Hash Function Implementation for Rubik's Cube Scramble in Multi-Blind Competition

Dhafin Rayhan Ahmad - 13518063
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: dhafindiamle@gmail.com

***Abstract*—In the last few years, people around the world have been trying to improve their knowledge and skills to get advanced on the twisty puzzle Rubik's cube. There were various customizations and modifications have been made to the original six-sided cubic puzzle. Methods on solving the cube also grow to push any existing limitation in the world of speedsolving. As blindfolded solving is introduced to the cubing community, the mechanism of Rubik's cube is getting more covered in studies on the cube itself, including relation possibilities between hash function and the cube.**

Keywords—Rubik's cube; hash function; scramble; multi-blind; competition

I. Introduction

Recently, people have been spending their time on a widely popular twisty puzzle, the Rubik's cube. Since its popularity in 1980s, a lot of puzzle-enthusiast had tried to beat each other in term of solving time. Some even tried to solve the cube in various way, such as one-handed, with feet, or the hardcore one, blindfolded. Several methods have been invented in order to solve the Rubik's cube. Most of the methods combine the advantages of intuition and memorization. On the other hand, the blindfolded solving requires a one to have strong intuition about how the pieces are moving in the Rubik's cube. Therefore, people with great intelligence – notably high IQ point – will most likely to learn blindfolded solving faster than the others. Although, with a lot of practice, anyone can achieve the same ability.

With eyes closed while solving the Rubik's cube, one cannot track on where the pieces are going. So before starting the solve, he should memorize where the pieces should go (with eyes open, of course), and these information are used to determine his turns while on the solving stage. This blindfolded phase requires him to turn the cube very carefully, as a single mistake can already ruin the whole solve. But with a lot of practice, and an appropriate turning technic – also called "fingertricks" – people can smash the blindfolded phase just as fast as a normal solve, without worrying about messing it up. Although, no blindfolded solver can guarantee a one hundred percent of success on that speedy rhythm.

Blindfolded solving – or the abbreviation, blindsolving – methods are keep being developed in years. Some blindsolvers are inventing the easier method on solving the cube blindfolded, which aim to approach more people into blindsolving. While the others, are keep inventing a more advanced one, as an attempt to push their limits down with optimal memorization and execution methods. But what we have to know is that, all these methods are based on a basic concept, often called "piece-tracking". This concept can even be made much simpler with the present of graph representation. With a good understanding on the relation between the two, one can already dig into the world of blindfolded solving.

II. Rubik's Cube, Multi-Blind, and Hash Function

A. Rubik's Cube

The Rubik's cube is a six-sided three-dimensional puzzle, each side usually colored with different colors each other. The cube was invented in 1974 by Hungarian sculptor and professor of architecture Ernő Rubik, and get licensed to be sold by Ideal Toy Corporation in 1980. After this agreement, the puzzle start to spread wide around the world, starting the crazy age of solving the Rubik's cube.

The sides of Rubik's cube are recognized with the difference of sticker color sticked on the cube surface, each one of these six colors: white, yellow, green, blue, red, and orange. A popular coloring scheme is to place two similar colors at the opposite side to each other (e.g., red is opposite to orange), although other forms of scheme are also found in the other part of the world, such as the Japanese-scheme in Japan. Some people even use different colors on the cube, using their own color choice in contrast of the six popular one.

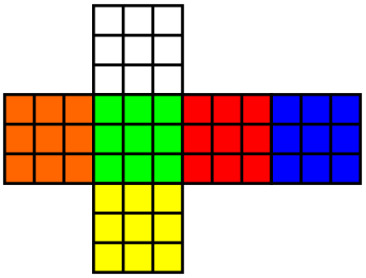

Fig. 1. The standard color scheme for the Rubik's Cube.

The Rubik's cube can be seen as a set of faces, which each side has 9 stickers, resulting in a total of 54 faces. A better way to view the cube is to partition the cube into pieces, which is

more practical in any solving method. This also gives anyone who learns the cube a better understanding of how the mechanisms work. The puzzle is broken down into three main parts, which is the corners, the edges, and centers. On a normal 3x3x3 Rubik's cube, there are 8 corner pieces, 12 edge pieces, and 6 center pieces. A corner piece can be defined as the piece that has three colors on it, edge piece is where the piece has two colors on it, whereas the center piece has only one color attached to it.

Fig. 2. From left to right, showing the Rubik's cube in highlight of: the corner pieces, the edge pieces, and the center pieces.

A corner piece can only go to the place of the other corners, and so for edges and centers. By this means that, for example, an edge piece cannot go to the place of a corner piece. The position of center pieces are not changing to each other by any turn, so they are already in their fixed position. Due to this fact, any turns applied to the cube are just actually edges and corners messing around the center pieces.

Some terms are defined for the Rubik's cube to make discussion easier. In a blindfolded solving world, it is required to be familiar to the terms of Rubik's cube turning, memorizing, and tracking. Notations are used to make the terms even simpler to understand. These terms will be discussed in a later section.

B. Multi-Blind

Multi-Blind, or multiple blindfolded solving, is a term used in competition format where people are trying to do multiple blindsolves at once. Blindsolving, or blindfolded solving, is used to describe the action where someone is attempting to solve the Rubik's cube blindfolded, using the information he got before by observing the cube with eyes open.

Cubers – a term for people who like to play the Rubik's cube and its variant – have developed several methods in solving the Rubik's cube blindfolded. The method of blindfolded solving is different with sighted solving. While on a sighted solve you can see what case you're getting on after applying some moves, in blindfolded solve you have to know where the pieces are going mid-solve.

Fig. 3. A cuber attempting a blindfolded solve – unaware with the situation of his house.
(Source: https://ruwix.com/the-rubiks-cube)

The base concept of doing a sighted solve is to complete the cube layer by layer, and the more advanced methods might do the layers simultaneously. On the other hand, blindsolving has the base concept called "tracking." During the memorization phase, one who do a blindfolded attempt will try to track on two stuffs: where the corners are going, and where the edges are going; each of them started with a piece of his choice as a "buffer." These tracking are translated into the memorization method he prefers, usually by using a letter representation for each sticker, and forms words from them. This memorization later be translated on the solving phase, or the execution phase, by doing algorithms that only affect few pieces once, so it is easier to maintain the tracking of the whole cube.

An example of method used to solve the Rubik's cube blindfolded is the beginner method of blindsolving, *OP/OP*. This execution method was founded by Stefan Pochmann, before he invented his new method for corners which he called R2, and a method for edges named M2. OP is the abbreviation for *Old Pochmann*, called this way as it's Stefan's older method. OP/OP means doing both edges and corners execution with his old methods.

The concept of the method is quite simple, it uses buffer pieces as a starting piece, and then we track on where the piece should go to get the solved state. The buffer piece for corners is UBL (the details of these notations can be referred in the notation and convention section), and the buffer for edges is UR. An algorithm is used to shoot the corner buffer to a target sticker, which is on the position of RFD. A different algorithm is used to shoot the edge buffer to a target sticker on UL. To shoot the buffer to a different target mentioned before, we use set-up moves.

To determine the set-up moves, we first eliminate the side that should not be turned on the set-up. For corners, since the buffer is on UBL, therefore the sides U, B, and L can not be a part of the set-up algorithm. This means that our set-up algorithm would contains only combination of D, F, and R turns.

A more advanced method uses direct 3-cycle to move the pieces along the solves. The method is often referred as *3-style*. This idea of this method is to use the commutator concept to move three pieces at a time without messing up the others. An example commutator would be $[R\ U\ R', D]$ which move UFR to RFD, RFD to FLD, FLD to UFR. It forms a 3-cycle of $UFR - RFD - FLD$.

C. Hash Function

A hash function is a process that transforms any random dataset in a fixed length character series, regardless of the size of input data. The output is called hash value or code, digest, image, or hash. The term "hash" can be both referenced to the hash function or to the hash value, which is the output of applicating this function on a particular message. The data that are to be run through the hash function are called message. The set formed by all possible messages is the message domain or message space. Hash functions are widely used for various cryptographic applications, e.g., for storing of password hashes or key derivation.

Hash function can be used to process Rubik's cube scramble notation into hexadecimal digest. For example, the scramble "L D' F U' F L U B' U2 F2 B R2 U2 R2 U2 B' U2 R2 F2 R' U Fw Uw" will be transformed into the following digest:

ff26da56de7655d550622c365797c6ca468b5344c7bd96dad865afc4bb3da29d

In this example, the scramble is translated into a set of bits, from which, after a series of operations, a 256-bit string is obtained (here represented by its value in hexadecimal notation).

The optimal properties of a hash function are:

- It may be played on digital contents of any size and format: at the end of the day, for a computer, all types of digital content (text, images, videos, etc.) are numbers.
- Any given input may produce a fixed size numerical output.
- This output is deterministic, that is; the same input message or dataset always yields the same output.
- Reconstructing the original input from the hash function output must be extremely difficult, if not outright impossible.
- A minimum variation in the original message (one bit) must yield a completely different hash (diffusion).
- Taking an input message, finding another message with the same digest must be extremely difficult (weak collision).
- Finding any two messages that yield the same summary must be also extremely difficult (strong collision).
- The hash algorithm must cover the entire hash space uniformly, which means that any output of a hash function has, in principle, the same probability of occurrence as any other. Therefore, all values in the hash space may be an output of the hash function.

In general, hash functions work as follows: the input message is divided into blocks. Then the hash for the first block, a value with a fixed size, is calculated for the first block. Then, the hash for the second block is obtained and added to the previous output. This process is repeated until all blocks are calculated.

III. Notation and Convention

Several notations are used to cover the discussion of turns, permutations, and piece names in the Rubik's cube. Some conventions are also to be introduced to keep the explanation simple.

The Rubik's cube would generally be drawn as a three-dimensional cube showing three sides of it, while the rest are not shown due to perspective angle. Some figures will use translucent cube drawing to show the back sides of the cube. The six sides of the cube are named by the direction their faces are pointing to (*up*, *down*, *left*, *right*, *front*, *back*). The convention is to say the side that appears on the upper part of the figure is *up*, the one on the left is *front*, and the other one is *right*. Therefore, the sides not shown on the figure are *down*, *back*, and *left*, each of them is opposite to *up*, *front*, and *right*, respectively. In the default solved state case shown on this paper, as appears on Fig. 8, the *up* would be the yellow side, *front* would be the red side, and the green side for *right*.

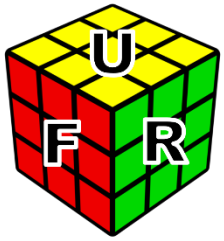

Fig. 4. The three sides of the cube, each letter denoting the initial of their side names.

From here on, we will be using a shorter way to call the six sides, by their initials. For example, U is for *up*, R is for *right*, and so on. On a turning sequence, or called an algorithm, moves are written as the six initials, telling which face needs to be turned 90° clockwise. An apostrophe modifier tells us to turn the side 90° counterclockwise (instead of clockwise). Another modifier is to put the number 2 at the end of a letter to denote the 180° turn. Algorithms should be executed in the order as they appear. An example algorithm is $R\ U'F2$, which read as "turn the *right* face 90° clockwise, and then turn the *up* face 90° counterclockwise, and then turn the *front* side 180°."

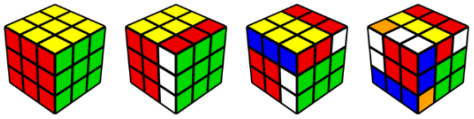

Fig. 5. From left to right: the puzzle on its solved state; R applied; $R\ U'$ applied; and finally the whole algorithm $R\ U'F2$ is applied.

Beside face turns, there are also turns that affect the middle part of the cube called slice turns. The three slice turns are M, S, and E. Slice M is the slice between *left* and *right* sides, being turned just as the L move. Slice S is the one between *front* and *back* sides, following the turning of F move. The last, E slice is between up and down sides, turns as the D face turns. Modifiers also applies to slice moves. Fig. 10 shows the solved state cube after being applied by the slice moves.

Fig. 6. Appearance of the cube after being applied: the M move, the S move, and the E move; respectively, each from a solved state.

Another important thing to note is the commutators and conjugates notation. A commutator is an algorithm in the form of $A\ B\ A'\ B'$, where either A and B can be a set of turns or just a single turn. Whenever an algorithm meets this condition, it can be written in a shorter notation, $[A, B]$. For example, the commutator $[U'R2\ B, L]$ in its longer form is $U'\ R2\ B\ L\ B'R2\ U\ L'$. Notice that the inverse of an algorithm is made by reading the algorithm backward and inversing every individual move on it (180° turn moves maintain the same).

A conjugate is where an algorithm is in the form of $A\ B\ A'$. The form can be written as $[A\colon B]$. We can combine conjugates and commutators on commutator, as in $[D\colon [U'\ R'\ U, M']]$, which read as $D\ U'\ R'\ U\ M'\ U'\ R\ U\ M\ D'$.

To build an easier communication, the community of Rubik's cube define a standard guide for naming each sticker place for

the cube. Rather than saying "the yellow sticker on the green-yellow-red corner," it is more convenient to say it by the initials of the face associated to the sticker. For example, we would refer the red sticker on the yellow-red-green corner as FRU. The first initial is indicating on which side is the sticker we're meant to, followed by another two sides of the corner, preferably in counterclockwise cycle (saying FUR is still acceptable though). The same goes for the edges, for instance, the yellow sticker on the green-yellow edge is called UR. Please be aware that this labelling system is dependent on the cube orientation we use.

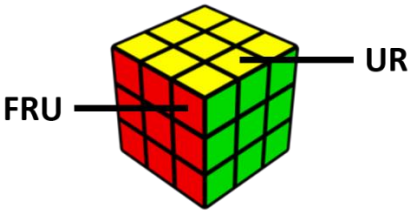

Fig. 7. Example of the labelling system.

Another easy way of naming the stickers is by using letters. For blindfolded solving, it is often encouraged to use this lettering system to make the memorization easier. The letters translate from the normal labelling system as shown on Table 1.

TABLE I. THE LETTERING SYSTEM FOR CORNERS

Sticker Label	Sticker Letter
UBL	A
URB	B
UFR	C
ULF	D
FUL	E
FRU	F
FDR	G
FLD	H
RUF	I
RBU	J
RDB	K
RFD	L
BUR	M
BLU	N
BDL	O
BRD	P
LUB	Q
LFU	R
LDF	S
LBD	T
DFL	U
DRF	V
DBR	W
DLB	X

As seen on Table 1, capital letters are used. To make it different, the edge stickers use noncapital letters instead.

TABLE II. THE LETTERING SYSTEM FOR EDGES

Sticker Label	Sticker Letter
UB	a
UR	b
UF	c
UL	d
FU	e
FR	f
FD	g
FL	h
RU	i
RB	j
RD	k
RF	l
BU	m
BL	n
BD	o
BR	p
LU	q
LF	r
LD	s
LB	t
DF	u
DR	v
DB	w
DL	x

IV. APPLYING HASH FUNCTION ON MULTI-BLIND SCRAMBLES

In a multi-blind competition, competitors are given several cubes that are needed to be solved blindfolded. These cubes are scrambled using pre-generated scramble sequences, which have to be the same among all competitors. To ensure fairness in the competition, organizers need to verify that all competitors have the same scramble for all cubes they attempt. Sometimes, the number of cubes is too many to verify that it will take a long time to do the verification. To make verification easier, take less time but still ensure security, competition organizers can apply a hash function to the scrambles, and check the digest to verify all the scrambles.

The scrambles for all cubes are treated as a single string, delimited by newlines. These scramble sequences are then to be processed by the hash function, to generate a digest that is used to verify all scrambles for all competitors. The function we're using here is a commonly used hash function, SHA-256. This is a moderately sufficient hash algorithm to verify Rubik's cube scramble sequences.

In the program that we made, we take input from texts that contain scramble sequences for each competitor that participate in the multi-blind contest. The scrambles are then processed by the SHA-256 function implementation in Python. A function will determine if both scramble sequences are exactly the same for all cubes. For example, if a competitor has scramble sequences like this:

D L' B R F' D F' B2 U' L' F2 B2 D2 R' F2 D2 F2 L' U2 R Rw' Uw'

F2 D' B2 L' U2 B2 F2 L' R2 U2 R2 D2 U2 F2 B' L B L2 F' L2 R Rw Uw2

U' F2 D2 B U2 B R2 D2 L2 F2 R2 U2 F2 D' L' U' B R F' D F Rw2 Uw

U' L D2 F2 U R2 D F2 R2 D R2 U R D2 B' L D F' R Rw Uw

D' L2 R2 F2 U' F2 D L2 U' B2 D2 R2 L F R U2 R' D F' D2 R'

L D' F U' F L U B' U2 F2 B R2 U2 R2 U2 B' U2 R2 F2 R' U Fw Uw

L2 B' L2 F2 L2 D L2 R2 D L2 U B2 U2 L R2 F L2 F2 L' B' Fw' Uw'

R' B2 D L2 D' R2 F2 D' R2 U F2 L2 D2 R' F' L2 D' F2 R U' R'

D2 L2 D2 U2 F L2 D2 B2 F R2 F2 U' L U R' B D U2 F R2 U2 Rw' Uw2

R U' L2 D' R2 U R2 B2 U' F2 D2 L2 F L2 U' F D2 L B2 D' F Fw' Uw

F R2 U2 B' D2 B U2 R2 B2 F' R2 B' L D U2 L2 D2 B U' F2 L Fw Uw'

F L2 F2 L2 D' R2 U R2 U2 F2 D2 U L B' R' F2 D L U' B2 R' Rw Uw'

D L' F2 D2 B R F' U2 R' B2 U D L2 D R2 B2 U' F2 L2 Rw Uw'

D' F' B D2 B2 L2 D R U2 F2 U2 B D2 R2 L2 B U2 F' D2 L' Rw Uw2

R2 L' U L B' R' F2 L D' R2 F2 D2 R' U2 R D2 L2 B2 L D2 Fw Uw

R2 D2 B2 F D2 R2 B2 U2 R2 U2 L2 U' F R' B' R F2 R2 F' L2 Fw

B R' B L2 F L2 F' L2 F2 L2 F' D2 B' R U' B' F D' R B2 Fw Uw2

F2 U' F2 U R2 B2 D L2 U L2 F2 B U B2 U2 L2 U' L' R' B U Fw Uw'

U F' U2 D L2 B L' F R2 L2 U2 B2 U' B2 L2 D2 F2 L2 D' B' Uw2

D2 F2 R' F2 U2 L2 R' F2 R2 U2 D R B2 U2 L B L' U' B2 Rw Uw2

If another competitor has exactly the same scramble sequences, the program will produce output as follows:

Digest 1: c516f0346ac4d977fee80e1af78436ab958ed8331d8835ead54b42539a9f066a

Digest 2: c516f0346ac4d977fee80e1af78436ab958ed8331d8835ead54b42539a9f066a

Both scramble sets are the same

Another example, if a different competitor has a slightly different scramble, that is an F move is mistaken to be an F' move in the tenth scramble, so the whole scramble sequences look like this:

D L' B R F' D F' B2 U' L' F2 B2 D2 R' F2 D2 F2 L' U2 R Rw' Uw'

F2 D' B2 L' U2 B2 F2 L' R2 U2 R2 D2 U2 F2 B' L B L2 F' L2 R Rw Uw2

U' F2 D2 B U2 B R2 D2 L2 F2 R2 U2 F2 D' L' U' B R F' D F Rw2 Uw

U' L D2 F2 U R2 D F2 R2 D R2 U R D2 B' L D F' R Rw Uw

D' L2 R2 F2 U' F2 D L2 U' B2 D2 R2 L F R U2 R' D F' D2 R'

L D' F U' F L U B' U2 F2 B R2 U2 R2 U2 B' U2 R2 F2 R' U Fw Uw

L2 B' L2 F2 L2 D L2 R2 D L2 U B2 U2 L R2 F L2 F2 L' B' Fw' Uw'

R' B2 D L2 D' R2 F2 D' R2 U F2 L2 D2 R' F' L2 D' F2 R U' R'

D2 L2 D2 U2 F L2 D2 B2 F R2 F2 U' L U R' B D U2 F R2 U2 Rw' Uw2

R U' L2 D' R2 U R2 B2 U' F2 D2 L2 F L2 U' F D2 L B2 D' F Fw' Uw

F R2 U2 B' D2 B U2 R2 B2 F' R2 B' L D U2 L2 D2 B U' F2 L Fw Uw'

F L2 F2 L2 D' R2 U R2 U2 F2 D2 U L B' R' F2 D L U' B2 R' Rw Uw'

D L' F2 D2 B R F' U2 R' B2 U D L2 D R2 B2 U' F2 L2 Rw Uw'

D' F' B D2 B2 L2 D R U2 F2 U2 B D2 R2 L2 B U2 F' D2 L' Rw Uw2

R2 L' U L B' R' F2 L D' R2 F2 D2 R' U2 R D2 L2 B2 L D2 Fw Uw

R2 D2 B2 F D2 R2 B2 U2 R2 U2 L2 U' F R' B' R F2 R2 F' L2 Fw

B R' B L2 F L2 F' L2 F2 L2 F' D2 B' R U' B' F D' R B2 Fw Uw2

F2 U' F2 U R2 B2 D L2 U L2 F2 B U B2 U2 L2 U' L' R' B U Fw Uw'

U F' U2 D L2 B L' F R2 L2 U2 B2 U' B2 L2 D2 F2 L2 D' B' Uw2

D2 F2 R' F2 U2 L2 R' F2 R2 U2 D R B2 U2 L B L' U' B2 Rw Uw2

It will produce an output as follows:

Digest 1: c516f0346ac4d977fee80e1af78436ab958ed8331d8835ead54b42539a9f066a

Digest 2: feba70e47ef28f95b0c3b74e44f48db5a2124441ddc458773c7fc5488cd4811c

Scramble sets are different

As we notice, a single move difference in the tenth scramble makes a significant change in the output digest. Therefore, the use of hash function in the case of verifying multi-blind scrambles for fairness among the competitors is proven to be effective, and yet easier for organizers to use this instead of verifying all the scrambles manually.

V. Appendix

All Rubik's cube models shown in this paper are generated from an open-source visual cube generator from http://cube.crider.co.uk/visualcube.php.

VI. Acknowledgment

All praise to Allah, only with His guidance I could finish this paper. After that, I thank Dr. Ir. Rinaldi Munir, M.T. for his guide in understanding the cryptography in the class. Special thanks to Ernő Rubik for his amazing invention on the cube. I also appreciate to the great community of cubing, especially Stefan Pochmann, for developing such a great concept in the world of blindfolded Rubik's cube solving.

REFERENCES

[1] C. Paar & J. Pelzl, *Understanding Cryptography*. Berlin: Springer-Verlag Berlin Heidelberg, 2010.

[2] D. Rayhan, *Graph Application in Rubik's Cube for Blindfolded Solving*. 2019.

[3] D. Rayhan, *Regular Expression Application in Rubik's Cube Algorithm for Eliminating Redundant Moves*. 2020.

[4] Agencia española de protección de datos, *Introduction to the hash function as a personal data pseudonymisation technique*. 2019.

[5] https://www.rubiks.com/en-us/about (*accessed on December 19th, 2021*)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Desember 2021

Dhafin Rayhan Ahmad (13518063)