proposition 11.7 proves Proposition 11.9 as well. This completes the proof of Theorem 11.1.

Remark 11.15. Earlier, we calculated that there were 519,024,039,293,878,272,000! possible congurations of the Rubik's cube; now, Theorem 11.1 tells us that only 1 12 of those are valid. This means the total number of solvable positions for the original cube is $\frac{519,024,039,293,878,272,000}{12} = 43,252,003,274,489,856,000$.
Which is a lot of configurations!

Group Structuring in the Bi-colored Cubes

The Green and Yellow cube is a physical representation of the Klein 4 group. This group is the unique non-cyclic group of order four. It has three elements of degree two, along with the fourth element which is the identity. It is produced

when we combine the cyclic group of order two of rotating the cube 180° around the z axis, with the cyclic group of order two of rotating the cube 180° around the y axis, and then flipping the cube around the z axis 90° (Can you argue that this element is also of order two because of the symmetry the cube has?).

The Red and Blue cube is a physical representation of the S_3 group. This is the set of permutations of a group of three. We have an element of order 3 which is produced when the cube is turned 120° along a diagonal. We also have an element order 2 which is made by flipping the cube along this diagonal. These two elements are not commutative, and so therefore are not abelian. This is why we get the group S_3, which has 6 elements.

Counting Arguments

Green and Yellow Cube

We began counting the possible states of the Green and Yellow cube by first thinking as if we had dismantled the cube, and are thinking of the ways we could reconstruct it again. We started by determining the number of placements the corner cubies could take. Since there are 8 corner cubies, and since four of the corners have two yellow sides with one green side, and the other four cubies have two green sides with one yellow side, once the placement is decided for one group of cubies, the placement of the other four does not matter. In this way, we count the placement of the corner cubies as $\frac{8!}{4!4!} = 70$

We then must account for the orientations of the corner cubies. Since each of the eight corner pieces can be oriented in three distinct ways within each placement, we count the orientation of the corner pieces as $3^8 = 6561$.

Now accounting for the edge pieces, we must start by counting the number of ways we can place them on the cube. We have twelve edge pieces, two of them being all yellow, two of them being all green, and the remaining eight having one yellow side and one green side. We can count the number of placements of these edge pieces by taking the number of possible orientations, 12! and dividing by the orientations of each of the similar groups of pieces. And so we have $\frac{12!}{8!2!2!} = 2970$ possible placements of the edge pieces.

Taking into consideration the orientation of these pieces, we must recognize that four cubies, the two all yellow edge pieces and two all green edge pieces, will not create a new cube state if they are flipped. This means we only need to account for the edge pieces that when flipped, will create a new cube state. Since each cubie can only be situated in two ways within a certain spot, we have $2^8 = 256$ orientations of the edge pieces. Combining all of the previous arguments, we find that there are
$\frac{8!\cdot 3^8 \cdot 12!\cdot 2^8}{4!\cdot 4!\cdot 8!\cdot 2!\cdot 2!\cdot 2} = 174,596,083,200$
possible constructions of the Green and Yellow Rubik's Cube.

It must be noted that not all of the aforementioned constructed states will be solvable. In the original Rubik's Cube, one cannot reconstruct the cube back into the solved state with the exemption of one piece being in the correct position, but incorrect orientation. However, in the Green and Yellow cube, if one edge piece is in the correct position but with the incorrect orientation, we can make this cube solvable again by simply flipping the orientation of another edge piece. This edge piece must be carefully chosen, and by choosing this piece to be one of the all green or all yellow sides, we are able to make the cube solvable, without actually changing the state of the cube, therefore, cube states with all of the edge pieces in the correct positions and with one piece being in the incorrect orientation, are solvable.

Another limitation the original cube has is that if the cube is solved with the exemption of two edge pieces being swapped, two other edge pieces must be swapped as well in order for the cube to be solvable. Applying this to the Green and Yellow cube, if two edge pieces are swapped, with all of the rest of the cube being solved, we can make this cube solvable by creatively choosing two other edge pieces, namely the two all yellow pieces or the two all green pieces, to swap with eachother to make the cube solvable. In this way, the Green and Yellow cube will be solvable if only two edge pieces are swapped, where the original Rubik's Cube is not.

The original Rubik's Cube is also not solvable if it is solved save for one corner, while in its correct placement, has the wrong orientation. There must be at least one other corner with the wrong orientation in order for the cube to be solvable. Since we cannot subtly switch the orientation of a corner piece as we could an edge piece, this counts for our total constructions being reduced by a factor of three. This means our total number of solvable positions for the Green and Yellow cube is $\frac{174,596,083,200}{3} = 58,198,694,400$.

Red and Blue Cube

We approached counting the number of possible states the Red and Blue cube could achieve in much the same way as we approached the Green and Yellow cube. Starting by determining the number of placements the corner cubies can take, we realize that of the 8 corner cubies, three of them have two red sides with one blue side, three have two blue sides with one red side, and there is one corner with all sides being blue and one corner with all red sides. We must account for the placements of the three corners with two blue sides and one red side, as well as the placement of the three corners with two red sides and one blue side. In this way, we count the placement of the corner cubies as $\frac{8!}{3!3!} = 1120$

We then must account for the orientations of the corner cubies. Since only six of the eight corner pieces can be oriented in three distinct ways within each

placement, we count the orientation of the corner pieces as $3^6 = 729$.

Now accounting for the edge pieces, we must start by counting the number of ways we can place them on the cube. We have twelve edge pieces, six of them having one blue side and one red side, three of them being all blue, and the remaining three being all red. We can count the number of placements of these edge pieces by taking the number of possible orientations, 12! and dividing by the orientations of each of the similar groups of pieces. And so we have $\frac{12!}{6!3!3!} = 18,480$ possible placements of the edge pieces.

Taking into consideration the orientation of these pieces, we must recognize that four cubies, the three all red edge pieces and three all blue edge pieces, will not create a new cube state if they are flipped. This means we only need to account for the edge pieces that when flipped, will create a new cube state. Since each cubie can only be situated in two ways within a certain spot, we have $2^6 = 64$ orientations of the edge pieces. Combining all of the previous arguments, we find that there are $\frac{8!\cdot 3^6\cdot 12!\cdot 2^6}{3!\cdot 3!\cdot 6!\cdot 3!\cdot 3!\cdot 3} = 321,889,075,200$ possible constructions of the Red and Blue Rubik's Cube.

It must be checked again to see if all of the cube states found will be solvable. We must again check that if the cube that is reconstructed back into the solved state with the exemption of one piece being in the correct position, but incorrect orientation, will be solvable. If one edge piece is in the correct position but with the incorrect orientation, we can make this cube solvable again by simply flipping the orientation of another edge piece. This edge piece, in the Red and Blue cube can be carefully chosen to be one of the all red or all blue sides, and thus we are able to make the cube solvable, without actually changing the state of the cube, therefore, cube states with all of the edge pieces in the correct positions and with one piece being in the incorrect orientation, are solvable.

Again, if the original Rubik's Cube is solved with the exemption of two edge pieces being swapped, two other edge pieces must be swapped as well in order for the cube to be solvable. Applying this to the Blue and Red cube, if two edge pieces are swapped, with all of the rest of the cube being solved, we can make this cube solvable by creatively choosing two other edge pieces, namely two of the three all red pieces or two of the three all blue pieces, to swap with eachother to make the cube solvable. In this way, the Red and Blue cube will be solvable if only two edge pieces are swapped, where the original Rubik's Cube is not.

Finally, we must compare our Red and Blue Rubik's Cube against the original Rubik's Cube in that it is not solvable if it is solved save for one corner, while in its correct placement, has the wrong orientation. There must be at least one other corner with the wrong orientation in order for the cube to be solvable. Here, we can again intentionally choose the other corner to be switched as either of the two corners where all three sides are of a single color. This again makes

our cube solvable without changing the cube state, which means that in any way the Red and Blue Cube is taken apart and reconstructed, every resulting cube state will be solvable.

Computing God's Number

The original idea of the computer program created was to have it take a solved cube, perform all eighteen permutations in the half-tern metric on that cube, and save each resulting state to a file. We also wanted the program to be able to compare states within the new file to previous ones in order to eliminate any duplicates that would arise. Then, the program would take every newly occurring state that is saved in this file, perform the eighteen moves on each of them, save the resulting states, delete duplicated states and so on. Each new resulting file would be considered a "depth" with the solved state being depth zero, the first file being depth 1, as each state within that file is exactly one move away from the solved state, and so on. When finally a distance N file returns zero states, we will know it is only possible to be N-1 moves away from the solved state. As mentioned previously, the Original Rubik's Cube has a depth of 20, and finding the respective depths, and size of these depths of our cubes will help in determining which is easier.

In order to transform our three dimensional cubes into something our program could easily permute, we split the corner and edge cubies into separate groups, and labeled each side of the cubies in each group from 0 to 23, and created a sequence of 48 bits where each side of each cubie corresponded with either a 0 or 1 in the sequence, where 0 and 1 each represented a color. When the program receives a sequence, it permutes the bit sequence according to the 18 permutations, or moves allowed within the half-turn metric, and the resulting sequences represent the resulting cube states.

The program also runs each new cube state through a series of symmetries. One symmetry is color inversion. This is because if you are given two cubes with the same orientations, but with the colors inverted, it would take the same moves to solve both cubes. Our program counts each of these cube states as being the same. It also takes each cube through a series of symmetries associated with it.

The first symmetries we check on the Red and Blue Cube are by rotating the physical cube along a diagonal. This is done by rotating the cube on the y-axis counter-clockwise 90°, and then rotating the cube along the x-axis clockwise 90°, this has an order of three. Then we look at flipping the orientation of the red and blue colors, which is a 180° rotation on the y-axis, and then a 90° counter-clockwise rotation of the cube on the x-axis, which has order two. We then must run through the diagonal rotations on these newly flipped cubes. This accounts for six cube states. Now we note the inversion of each of these cube states. Whichever of the 12 cube states' bit strings is the smallest, the

program saves.

The Green and Yellow cube goes through different state checks. The first is the symmetric 180° flip along the Z axis. Next, we check the cube state that physically switches the colors, created by rotating the cube 180° around the y axis, and then flipping the cube around the z axis 90°. We perform this rotation on the cube, as well as run the resulting cube through the symmetric fl ip.This gives us four resulting states. Finally, we perform a color inversion on each of these four cubes, and the program saves the smallest of the eight bit strings.

Results

The following table maps out how many new cube states the program found, and at what depth the states were found. For example, there are only 4 cube states that are one move away from the solved Yellow and Green Cube, and there are 48 new states that can be achieved with two permutations of the cube.

Depth	YG Cube	BR Cube
0	1	1
1	4	3
2	48	42
3	593	540
4	7,327	7,199
5	92,007	95,189
6	1,154,866	1,250,962
7	14,418,359	16,333,778
8	199,293,154	210,637,384
9	1,769,851,087	2,582,582,687
10	8,359,388,826	Not found yet
States Found	10,344,206,272	

Taking into account the number of states we have found, along with the size of the depths, it seems as though the Green and Yellow cube is the "easier" cube. With a wider tree and fewer positions to account for, we are assuming that it will generally take fewer moves to solve than it will the red and blue cube. The next step in our research will be to modify the optimal solution solver that Herbert Kociemba created, to work on our cubes and determine which cube states take the most moves to solve. This means that the Green and Yellow cube is the more intermediate level of cube. Therefore, if someone is not willing to tackle the complexities of the Origninal Rubik's Cube, but still wants a sufficient challenge, the Green and Yellow cube would sufficiently fit that need.

Code

The following code is the file containing all of the functions called in the proceeding files.

```
  unsigned long int bit_permute_step(unsigned long int x, unsigned long int m, int shift)
  {
int t;
t = ((x >> shift) ^ x) & m;
x = (x ^ t) ^ (t << shift);
return x;
  }


//The most significant 16 bits are zeros
//The next 24 significant bits represent the edge pieces
//The least 24 significant bits represent the corner pieces

unsigned long long int Front1(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x= (x & 0x00c333ff)
| ((x & 0x00000800) << 9)
| ((x & 0x00000400) << 11)
| ((x & 0x00080000)>> 5)
| ((x & 0x0000c000) >> 4)
| ((x & 0x00040000) >> 3)
| ((x & 0x00300000) >> 2);


   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x00e07e07)
| ((x & 0x00000020) << 13)
| ((x & 0x00000018) << 16)
```

```
| ((x & 0x00020000) >> 11)
| ((x & 0x00018000) >> 8)
| ((x & 0x00180180) >> 4)
| ((x & 0x00040040) >> 1);


    newstate=newstate+x;

 return newstate;
}

  unsigned long long int Front2(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00c333ff)
  | ((x & 0x00008000) << 5)
  | ((x & 0x00004800) << 7)
  | ((x & 0x00000400) << 9)
  | ((x & 0x00080000) >> 9)
  | ((x & 0x00240000) >> 7)
  | ((x & 0x00100000) >> 5);


newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;


x = (x & 0x00e07e07)
  | ((x & 0x000001f8) << 12)
  | ((x & 0x001f8000) >> 12);

    newstate=newstate+x;


 return newstate;
}
```

```
    unsigned long long int Front3(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00c333ff)
  | ((x & 0x00000800) << 9)
  | ((x & 0x00000400) << 11)
  | ((x & 0x00080000) >> 5)
  | ((x & 0x0000c000) >> 4)
  | ((x & 0x00040000) >> 3)
  | ((x & 0x00300000) >> 2);


newstate=x;
    newstate=newstate<<24;

x=cube&0x00FFFFFF;


x = (x & 0x00e07e07)
  | ((x & 0x00020020) << 1)
  | ((x & 0x00018018) << 4)
  | ((x & 0x00000180) << 8)
  | ((x & 0x00000040) << 11)
  | ((x & 0x00180000) >> 16)
  | ((x & 0x00040000) >> 13);


    newstate=newstate+x;


 return newstate;
}

  unsigned long long int Right1(unsigned long long int cube)
{

unsigned long int x;

unsigned long long int newstate=0;

x=cube>>24; //x has edge info
```

```
x = (x & 0x00fcc0ff)
  | ((x & 0x00000f00) << 2)
  | ((x & 0x00002000) << 3)
  | ((x & 0x00001000) << 5)
  | ((x & 0x00020000) >> 9)
  | ((x & 0x00010000) >> 7);;

//x now has the permuted edge info

newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

// x now has info from x

x = (x & 0x001c7e38)
| ((x & 0x00000100) << 7)
| ((x & 0x000000c0) << 10)
| ((x & 0x00c10000) >> 16)
| ((x & 0x00020000) >> 15)
| ((x & 0x00008000) >> 14)
| ((x & 0x00200000) >> 13)
| ((x & 0x00000002) << 20)
| ((x & 0x00000004) << 21)
| ((x & 0x00000001) << 22);

// x now has permuted corner info

    newstate=newstate+x;

// newstate now has corner info from x

 return newstate;
}

  unsigned long long int Right2(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24; //x has edge info

x = (x & 0x00fcc0ff)
```

```
  | ((x & 0x00000300) << 4)
  | ((x & 0x00000800) << 5)
  | ((x & 0x00000400) << 7)
  | ((x & 0x00020000) >> 7)
  | ((x & 0x00010000) >> 5)
  | ((x & 0x00003000) >> 4);

  newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

// x now has info from x
x = (x & 0x001c7e38)
  | ((x & 0x00000004) << 5)
  | ((x & 0x00038001) << 6)
  | ((x & 0x00000002) << 7)
  | ((x & 0x00000100) >> 7)
  | ((x & 0x00e00040) >> 6)
  | ((x & 0x00000080) >> 5);

// x now has permuted corner info

    newstate=newstate+x;

// newstate now has corner info from x

 return newstate;

}

   unsigned long long int Right3(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00fcc0ff)
  | ((x & 0x00000200) << 7)
  | ((x & 0x00000100) << 9)
  | ((x & 0x00020000) >> 5)
  | ((x & 0x00010000) >> 3)
  | ((x & 0x00003c00) >> 2);
```

```
   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x001c7e38)
  | ((x & 0x00400000) >> 22)
  | ((x & 0x00800000) >> 21)
  | ((x & 0x00200000) >> 20)
  | ((x & 0x00000100) << 13)
  | ((x & 0x00000002) << 14)
  | ((x & 0x00000004) << 15)
  | ((x & 0x000000c1) << 16)
  | ((x & 0x00030000) >> 10)
  | ((x & 0x00008000) >> 7);


    newstate=newstate+x;

 return newstate;

}


   unsigned long long int Left1(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x0033ffcc)
  | ((x & 0x000c0000) << 4)
  | ((x & 0x00c00000) >> 18)
  | ((x & 0x00000003) << 18)
  | ((x & 0x00000030) >> 4);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;
```

```
x = (x & 0x00e381c7)
  | ((x & 0x00000c20) << 8)
  | ((x & 0x00000008) << 9)
  | ((x & 0x00000010) << 10)
  | ((x & 0x00000200) << 11)
  | ((x & 0x00100000) >> 17)
  | ((x & 0x000c0000) >> 14)
  | ((x & 0x00004000) >> 4)
  | ((x & 0x00001000) >> 3)
  | ((x & 0x00002000) >> 2);


    newstate=newstate+x;

 return newstate;

}

    unsigned long long int Left2(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x0033ffcc)
  | ((x & 0x00c00000) >> 22)
  | ((x & 0x00000030) << 14)
  | ((x & 0x000c0000) >> 14)
  | ((x & 0x00000003) << 22);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x00e381c7)
  | ((x & 0x00004000) << 4)
  | ((x & 0x00002038) << 6)
  | ((x & 0x00001000) << 8)
  | ((x & 0x00100000) >> 8)
  | ((x & 0x00080e00) >> 6)
  | ((x & 0x00040000) >> 4);
```

```
    newstate=newstate+x;

 return newstate;

}

    unsigned long long int Left3(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x0033ffcc)
  | ((x & 0x00000003) << 4)
  | ((x & 0x000c0000) >> 18)
  | ((x & 0x00000030) << 18)
  | ((x & 0x00c00000) >> 4);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x00e381c7)
  | ((x & 0x00000800) << 2)
  | ((x & 0x00000200) << 3)
  | ((x & 0x00000400) << 4)
  | ((x & 0x00000030) << 14)
  | ((x & 0x00000008) << 17)
  | ((x & 0x00100000) >> 11)
  | ((x & 0x00004000) >> 10)
  | ((x & 0x00001000) >> 9)
  | ((x & 0x000c2000) >> 8);


    newstate=newstate+x;

 return newstate;

}
```

```
     unsigned long long int Up1(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00ff0ff0)
  | ((x & 0x00002002) << 1)
  | ((x & 0x00001001) << 3)
  | ((x & 0x0000000c) << 10)
  | ((x & 0x0000c000) >> 14);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x00e071f8)
  | ((x & 0x00038000) << 3)
  | ((x & 0x00000004) << 14)
  | ((x & 0x00000001) << 15)
  | ((x & 0x00000002) << 16)
  | ((x & 0x00000800) >> 10)
  | ((x & 0x001c0200) >> 9)
  | ((x & 0x00000400) >> 8);


    newstate=newstate+x;

 return newstate;

}

     unsigned long long int Up2(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00ff0ff0)
  | ((x & 0x0000000a) << 11)
  | ((x & 0x00000005) << 13)
```

```
  | ((x & 0x0000a000) >> 13)
  | ((x & 0x00005000) >> 11);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x00e071f8)
  | ((x & 0x00000e00) << 6)
  | ((x & 0x00100000) >> 19)
  | ((x & 0x00040000) >> 18)
  | ((x & 0x00080000) >> 17)
  | ((x & 0x00000004) << 17)
  | ((x & 0x00000001) << 18)
  | ((x & 0x00000002) << 19)
  | ((x & 0x00038000) >> 6);


    newstate=newstate+x;

 return newstate;

}

     unsigned long long int Up3(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00ff0ff0)
  | ((x & 0x00000003) << 14)
  | ((x & 0x00003000) >> 10)
  | ((x & 0x00008008) >> 3)
  | ((x & 0x00004004) >> 1);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;
```

```
x = (x & 0x00e071f8)
  | ((x & 0x00000004) << 8)
  | ((x & 0x00000e01) << 9)
  | ((x & 0x00000002) << 10)
  | ((x & 0x00020000) >> 16)
  | ((x & 0x00008000) >> 15)
  | ((x & 0x00010000) >> 14)
  | ((x & 0x001c0000) >> 3);


    newstate=newstate+x;

 return newstate;

}

     unsigned long long int Down1(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x000ffc3f)
  | ((x & 0x00000080) << 15)
  | ((x & 0x00000040) << 17)
  | ((x & 0x00300000) >> 12)
  | ((x & 0x00800000) >> 3)
  | ((x & 0x00000300) >> 2)
  | ((x & 0x00400000) >> 1);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x001f8e07)
  | ((x & 0x00000038) << 3)
  | ((x & 0x000001c0) << 15)
  | ((x & 0x00804000) >> 11)
  | ((x & 0x00402000) >> 9)
  | ((x & 0x00201000) >> 7);
```

```
    newstate=newstate+x;

 return newstate;

}

    unsigned long long int Down2(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x000ffc3f)
  | ((x & 0x00000200) << 13)
  | ((x & 0x000000c0) << 14)
  | ((x & 0x00000100) << 15)
  | ((x & 0x00800000) >> 15)
  | ((x & 0x00300000) >> 14)
  | ((x & 0x00400000) >> 13);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x001f8e07)
  | ((x & 0x00000100) << 4)
  | ((x & 0x00000080) << 6)
  | ((x & 0x00000040) << 8)
  | ((x & 0x00e00000) >> 18)
  | ((x & 0x00000038) << 18)
  | ((x & 0x00004000) >> 8)
  | ((x & 0x00002000) >> 6)
  | ((x & 0x00001000) >> 4);


    newstate=newstate+x;

 return newstate;

}
```

```
    unsigned long long int Down3(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x000ffc3f)
  | ((x & 0x00200000) << 1)
  | ((x & 0x000000c0) << 2)
  | ((x & 0x00100000) << 3)
  | ((x & 0x00000300) << 12)
  | ((x & 0x00800000) >> 17)
  | ((x & 0x00400000) >> 15);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x001f8e07)
  | ((x & 0x00004020) << 7)
  | ((x & 0x00002010) << 9)
  | ((x & 0x00001008) << 11)
  | ((x & 0x00e00000) >> 15)
  | ((x & 0x000001c0) >> 3);

    newstate=newstate+x;

 return newstate;

}

    unsigned long long int Back1(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00fcff03)
  | ((x & 0x0000003c) << 2)
  | ((x & 0x000000c0) << 10)
  | ((x & 0x00030000) >> 14);
```

```
   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x00bfebfd)
  | ((x & 0x00000400) << 2)
  | ((x & 0x00000002) << 9)
  | ((x & 0x00001000) << 10)
  | ((x & 0x00400000) >> 21);

  x=x&0x00FFFFFF;

x = (x & 0x00dfddfb)
  | ((x & 0x00000200) << 4)
  | ((x & 0x00000004) << 7)
  | ((x & 0x00002000) << 8)
  | ((x & 0x00200000) >> 19);

  x=x&0x00FFFFFF;


  /*
x = (x & 0x007fb7fe)
  | ((x & 0x00000800) << 3)
  | rol(x & 0x00804000, 9)
  | ((x & 0x00000001) << 11);
*/

x = bit_permute_step(x, 0x00400400, 1);  // Butterfly, stage 0
x = bit_permute_step(x, 0x00000044, 8);  // Butterfly, stage 3
x = bit_permute_step(x, 0x00000040, 16);  // Butterfly, stage 4
x = bit_permute_step(x, 0x00000004, 4);  // Butterfly, stage 2
x = bit_permute_step(x, 0x00000001, 2);  // Butterfly, stage 1
x = bit_permute_step(x, 0x00000044, 8);  // Butterfly, stage 3
x = bit_permute_step(x, 0x00400400, 1);  // Butterfly, stage 0


  x=x&0x00FFFFFF;

/*
x = (x & 0x001f81f8)
  | ((x & 0x00000400) << 2)
```

```
  | ((x & 0x00000800) << 3)
  | ((x & 0x00000200) << 4)
  | ((x & 0x00000004) << 7)
  | ((x & 0x00002000) << 8)
  | rol(x & 0x00804002, 9)
  | ((x & 0x00001000) << 10)
  | rol(x & 0x00400001, 11)
  | ((x & 0x00200000) >> 19);
  */


    newstate=newstate+x;

 return newstate;

}

    unsigned long long int Back2(unsigned long long int cube)

{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00fcff03)
  | ((x & 0x0000000c) << 4)
  | ((x & 0x00000030) << 12)
  | ((x & 0x00030000) >> 12)
  | ((x & 0x000000c0) >> 4);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

x = (x & 0x001f81f8)
  | ((x & 0x00000006) << 11)
  | ((x & 0x00000e00) << 12)
  | ((x & 0x00000001) << 14)
  | ((x & 0x00004000) >> 14)
  | ((x & 0x00e00000) >> 12)
  | ((x & 0x00003000) >> 11);
```

```
    newstate=newstate+x;


 return newstate;

}

    unsigned long long int Back3(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24;

x = (x & 0x00fcff03)
  | ((x & 0x0000000c) << 14)
  | ((x & 0x00030000) >> 10)
  | ((x & 0x000000f0) >> 2);

   newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

/*
x = (x & 0x001f81f8)
  | ((x & 0x00000004) << 19)
  | rol(x & 0x00000802, 21)
  | ((x & 0x00400000) >> 10)
  | rol(x & 0x00800401, 23)
  | ((x & 0x00200000) >> 8)
  | ((x & 0x00000200) >> 7)
  | ((x & 0x00002000) >> 4)
  | ((x & 0x00004000) >> 3)
  | ((x & 0x00001000) >> 2);
  */

x = bit_permute_step(x, 0x000000e0, 16);
x = bit_permute_step(x, 0x0000006a, 8);
x = bit_permute_step(x, 0x00000200, 2);
x = bit_permute_step(x, 0x00000445, 1);
x = bit_permute_step(x, 0x00001203, 2);
x = bit_permute_step(x, 0x0000040a, 4);
x = bit_permute_step(x, 0x0000004a, 8);
```

```
x = bit_permute_step(x, 0x000000e0, 16);

    x=x&0x00FFFFFF;

    newstate=newstate+x;


 return newstate;

}

  unsigned long long int Symmetry(unsigned long long int cube)
{
unsigned long int x;
unsigned long long int newstate=0;

x=cube>>24; //x has edge info

x = ((x & 0x0000000c) << 4)
  | ((x & 0x00008000) << 5)
  | ((x & 0x00004802) << 7)
  | ((x & 0x00002401) << 9)
  | ((x & 0x00001000) << 11)
  | ((x & 0x00000030) << 12)
  | ((x & 0x00030000) >> 12)
  | ((x & 0x00800000) >> 11)
  | ((x & 0x00480200) >> 9)
  | ((x & 0x00240100) >> 7)
  | ((x & 0x00100000) >> 5)
  | ((x & 0x000000c0) >> 4);

  newstate=x;
    newstate=newstate<<24;

// info from x now resides in newstate

   x=cube&0x00FFFFFF;

// x now has info from x
x = ((x & 0x00000006) << 11)
  | ((x & 0x00000ff8) << 12)
  | ((x & 0x00000001) << 14)
  | ((x & 0x00004000) >> 14)
  | ((x & 0x00ff8000) >> 12)
  | ((x & 0x00003000) >> 11);
```

```
// x now has permuted corner info

    newstate=newstate+x;

// newstate now has corner info from x

//std::cout << std::hex << newstate << "\n";


 return newstate;

}

unsigned long long int Invert(unsigned long long int cube)
{
cube=~cube&0x00FFFFFFFFFFFF;

return cube;
}

 unsigned long long int Smallest(unsigned long long int cube)
 {
 unsigned long long int temp2, temp3, temp4, smallest;

temp2 = Symmetry(cube);
//std::cout << std::hex << temp2 << std::endl;
temp3= Invert(cube);
//std::cout << std::hex << temp3 << std::endl;
temp4=Symmetry(temp3);
//std::cout << std::hex << temp4 << std::endl;

if (cube<temp2)
{
smallest = cube;
}
else
{
smallest = temp2;
}
if (smallest > temp3)
{
smallest = temp3;
}
if (smallest > temp4)
{
smallest = temp4;
```

```
}

return smallest;
}
```

The following program contains the permutations and symmetries for all 18 moves of the Red and Blue Cube.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <bitset>
#include <iomanip>
#include <set>
#include "rubiksfunctions.h"
using namespace std;


int main ()
{

   unsigned long long int rbsolved=0x16FD09978395;
unsigned long long int gysolved=0x9E9D229B29B4;
unsigned long long int emptycube=0;
unsigned long long int onecube=0xFFFFFFFFFFFF;
unsigned long long int temp1;
unsigned long long int temp2;
unsigned long long int temp3;
unsigned long long int temp;

  ofstream outputFile;
  outputFile.open("/../../../data/tlpieper/RBDistance10.txt");

  temp3 = 0;
  std::ifstream inFile("/../../../data/tlpieper/RBD9.txt");
  while(!inFile.eof())
    {
      inFile >> temp;

      outputFile << RBSmallest(Up1(temp))    << endl;
      outputFile << RBSmallest(Up2(temp))    << endl;
      outputFile << RBSmallest(Up3(temp))    << endl;
```

```
      outputFile << RBSmallest(Down1(temp))  << endl;
      outputFile << RBSmallest(Down2(temp))  << endl;
      outputFile << RBSmallest(Down3(temp))  << endl;
      outputFile << RBSmallest(Right1(temp)) << endl;
      outputFile << RBSmallest(Right2(temp)) << endl;
      outputFile << RBSmallest(Right3(temp)) << endl;
      outputFile << RBSmallest(Left1(temp))  << endl;
      outputFile << RBSmallest(Left2(temp))  << endl;
      outputFile << RBSmallest(Left3(temp))  << endl;
      outputFile << RBSmallest(Front1(temp)) << endl;
      outputFile << RBSmallest(Front2(temp)) << endl;
      outputFile << RBSmallest(Front3(temp)) << endl;
      outputFile << RBSmallest(Back1(temp))  << endl;
      outputFile << RBSmallest(Back2(temp))  << endl;
      outputFile << RBSmallest(Back3(temp))  << endl;
  temp3++;
    }
cout << temp3 << endl;
  return 0;
}
```

The following code contains all of the 18 permutations of the Green and Yellow Cube.

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <ctime>
#include <bitset>
#include <iomanip>
#include <set>
#include <string>
#include "rubiksfunctions.h"
using namespace std;

   int main ()
   {

unsigned long long int rbsolved=0x16FD09978395;
unsigned long long int gysolved=0x9E9D229B29B4;
unsigned long long int emptycube=0;
unsigned long long int onecube=0xFFFFFFFFFFFF;
unsigned long long int temp1;
unsigned long long int temp2;
unsigned long long int temp3;
```

```
ofstream outputFile;
outputFile.open("/../../../data/tlpieper/GYDistance10.txt");

temp3 = 0;
std::ifstream inFile("/../../../data/tlpieper/GYD9.txt");
while(!inFile.eof())
{
inFile >> temp1;
temp2= GYSmallest(Down1(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Down2(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Down3(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Up3(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Up2(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Up1(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Right1(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Right2(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Right3(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Left3(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Left2(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Left1(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Back1(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Back2(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Back3(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Front3(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Front2(temp1));
{outputFile << temp2 << endl;}
temp2= GYSmallest(Front1(temp1));
{outputFile << temp2 << endl;}
temp3++;
}
```

```
cout << temp3 << endl;

return 0;
   }
```

Bibliography

[1] Chen, Janet, *Group Theory and the Rubik's Cube.* Available at http://www.math.harvard.edu/ jjchen/docs /Group%20Theory%20and%20the%20Rubik's%20Cube.pdf.

[2] Rokicki, Tomas, *God's Number is 20*,. Available at http://www.cube20.org