Abstract (in Polish)

Tematem pracy inżynierskiej jest analiza metod rozwiązywania kostki Rubika oraz optymalizacja metod stosowanych w celu jej ułożenia. Aby wprowadzić w temat kostki Rubika w pierwszym rozdziale są zawierane podstawowe informacje dotyczące zasad technicznych kostki. Pobocznym zagadnieniem, którego się podjąłem jest wyznaczenie wzoru na ilość możliwych stanów kostki Rubika w zależności od rozmiaru kostki, to znaczy wyrażenie za pomocą wzoru, gdzie zmienną jest rozmiar kostki, wszystkich możliwych kombinacji ułożeń kostki Rubika o danej wielkości.

Aby osiągnąć główny cel pracy to znaczy porównanie wydajności różnych metod układania kostki Rubika zdefiniowałem konkretne strategie optymalizacji tych metod. Następnie napisałem kod w języku R odpowiedzialny za układanie kostki różnymi metodami oraz ich optymalizacje, który pozwolił mi testować metody w programie Rstudio. Głównym elementem pracy jest analiza otrzymanych wyników oraz ich walidacja. Analiza zakłada porównanie (i) metod między sobą; (ii) zoptymalizowane i niezoptymalizowane wersje tych samych metod; i na koniec (iii) otrzymane wyniki z wynikami z innych prac.

Słowa kluczowe:

kostka Rubika, pomieszanie, metoda LBL, metoda CFOP, optymalizacja

Dziedzina nauki i techniki:

Nauki przyrodnicze, Matematyka, Nauki o komputerach i informatyka, Inżynieria informatyczna

Abstract

The main subject of this thesis is the studying the Rubik's cube solving methods and the analysis of their efficiency The study starts with introduction into basic information concerning technical and mathematical aspects of Rubik's cube. Then the general formula for the number of possible scrambled states of Rubik's cube of any size was derived. This formula allows to give the answer about the number of combinations there can occur on Rubik's cube of any size of the cube as a variable.

To achieve the main goal of the thesis - comparing the efficiency of the different methods of solving Rubik's cube - the particular strategies of their optimization were defined. The special programming code for solving Rubik's cube and further optimization of selected methods was developed and tested in Rstudio. The crucial part of the thesis is the analysis of the achieved results and their validation. The methodology of results analysis presupposed the comparison of (i) the methods against each other; then (ii) their optimized and not optimized versions; and finally (iii) our experimental results against results from extant literature.

Keywords:

Rubik's cube, scramble, LBL method, CFOP method, optimization

Field of Science and Technology:

Natural sciences, Mathematics, Computer and information sciences, Information engineering

Table of contents

LIST OF KEY SYMBOLS AND ABBREVIATIONS

LBL - Layer By Layer method

F2L - First 2 Layers

OLL - Orientation of Last Layer

PLL - Permutation of Last Layer

CFOP - Cross - F2L - OLL - PLL

INTRODUCTION

Rubik's cube nowadays becomes one of the bestselling toys or puzzles and is considered very interesting for both youth and adults for a number of reasons. First of all, it looks so simple, when a person takes Rubik's cube in their hands and tries to play with it or solve it, at the start, it looks easy, whereas it is very complicated and almost no one can figure out a way to solve it without any help. This is why there is a lot of content on the Internet that describes it. For example how to solve it, how to solve it in as short a time as possible, there are a lot of videos of people solving it in just a couple of seconds. The current world record is equal to 3.47 seconds, which sounds impossible, but actually is true. These are only some of the reasons why the community of people interested in it keeps growing and growing.

On the other hand, many people also say that solving a Rubik's cube is a metaphor for any other problem. When we start to solve any big problem, we can say that there is a lot to do, we can take the example of creating and running a business. Similarly, as in unsolved Rubik's cube, there are a lot of ways we can start, we can choose any of them, we can choose industry, type of company, and many other things. Then when we are in the middle of solving Rubik's cube we see that it is getting harder and harder, we need to remember that by solving the next parts we have to be careful not to mess up any progress made previously. Similar is the case for running a business, when we already have a business and want to expand it starts to get harder, not we have limited options, we cannot do anything we want to expect to get better, we have to carefully plan our action and predict what the consequences will be. Finally, when we get to a situation where we have Rubik's cube almost solved, we are in a situation more difficult than any before. It is now when we have to experiment, try different things, and see which ones can help and which ones are not a good idea. It is similar to running a business, when we already have a thriving business and almost everything is perfect, but we want it to be 100% perfect, then it is really hard to improve certain aspects to make it perfect. Often when we find a way to improve it we find out that a given solution causes a number of problems in some other area and to find that specific solution to a given situation that helps us solve current problems without creating any new ones is equivalent to solving the last step of Rubik's cube. Also, another thing, which is similar in those two problems is the aspect of randomness. In Rubik's cube sometimes you get lucky situations and sometimes the opposite of that. In running a business, you can also sometimes get lucky when for example it turned out that starting a company in a given industry was a very good idea because that industry turns out to grow very fast. And again sometimes this is a situation, which we can predict, for example, that the high technology industry is good to invest in, but sometimes some situations are impossible to predict at an early stage of problem-solving.

The aim of the thesis is to find out the optimal method of solving Rubik's cube. The first chapter presents theoretical aspects of Rubik's cube, these are construction of the cube and types of moves. Second chapter describes math that concerns Rubik's cube and there we calculated the number of possible states of $n \times n \times n$ Rubik's cube. Third chapter focuses on describing methods of solving Rubik's cube as well as choosing the ones, which I will implement in my thesis. In the fourth chapter I go over my experimental setup, that is explaining step by step what actions

I need to take in order to implement, optimize and analyze the methods of solving Rubik's cube, which I chose before. Fifth chapter describes the optimizing approaches for implemented methods, that means the way in which I will try to improve the methods implemented by myself in the first place. Sixth one concerns analysis of results achieved by myself before and comparing them against each other as well with the results obtained in extant literature.

1. TECHNICAL DESCRIPTION OF RUBIK'S CUBE

Erno Rubik, Hungarian inventor, architect and professor of architecture, created the Rubik's Cube in 1974 (Mcaleer S & Baldi P, 2018). Firstly he named it the Magic cube and after some time he decided to rename it to Rubik's cube. Since then over the globe it has been sold over 350 million copies of that puzzle. Creation of Rubik's cube also inspired many other inventors to create more and more new puzzles with similar characteristics to Rubik's cube, but different in terms of shape, size or types of movement.

In this chapter, we focus on classic $3 \times 3 \times 3$ Rubik's cube and aim to establish the main technical facts that describe the object of our study - Rubik's cube, namely the (i) construction of cube, (ii) types of moves in Rubik's cube.

1.1. Construction of cube

First, we need to realize that Rubik's cube elements are not stickers but smaller cubies. We can look for definitions of them in many thesis, for example we can see in the work of Da-Xing Zeng et al. (2018) or the work of Stefan Pochmann (2008) fully described construction of cube, which I will briefly describe here. By cubies we understand one element, which can consist of one, two or three stickers, which may seem separated, but in fact they are one element. To show what it means we can look at Figure 1.1:

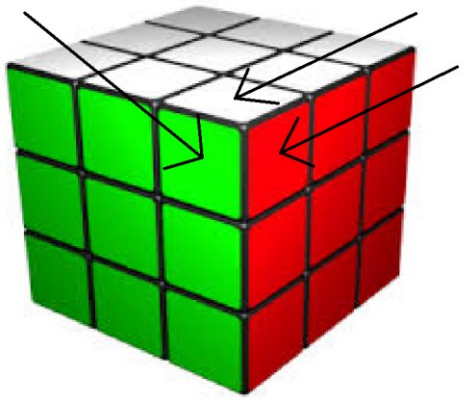

Fig. 1.1. Construction of a Rubik's cube Source: https://rcube.pl/jak-ulozyc-kostke-rubika-3x3x3-lbl (edited)

All elements pointed by arrows are one non-separable element, in a way that by scrambling the cube these stickers would always be connected and we can't disconnect them. Having this information we can see that Rubik's cube is made out of 26 smaller cubies (, we subtract 1 because we don't count the inside cubie).

In classic Rubik's cube, there are no two identical elements, each one is unique and there is only one solve state. We can distinguish three types of elements. These are centers; edges; corners.

The main feature that characterizes each element is the number of colors it has. Centers have one color, edges two colors, and corners three colors

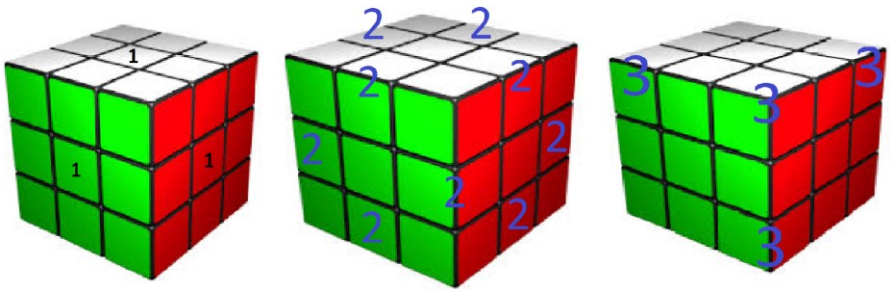

Fig. 1.2. Elements of a Rubik's cube. Source: https://rcube.pl/jak-ulozyc-kostke-rubika-3x3x3-lbl (edited)

As we see in figure 1.2, 1s are centers (one color); 2s are edges (two colors); 3s are corners (three colors). The easy thing to notice is that each type of element can only switch places with the element of the same type, e.g. corner can't switch places with edge or center and so on.

1.2. Types of moves in Rubik's cube

Second, we need to define and generalize the types of Rubik's cube movement. There is a commonly defined way to name all types of moves in order to simplify the notation and generalize it, so that as many people as possible would use that set of terminology, because it helps in communication if everyone uses the same terminology. The terminology is available in many papers, for example it is presented in Jaslyn Lek's (2009) or in the work of Stefan Pochmann (2008). Each move consists of turning one face of a cube by 90^{o} to one side or another, or by 180^{o} (no matter which side). We assume that we are holding the cube as follows (Figure 1.3):

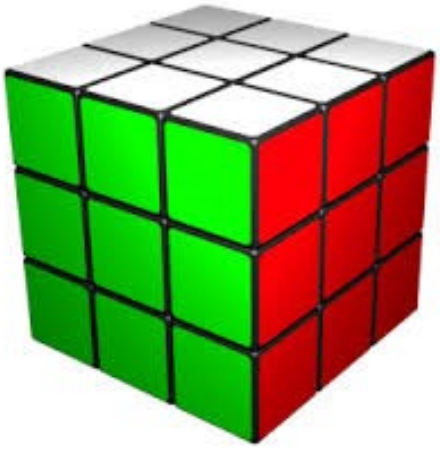

Fig. 1.3. Holding Rubik's cube. Source: https://rcube.pl/jak-ulozyc-kostke-rubika-3x3x3-lbl

In this case, the green face is in front of us, the red one is on the right and the white is on top. Each of the basics moves is understood as follows (Figure 1.4):

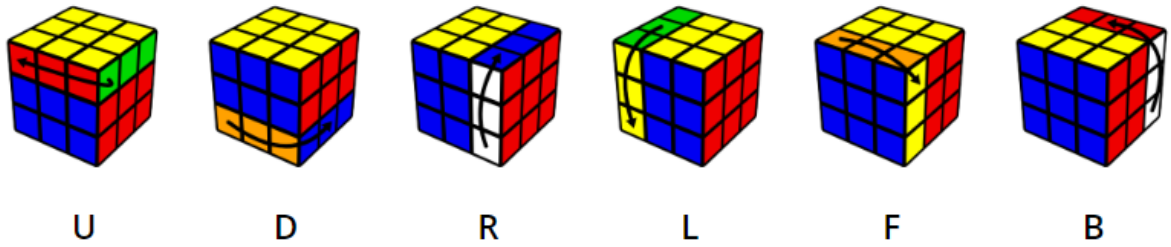

Fig 1.4. Basic moves of a Rubik's cube. Source: https://jperm.net/3x3/moves

Then there is the opposite of these moves (Figure 1.5):

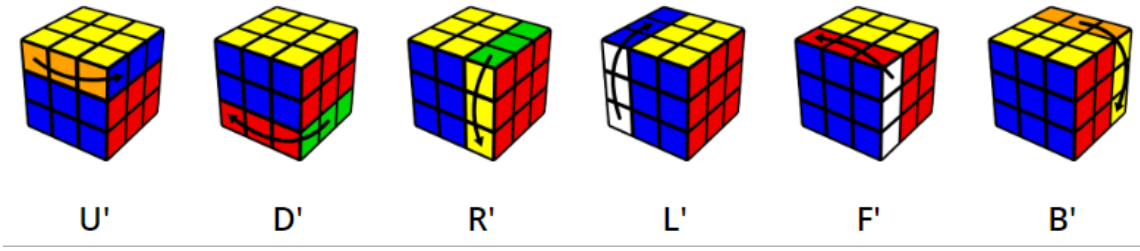

Fig 1.5. Opposite moves of a Rubik's cube. Source: https://jperm.net/3x3/moves

And there are also middle moves, also called slice moves. They could be replaced by making two moves of opposite faces in the same direction, but by defining them it is easier to write down some algorithms (Figure 1.6).

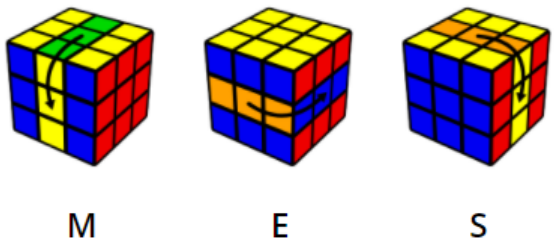

Fig 1.6. Slice moves of a Rubik's cube. Source: https://jperm.net/3x3/moves

There are also of course the opposite of slice moves M', E', and S'. They are easily the same moves but in other directions.

There are also moves which I mentioned before, they can easily be understood by making any move two times in the same direction. An important thing to notice is that they can be understood by making two moves, but we count moves as only one move.

These are all moves that are commonly defined, and we will use only those moves in further parts of my thesis. There are also cube rations, which are not exactly moving, but rotations that help to solve the cube to make it more comfortable. They rotate the whole cube along x, y,, and z-axis, but the important thing to remember that, they are not considered as a move, because in fact, we are not doing turning any face of Rubik's cube, but we are just changing the angle from which we are looking at the cube (Figure 1.7).

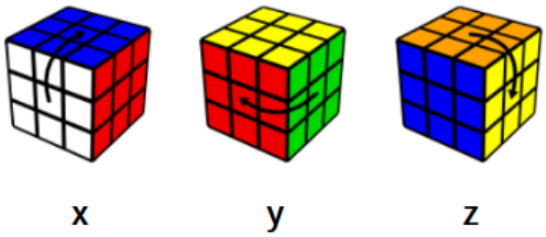

Fig 1.7. Rotations of a Rubik's cube. Source: https://jperm.net/3x3/moves

2. RUBIK'S CUBE MATHEMATICAL BACKGROUND

Rubik's cube is a very interesting toy and puzzle also from a mathematical point of view. It is mainly caused by the number of possible scramble states. Of course, some of those scrambled states are one move away from solving them but we still count them as scrambled. The main goal of this section is to explore classical $\mathbf{3} \times \mathbf{3} \times \mathbf{3}$ Rubik's cube and then derive a universal formula for the number of scrambled states of $\boldsymbol{n} \times \boldsymbol{n} \times \boldsymbol{n}$ Rubik's cube.

2.1. $\mathbf{3} \times \mathbf{3} \times \mathbf{3}$ Rubik's cube

In this chapter we will focus on classic $\mathbf{3} \times \mathbf{3} \times \mathbf{3}$ Rubik's cube. The number of possible states of Rubik's cube is commonly known and was calculated many times, by many different people in many thesis, for example in the work of Toshniwal and Golhar (2019); Richard E. Korf (1997); Da-Xing Zeng et al. (2018); Jaslyn Lek (2009); W. D. Joyner (1996); Duberg and Tideström (2015) and many others. Nevertheless I will show the proof of that and show each step of procedure. In order to do so, we have to consider corners and edges of the cube independently. The specification of Rubiks' cube is that each scramble state of corners can meet each scramble state of edges. So firstly we will have to find out the number of possible states of scrambling corners and then do the same for edges. Knowing these numbers we will easily multiply them and get the results.

First we will need to discuss the number of possible scrambled states of corners. There are 8 corners in the cube, so first we need to calculate how many ways we can place them on the cube. First one we can place on the cube in 8 ways, the second one in 7 places, the third one in 6 places, and so on. So to calculate that we just need to calculate 8 factorials (8!). This number is equal to 40.320.

Next, we need to find out the number of possible orientations of each of the 8 corners. It is pretty straightforward that each corner has three possible orientation states. So we would easily assume that if we have three ways to twist each corner we would have 3^{8} possible orientations of all corners. But there is one little hitch. The cube is designed in such a way that we can never obtain a scramble where for example one corner is twisted and all others are fixed, (Figure 2.1):

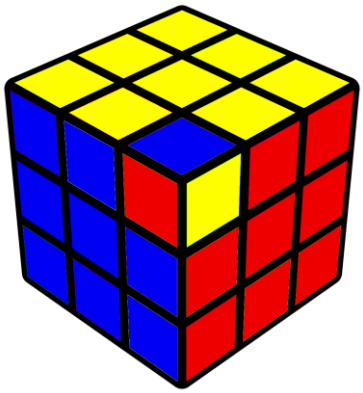

Fig 2.1. Unsolvable Rubik's cube. Source: https://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube (edited)

This state is impossible to solve and also impossible to obtain in the classic scramble. That would be only possible by unsticking and placing back stickers in some other places. Overall we can summarize that the orientation of the last corner is always dependent on the other seven corners. So the number would not be equal to 3^8, but 3^7. This number is equal to 2.187.

Then to find out the overall number of scrambled positions of corners we just need to multiply these two obtained number and we get the result:

$$8! \cdot 3^7 = 40320 \cdot 2187 = 88179840$$

We see that this is already a huge number, and here we only consider corners. To give an image of how big this number is we can say that if we move the cube randomly the chance of corners being solved is 1 in 88 179 840, which is comparable to the chance of winning Euro jackpot, and is 6 times more likely than winning in Lotto.

Next, we will focus on the edges. The method we will use is similar to previously. We know that there are 12 edges on the cube. So first we calculate the number of ways we can place them on the cube. First one we can place on the cube in 12 ways, the second one in 11 places, the third one in 10 places, and so on. So to calculate that we just need to calculate 12 factorials (12!). But again there is one little hitch. The last two edges have only one possible way of putting them into the cube, and it is again caused by the design of the cube, so we can never obtain a scramble (Figure 2.2):

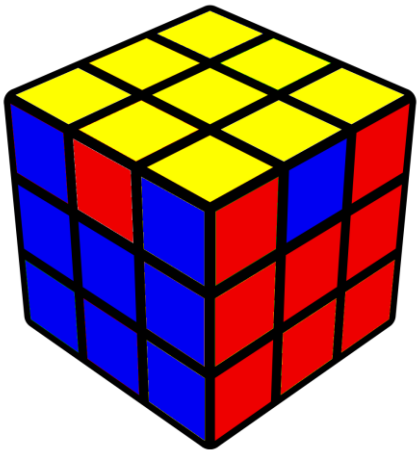

Fig 2.2. Unsolvable Rubik's cube. Source:
https://en.wikipedia.org/wiki/Optimal_solutions_for_Rubik%27s_Cube (edited)

If the whole cube was solved except for two edges, which would be swapped, this we call impossible to state, because it is impossible to solve, and also impossible to obtain in the classic scramble. So the number of possible permutation would be

$$12 \cdot 11 \cdot 10 \cdot \ldots \cdot 3 \cdot 1 = \frac{12!}{2} = 479001600 : 2 = 239500800$$

We also have to consider the orientation of all edges. Unlike corners, edges can be oriented in one of two ways. So we intuitively would guess that there are 2^{12} ways to orient edges, but similarly to corners orientation of the last edge is dependent on the rest of the edges, so it is 2^{11} (which is equal to 2048). Then we need to multiply these numbers and we get:

$$\frac{12!}{2} \cdot 2^{11} = 239500800 \cdot 2048 = 490497638400$$

We see that this number is even bigger than the number in the case of corners

Lastly, we need to multiply numbers which we achieved in the case of corners and edges, and we get:

$$88179840 \cdot 490497638400 = 43252003274489860000 = 4.3 \cdot 10^{19}$$

It is a number of magnitudes that is hard to imagine or even find a real-life example that uses such big numbers. To show how big it is, we can image a moving cube randomly and we see that the probability of solving it by chance is 1 in 43252003274489860000. It is orders of magnitude smaller than for example a chance of winning Euro jackpot, or even winning it two times in a row. It is similar to picking 20 out of 80 numbers and guessing all of them correctly. But actually, it is even smaller than that, but slightly.

2.2. $n \times n \times n$ Rubik's cube

In order to generalize the number of possible scrambles for the $n \times n \times n$ Rubik's cube, we have to distinguish two types of Rubik's cubes, with (i) odd layered and (ii) even layered.

2.2.1. Odd layered cubes

Firstly we will focus on odd layered cubes. We will take as an example $5 \times 5 \times 5$Rubik's cube. As it is easy to observe it has many more elements, because it is much bigger, so we have to distinguish what are those.

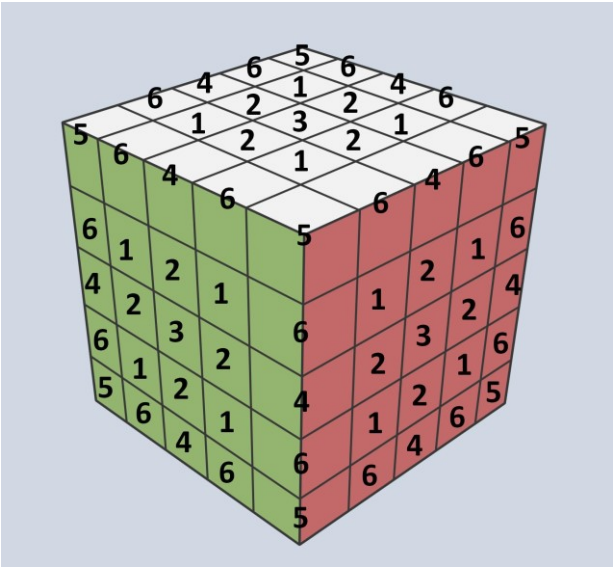

Fig 2.3. $5 \times 5 \times 5$ Rubik's cube Source: https://www.wikihow.com/Solve-a-5x5x5-Rubik%27s-Cube (edited)

We can notice in Figure 2.3. different types of elements: *1* - corners of centers; *2* - edges of centers; 3 - centers; 4 - edge; 5 - corners; 6 - outers edges.

If we think a few moments about this cube, we see that if we focus only on centers, edges, and corners (3,4,5) we could solve only those elements exactly like $3 \times 3 \times 3$. The only problem is the rest of the elements, but this is not how we would solve it, but this is the topic for another

time. For now, the important fact is that each of those groups of elements can be scrambled independently, what it means is that we can swap elements from each group individually, so for example all corners of centers (ones) can be mixed up, all outer edges (sixes) can also be mixed up independent from other elements. So in order to calculate the overall number of possible scramble states we need to calculate the number of ways in which each group of elements can be scrambled and then multiply the results. But we have to remember that we want to do this for any odd layered cube, not only $5 \times 5 \times 5$, but I will get to that later. For a start we can use our previous observation, that centers, edges, and corners (elements 3, 4, 5 on the picture) can be scrambled exactly like $3 \times 3 \times 3$ cube, so for those elements, we do not have to do anything else, so we just need to focus on corners of centers, edges of centers and outer edges (elements 1,2,6). Firstly we will focus on the outer edges. We know that on each edge of the cube there are two outer edges, there are 12 edges, so that gives us 24 outer edges. The construction of the cube does not give any constraints about scrambling them, we can swap each two of them independently from the rest of the cube, so that gives 24 factorial (24!). But that is only the case for the $5 \times 5 \times 5$ cube, but what about $7 \times 7 \times 7$ or $9 \times 9 \times 9$? We have to think about construction once again. In the case of $5 \times 5 \times 5$ we have the edge of a length 5, where 2 are corners and 1 is the edge, so the rest two are outer edges, in case of $7 \times 7 \times 7$ it is similar, but there are four remainings because the number of edge pieces and corners is fixed so all the extra pieces need to be outer edges. In the case of $9 \times 9 \times 9$ there are six remaining pieces, so there are 6 outer edges. We can observe a simple linear function (Table 2.1):

Table 2.1 number of outer edges in one edge in $n \times n \times n$ Rubik's cube

Order of cube	Number of outer edges in one edge
$3 \times 3 \times 3$	0
$5 \times 5 \times 5$	2
$7 \times 7 \times 7$	4
$9 \times 9 \times 9$	6
$11 \times 11 \times 11$	8
$13 \times 13 \times 13$	10
$15 \times 15 \times 15$	12
$17 \times 17 \times 17$	14

Source: author's elaboration

We can see that a number of outer edges in each edge of $n \times n \times n$ cube can be expressed by a simple formula $n - 3$, where *n* is the order of the cube. Another important thing to notice is that each set of outer edges can only be mixed with each other, what it means is that the 1st outer edge piece can not be swapped with the 2nd outer edge piece, 3rd outer edge piece, etc. So what is actually happening in for example $9 \times 9 \times 9$ cube is that instead of having 72 ($12 \cdot$

$(9-3)$) outer edges that can be swapped with each other, we have 3 groups of 24 edges that can be swapped within each group, so the number, in this case, would not be 72 factorial (72!), but $(24!)^3$, because we have three groups, where 3 is, in this case, the number of groups that we have (which in case of $5 \times 5 \times 5$ is simply one, that is why it is $(24!)^1$). We can make a conclusion that the number of outer edge groups is always equal to half of the number of outer edges on one edge and that is equal to $\frac{n-3}{2}$. So the final number of combinations for outer edges can be expressed by the formula:

$$(24!)^{\frac{n-3}{2}} \tag{2.1}$$

where *n* of course means the size of the cube.

The next things to consider are corner centers and edge centers. In both these cases, the situation will be very similar and analogous. Firstly we can focus on edge corners, first what we can observe is that there are four of them on each face, so we have 24 of them. There are also no restrictions, where each one can be, depending on the rest of the pieces, so there is 24 factorial (24!) ways to arrange them. There is one hitch, the four pieces that belong to one face are indistinguishable, so there are actually 4 factorials (4!), which is 24 ways to arrange four pieces on each side. Based on those observations we have to divide the number we achieved, that is 24! by 24 six times, because there are of course six faces. So the final number we obtained is equal to $\frac{24!}{24^6}$. We repeat all those steps for corner centers, and the way of thinking is exactly the same as before, so also the number is equal to$\frac{24!}{24^6}$. Finally, we achieved the following result:

$$\left(\frac{24!}{24^6}\right)^2 \tag{2.2}$$

But this number is good only in the case of $5 \times 5 \times 5$ cube. What about the higher order of cubes? We will be thinking similarly, but we have to check how many elements similar to corner centers and edge centers are there in each of the cubes. Below there is a view of a grid of center of one face of $7 \times 7 \times 7$cube (Figure 2.4):

1	4	2	5	1
5	6	7	6	4
2	7	3	7	2
4	6	7	6	5
1	5	2	4	1

Fig 2.4. Grid of center of one face of $7 \times 7 \times 7$cube. Source: author's elaboration

We see that there are overall 25 tiles in that center. We have to assign each of them to one group in a way that we categorize them into some groups, as previously. This time we will be not naming them because there are too many of them. I divide the grid into four rectangles and one small square. The division is not accidental. The square is of course fixed center, that is always only one and this is the element from classical $3 \times 3 \times 3$ Rubik's cube, so we can forget

about it for now. Then we are left with four rectangles with six pieces each. Then when we think about turning the face we would have to turn in for example 90^{o} clockwise. Let's see what we obtained (Figure 2.5):

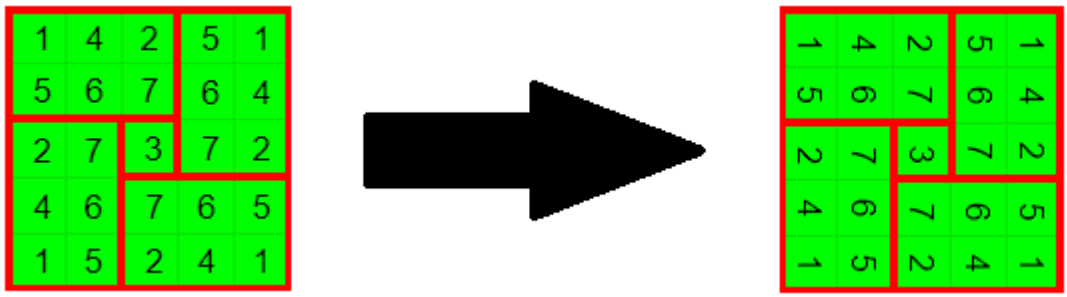

Fig 2.5. Turned grid of the center of one face of $7 \times 7 \times 7$ Rubik's cube. Source: author's elaboration

When we look at the results of turning face we can easily observe that in place of each *1s* there is again *1*, in place of each *2s* there is again *2,* and so on. That is why I divided the tiles and numbered them in a given way. This shows us that, no matter how we would turn the face each group scramble only within each group. So we see that there are 6 different tiles in each rectangle and that tells us that there are six 6 groups of different centers pieces in $7 \times 7 \times 7$ Rubik's cube. Now we can check what will be in case of $9 \times 9 \times 9$. We will proceed in a similar way (Figure 2.6).

Fig 2.6. Grid of center of one face of $9 \times 9 \times 9$cube. Source: author's elaboration

Similarly, as in $7 \times 7 \times 7$ there are some new pieces, there are 13 groups but again we will exclude the group number 3, so we are left with 12 groups (for the same reasons as in $7 \times 7 \times 7$ Rubik's cube). As I intentionally highlighted in the picture we can divide each face into four rectangles and one square (which is group 3). We will focus on rectangles. The first thing that is important to notice, that each of them contains one and only one copy of each number from 1 to 13 (except 3), and also in an identical configuration. What it means is that we can turn that face 90^{o} in any direction or 180^{o} and the face will still look the same. This means that when we want to find out how many groups of center tiles there will be in a cube, we can simply count the numbers in each rectangle. We can do this in one of the two following ways.

The first method is based on measuring the length of two adjacent sides of a rectangle, in the case of $5 \times 5 \times 5$ this is 1x2, in case $7 \times 7 \times 7$ of 2x3, in case of $9 \times 9 \times 9$ 3x4, and so on, we can see the pattern and could be presented by the formula:

$$\frac{n-3}{2} \cdot \frac{n-1}{2} \tag{2.3}$$

where *n* of course means the size of the cube.

The second way is to observe that the grid of center of $n \times n \times n$ cube is always *(n-2)x(n-2)*, so this is equal to the sum of the four rectangles of unknown size and 1 square of size 1, so we can subtract 1 from the area of a grid of center square and divide by four to get:

$$\frac{(n-2)^2-1}{4} \tag{2.4}$$

Of course, these two formulas are equivalent, and if we transform one of them, we can easily achieve the other one. It is just a matter of choice, which method we find more pleasant or easier to use. Nonetheless, we calculated this for a reason, we substitute this formula as a number of groups of centerpieces in $n \times n \times n$ cube to isolate the number of possible scrambles states of these groups combined and we get:

$$\left(\frac{24!}{24^6}\right)^{\frac{(n-3)}{2}\cdot\frac{(n-1)}{2}} = \left(\frac{24!}{24^6}\right)^{\frac{(n-2)^2-1}{4}} \tag{2.5}$$

where *n* of course means the size of a cube.

The final view of the generalized formula is the following:

$$\left(\frac{24!}{24^6}\right)^{\frac{(n-2)^2-1}{4}} \cdot (24!)^{\frac{n-3}{2}} \cdot 43252003274489860000 \tag{2.6}$$

2.2.2. Even layered cubes

Next, we will focus on even layer cubes, because for the reasons I will explain the formula will differ a little bit. Some steps will be similar, but nonetheless, it would be easier to consider them independently.

First what we have to notice is that in odd layered cubes there are no fixed centers, like it has placed in odd layered cubes, all elements can be swapped all around the cube. The next observation is that there are no edge elements like in odd layered cubes, there are only edge pieces that are equivalent to outer edges in odd layer cubes and this is what we will call them because they are essentially the same thing. So firstly we can focus on them. All calculations will be analogous to odd layer cubes, there is one thing we need to do, that is to isolate the number of groups of outer edges based on *n*, that is the order of the cube. In each case edge length constitute two corner elements and the rest are outer edges, but there are two from each group, so we will have to divide the achieved number by two, so we get $\frac{n-2}{2}$, and the final number of possible scrambles of outer edges is given by the formula:

$$(24!)^{\frac{n-2}{2}} \tag{2.7}$$

The next group we have to consider are corners, this is identical to in case of $3 \times 3 \times 3$ Rubik's cube, so I will not repeat the steps I did in order to achieve this number:

$$8! \cdot 3^7 = 40320 \cdot 2187 = 88179840 \tag{2.8}$$

It is also worth mentioning that this number is fixed, is not dependent on the size of the cube, and if we think about it for a moment it makes sense because the number of corners in any cube will be equal to 8.

Lastly, we have to consider center pieces. It will also be a similar process to odd layered cubes, but we have to find out the number of groups of them, similar as before because the number of possible scrambles states of each group of center pieces is the same as before. In $4 \times 4 \times 4$ Rubik's cube, the case is simple because we see only four center pieces, and they all can swap with each other. If we would look at $6 \times 6 \times 6$ there are 16 pieces, but not each of them can be swapped with each of them. We can again try to visualize it by looking on the grid, we can do it for $4 \times 4 \times 4$, $6 \times 6 \times 6$, and $8 \times 8 \times 8$ Rubik's cube (Figure 2.7):

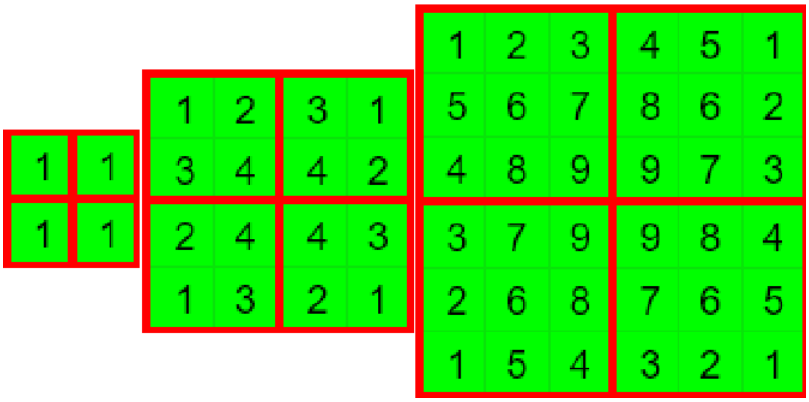

Fig 2.7. Grid of center of one face of $4 \times 4 \times 4$, $6 \times 6 \times 6$ and $8 \times 8 \times 8$ Rubik's cubes.
Source: author's elaboration

Again I highlighted the division of all elements into parts, that contains a member of each center pieces group. That means that all 1s can finish scrambling only in places of other 1 (it might be another color place). The same concerns other numbers. Again we have to isolate the number of groups of center pieces, we see that this is very similar to what we did in the case of odd layer cubes. we have to calculate the number of all center pieces in each case, and this is, of course, equal to *(n-2)x(n-2)*, and then we just need to divide this number by four, because we see that we can divide the whole grid into four equal squares, so the final formula for the number of possible scrambled states of center pieces is given:

$$\left(\frac{24!}{24^6}\right)^{\frac{(n-2)^2}{4}} \tag{2.9}$$

Then we have almost all of the parts we need to isolate the general formula for scrambled states of even layered cubes. We can sum up what we already have:

$$\left(\frac{24!}{24^6}\right)^{\frac{(n-2)^2}{4}} \cdot (24!)^{\frac{n-2}{2}} \cdot 88179840 \tag{2.10}$$

But as we said before even layered cubes differ a little bit from odd layered cubes. Another very important fact, which comes from the fact, that there are no fixed centers in even layered cubes, which provide recognition of each face, which is necessary to solve the cube, in another case we may end up with an unsolvable situation, and we would have to change the solution in a way that e.g. red face should be green, green should be orange, orange should be blue and blue should be red. But this wide topic is a topic for another time, an important thing to remember out of that is that because of that we may end up with 24 identical scrambles, which can only differ by rotation of a cube, but it is exactly the same scramble. We want to eliminate such a situation, that is why we will have to additionally divide our final formula by 24, so our final formula is given:

$$\left(\frac{24!}{24^6}\right)^{\frac{(n-2)^2}{4}} \cdot (24!)^{\frac{n-2}{2}} \cdot 3674160 \tag{2.11}$$

2.3. Generating results of a number of possible scramble states of $n \times n \times n$ Rubik's cube

Now, when we have a general formula for all sizes of Rubik's cubes, we could try to calculate those numbers. Due to the limited computing power that my computer possesses, we will be able to calculate them until results will exceed my capabilities. Main results are placed in Table 2.2.

Table 2.2. Number of possible scrambles of $n \times n \times n$ Rubik's cube

Size of cube	Number of possible scrambles states
$4 \times 4 \times 4$	7 401 196 841 564 901 869 874 093 974 498 574 336 000 000 000
$5 \times 5 \times 5$	282 870 942 277 741 856 536 180 333 107 150 328 293 127 731 985 672 134 721 536 000 000 000 000 000
$6 \times 6 \times 6$	157 152 858 401 024 063 281 013 959 519 483 771 508 510 790 313 968 742 344 694 684 829 502 629 887 168 573 442 107 637 760 000 000 000 000 000 000 000 000
$7 \times 7 \times 7$	19 500 551 183 731 307 835 329 126 754 019 748 794 904 992 692 043 434 567 152 132 912 323 232 706 135 469 180 065 278 712 755 853 360 682 328 551 719 137 311 299 993 600 000 000 000 000 000 000 000 000 000 000 000
$8 \times 8 \times 8$	35 173 780 923 109 452 777 509 592 367 006 557 398 539 936 328 978 098 352 427 605 879 843 998 663 990 903 628 634 874 024 098 344 287 402 504 043 608 416 113 016 679 717 941 937 308 041 012 307 368 528 117 622 006 727 311 360 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000

Source: author's elaboration

To check the correctness of our results. The paper https://artofproblemsolving.com/wiki/index.php/Rubiks_cube confirms that all of our results are matching the official numbers presented in that publication.

We see that the numbers are getting so high so fast that we will be unable to draw any reasonable conclusions out of that if we keep going. But there is one thing we can do about this.

When generally working, comparing, and analyzing such huge numbers, instead of comparing them in a normal way, we can compare their order of magnitude. Order of magnitude is simply speaking the number of powers of 10, that the number contains. It is almost the same as the number of digits in a number (the number of digits in a given number is always 1 bigger than the order of magnitude of a given number). For example, the order of magnitude of number 123456789 is 8, and of course, the number of digits is 9. In order to find out the order of magnitude, we need to put in the number into decimal logarithm and take the floor function of that number, because these numbers are usually not a whole number and we know that the order of magnitude is expressed by whole numbers. So to sum up what we want to do is to superimpose a decimal logarithm on a number of possible scrambles.

Assuming *f(n)* is a function describing the number of scrambles (for odd layer cubes) and *n* is variable, the function is stated:

$$f(n) = \left(\frac{24!}{24^6}\right)^{\frac{(n-2)^2-1}{4}} \cdot (24!)^{\frac{n-3}{2}} \cdot 43252003274489860000 \quad (2.12)$$

Now we superimpose a decimal logarithm on a function, we achieve:

$$log[f(n)] = log\left[\left(\frac{24!}{24^6}\right)^{\frac{(n-2)^2-1}{4}} \cdot (24!)^{\frac{n-3}{2}} \cdot 43252003274489860000\right] \quad (2.13)$$

Now we can transform the formula:

$$log[f(n)] = \frac{(n-2)^2-1}{4} \cdot log\left(\frac{24!}{24^6}\right) + \frac{n-3}{2} \cdot log(24!) + 19.636 \quad (2.14)$$

We could make further transformation, but I think in this form the formula is both clear and easy to substitute.

Next, we have to do the same operations for even cubes, so I will not repeat all the steps just give the final formula:

$$log[f(n)] = \frac{(n-2)^2}{4} \cdot log\left(\frac{24!}{24^6}\right) + \frac{n-2}{2} \cdot log[(24!)] + 6.565158 \quad (2.15)$$

Where *n* is an (odd) size of a cube.

Now, when we have established a general formula for the order of magnitude of a number of possible scrambles, we can substitute for the values (Table 2.3):

Table 2.3 Order of magnitude of the number of possible scrambles of $n \times n \times n$ Rubik's cube

n	log[*f(n)]*	*n*	log[*f(n)]*	*n*	log[*f(n)]*	*n*	log[*f(n)]*
4	45	16	933	28	2937	40	6058
5	74	17	1054	29	3152	41	6366
6	116	18	1189	30	3379	42	6686
7	160	19	1326	31	3610	43	7010
8	217	20	1477	32	3853	44	7346
9	277	21	1639	33	4099	45	7685
10	349	22	1795	34	4358	46	8037

11	425	23	1963	35	4619	47	8391
12	513	24	2145	36	4893	48	8759
13	603	25	2328	37	5170	49	9129
14	707	26	2525	38	5460	50	9512
15	813	27	2724	39	5752	51	9897

Source: author's elaboration

Looking at table 2.3 we see that these numbers are pretty huge, when we think about that, they are only the number of digits of some other numbers. Most of these numbers we will probably not be able to write down in a reasonable time. For example, the biggest mass-produced Rubik's cube is $17 \times 17 \times 17$, based on the numbers in the table we see, that if we want to write down the number of possible scramble states of that cube, we would have to write down 1055 digits. That shows us the vastness of that number. The results of calculating the order of magnitude of the number of possible scramble states of $n \times n \times n$ Rubik's cubes are presented in Figure 2.6.

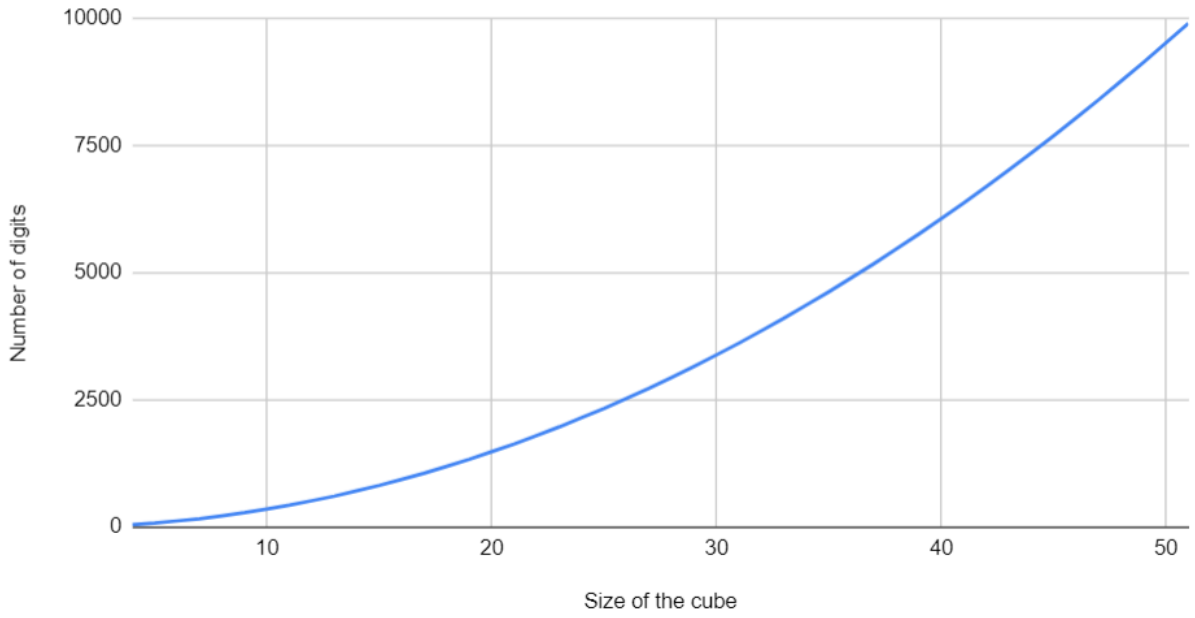

Fig 2.8. Order of magnitude of the number of possible scramble states of $n \times n \times n$ Rubik's cubes.
Source: author's elaboration

We can observe that the odd function and even function looked to be on the same curve. This may seem strange because the formulas are different, but the differences are imperceptible. But we have to remember that this is the plot of orders of magnitude, not the actual numbers, so it is even easier not to notice differences. Next, we easily can observe that the plot looked really 'nice', what I mean by that is that it looked like some simple function, probably a quadratic function. We can go back to our formulas 2.12 and 2.14 to analyze it:

$$log[f(n)] = log[(\frac{24!}{24^6})^{\frac{(n-2)^2-1}{4}} \cdot (24!)^{\frac{n-3}{2}} \cdot 43252003274489860000] \quad (2.16)$$

$$log[f(n)] = \frac{(n-2)^2}{4} \cdot log(\frac{24!}{24^6}) + \frac{n-2}{2} \cdot log[(24!)] + 6.565158 \quad (2.17)$$

We can observe that these both are quadratic equations and the elements $log(\frac{24!}{24^6})$; $log(24!)$; 19.636; 6.565158 are only coefficients, for which we could calculate the exact value, but for the simplicity of formula it is better to leave it in this form

Then we are left with the terms with our variable, they are in first or second power, these are:

$\frac{n-3}{2}; \frac{n-2}{2}$ - these are in 1st power;

$\frac{(n-2)^2-1}{4}; \frac{(n-2)^2}{4}$ - these are in 2nd power, 1st power, and a term without variable because when we transform it we will get term:

$\frac{n^2-4n+3}{4}; \frac{n^2-4n+4}{4}$

To sum up in both formulas in fact we only have terms with the variable in 2nd power, 1st power, and terms without a variable, so now it is clearly visible that it is a quadratic formula. Now we can easily understand where the shape of a plot came from.

3. OVERVIEW OF METHODS TO SOLVE RUBIK'S CUBE

The methods of solving Rubik's cube could be categorized into two groups: the ones aimed for humans to solve it; and methods designed for computers. The methods designed for computers are often much too complicated for humans or are 'invented' by computers. What it means is that a very popular approach to solving Rubik's cube nowadays is to teach a computer how to do that using artificial intelligence or/and machine learning. There are many papers that describe such an approach and discuss the results, for example the paper of McAleer, Agostinelli, Shmakov and Baldi (2018); Richard E. Korf (1997); Timo Mantere (2012). This is a very interesting approach, which often ends up with the computer solving a cube in a way that is too complicated for humans and we would not be able to do it in a given way, but thanks to the computer's capabilities, that is computing power it is possible from them. In this thesis, I decided to focus on methods that are more suited for humans and the ones which were invented by humans for humans.

This section aimed to provide an overview and analysis of the most popular and well-defined methods of solving Rubik's cube. There are many methods that we do not include in our analysis and further experiments for different reasons. Different methods differ mainly by the number of algorithms used, level of intuition, complexity, and versatility. In general, there is a simple relationship between the number of algorithms and the number of moves/times needed to solve the cube. The more algorithms the more specific cases you can solve by one/two algorithms instead of for example four or more. This is a very important thing for people who want to solve Rubik's cube. When people are learning to solve Rubik's cube, usually a very important thing is easiness because most people would get very frustrated and discouraged after a certain amount of time if they did not succeed. The absolute minimum of algorithms needed to remember is 4, but the solving experience is rather long, not efficient at all and some people might consider it wearisome. On the contrary in theory we could find out the shortest sequence of moves that needed to be applied in order to solve each individual scramble in an efficient way, but of course, knowing the number of possible scrambles it is rather impossible, but this would be perfect solutions. We need to compromise those two approaches. Methods with higher amounts of algorithms are usually intended for people who want to solve Rubik's cube efficiently, that is in the shortest possible time or in the shortest possible number of moves because they require more practice. Whereas methods with low numbers of algorithms are more suited for beginners.

To a theoretical computer scientist, the Rubik's Cube and its many generalizations suggest several natural open problems. What are good algorithms for solving a given Rubik's Cube puzzle? What is an optimal worst-case bound on the number of moves? What is the complexity of optimizing the number of moves required for a given starting configuration? (Demaine and Lubiw, 2014). There are several computational approaches for solving the Rubik's Cube Duberg and Tideström (2015); Stefan Pochmann (2008); Steinparz, Hinterreiter, Stitz and Streit (2019); Da-Xing Zeng et al. (2018); Jaslyn Lek (2009). In the group of Non-evolutionary Approaches the three most important being the work of Thistlethwaite, Kociemba, and Rokicki (El-Sourani, Hauke and Borschbach, 2010). Thistlethwaite's Algorithm (TWA) uses the pre-

calculated lookup-tables, sequences are put together that move a Cube from one group into another until it is solved (Thistlethwaite, M.B, 1981). Kociemba's Algorithm reduces the number of needed subgroups to only 2. Rokicki divides the problem into 2 billion cosets, each containing around 20 billion related configurations. With this method he was able to push the up- per bound to 22 moves sufficing to solve the Cube from any initial scrambled configuration (El-Sourani, Hauke and Borschbach, 2010; Rokicki, T.).

An evolutionary approach is presented by Herdy, which devised a method that solves the Cube using pre-defined sequences as mutation operators that only alter a few cubies (Herdy, M. &Patone, G, 1994). Another approach developed by Borschbach and Grelle (Borschbach, M. & Grelle, C, 2009), that first transforming the Cube into a 2x2x3 solved state, then into a subgroup where it can be completed using only two adjacent faces (El-Sourani, Hauke and Borschbach, 2010).

The methods described in this theses include the competitive approaches for solving the cube rely on increasing numbers of algorithms and increasing complexity of the intermediate goal states: (i) LBL (Layer by layer) method, designed by David Singmaster, which was published in „Notes on Rubik's 'Magic Cube'" in 1981 (Fogarassy, 2016); CFOP (Cross-F2L-OLL-PLL) method - he Friedrich method (Madan, 2020); (ii) Roux and ZZ method, which aimed to reduce the number of necessary moves to solve the cube (Madan, 2020); (iii) Petrus "SpeedCubing" method (Petrus, L); (iv) corners first or block-building method - is the basis of one of the fastest, Gilles Roux's method (El-Sourani, Hauke and Borschbach, 2010).

Generally, we can say that the best method would require a low number of algorithms but at the same time keeping the solve time/number of moves low. In reality, there is no such method, you need to find an equilibrium between those two factors. Of course, that is the case for humans, because it is not that easy to remember hundreds of algorithms for humans, and recognize each case immediately (in the case of time-solving). But that is not the case for computers, they can memorize almost unlimited numbers of algorithms. Eventually, the time of looking for the right case can be determined by the computing power of a given device and the complexity of programs, but in the case of a number of moves, this is not a problem. That is why I expect the methods with the highest number of algorithms to be the most efficient in the case of the number of moves.

There is always a solution to the cube that requires a maximum of 20 moves to solve the cube. It is commonly called 'God's number'. They have not defined a solution to each scramble as the number of them is so high. Some supercomputers are able to find those solutions but it is almost impossible to do it by a human in a reasonable time, let's say a couple of hours. So of course I do not expect such results and we will not aim for such results.

3.1. Layer - By - Layer (LBL) method

Layer - By - Layer (Fogarassy, 2016; Duberg and Tideström, 2015) method is commonly considered the easiest method to learn. It is safe to say that over 95% of people who learn how to solve Rubik's cube start with this method. It has a low number of algorithms required to memorize, which make it easy for people, also solving the cube with this method does not take too much time and that is also important because this could be discouraging for people who want to learn how to solve Rubik's cube. LBL compromises the number of algorithms and pleasant solving experience and, it consists of the following 7 steps:

3.1.1. White cross step

This step consists of putting four edges in correct places that are half wight, these are white-green, white-red, white-blue, white-orange (Figure 3.1). This step is pretty intuitive, and often people do not need to learn any algorithms, but when someone has problems it may require some algorithms, but the number is a maximum of 4. At this moment it is important to point out that instead of white/yellow (in the next steps) it could be any other two opposite colors like red/orange or blue/green, but it is agreed that the color to start with is usually white. Of course, professional speed-cubers can start with any other color, but that increases the level of difficulty significantly. Also for now on each next step we assume, we will not break any progress previously made.

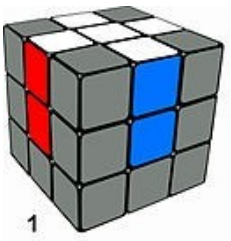

Fig 3.1. The first step of solving Rubik's cube by LBL method, white cross
Source: https://pl.wikipedia.org/wiki/Layer_by_Layer

3.1.2. White corners step

It consists of putting four corners that are partially white in the correct places. These are white-green-red, white-orange-green, white-blue-orange, white-red-blue. Now we have finished the first layer (as the name said layer by layer), and now we will keep making the next layers. This step requires 1 algorithm, but the algorithm will be repeated multiple times (Figure 3.2).

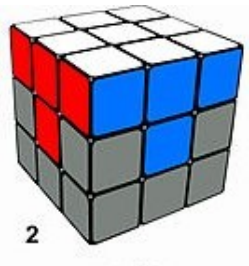

Fig 3.2. Second step of solving Rubik's cube by LBL method, white corners
Source: https://pl.wikipedia.org/wiki/Layer_by_Layer

3.1.3. Middle layer (second layer)

This step consists of putting all middle layer edges in correct places, these are red-green, green-orange, orange-blue, blue-red. Now we have finished the first two layers, all that is left for us to do is to solve the last layer. Of course, it is the most complicated step, and will require us the most time. This step requires 2 algorithms, and they also will be repeated multiple times (Figure 3.3).

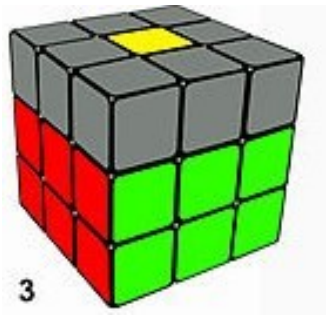

Fig 3.3. The third step of solving Rubik's cube by LBL method, the middle layer
Source: https://pl.wikipedia.org/wiki/Layer_by_Layer

3.1.4. Yellow cross orientation step

It consists of placing all half yellow edges in the correct orientation, but they might be swapped, one with each other. These edges are yellow-green, yellow-red, yellow-orange, yellow-blue. This step also requires 1 algorithm (Figure 3.4).

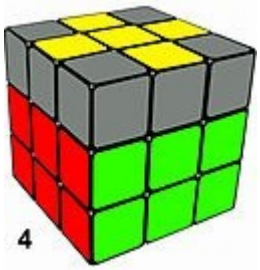

Fig 3.4. The fourth step of solving Rubik's cube by LBL method, yellow cross orientation
Source: https://pl.wikipedia.org/wiki/Layer_by_Layer

3.1.5. Yellow cross permutation step

It consists of permuting previously turned edges so that they will be not only correctly oriented but also correctly placed. This step requires 1 algorithm but may be repeated two times (Figure 3.5).

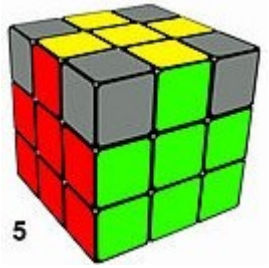

Fig 3.5. The fifth step of solving Rubik's cube by LBL method, yellow cross permutation
Source: https://pl.wikipedia.org/wiki/Layer_by_Layer

3.1.6. Yellow corners permutation step

It consists of permuting all partly yellow corners in the correct place, but not necessarily correctly oriented (twisted). This step requires 1 algorithm, but also may be repeated multiple times (Figure 3.6).

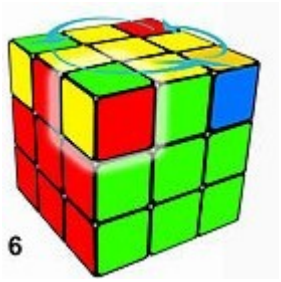

Fig 3.6. The sixth step of solving Rubik's cube by LBL method, yellow corners permutation
Source: https://pl.wikipedia.org/wiki/Layer_by_Layer

3.1.7. Yellow corners orientation step

It is the last step, and it consists of orienting (twisting) yellow corners so that they will be correctly oriented, because, after the last step, they are in the right places, but probably incorrectly oriented. This is the step when the last (third) layer is solved. This step requires 1 algorithm but will be repeated multiple times (Figure 3.7).

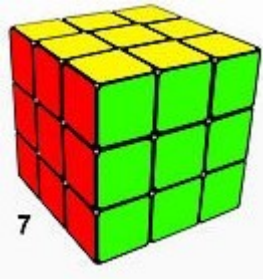

Fig 3.7. The seventh step of solving Rubik's cube by LBL method, yellow corners orientation
Source: https://pl.wikipedia.org/wiki/Layer_by_Layer

After performing all these steps Rubik's cube should be finally solved. Depending on how easy the white-cross goes the number of algorithms necessary to remember is from 7-10, which is really low, compared to the methods I will present later.

As we mentioned before this method is excellent for beginners, because it is rather easy, but definitely is not that good considering the numbers of moves/time necessary to solve. In case of time it is possible to solve Rubik's cube under 1 minute, maybe 40/45 second, but lower than that is rather impossible, or very hard.

While implementing this method I will need a list of algorithms, which will help me implement this method. I helped myself with the algorithm from the website: https://rcube.pl/jak-ulozyc-kostke-rubika-3x3x3-lbl , where they are very well described and helped me write the code easily.

3.2. CFOP (Cross – F2L – OLL – PLL) method

The next method we want to present is CFOP (Friedrich/CFOP), the name might seem strange, but it is easily the shortcut from C - Cross; F - F2L (first two layers); O - OLL (orientation of the last layer); P - PLL (permutation of the last layer). This method is much more advanced and is more suited for people who are more advanced and have bigger experience in solving Rubik's cube (Duberg and Tideström, 2015; Madan R, 2020). It required much more algorithms, but at the same time, there are fewer steps, because only 4 instead of 7 like in LBL. This method is the most popular method for professionalists because it lets you solve the cube in under 10 seconds. Of course, this requires a lot of training and memorizing algorithms, but going under 40/30 seconds is pretty common with this method, even without that much training. CFOP method consists of the following four steps.

3.2.1. White cross step

This step is exactly the same as before, in LBL, so there is not much to say, but in this method, it is definitely required to be able to solve cross intuitively, because this method is usually used to solve in the lowest time possible (Figure 3.8).

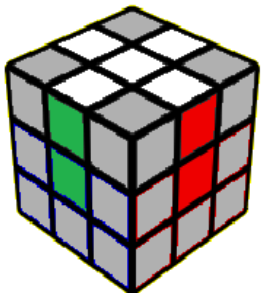

Fig 3.8. The first step of solving Rubik's cube by CFOP method, white cross
Source: https://pl.wikipedia.org/wiki/Metoda_Fridrich

3.2.2. F2L - first two layers step

This step is actually the combination of step 2 and 3 from LBL, but it consists of solving two-piece at the same time, for example, white-blue-red corner and blue-red edge, we need to build the 1x1x2 block in a way, that they will be next to each other and match each other, and then put them both at the same time into correct places (Figure 3.9).

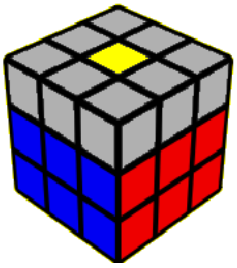

Fig 3.9. The second step of solving Rubik's cube by CFOP method, F2L
Source: https://pl.wikipedia.org/wiki/Metoda_Fridrich

The example is presented below (Figure 3.10):

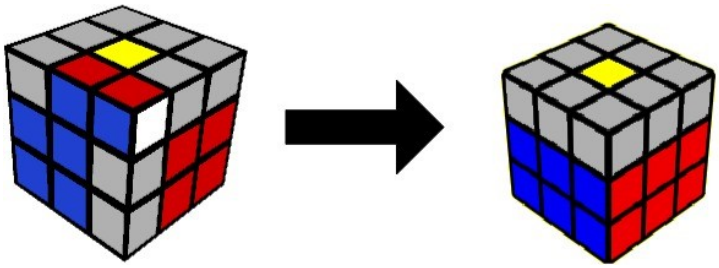

Fig 3.10. Example of F2L case Source: https://pl.wikipedia.org/wiki/Metoda_Fridrich (edited)

In this step there can occur 42 different cases of positions each corner-edge pair, but it also includes the case where it is already solved, so forgetting this one case it leaves us with 41 cases, and generally, for people with high skills, some of them, or maybe even most of them (or sometimes all of them) can be done intuitively, but if we assume our skills are not that advanced we need for that step 41 algorithms.

3.2.3. OLL - orientation of last layer step

Starting from this step, we can see a different approach to solve the rest of the cube than in an LBL method. Instead of solving edges first and corners later, now we want to correctly orient all unsolved elements in a way that all of them are facing a yellow sticker up, but they can be permuted incorrectly. This step required 57 algorithms to memorize, so it is pretty much, especially for someone whose skills are not on a high level (Figure 3.11).

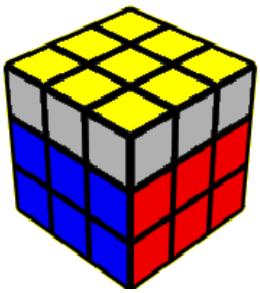

Fig 3.11. The third step of solving Rubik's cube by CFOP method, OLL
Source: https://pl.wikipedia.org/wiki/Metoda_Fridrich

3.2.4. PLL - permutation of last layer step

This step is the last one and it is supposed to solve the rest of the cube. From the previous step, we know that all pieces are oriented correctly, so we only need to permute them, so that all elements will be in the correct places. This step requires learning 21 different algorithms (Figure 3.12).

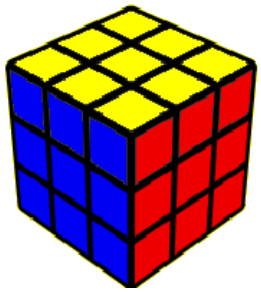

Fig 3.12. The fourth step of solving Rubik's cube by CFOP method, PLL
Source: https://pl.wikipedia.org/wiki/Metoda_Fridrich

We see that this method is much more condensed than LBL. We can think about it in a way that every step (except the first one) compresses two steps from previous steps. Steps 2 and 3 from LBL are equivalent to step 2 in CFOP, steps 4 and 7 from LBL are equivalent to step 3 from CFOP, and steps 5 and 6 are equivalent to step 4 in CFOP. While solving with this method in each step we need to recognize the situation and then apply the correct algorithm, because in each step we know an algorithm for each possible situation. But there is a price for that. The total number of algorithms needed to learn is 119. It is definitely a lot, but only for a human.

This time again I needed a reliable source of algorithms for my implementation and I managed to found a list of algorithms for each of last three steps, which was created by one of the best speed-cuber in the world - Feliks Zemdegs, which guarantee they are correct, these papers are :

- F2L Algorithms (First 2 Layers)
- OLL Algorithms (Orientation of Last Layer)
- PLL Algorithms (Permutation of Last Layer)

  All of them are developed by Feliks Zemdegs and Andy Klise.

4. EXPERIMENTAL STEUP

4.1. Experiment Plan

Main goal of our work is to find out which method of solving Rubik's cube is most efficient. The aspect which we will focus on this work as a criteria for efficiency evaluation is the number of moves required to do in order to solve the Rubik's cube. Overview of my plan can be states as follows:

1. Choosing the tool for experiment performing
2. Building the instrument for solving Rubik's cube method simulation
3. Generating scrambles
4. Simulating the solves and generating the results
5. Defining optimizing strategy
6. Building the instrument for performing Rubik's cube solving optimization
7. Performing the optimization experiments
8. Results analyzing and validation

4.2. Tools and Instruments

In order to realize the experimental plan, we needed to choose a programming language and a program that will help me do this. R language was chosen for our work. The main reason is the package '*cubing*' that is already created in R and is open to use, it was created by Alec Stephenson (2018) for visualizing, animating, solving and analyzing the Rubik's cube. The basic functions that we needed to use are *getCubieCube()* and *move()*. The first one lets create an object that represents Rubik's cube and the second one allows to do certain moves, which was of course necessary to solve Rubik's cube. The general form of an object of type *CubieCube* consists of 5 lists of integers. They store information about arrangement of elements on a cube. Each lists has a name that is a shortcut form information that it represents, because each of them store specific information, that is:

Table 4.1 Explanation of lists inside *CubieCube* object

Name of list	Explanation of shortcut	Information stored
cp	corners permutation	It stores information about position of each corner piece
ep	edges permutation	It stores information about position of each edge piece
co	corners orientation	It stores information about rotation of each corner piece
eo	edges orientation	It stores information about rotation of each edge piece
spor	spatial orientation	It stores information about location of the fixed center pieces

Source: https://cran.r-project.org/web/packages/cubing/vignettes/cubingintro.pdf

As we see in Table 4.1, inside the object *CubieCube* we have all necessary information about elements in the Rubik's cube. In other words, based on those 5 lists we could place stickers in a Rubik's cube and get exactly the same cube that is supposed to be stored in a given object. Based on that information the package '*cubing*' lets us visualize the objects by plotting them. The plot function shows us a 3D object which is Rubik's cube on a two dimensional surface, as it is shown in Figure 4.1.

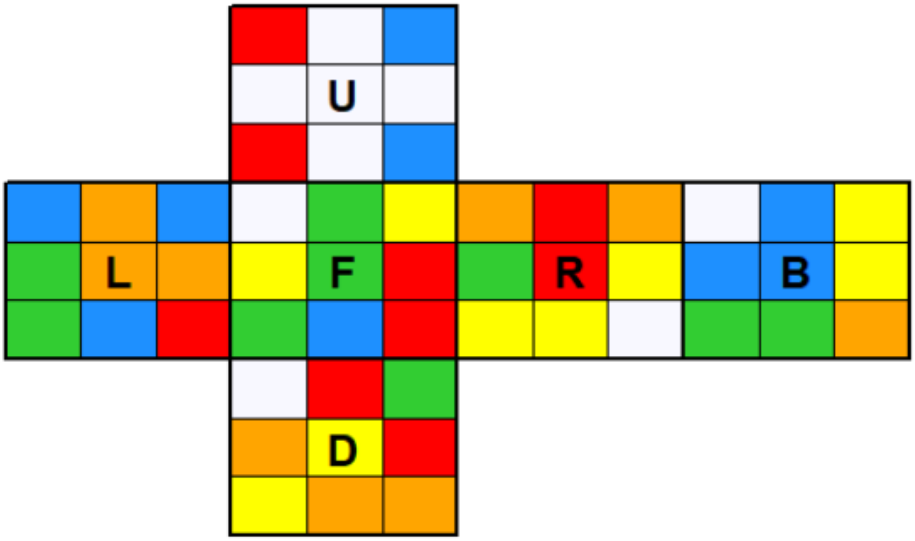

Fig 4.1. Example of 2D representation of *CubieCube* object
Source: Rstudio

This visualization is very helpful in the process of constructing algorithms and checking whether our program works properly.

4.3. Generating scrambles

One of the main conditions for the implementation of the Rubik's cube solving optimization experiment being carried out is to ensure the correctness of the Rubik's cubes scrambling stage. To do that we have to prepare some lists of moves that I will need to apply on the solved Rubik's cube to scramble it. Such a list we will simply call a scramble. Then I can apply scrambles on a solved Rubi's cube and test my program. I will check whether each scramble is solved by my program. I will do that by running some function with a loop, and then, when some mistakes would occur I could follow the solve process of the given scramble to locate the mistake and correct it. Later it will be needed to solve the Rubik's cube and generate results. I could use the build-up function in R from package "cubing". Here is why I decided to use a different approach. If I would generate random scrambles while generating results, there is always a chance that for one method scrambles will be more 'lucky' and for another 'unlucky'. By generating the scrambles firstly, I guarantee that my results will be 'fair'. Why? Because each method will be tested on exactly the same sample, so the only thing that will differ results of each method is actually the methods themselves. I decided to use an online generator of scrambles from the website: https://ruw4ix.com/puzzle-scramble-generator/. I downloaded 3000 scrambles and did

some text processing, where I used notepad++ which was the perfect tool for that situation. I could write there a macro, which allows me to adjust the scrambles to the form that can be used by R. Next I could upload generated scrambles into the Rstudio and start working with them. This is the final form of the scrambles (Figure 4.2):

```
scramble1 <- "D B2 R' B L' F2 B' R2 F' D2 F R L D U2 L' R2 B2 L R F2 U2 L' R F'"
scramble2 <- "L2 B2 U F U' L2 R' B2 D L' B U' B U L R U' B2 L' F' B2 U' R' F2 B'"
scramble3 <- "B' L U' B F2 R' F R' L' B' R' F2 U2 F' R2 D U2 F' U2 R2 U2 F B L2 F2"
scramble4 <- "U B' D' U' R F R F' L' R' U' F' B R2 L F' R U' R2 D R U2 L2 F2 U'"
scramble5 <- "D L2 F2 R2 F' U2 F' B2 U D' R2 U B2 R D' R' D2 U' L2 U L B R' B2 U"
scramble6 <- "B' U' R U2 D2 B' F' R F L2 U' R2 L D' F2 D2 F2 R L' D L B' U R' L'"
scramble7 <- "D F2 D' B R2 F2 D R' L D' L D' R' U2 D B2 L2 R U B' L B2 U D2 L"
scramble8 <- "U' D2 L' R2 U B F L' D' R D2 U' F2 L D' U' F U2 L' R F2 B2 L F2 D"
scramble9 <- "B2 F2 R2 D' R2 U F L' D U2 L2 D' R' D L2 F2 B U2 F2 R2 L U R' B L2"
scramble10 <- "R2 L' B F' L' U' F' U D' B' F U L B R2 L B' F2 L F' L2 F2 U2 F L"
scramble11 <- "R' F2 R' B2 D2 L' R B' D U R' B2 R' U F U2 R U2 R F2 L' U' B' L2 B"
scramble12 <- "L' F2 R' F' D B F2 L' U2 R' D' R F2 U2 F2 U2 F' R F' R' D B2 U R B2"
scramble13 <- "D' L2 B' U2 R F2 R' U R L' D R' U2 D' F L' B2 L' U R2 F' R2 D2 F2 R'"
scramble14 <- "D' L' U D' R2 F2 D' L2 F' D2 L2 U2 B2 U2 F R2 F' B R U D' F2 R' L' U2"
scramble15 <- "L' R D U F D' U' R U2 D R2 D' F2 L2 B R2 L U L2 U D2 R2 F' B D"
scramble16 <- "F B' D' B' D' R L' D' F L' R U D B2 D R B D2 L' U R F2 D F B2"
scramble17 <- "F' R U' D2 R L' F' R' L' F U2 R2 B D F L' R D L' F U2 L2 F' D' L'"
scramble18 <- "D' U F L R' B' L U' B' L R D2 F2 D F D' F2 L2 B2 R B F' L2 B' U'"
scramble19 <- "R2 B L' U L' B' F' U' R2 L B U F2 L R2 B2 U2 D2 L2 R2 D' U2 F D' L"
scramble20 <- "F D' R2 B' F' D L D2 B' D F' L' B F U L2 B U2 D' L B' D' B' L2 F'"
```

Fig 4.2. Examples of scrambles generated and adjusted to R
Source: https://ruwix.com/puzzle-scramble-generator/ (adjusted to proper form by author)

As we see the letters in each scramble stand of course for each move as it was explained in chapter 1.2. Having those lists we can scramble Rubik's cube, so for example the moves " D B2 R' " would tell us to turn down the face, then double turn back face, and then turn right face in opposite direction.

5. RUBIK'S CUBE SOLING OPTIMIZATION STRATEGY

In this section, we aim to realize the main goal of our study - to find a way to optimize the results of solving Rubik's cube by selecting two methods. As the main criterion of optimization, we will use the number of moves during the solving process.

Unfortunately, not each step of each method can be optimized. Of course, a method as a whole can be called optimized when each possible to optimize step will be optimized. It is the case, because not every step can be optimized and it is caused by the way that each method is defined. That easily means that the inventor of a given method did not give us a room for improvement. Some steps are defined in such a way that by following the rules of a given method we cannot optimize this step. That means that of course some steps could be done in a better, more efficient way, but that would require us to use a different approach and not the one which is given by the inventor of the method. Usually when the step can not be optimized that means that it is defined in way that there is strictly said in what situation what we need to do and there is one and only one possibility, if we would like to do a given step differently we can of course do this, but then we can not say that we solve the Rubik's cube using that method. Steps which we can optimize usually consist of solving some parts of the cube that need multiple elements to be solved. For example, cross (Fig 2.1) consists of solving 4 edges and we have to solve each of them separately and solving each of them is already optimized by the inventor of the method, so for example we do not use then 8 moves, when 4 are enough. But the order is significant, because it matters which one we choose to do as a first, which as a second and so on. This is where we have room for improvement. We will be trying to figure out the best possible order of solving those elements. That of course means solving them in as few moves as possible. So first of all we have to establish which step can be optimized (Table 5.1).

Table 5.1 Possibility of optimizing each method of solving Rubik's cube

LBL steps	Possibility of optimization	CFOP steps	Possibility of optimization
Cross	YES	Cross	YES
First layer corner	YES	F2L	YES
Second layer	YES	OLL	NO
The orientation of the last layer cross	NO	PLL	NO
Permutation of last layer cross	NO		
Permutation of last layer corners	NO		
The orientation of last layer corners	NO		

Source: author's elaboration

We can see that overall, 5 of 11 steps can be optimized. Two main approaches have been chosen as optimization strategies: semi-optimized, and fully optimized. In the case of the Cross step, the optimization method is set by default as follows: first, we will solve the white-orange edge, then white-red edge, then white-green edge, and lastly white-blue edge. But there is no evidence that this method is optimal in the case of a Cross step. Now we can show what

these two approaches will consist of. In this regard, both methods will be applied to all optimization steps.

5.1. Semi-optimized strategy

In the semi-optimized strategy, we will firstly check which out of 4 available options takes the fewest moves to solve, then choose it, and then repeat that step two times in order to pick the second option and the third option, and obviously, for the last one, there is only one left. It probably will be a good improvement, but it is not hard to notice that it is not the most efficient version, there is the possibility that a non-optimized version would be better, than a semi-optimized version. It may seem strange, but that is the case. Why? Because in this part there is, of course, a lot of randomness included, because for example (in some particular scramble) we can do some step in an inefficient way, and then the next step will accidentally get very short to solve, and in the case when we optimize some step it may turn out that it will accidentally cause next one to be longer. So the optimization process will in general shorten the number of moves required to solve Rubik's cube, but it will not work in 100% cases as we planned. But of course, we want to generally achieve better results, not better results in some particular scramble. That is why it is anyway worth doing. In Figure 5.1 we have a scheme of how the algorithm will be working. We see that in fact there are three steps repeated and then there is setting the order and execution of them.

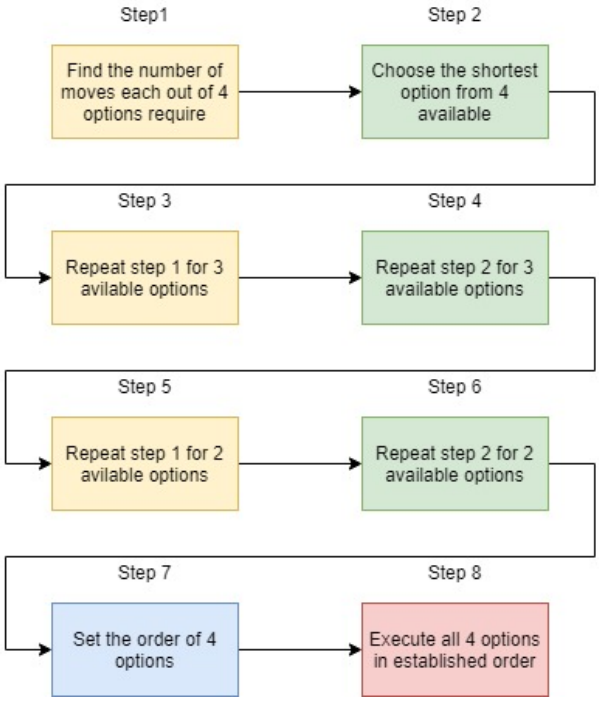

Fig 5.1. The algorithm of the semi optimized strategy
Source: author's elaboration

5.2. Fully-optimized strategy

In the fully-optimized strategy, we will consider all possible orders of solving those edges. It is not hard to calculate that there are 24 possible orders of those edges, because the first one we can choose in 4 ways, the second one in 3 ways, third in 2 ways and last 1 in one way. Then we have to multiply those numbers and we get $4 \cdot 3 \cdot 2 \cdot 1 = 24$. Knowing that we will only have to check each of them and choose one option with the lowest number of moves. This approach will greatly decrease the chances of this optimized version being worse than the non-optimized. Again, why? Because we consider each step separately and in each step this approach guarantees the best possible number of moves, but after this step, the next step may meet an unlucky position and may need to make so many moves, that in general, it will be not efficient to optimize. The perfect optimized version would consider each case of each step with each case of another step and so on and so on. But this would require powerful computing power, which I am afraid I can not provide with my equipment, but ideally, this would be the perfect version. What I can afford is to optimize each step separately. But I think that this version of optimization will be very close to the one I described as too complicated to run the simulation on my computer. For sure my fully-optimized version of each step will be better (or eventually the same) than the non-optimized version of that step. Also definitely I can say that in most cases (scrambles) my fully-optimized version will decrease the number of moves required to solve the whole Rubik's cube, comparing it to the non-optimized version. In Figure 5.2 we see the scheme of how the fully optimized algorithm will work. We see that we can end this algorithm only when all possible orders are checked and this is what we wanted to achieve.

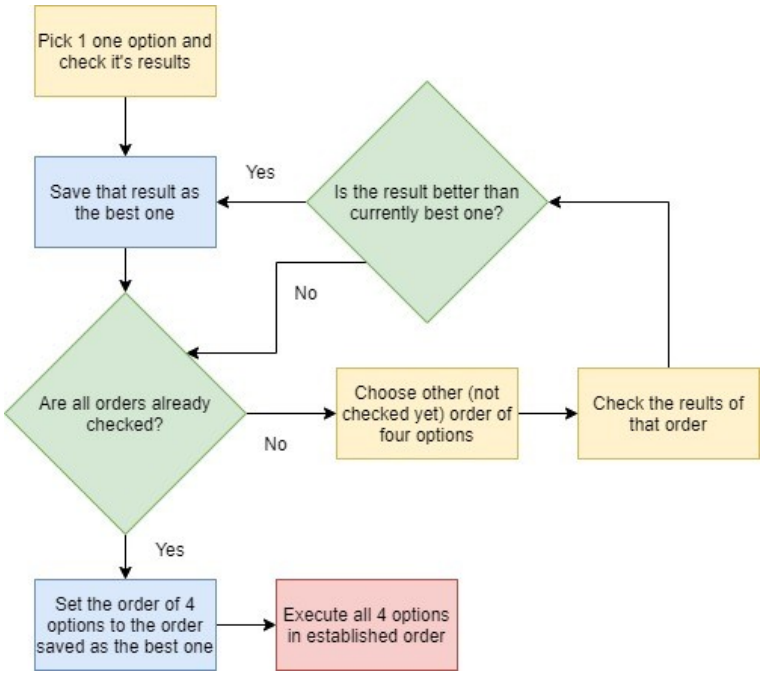

Fig 5.2. The algorithm of the fully optimized strategy
Source: author's elaboration

6. EXPERIMENT RESULTS

This section is devoted to the presentation and analysis of the results of the experiments. In accordance with the settings of the experiment, the plan for the experiment and the subsequent analysis of its results consists of the following stages: (1) Analysis of the efficiency of not optimized two selected solving Rubik's cube methods (LBL and CFOP); (2) Comparison not optimized methods results; (3) Analysis of the efficiency of semi- and fully-optimized strategy for two selected solving Rubik's cube methods; (4) Comparison of methods against each other or not optimized strategies versus optimized ones; (5) Results validation. The R-language code for realizing all steps of experiments presented in GitHub repository[1].

6.1. Analysis of the efficiency of not optimized solving Rubik's cube methods

6.1.1. LBL

As the main goal of my whole work is to find out the level of efficiency of each method, the first things that should be included are basic information and statistics about this method. As I mentioned previously, I simulated 3000 solves, so this is my sample for those statistics (Table 6.1).

Table 6.1 Basic statistics of the number of moves of the LBL method, not optimized

	Mean	Median	Standard deviation	Minimum	Maximum
LBL	125.3	124	17.7	55	176

Source: author's elaboration

As we see the mean is almost the same as the median. That tells us that presumably, the histogram will be quite symmetric. We can easily check the (Figure 6.1):

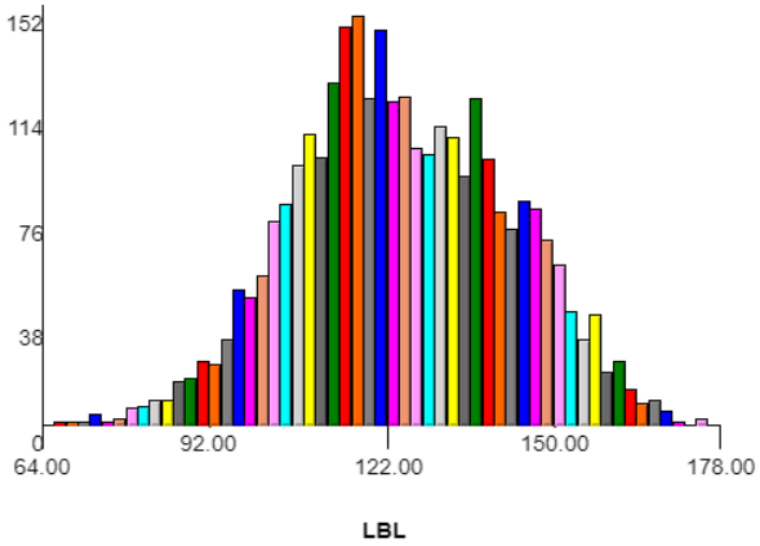

Fig 6.1. Histogram of frequency of the number of moves during the solving process by LBL method
Source: author's elaboration

[1] https://github.com/bryla121/Efficiency-of-methods-of-solving-Rubik-s-cube-implemented-in-R

We see that it is in fact the case. What we can also observe is that the shape of the obtained histogram in Fig 6.1 is really close to normal distribution. It is not perfect for a simple reason: the number of simulations. If the number would be smaller the shape would differ from the normal distribution even more. On the other hand, if we would increase the number of simulations the shape would be more and more similar to normal distribution. As I previously said it would be too problematic to run more than 3000 simulations, so this is an appropriate number of observations we can use, because we see that the shape is anyway close enough. But of course, we could increase the number of simulations endlessly and get more accurate results each time, but it is not necessary.

Looking at Table 6.1 we see that the minimum and maximum are pretty much far from each considering that the difference between them is equal to 121, whereas the standard deviation is equal to only 17.7. But of course, these are only singular observations, which occurred as a result of great luck or great bad luck. We see in Figure 6.1 that they do not occur multiple times, but it is surely possible, it is just very unlikely.

The next thing is partial results, which are the results of each step (Table 6.2). Looking at those statistics we will be able to find out information like, which step takes the highest number of moves to execute. This will be also important numbers considering future comparison with different methods and optimization of methods. Here we have the statistics for each step:

Table 6.2 Basic statistics of each step of the LBL method, not optimized

	Mean	Median	Standard deviation	Minimum	Maximum
Cross	10.442	11	2.272	2	16
White corners	18.476	18	3.870	6	28
Second layer	36.688	38	7.959	7	60
Yellow cross orientation	5.948	6	2.972	0	12
Yellow cross permutation	8.790	9	4.580	0	17
Yellow corners permutation	9.429	8	4.466	0	16
Yellow corners orientation	35.126	27	13.870	0	52

Source: author's elaboration

We can compare the share of each step in the whole solve process. As we see on the pie chart (Figure 6.2), the proportion of shares of each step are far from being equal. We see that each of the second layer and yellow corners orientation take almost 30% of the whole solve process. I will go over why each step takes a given number of steps in a moment when I will focus on each step individually. For now, while looking at Fig 6.2 it is important to remember that it is an average percentage share, it does not mean that some step in each scramble took exactly that much time out of the whole solve process. In some particular scramble, it can happen that some

steps take 0 moves, or two times more moves than the average, but this pie chart is only supposed to show us the overall proportions.

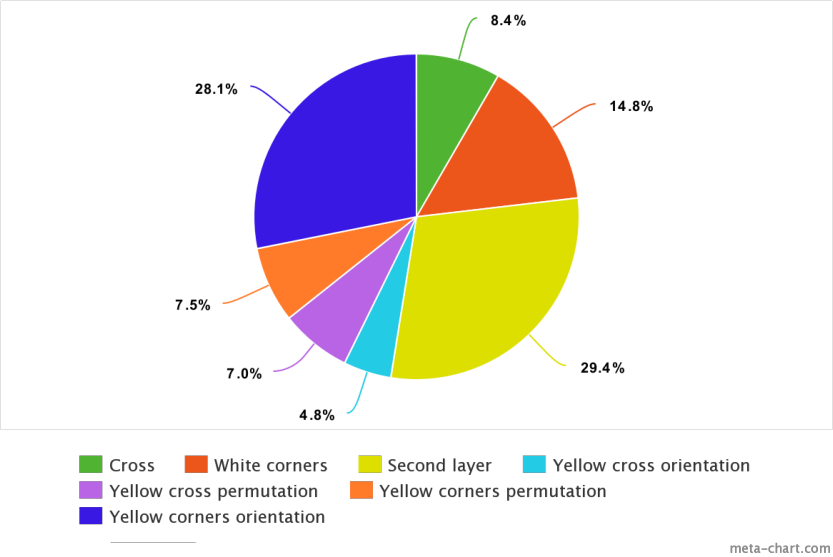

Fig 6.2. Pie chart of share of each step of the whole solve process, LBL method, not optimized
Source: author's elaboration using https://www.meta-chart.com/

Now we can analyze the results on the histogram form of each step (Figure 6.3). So, in the first step, we see that the shape is quite similar to the shape of the histogram in Fig 6.1, which is quite logical, that tells us that there are no unexpected results in this step. We see that the numbers circle around 10, so comparing this to the number of the whole solve is not that high as it is also seen in Fig 6.2 it takes about 8.4% of the whole solve. The range is quite high, as we see in table 6.1 it is equal to 14, whereas the standard deviation is equal only 2.27 as it is stated in table 6.1, but again this is caused by singular outliers and they do not happen frequently as it is seen in Fig 6.3.

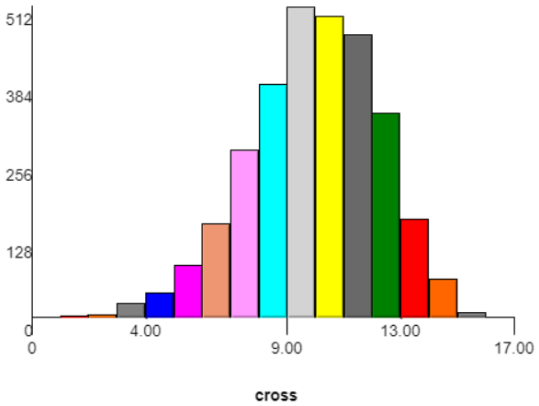

Fig 6.3. Histogram of frequency of the number of moves during the solve process by LBL method step 1, cross
Source: author's elaboration

In the case of white corners (step 2) again we see that the shape is similar to the shape (Figure 6.4) of the histogram in Fig 6.1, it is close to normal distribution. Here we see that the number of moves is higher than in the first step (Fig 6.3), so this step definitely takes more moves

to complete than the white cross. One of the reasons for that is of course that now we have to be careful not to destroy any progress previously made. Also as we see in Fig 6.2 this step takes on average 14.8%. As we see in table 6.1 again the range is equal to 22 and it is quite high compared to the standard deviation which is equal to 3.87 and it is rather small, but as previously the outliers do not happen too frequently.

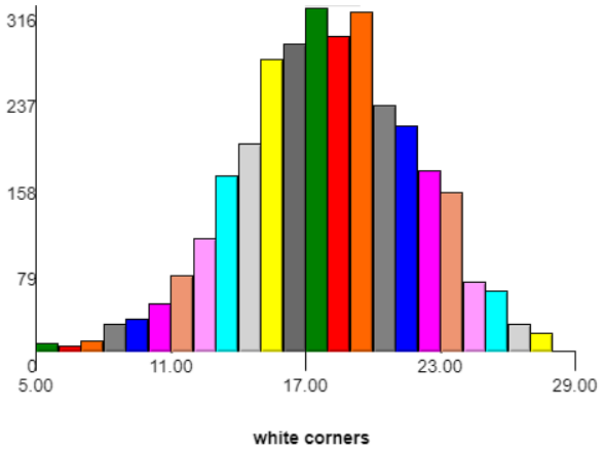

Fig 6.4. Histogram of frequency of the number of moves during the solving process by LBL method step 2, white corners
Source: author's elaboration

In the case of the second layer (step 3), the first time we see something strange considering the histogram in Figure 6.5. The overall shape looks a little bit like the normal distribution, but there are visible significant gaps or/and a decrease in frequency.

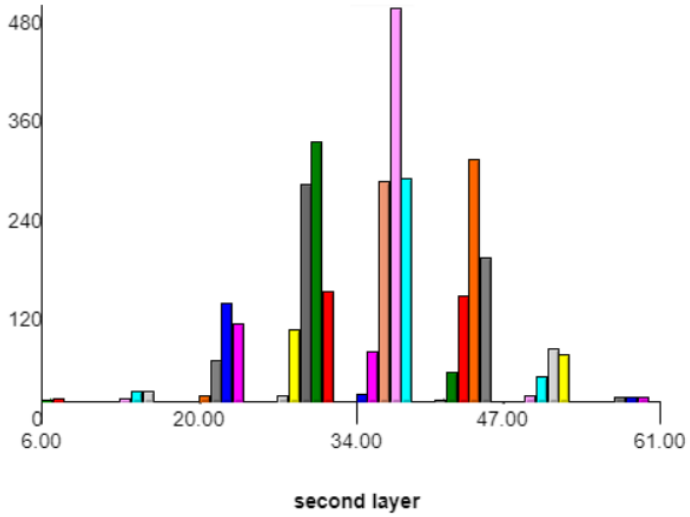

Fig 6.5. Histogram of frequency of the number of moves during the solve process by LBL method step 3, the second layer
Source: author's elaboration

These conclusions might seem strange but there is a reason for that. As we mentioned in chapter 3.1.3. this step consists of putting four edges (that should be placed in the middle layer) in the correct places. This comes from the fact that in order to put one edge in correct places we

have to put the given edge in some place (that is strictly specified) with some so-called ‘set-up moves’ and then and only then execute some algorithm. What does it mean?

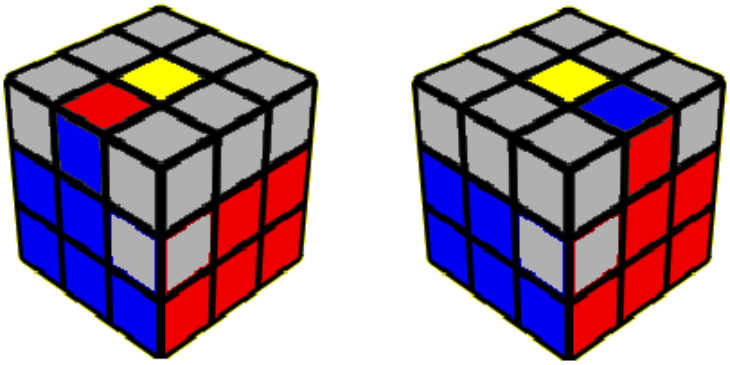

Fig 6.6. Effect of setup moves in the second layer
Source: author’s elaboration

As it is shown in Figure 6.6 these are two situations which we want to achieve as a result of ‘set-up moves’ and then we can execute an algorithm, which places (in this case) red-blue edge in the correct place. Those setup moves are actually usually zero moves (if we already have one of the cases from the picture), one move or eventually more moves if the given edge is in the second layer but in the wrong place, then we have to execute the algorithm once again in order to exchange the edge with any from the upper layer. This is why the whole second layer consists of some number of ‘set-up moves’ and some number of times the algorithm needs to be executed. The length of the algorithm is equal to 7. So to sum the number of moves in this step can be expressed by the formula:

$$A \cdot (7 + S) \tag{6.1}$$

where: *A* means number of times an algorithm needs to be executed; *S* means a number of set-up moves (0 or 1).

Looking at the formula we see that it is impossible to achieve results like e.g. 10, 19, 25 and so on. These numbers are the gaps that are visible in Figure 6.5.

Based on Figure 6.2 we see that this step requires a lot of moves to complete, this is the longest part in the whole solve process, it is equal to 29.4%. As we see in the table 6.1 Range is also very high, because is equal to 53, but looking at the formula 6.1 we see that the number is highly dependent on the value of *A*, if it is high the whole value will be high, and if it is low the number will be low. But actually, in Figure 6.5 we see that the value of *A* is normally distributed because the most frequently it is equal to 5, also often 4 and 6, sometimes 3 and 7, and other values occur very rarely.

In the case of the yellow cross orientation (step 4) again we see a strange histogram in Figure 6.7. But again there is a reason for that. Similarly as before the number of moves in this step is dependent on the number of times some algorithm is needed to be executed in order to finish this step. Here the case is very simple, we need to do the algorithm 0 times, 1 time or 2

times. There is no other option. The length of the algorithm is 6. The shape is rather non-normal distribution this time because this time the number of times the algorithms have to be executed is closely related to the construction of the cube and each case. I will not go over each case, but I will just say that there is 1 possible state of edges, where there is no need to execute an algorithm. There is also only one possibility that the algorithm will be executed two times, and 6 possibilities that the algorithm will be executed once. When we take that into consideration the histogram in Fig 6.7 starts to look quite logical. Here there is no sense in analyzing range and standard deviation from Table 6.1 because we only have three distinct values in this data set.

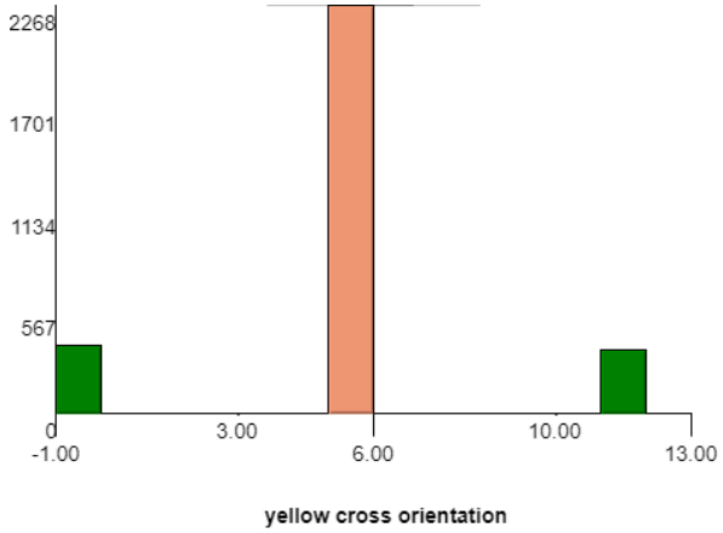

Fig 6.7. Histogram of frequency of the number of moves during the solving process by LBL method step 4, yellow cross orientation
Source: author's elaboration

The next step is a yellow cross permutation and again we see that the histogram in Figure 6.8 does not look similar to a classical histogram. The situation is similar to the two previous steps (Fig 6.5 and 6.7). In this step, we also have three groups of columns, but this time they are two columns in each group. Again there is an algorithm, which can be executed 0, 1 or 2 times. Besides that there might occur situations where we need to do 'set-up move', but this time only one, independently of how many times we will execute the algorithm. The length of the algorithm is 8, so we can the formula for the number of moves in this step would be:

$$8 \cdot A + S \tag{6.2}$$

where: *A* means the number of times an algorithm needs to be executed (0, 1 or 2); *S* means the number of set-up moves (0 or 1).

Now we see why we did not achieve any results from the gaps in Fig 6.7, like 3,4,5,11,12,13 and so on. In this case, it does not make sense to focus on statistics like range or standard deviation from Table 6.1, because we have only 6 distinct values.

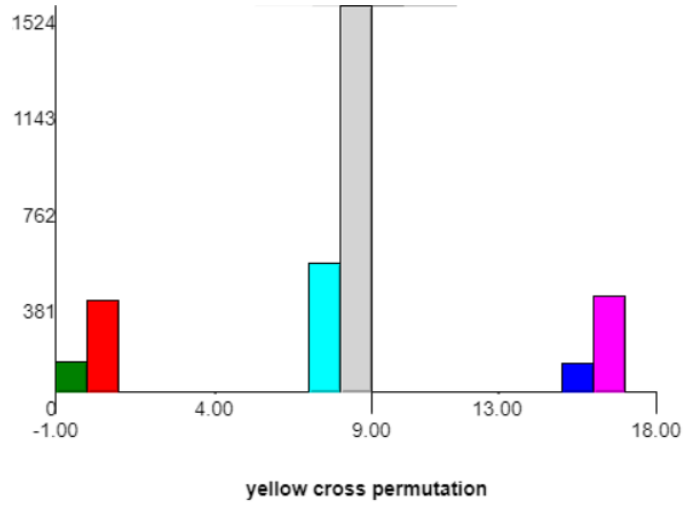

Fig 6.8. Histogram of frequency of the number of moves during the solve process by LBL method step 5, yellow cross permutation
Source: author's elaboration

The next step (Figure 6.8) is very analogical to step 4 (Figure 6.7) because we only have three columns. Again this comes from a fact that we might end up in a situation where we have to execute the algorithm 0, 1 or 2 times. The only difference is that the probability of each situation is different. There is one situation where we do not have to execute a given algorithm, 8 situations where we have to execute the algorithm 1 time and 3 situations where we have to execute the algorithm twice. Looking at the graph we see that the proportion is approximately 1:8:3. The proportion might not be exactly these, because we would have to get exactly 1 in 12 times case 1, 8 in 12 times case 2, and exactly 3 in 12 times case 3. We know that despite these are the right proportions, we would probably never get exactly these proportions. We see that the proportions of the high of each column in the histogram in Fig 6.9 show that too. This time again there is no sense to analyze range or standard deviation because we only have three columns.

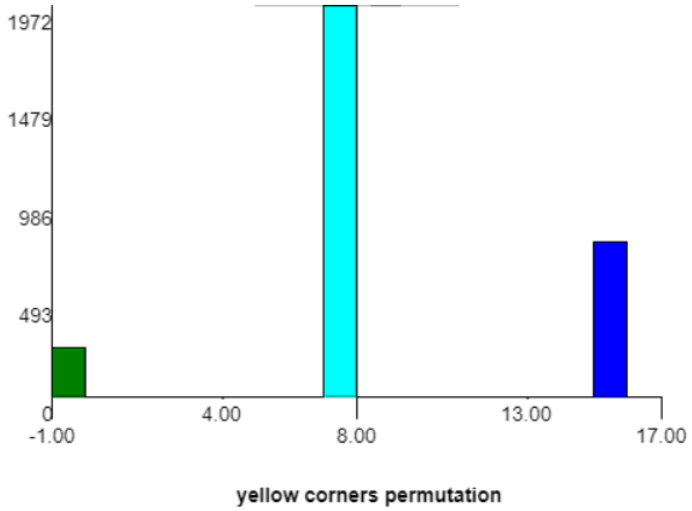

Fig 6.9. Histogram of frequency of the number of moves during the solve process by LBL method step 6, yellow corners permutation
Source: author's elaboration

The last step (Figure 6.10) is very analogical to step 4 (Fig 6.8) because we only have three groups of columns. Again this comes from the fact that we need to execute a given algorithm 0,1 or two times. The length of the algorithm is 24, but it also requires some 'set-up' moves, and the number of them can be equal to 2, 3, or 4 unless we do not need to execute the algorithm, then, of course, we do not need to do 'set-up' moves so then it could be equal to 0. So again we can easily isolate the formula:

$$24 \cdot A + S \tag{6.3}$$

where: *A* means the number of times an algorithm needs to be executed (1 or 2); *S* means the number of set-up moves (2, 3, or 4)

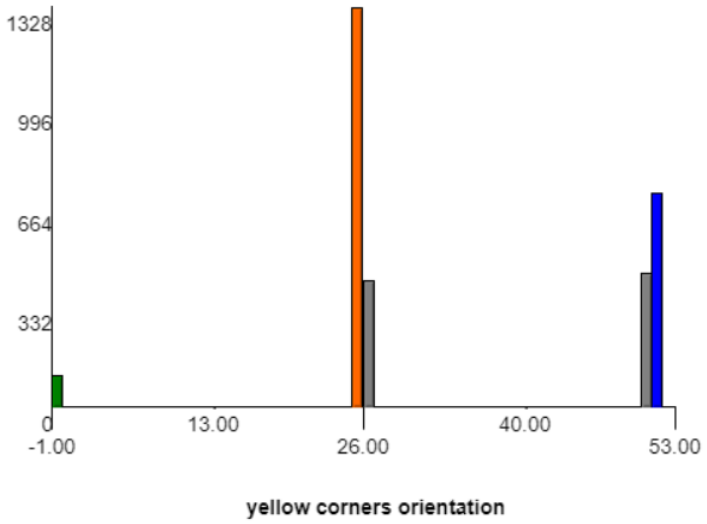

Fig 6.10. Histogram of frequency of the number of moves during the solving process by LBL method step 7, yellow corners orientation
Source: author's elaboration

So now, looking at the formula 6.3 we see that we can not obtain values from the gaps, these are 1 to 25 and 28 to 50. There are in fact 27 different situations that can occur at this step. We can easily make a probability table for each number of moves.

Table 6.3 Probability of each number of moves in the seventh (last) step of LBL

Number of moves	Probability of occurrence
0	$\frac{1}{27}$
26	$\frac{12}{27}$
27	$\frac{4}{27}$
51	$\frac{4}{27}$
52	$\frac{6}{27}$

Source: author's elaboration

Looking at Table 6.3 we see that the probability of occurrence of each number moves agrees with the histogram in Figure 6.10. And once again there is no sense in analyzing the range or standard deviation because we only see 5 columns here. Also looking at Figure 6.2 we see

that this step also takes a lot of time because on average it takes 28.1% of the whole solve process, which is almost that much as step 2, the second layer.

6.1.2. CFOP

Next, we will focus on the second method, which is CFOP. Firstly we will generate basic statistics to have the overall image of the numbers that are results of this method:

Table 6.4 Basic statistics of the number of moves of the CFOP method, not optimized

	Mean	Median	Standard deviation	Minimum	Maximum
CFOP	63.598	64	5.755	39	85

Source: author's elaboration

This time we see the mean is almost the same as the median too. That tells us that presumably, the histogram will be quite symmetric. We can easily check that.

We see that the shape of the histogram in Fig 6.11 is again close to the normal distribution, so it is similar to the previous method (Fig 6.1), but of course is different in the case of numbers, because numbers are much lower than before. Overall based on the histogram in Fig 6.11 and Table 6.4 we see that the results are more condense, the range is rather low. We also see that the standard deviation is at a low level.

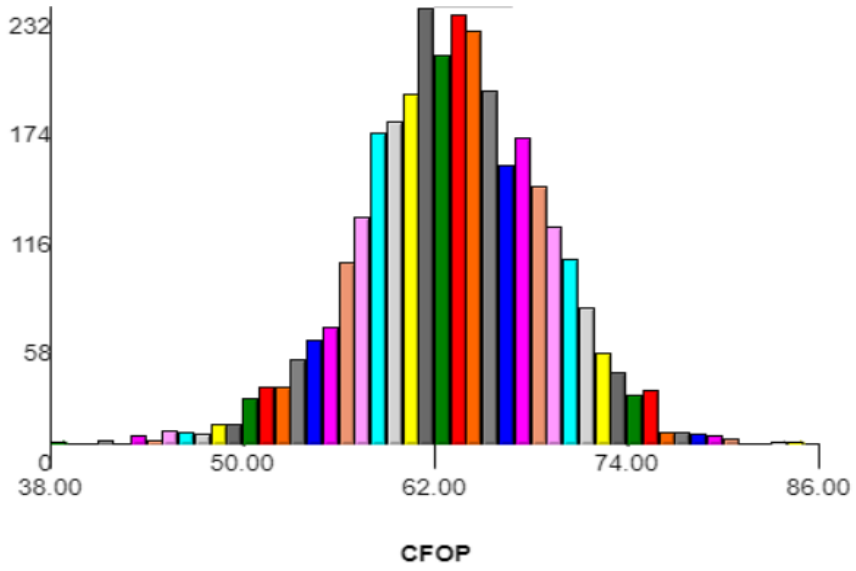

Fig 6.11. Histogram of frequency of the number of moves during the solving process by CFOP method
Source: author's elaboration

The next thing is partial results, which are the results of each step. Looking at those statistics we will be able to find out information like, which step takes the highest number of moves to execute. Here we have the statistics for each step (Table 6.5):

Table 6.5 Basic statistics of each step of the CFOP method, not optimized

	Mean	Median	Standard deviation	Minimum	Maximum
Cross	10.442	11	2.272	2	16
F2L	30.016	30	3.767	16	48
OLL	9.513	10	2.224	0	15
PLL	13.626	14	2.997	0	19

Source: author's elaboration

We can compare the share of each step in the whole solve process. In Figure 6.12 we see that the second step, that is F2L takes 47.2% of the whole solve process, it is a lot, almost half of that. This is the part that takes the biggest part of time also for humans, there is always the most place for improvement in this step. Also, we see that cross and OLL on an average similar amount of moves to be completed and PLL takes a little bit more moves on average.

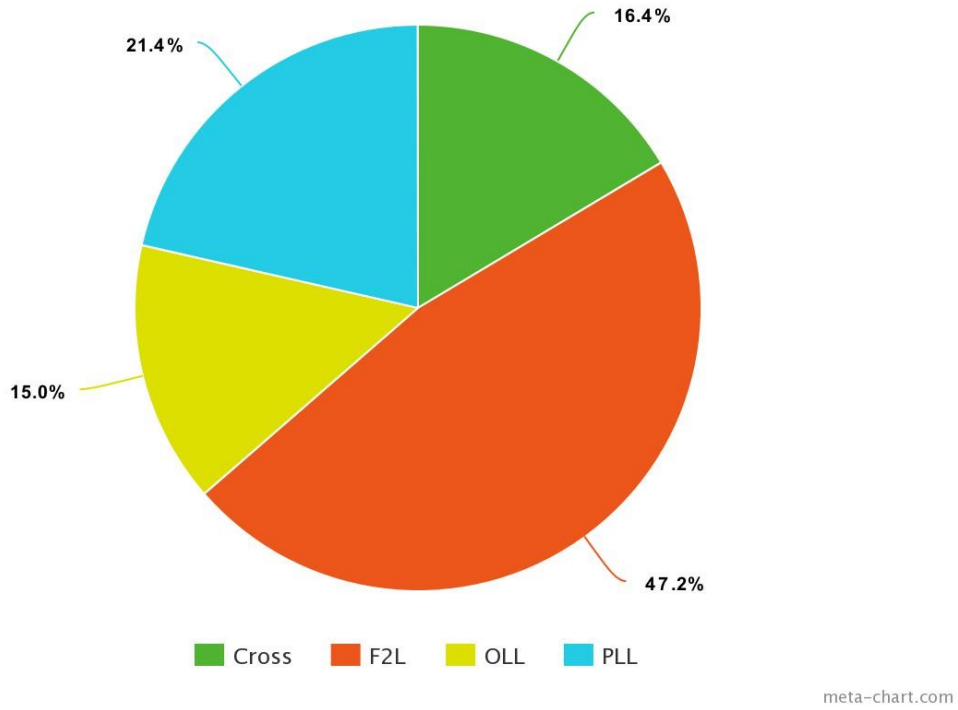

Fig 6.12. Pie chart of share of each step of the whole solve process, CFOP method, not optimized
Source: author's elaboration using https://www.meta-chart.com/

Now we can look for detailed results from each step, we can skip cross because this step is identical to LBL first step, so every information would be identical as before except the share in the whole solve process.

As I mentioned before (section 3.2.2.) this step is analogous to step 2 and 3 from LBL combined, so it is for sure more efficient than in the case of LBL. In Figure 6.13 we see that again the shape of the histogram is close to normal distribution. There are some outliers, especially on the right side of the histogram and that tells us that there are some situations that are very unlucky and cause a high number of moves to finish that step. On the left side, we do not see that many

outliers and they are not as far from the middle part of the histogram as in the case of the right side. But those are anyway singular observations and we do not have to worry about them. Looking at the histogram and in Table 6.5 we see that most of the simulations are between 24 and 32 and the mean is equal to 26.652. This is also a step which is considered the hardest to solve for humans efficiently and for now, our results are not extraordinary, but of course, this is not an optimized version. We will be able to compare our results to humans or different programs when it will be optimized.

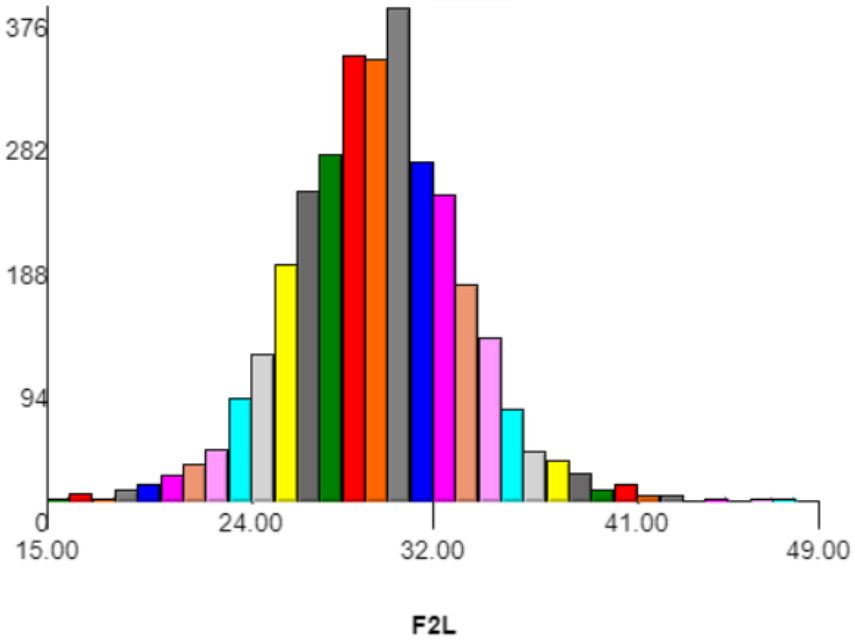

Fig 6.13. Histogram of frequency of the number of moves during the solving process by CFOP method, step 2, F2L
Source: author's elaboration

In Figure 6.14 we see a histogram of step 3 of CFOP solve that is OLL (orientation of the last layer). We see that this time the shape differs from the shape of normal distribution. There is a simple reason for that. In this step, there might occur several different situations and for each of them, there is an algorithm, which needs to be executed in order to finish this step. Each case has a given probability of appearance and consists of some number of moves. As we see in Figure 6.14 all numbers do not appear because there are no algorithms for this part that take for example 1 to 5 moves or 13 moves. There are 58 different cases, but that includes the case when this part is 'luckily' solved and we can skip this step. So overall there are 57 different algorithms in this step.

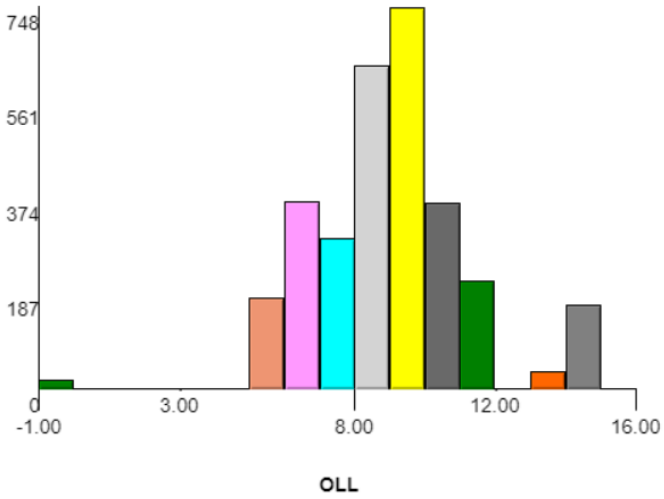

Fig 6.14. Histogram of frequency of the number of moves during the solving process by LBL method, step 3 OLL
Source: author's elaboration

Table 6.6. contains the probability of how often given numbers of moves are required to finish this step:

Table 6.6 Probability of occurrence the situation which requires a given number of moves in OLL

Number of moves	Probability
0	$\frac{1}{216}$
6	$\frac{12}{216}$
7	$\frac{28}{216}$
8	$\frac{20}{216}$
9	$\frac{44}{216}$
10	$\frac{54}{216}$
11	$\frac{27}{216}$
12	$\frac{16}{216}$
14	$\frac{2}{216}$
15	$\frac{12}{216}$

Source: author's elaboration

Looking at table 6.6 we see the probability of each situation, sums up to 1. Knowing that we can calculate the expected value of the number of moves in this step. We can calculate this by using a simple formula:

$$E[X] = \Sigma\, x \cdot p(x) \tag{6.4}$$

Where: *x* is the number of moves, *p(x)* is a probability of a given situation

We can substitute for value and we achieve the expected value equal to 9.541. When we look at table 6.5 we see that the actual value obtained during simulations, that is mean of the results is equal to 9.486. We see that these values are almost the same, they differ by 0.055 move, which is a really low value. But we could get even more precise to the expected value. We would have to simply increase the number of simulations. The more observations the closer the actual mean would get to the expected value. I think that anyway for the sample of size 3000 the difference which is equal to 0.055 moves is a really good result.

In Figure 6.15 we see a histogram of step 4 of CFOP solve that is PLL (permutation of the last layer).

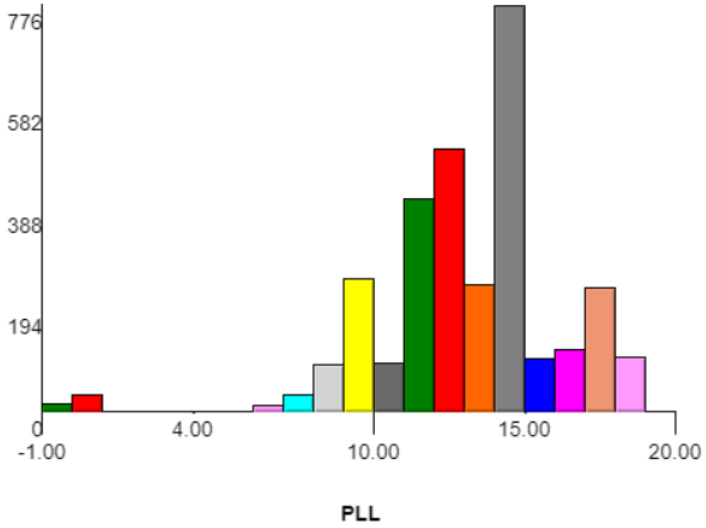

Fig 6.15. Histogram of frequency of the number of moves during the solving process by LBL method, step 4 PLL
Source: author's elaboration

Again we see that the shape is irregular, but again there is a simple reason for that. Similar to the previous step (OLL) in this step we have a number of situations that can occur and for each of them there is an algorithm that we have to execute in order to finish solving the Rubik's cube. But this time there is one additional thing. Sometimes we have to do one additional move that is not a part of the algorithm, but we need to do this because only executing the algorithm can be not enough. I will present this in Figure 6.16:

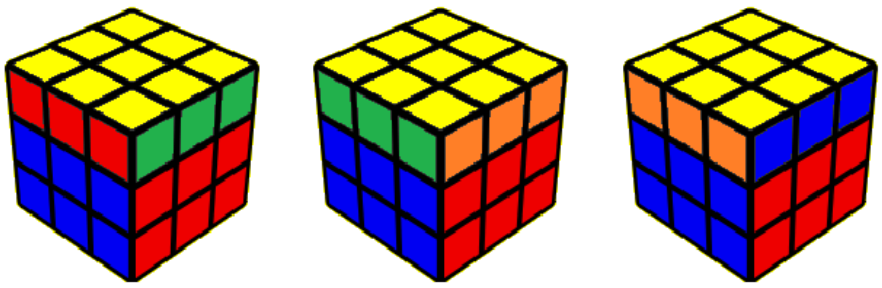

Fig 6.16. .Almost solved Rubik's cubes Source: https://pl.wikipedia.org/wiki/Metoda_Fridrich (edited)

In Figure 6.16 we see that these Rubik's cubes are clearly almost solved and we do not need to know any algorithms in order to solve them, but of course, that does not mean that we do not have to count those moves while solving Rubik's cube. This is the one additional move that we often have to do. It is also not hard to calculate that there is 1 in 4 possibilities of not making that move and 3 in 4 possibilities that we will have to make that move (all presented in Figure 6.16) . Why? Because there are simply four ways a solved last layer can be rotated. Besides that additional move, we can again make a table of the probability of how often given numbers of moves are required to finish this step (analogous to Table 6.6).

Table 6.7 Probability of occurrence the situation which requires the given number of moves in PLL algorithm

Number of moves	Probability
0	$\frac{1}{72}$
7	$\frac{1}{72}$
9	$\frac{8}{72}$
11	$\frac{8}{72}$
12	$\frac{16}{72}$
14	$\frac{24}{72}$
15	$\frac{2}{72}$
16	$\frac{2}{72}$
17	$\frac{6}{72}$
18	$\frac{4}{72}$

Source: author's elaboration

This time we have to remember that in this table we exclude the optional additional move, which I talked about previously. We will get back to that. Of course, the probability once again sums up to 1. Now once again we can calculate the expected value of the number of moves in this step. We can calculate this by using formula 6.4 and the value we achieve is equal to 12.93. But this is not the final result, because now we have to take into consideration the additional move that we sometimes have to make. As we remember there was only one move, and the probability of making that move was ¾, so again we can use formula 6.4 to calculate how many additional moves we expect and the result here is not hard to calculate, it is simply 0.75 because $1 \cdot 0.75 = 0.75$. Knowing that we can add both expected values to each other and we get $12.93 + 0.75 = 13.68$. We need to go back to table 6.5 to look at what are our actual results and we see that it is equal to 13.613. The difference between the expected value and the actual result is equal to 0,067. Again it is very low, so our predictions were correct, but of course, we could probably get

even closer to the expected value if we increase the sample size, but as I mentioned before in chapter 4, generating more than 3000 results would take a lot of time, because of my limited computing power.

6.1.3. LBL vs CFOP

Knowing the basic statistics of each method and where that comes from, after analyzing each method we can compare two methods against each other. Firstly we can cumulate the basic statistics of each method in one place to compare them easier (Table 6.8):

Table 6.8 Statistics of comparison of LBL and CFOP, not optimized

LBL	Mean	Min	Max	CFOP	Mean	Min	Max
Whole solve	125.3	55	176	Whole solve	63.598	39	85
White corners and the second layer	55.163	20	82	F2L	30.016	16	48
Yellow cross and corners orientation	41.074	0	64	OLL	9.513	0	15
Yellow cross and corners permutation	18.220	0	33	PLL	13.626	0	19

Source: author's elaboration

When comparing the results we need to remember what steps are equivalent to each other in both methods. As it is described in chapter 3.2. steps 1 are the same in both methods, so there is no sense in comparing them, steps 2 and 3 from LBL (white corners and second layer) are equivalent to step 2 from CFOP (F2L), steps 4 and 7 from LBL (yellow cross orientation and yellow corners orientation) is equivalent to step 3 from CFOP (OLL) and steps 5 and 6 from LBL (yellow cross permutation and yellow corners permutation) are equivalent to the step 4 from CFOP (PLL). Knowing that we will compare them in that way and it is shown in that way in Table 6.8.

Firstly we will focus on the whole solve process. Looking at Table 6.8 we see that the average results differ by almost 62 moves. This is a lot considering the mean in the case of CFOP is 63.598. The mean in the case of LBL is almost twice as big as in the case of CFOP. Also, we see that the minimum in the case of LBL is only about 9 moves lower than the mean of CFOP. The worst-case in CFOP (that is 85) is still 40 moves below the average LBL result. Whereas the worst case for LBL (that is 176) is almost three times the average of CFOP (63.598) and over 4.5 times as much as the minimum of CFOP. We also see that both minimums (55 and 39) are not that far apart, but these are only singular observations so we shouldn't pay too much attention to them. Knowing all those differences we can go deeper and try to find in which steps the biggest difference occurs.

Next, we can compare the second and third steps from LBL and the second step from CFOP. In table 6.8 we can observe that in fact there is quite a big difference, but it is not huge. The mean in the case of LBL (55.163) is almost two times bigger than in the case of CFOP. (30.016). A similar relation is between maximums because in the case of LBL it is equal to 82 and

in the case of CFOP, it is 48. Whereas in the case of the minimum we see that they are very close to each other, 20 in the case of LBL and 16 in the case of CFOP, but as I said before these are only singular observations and we do not have to worry about them. An interesting thing to notice once again is that the maximum in the case of CFOP (48) is lower than the average in the case of LBL (55.163).

The next steps we can compare are steps 4 and 7 from LBL and step 3 from CFOP. Here looking in table 6.8 the first thing we notice is that there is a huge difference in the means of each method. The mean in the case of LBL (41.074) is over 4 times bigger than the mean in the case of CFOP (9.513). That is a really huge difference, and now we see where the overall difference comes from. On this step, LBL is on average 31.5 moves slower than CFOP or in other words, LBL needs over 431% moves that CFOP needs. A huge difference is also visible in maximums and the relation is almost identical. LBL method needs 64 moves and CFOP 15 moves. We can again say that in the worst-case LBL needs about 427% moves that CFOP needs. Unlike mean and maximums the minimums are the same, they are equal to 0, but as I explained in section 6.1.1. and 6.1.2. In some steps, there is a possibility to 'skip' a given step if we are 'lucky' and this is what happened in both cases. But it does not mean that we can forget other disparities and based on that and only that observations say that these methods in these steps are equal. This is why I take numerous statistics into consideration and not only one or two.

The last steps that we will compare are steps 5 and 6 from LBL and step 4 from LBL. Again we will base our conclusions on table 6.8. The first thing we can look at are means. We see that as previously CFOP mean (13.626) is lower than LBL (18.220), but slightly. They differ by 4.6 moves. Compared to previous differences, here both methods are closer to each other. Maximums differ a little bit more because in the case of LBL (33) it is almost two times higher than in the case of CFOP (19), but those are probably singular, not frequent observations, so we do not have to worry about them. Identically as in the previous comparison, the minimums are both equal to 0, but this is not something we should focus our attention on.

To sum up, we see that CFOP is much better than LBL and this will never change, but there is a price to pay for that. In the case of CFOP, as I explained in chapter 6.1.2. there are 119 algorithms that are needed to memorize, whereas in the case of LBL it is only about 7 to 10 algorithms. Of course in my simulations, this is not a problem, because as I said before computers do not have limitations about the number of algorithms to remember, but if we would consider such a situation for humans it would not be that simple because this definitely can be a problem That is why the CFOP method for computers is much better, but for humans not necessarily. Of course, this is the case now, when we have not optimized the methods yet, because as it was stated in Table 6.1 in the LBL method there is more room for improvement than in the CFOP method. So for now we can not without fail to say that LBL is worse than CFOP, we will have to do optimization and further analysis to find out.

6.2. Analysis of the efficiency of semi-optimized solving Rubik's cube methods

Now we can focus on optimization, firstly semi-optimized strategy. Important thing is that we will not analyze steps that can not be optimized, because we will just repeat what was already said in chapter 6.1. Of course, we will also talk about the whole solve process, because that is the essence of my work.

6.2.1. LBL

Similarly as in chapter 6.1 firstly we will focus on the LBL method. As it was previously stated in chapter 6.1 in the LBL method the steps that could be optimized were: cross, white corners, and the second layer (first, second, and third steps). But first, we will focus on the whole solve process to get an overall image of the level of improvement of our optimization.

Table 6.9 Basic statistics of the number of moves of the LBL method, not optimized and semi optimized

LBL	Mean	Median	Standard deviation	Minimum	Maximum
not optimized	125.3	124	17.7	55	176
semi optimized	115.7	115	17.2	51	161

Source: author's elaboration

We see in table 6.9 that definitely there is an improvement, but in general, we would not call that medium improvement, because in the case of means that is only about 8% improvement, from 125 to 115. Almost the same drop is in the case of the median. Of course, these are still good results, they are easily not amazing ones. Minimums are very similar and maximum dropped a little, but comparing it to the size of maximum, that is also not that big a drop. The standard deviation drops about 3%, so it is again not a big change, but this tells us that the results are more condense around the mean, so we can say that they are a little bit more predictable.

In Fig 6.17 we see a histogram of the frequency of the number of moves during the whole solve process. We see that the shape is also close to the normal distribution and also is similar to the histogram in Fig 6.1 where the results of LBL are not optimized. The main difference between those two histograms is that the one in Fig 6.17 is shifted more to the left

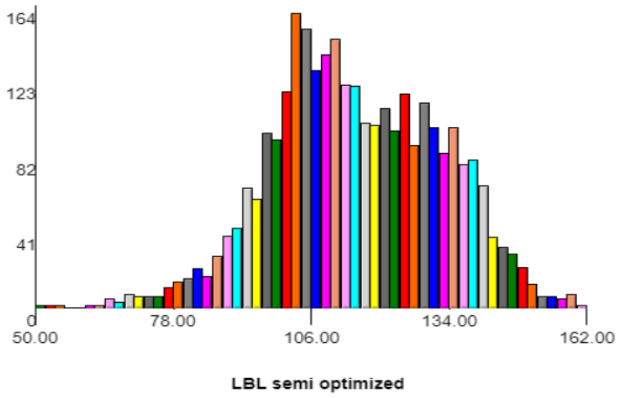

Fig 6.17. Histogram of frequency of the number of moves during the solve process by LBL method, semi optimized
Source: author's elaboration

Next thing we can try to find out is the percentage share of each step in the whole solve process. We see in Fig 6.8 that the share of steps that was not optimized has increased, which was of course easy to predict, because results we achieved in those steps are at the same level as before, but only when expressed in a number of moves. When expressing share in percentages the numbers increased, because the number of moves in whole solves decrease whereas in each of those steps stay at the same level. In contrast to that, we see that the percentage share of steps that were optimized decreased. Of course that was also expected, because that was the aim of our optimization.

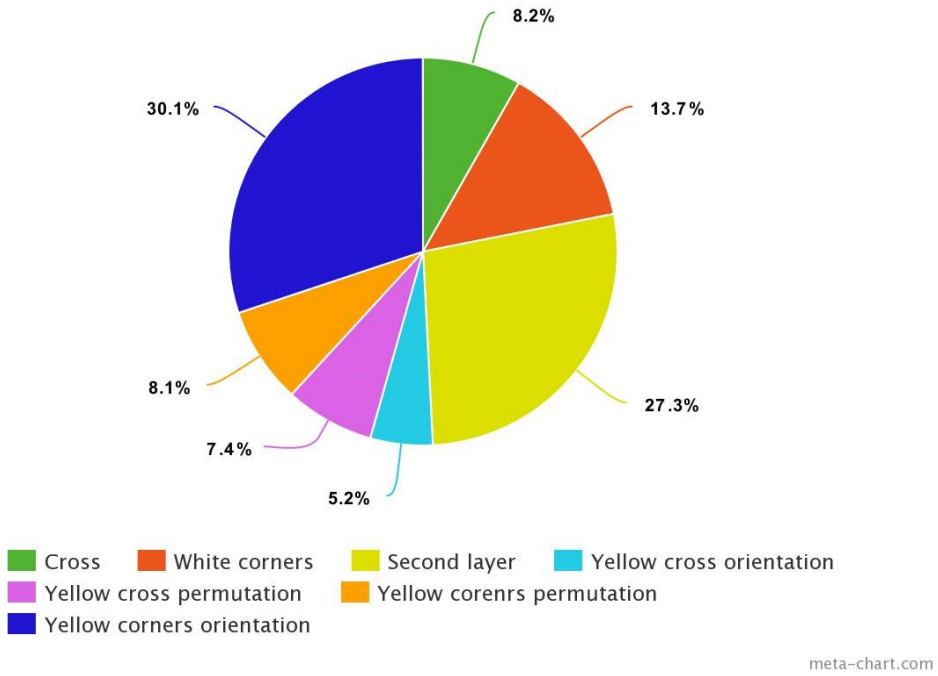

Fig 6.18. Pie chart of share of each step of the whole solve process, LBL method, semi optimized
Source: author's elaboration using https://www.meta-chart.com/

Next, we can focus on progress in each step. In Table 6.10 we see that in the first step, that is cross there is a slight improvement in the semi-optimized version, because the result drops from 10.442 to 9.503, so it is about almost 1 move better than the not optimized version. It is not much, but we have to remember this is a rather short step and that is a 9% decrease and I think that when we express the drop in percentages it starts to look much better. We rather could not expect the results to drop by about 40% or 50%. This is a reasonable decrease. We see that minimum stay on the same level, and this is rather expected, because it was impossible to solve a situation which was firstly solved in two moves, in one move. Maximum drops 1 move, but that is not valid information, because this is probably just a singular observation and that was a very unlucky situation, that even semi optimization did not quite help, because it is only one move lower than not optimized.

Table 6.10 Basic statistics of the number of moves of each step of the LBL method, not optimized and semi optimized

LBL	Not optimized			Semi optimized		
Step	Mean	Min	Max	Mean	Min	Max
Cross	10.442	2	16	9.503	2	15
White corners	18.476	6	28	15.840	6	27
Second layer	36.688	7	60	31.578	8	53

Source: author's elaboration

In the next step, which are white corners we see in table 6.10 that the decrease between not optimized and optimized versions in case of mean is from 18.476 to 15.840, that is 2.636 moves less. We see that this is a bigger decrease, not only looking at the number of moves but also looking at percentages because it is decreased by 14,3%. It is almost ⅙ of whole moves, so this is a pretty big decrease. In the case of minimums and maximums, we see that the relations are almost the same as in the case of the cross because minimums are the same in both cases, and the maximum in the semi optimized version is only one move smaller than in the not optimized version.

The last step that we can compare is the second layer. Again we see quite a significant decrease in means, because from 36.688 (in not optimized version) to 31.578 (in semi optimized version), so that is 5.11 moves less. Again we can check the percentage decrease and we see that the decrease is equal to almost 13.93%, so we see that this is a very similar percentage as in the previous step. We see that maximums are a little bit further from each other than they were in previous steps, but an interesting thing we can observe in minimums. We see that the minimum in the not optimized version is lower than in the semi optimized version. What does it mean? Our optimization does not work? No, it works and it works as I predicted, but singular observations like this can happen. I explained this in chapter 4.1. There is always a small possibility that if we do the first element in an efficient way (that is we choose to as the first one, the one which takes the fewest moves to finish), the next ones will be positioned in unlucky situations. Whereas it can also happen that, despite doing the first one in an inefficient way, the next ones will position in a lucky situation. This is exactly what happened here. Anyway, we do not have to worry about that, because these are only singular observations and as we saw in the mean statistics, our work to optimize the method has given good results.

To sum up we see that the semi optimize method gives us a decrease by 0.919 moves, 2.636 moves, and 5.11 moves, which gives us an average of 2.88 moves decrease per step. But better focus on percentage decrease and here we achieved decreases at the level of 9%,14,3%, and 13.93% and that gives as an average of 12.41% decrease per step. This is good because it is almost ⅛ moves less than in the not optimized version. Of course regarding the whole solve process, it is not that high, but we have to remember that 4 other steps are not possible to optimize, so in the case of semi-optimization defined by me, this is the best we can do.

6.2.2. CFOP

Now we will analyze the results of the semi optimized strategy of the CFOP method. Similarly, as in section 6.2.1, we will start from the table of basic statistics of this version and not optimize one, to see differences. We see that in Table 6.11 the overall decrease in the semi optimized version is visible, but is not very high. Firstly we can focus on means. We see that it dropped from 63.598 to 59.254 which is 4.344 moves less. It does not look to be a great improvement, but we can express it in percentages and it is a 7% decrease. It is not as much as in the LBL method but we have to remember that in this method there was less room for improvement. The Median dropped by 5 moves, the standard deviation is almost at the same level. In the case of minimums and maximums, we see 6 and 7 moves decrease, which is a little bit more than in the case of means or medians, but these are singular observations, so they are not that significant for analysis. Nonetheless, 33 moves is a very good result and it is very hard or almost impossible to achieve for humans in a reasonable time. Of course there for sure was a lot of luck included but anyway it is worth noticing such a great result.

Table 6.11 Basic statistics of the number of moves of the CFOP method, not optimized and semi optimized

CFOP	Mean	Median	Standard deviation	Minimum	Maximum
not optimized	63.598	64	5.755	39	85
semi optimized	59.254	59	5.468	33	78

Source: author's elaboration

Next, we can focus on each step individually, but we have to remember that in the case of CFOP, as I said previously in Table 4.1 we can only optimize two steps of that method, these are: cross and F2L. In section 6.2.1. we already analyze the cross and as I mentioned before this step is exactly the same in both methods, so I will not repeat the analysis of that step and now focus only on F2L. In Table 6.12 we see a comparison of F2L from not optimized and semi optimized versions of CFOP. Firstly we see that there has been a decrease by 3.364 moves between not optimized and semi-optimized methods. It is an 11.2% decrease so it is a pretty good improvement. We see that minimums differ only by 1 move, but in maximums, we see that there has occurred a big decrease, because from 48 to 38 moves. Of course, we could say that these were only singular observations, but in the not optimized version for sure there are observations between 38 and 48, especially lower ones, like 39, 40, or 41 and we see that optimization prevents us from achieving so high numbers. That means that the results are probably more condense around the mean.

Table 6.12 Basic statistics of the number of moves of F2L step of the CFOP method, not optimized and semi optimized

CFOP	Not optimized			Semi optimized		
Step	Mean	Min	Max	Mean	Min	Max
F2L	30.016	16	48	26.652	15	38

Source: author's elaboration

We can also try to generate once again the pie chart of the share of each step in the whole solve process. In Figure 6.19 we see that the share of F2L step decreased from 47.2% (Fig 6.12) to 45%. Also, we see that now cross and OLL take the same share of the whole solve process. Also knowing that OLL and PLL were not optimized we see that they share in the whole solve process increased.

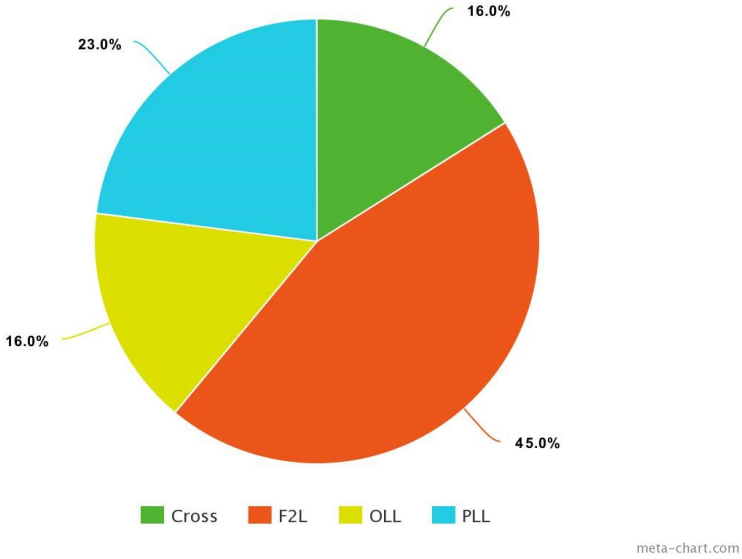

Fig 6.19. Pie chart of share of each step of the whole solve process, CFOP method, not optimized
Source: author's elaboration using https://www.meta-chart.com/

6.2.3. LBL vs CFOP

In this step of the analysis, we compare the results of semi optimized strategies for both methods. This time we will only compare the steps which we have optimized because outside of them nothing has changed. Of course, we will also compare the whole solve, because that also changed. In Table 6.13 I presented the results of semi optimization of both methods. We see that the relation between those methods are very similar as before optimization. Before (Table 6.8) the mean of LBL was almost twice as big as the mean of CFOP and we see that now it is very similar. In terms of the number of moves, the decrease in LBL is of course bigger, almost twice bigger, but when we look at the percentage decrease it is a slightly bigger decrease because the difference is only 0.83 pp.

Table 6.13 Statistics of comparison of LBL and CFOP, semi optimized

LBL	Mean	Decrease [moves]	Decrease [%]	CFOP	Mean	Decrease [moves]	Decrease [%]
Whole solve	115.702	9.6	7.66%	Whole solve	59.254	4.345	6.83%
White corners and the second layer	47.418	7.745	14.04%	F2L	26.652	3.364	11.21%

Source: author's elaboration

When comparing steps 2 and 3 from LBL and step 2 from CFOP we see that the decrease expressed in the number of moves is also bigger in the case of LBL and it is actually over two times bigger. When we look at percentage change we see that the difference between both methods is 2.83 pp. That tells us that in LBL we were able to make a little bit bigger progress. It is mainly caused by the fact that in LBL those were two separate steps and we have more room to improve because we could optimize each of them. In the case of CFOP, there was only one step so it was harder to optimize as well as in LBL.

Anyway, we have to notice that better results of optimization in LBL are only slightly better because the difference of improvements at level 0.83pp and 2.83pp are both rather low and we can say that they both are similar. Also, it would be irrational to expect totally different results in each of them when the approach was the same.

6.3. Analysis of the efficiency of fully optimized solving Rubik's cube methods

This section can focus on the fully optimized strategy of our methods. Firstly we will analyze both methods separately and show the improvement regarding the not optimized versions. Then we can compare the results we achieved in both methods and see whether in any of them they were more successful.

6.3.1. LBL

First, we start by analyzing the LBL method. Firstly we can look at the basic statistics in Table 6.14 of that method and compare them with the not optimized version. We see that the mean in has dropped from 125.3 to 112.85 and that gives us 12.45 moves less, or 10% less. I think this is quite a big drop. We see that in the case of medians the numbers are almost the same as means. Standard deviation dropped about 1 move, so that means that results are more condense around the mean. Minimum dropped 5 moves, but that does not give us much significant information, because these are probably singular observations. Whereas the maximum dropped 14 moves, that tells us that we do not have any results in the range 163-176, so results are more consistently smaller.

Table 6.14 Basic statistics of the number of moves of the LBL method, not optimized and fully optimized

LBL	Mean	Median	Standard deviation	Minimum	Maximum
not optimized	125.3	124	17.7	55	176
fully optimized	112.85	112	16.827	50	162

Source: author's elaboration

Now we can find out a level of optimization in each step. We need to remember that we only focus on those steps where optimization is possible because in the rest we would achieve the same results. Firstly we look to the values of means in table 6.15. We see that the average number of moves dropped from 10.442 to 8.473 and that gives us 1.969 moves less or in other words 18.86% decrease. That is a lot, we decreased the number of moves almost by ⅕. Again we see that the minimum stayed at the same level, this is not a surprise. Maximum dropped a

little bit, but this just tells us that in some unlucky situation even optimization can't do much. In the second step, that is white corners we see that the mean dropped from 18.476 to 14.236, and that gives as 4.24 moves decrease and this is a 22.95% decrease. Again this is a lot, which tells us that on average we decreased the number of moves by some number between ⅕ and ¼. Again, minimums and maximums do not tell us any new important information. In the third step, which is the second layer, we see that the mean dropped from 36.688 to 31.078, and that gives us 5.61 moves or 15.29% less. This is also quite a high decrease, but it is not that high as before. In maximums we see that there is an 8 moves difference, so that tells us that improvement excluded situations where there was needed from 53 to 60 moves, so we can say that the results in this step are probably more condense around mean. In minimums, we observe an identical situation as in the optimized version.

Table 6.15 Basic statistics of the number of moves of each step of the LBL method, not optimized and fully optimized

LBL	Not optimized			Fully optimized		
Step	Mean	Min	Max	Mean	Min	Max
Cross	10.442	2	16	8.473	2	14
White corners	18.476	6	28	14.236	6	26
Second layer	36.688	7	60	31.078	8	52

Source: author's elaboration

We can also find out the share of each step in the whole solve process and compare it to the primary share. In Fig 6.20 we see the share of each step primary and after optimization. We see that in all steps which were possible to optimize, the share has decreased respectively cross: 0.9pp; white corners: 2.2pp; second layer 1.9pp. Cumulative they stand for 52.6% and after fully optimizing them stand only for 47.6%, so that is a 5pp decrease.

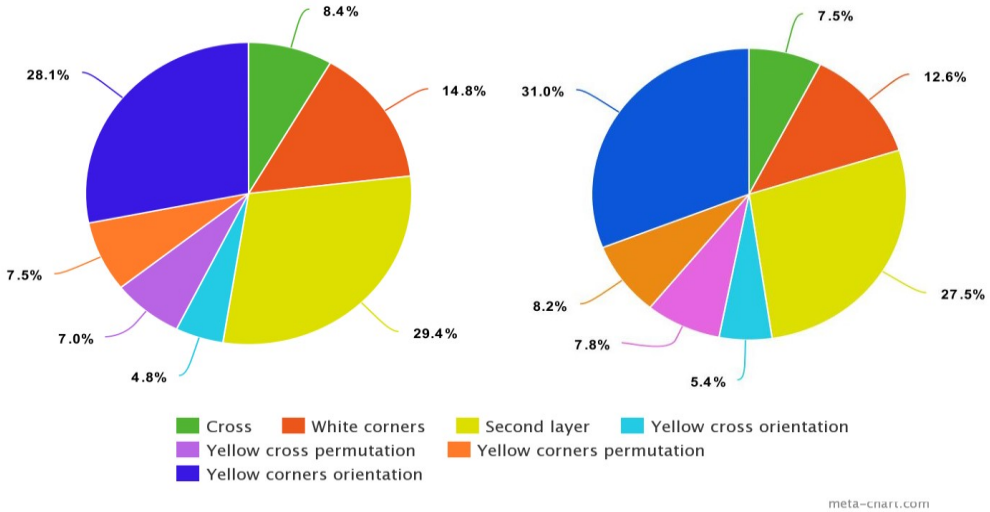

Fig 6.20. Pie chart of share of each step of the whole solve process, LBL method, not optimized and fully optimized
Source: author's elaboration using https://www.meta-chart.com/

6.3.2. CFOP

Next we want to analyze the second method, that is CFOP and its fully optimized version. Firstly we will compare it to not optimized version and interpret the results. In table 6.16 we see that the mean dropped from 63.598 to 55.265 and that gives us 8.333 moves less or in other words 13.1% decrease. That is a substantial decrease, because it is over ⅛ of the total number of moves. Ine medians we see a similar derase, it is a little bit bigger because it is equal to 9 moves. We also see that standard deviation dropped by 0.883 moves so that tells us that the results are more condensed around the mean. In minimums we see that it drops by 6 moves and the final best result is equal to 33 moves, which is an amazing result. In maximum we see 14 moves difference, which is also high and quite important, because that tells us that there are no results between 72 and 85 and that is quite a big range of results that we achieved to exclude.

Table 6.16 Basic statistics of the number of moves of the CFOP method, not optimized and fully optimized

CFOP	Mean	Median	Standard deviation	Minimum	Maximum
not optimized	63.598	64	5.755	39	85
fully optimized	55.265	55	4.872	33	71

Source: author's elaboration

Next we can look for each step statistics, but again we have to remember that in LBL the first step is also cross and we talk about results of fully optimization that step in chapter 6.3.2. So now we will focus only on the second step, that is F2L. We see in table 6.17 that the mean has dropped from 30.016 to 23.716 and that gives us 6.3 moves less or 20.99% decrease. This is an amazing result, we have managed to decrease average number of moves by ⅕. We see that the minimum has dropped only by 2 moves, but we have to remember that during the optimization process we were focusing on choosing the best order of placing elements in correct places and it is always possible that the order set by default in not optimized version is the best order. That is why such results will always appear from time to time, but of course they are not too common. In case of maximums we see that the number of moves in the worst case dropped from 48 to 34 and that is also significant information, because now we are not getting any results from range 35 to 48, and it is quite a big range.

Table 6.17 Basic statistics of the number of moves of F2L step of the CFOP method, not optimized and fully optimized

CFOP	Not optimized			Fully optimized		
Step	Mean	Min	Max	Mean	Min	Max
F2L	30.016	16	48	23.716	14	34

Source: author's elaboration

Next thing we can show is the percentage share of each of steps in the whole solve process. As we could expect in Fig 6.21 we see that the share of steps which were not optimized has increased and the share of steps that we optimized has decreased. In the case of the first step, which is cross, the share has dropped by 1.1pp and in the second step that is F2L, the share

has dropped by 4.3pp. In steps 3 and 4, these are OLL and PLL the share has of course increased slightly, despite that the results are the same. We can also generalize that the overall share of the part which we optimized has dropped by 5.4pp.

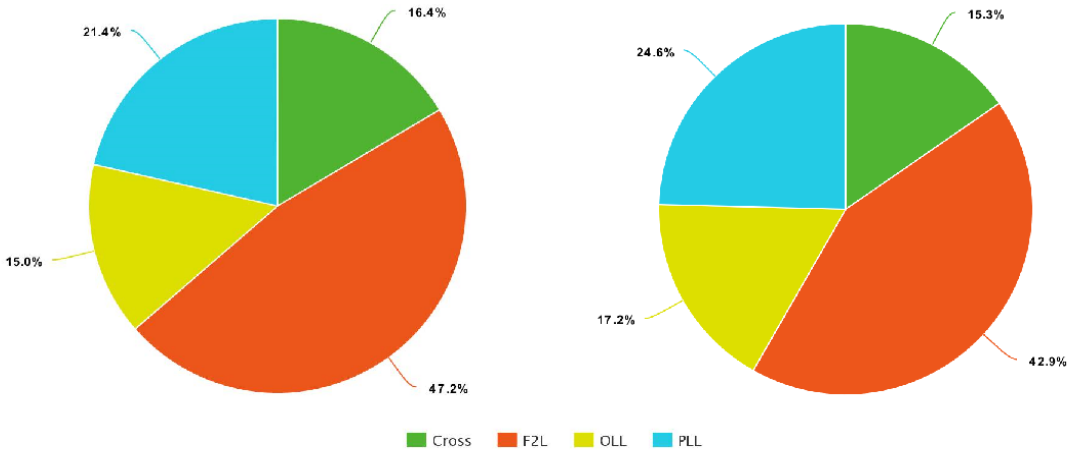

Fig 6.21. Pie chart of share of each step of the whole solve process, CFOP method, not optimized and fully optimized
Source: author's elaboration using https://www.meta-chart.com/

6.3.3. LBL vs CFOP

Next we want to compare results of full optimization in LBL and in CFOP to see where it was more important to optimize and where it gave better results. We will focus on the whole solve and on each step, but of course we will exclude the first step from both methods (cross) cause it is the same, so there is no sense in comparing the same thing. Also we have to remember that we are comparing steps 2 and 3 from LBL (white corners and second layer) against step 2 from CFOP (F2L), for the reasons I explained in chapter 2.2. In table 6.18 we see that the mean of CFOP is over two times smaller than the mean of LBL, that is quite a lot. Next thing we can see is that decrease in moves is higher in case of LBL, whereas decrease expressed in percentages compared to primary results is higher in CFOP by 3.1%

Table 6.18 Statistics of comparison of LBL and CFOP, fully optimized

LBL	Mean	Decrease [moves]	Decrease [%]	CFOP	Mean	Decrease [moves]	Decrease [%]
Whole solve	112.851	12.45	10%	Whole solve	55.265	8.333	13.1%
White corners and the second layer	45.314	9.849	17.85%	F2L	23.716	6.3	20.99%

Source: author's elaboration

In terms of comparing steps 2 and 3 from LBL and step 2 from CFOP we see that the mean is again almost twice as high in LBL as it is in CFOP. The decrease in moves is again lower in CFOP, but the decrease expressed in percentages is higher in case of CFOP by 3.14pp. It is not a surprise that the decrease in moves is in both cases higher in LBL than in CFOP, whereas

the decrease expressed in percentages is higher in both cases in CFOP than in LBL, because in LBL there was much more room for improvement. And we can imagine a situation where some other method requires on average 1000 moves to finish the solve and then reducing the total number of moves by 50 would be a high decrease in number of moves, but when we think about percentages decrease it would be only 5%. Based on this example we see that a better way to compare those results is to look on percentage decrease. Doing it that way shows us that the improvement was more impactful in case of CFOP.

6.4. Analysis of the efficiency of semi- and fully optimized solving Rubik's cube methods

6.4.1. Semi-optimized LBL vs. Fully-optimized LBL

This section can focus on comparing the level of improvement in both versions, that is semi optimized and fully optimized. Firstly we will focus on the whole solve process. In table 6.19 we see that the means do not differ significantly. They differ only by 2.85 moves which is equivalent to 2.46%. We see that the median is also 3 moves lower in the fully optimized version, so it is a similar situation to mean. We see that in case of standard deviation and minimums results are almost the same. Interesting things happen in case of maximums. We see that the maximum in case of fully optimized version is higher than in the not optimized version. Why? How is that possible? As I explained in chapter 4.2, fully optimized version guarantee at least as good results as semi optimized, but only in each step. It does not guarantee better results overall. What is the difference? There can occur some situation (and as we see by the results it happened), that doing a given step in the most optimized way can cause an unlucky situation in the next step, which may require more moves to finish, whereas less optimized version can meet then lucky situation and require fewer moves to finish.

Table 6.19 Basic statistics of the number of moves of the LBL method, semi optimized and fully optimized

LBL	Mean	Median	Standard deviation	Minimum	Maximum
semi optimized	115.7	115	17.2	51	161
fully optimized	112.85	112	16.827	50	162

Source: author's elaboration

Next we can focus on each step individually. In table 6.20 we see that in cross (step 1) the means differ by 1.03 moves or in other words by 10.84%. Decrease at level of 10% is quite high, but it is not an amazing result. In case of minimum it is the same, but again we cannot expect to finish a step in one move, when previously it was finished in two moves. Maximum decreased by 1 move, but this is not a big difference.

Table 6.20 Basic statistics of the number of moves of each step of the LBL method, semi optimized and fully optimized

LBL	Semi optimized			Fully optimized		
Step	Mean	Min	Max	Mean	Min	Max
Cross	9.503	2	15	8.473	2	14
White corners	15.840	6	27	14.236	6	26
Second layer	31.578	8	53	31.078	8	52

Source: author's elaboration

Next step is white corners (step 2) and again we see that the difference is equal to 1.604 moves or in other words 10.13%. This is a very similar difference as in step 1. Minimums again did not change and in maximums again we see only one move decrease and that tells us that when there is an unlucky situation even fully optimization can not do much. Last step, which we optimized is the second layer (step 3), here we see a decrease by 1.5 moves, in other words 4.75% decrease. This time we see much lower decrease and that tells us that in this case the difference between semi optimized and fully optimized version is rather negligible, especially that in this case the primary number of moves was high and in theory there was a lot of room for improvement. In case of minimums, again they are the same and maximum is one move lower than before.

Overall we can say that in the case of LBL method optimization as a whole was good, and gave great results, but the difference between two approaches: semi optimized and fully optimized was insignificant. And we have to keep in mind that a fully optimized version is more complex and requires more computing power. For example if we would need to make a decision which approach to use and we were limited by computing power then we would rather decide to use a semi optimized approach because the difference in results may be not worth wasting much more computing power. But of course if we do not have such limitations then even marginal improvement is worth doing and then we would choose a fully optimized version.

6.4.2. Semi-optimized CFOP vs. Fully-optimized CFOP

Next we will compare the results of semi optimization and fully optimization in CFOP method. Knowing that we compared only the first two steps in this method we will focus only on them and on the whole solve process of course. Firstly we will look at the results of the whole solve process. In table 6.21 we see that the mean in case of fully optimized version is 3.989 moves lower than in semi optimized version, in other words it is 6.73% less. In the case of medians we have almost exactly the same difference. Standard deviation is lower by 0.596 moves in case of fully optimized version than in semi optimized. Minimums are exactly the same and maximum is 7 moves lower in case of fully optimized version, so that tells us that we managed to exclude all observations between 72 and 78. We see that the overall difference between those two approaches is really good.

Table 6.21 Basic statistics of the number of moves of the CFOP method, semi optimized and fully optimized

LBL	Mean	Median	Standard deviation	Minimum	Maximum
semi optimized	59.254	59	5.468	33	78
fully optimized	55.265	55	4.872	33	71

Source: author's elaboration

Next we will focus on each step individually. But we remember that the first step in both methods LBL and CFOP is the same, so knowing that we already analyze the results of cross in LBL in the previous chapter we will not repeat that here. So we will only focus on the second step, this is F2L. In table 6.22 we see that the mean has decreased from 26.652 in the semi optimized version to 23.716 in fully optimized version and that gives us 2.936 moves less or in other words 11.02%. That is quite a big decrease because on average we shorten the number of moves by 1/9. We see that minimums differ only by one move and maximums by 4 moves, which is also not much.

Table 6.22 Statistics of comparison of CFOP, semi optimized and fully optimized

CFOP	Semi optimized			Fully optimized		
	Mean	Min	Max	Mean	Min	Max
F2L	26.652	15	38	23.716	14	34

Source: author's elaboration

Overall we can conclude that in CFOP we achieved good results, because when we compare two approaches of optimization we see that the overall difference is not much higher than in LBL. The overall difference between two approaches in a whole solve process is 6.74% so we see quite a significant decrease. Of course a bigger difference is when we compare not optimized version and semi optimized version, but when we compare two approaches we get once again significant level of improvement. So in the case of CFOP we can say that both approaches gave good results compared to the previous version, namely that (1) from not optimized to semi optimized version we reached 7% decrease and (2) from semi optimized to fully optimized version we reached 6.74% decrease. Especially that we have to remember that in this method there was not much place for improvement, because only two steps were possible to optimize. We can say that both versions of optimizing CFOP are worth implementing in order to achieve better results.

6.4.3. Results Validation

This section aims to compare our final results of optimized versions of both LBL and CFOP methods with results achieved by other researchers in extant literature and in internet resources.

Firstly, we can focus on LBL. It was a little problematic, because there is not much information about the average number of moves that are required to solve Rubik's cube with this method. The only information I managed to find was that this number is definitely higher than 100. But I found information about the RCSG method (Rubik's Cube Solution Guide). It is very similar to LBL and it is featured on the official Rubik's Cubes website. The results that were achieved by this method in work published by Duberg and Tideström (2015) are: mean equal to 134.61. We see in table 6.23 that the results that I managed to achieve with the LBL method are definitely better than the ones achieved by Duberg and Tideström (2015). The results before optimization are slightly better, because the difference is equal to 9.31 moves or in other words 6.92% less. We see that our final results, that is a fully optimized version is significantly lower than in RCSG method. The difference is equal to 21.76 moves or in other words 16.17% less, that is almost ⅙ moves less in fully optimized version of LBL, so that is really big difference. So to sum up we can say our implementation is rather good and gave good results.

Table 6.23 Validation of results of LBL method

	LBL not optimized	LBL semi optimized	LBL fully optimized	RCSG method, Duberg and Tideström (2015)
mean	125.3	115.7	112.85	134.61

Source: author's elaboration

Next, we can focus on the CFOP method. In this time, it was much easier to find the average number of moves used to solve Rubik's cube by this method. On different sources on the internet we can find many different numbers, but in general they oscillate between 50 to 70 moves. But anyway I will compare my results with the results from the same work of Duberg and Tideström (2015) as I already used it previously for comparison. In table 6.24 we see that unfortunately this time all my results are worse than the ones achieved by Duberg and Tideström (2015), however we see that the fully optimized version differs only by 0.955 moves, which is equal to only 1.73%. In not optimized and semi optimized versions we see that we got definitely worse results, but in case of fully optimized version the difference is almost negligible, so I think it is safe to say that the results are satisfiable.

Table 6.24 Validation of results of CFOP method

	CFOP not optimized	CFOP semi optimized	CFOP fully optimized	CFOP (Duberg and Tideström, 2015)
mean	63.598	59.254	55.265	54.31

Source: author's elaboration

7. CONCLUSIONS

In this thesis the analysis the efficiency of the different methods of solving Rubik's cube optimization was realized. For this purpose, first I derived the general formula for number of possible scrambled states of $n \times n \times n$ Rubik's cube, which I wanted to isolate and I managed to do that. As it is presented, results agree with information I managed to find on the Internet.

Considering the methods, I choose for comparison we see that the better method is definitely CFOP, compared to LBL, but we have to remember that it is a more advanced method and when people are about to solve Rubik's cube for the first time it is very hard to do it with CFOP method. It is also a method that required more computing power as it was more complex.

Speaking about results of optimization it is safe to say that they were satisfying. In the case of LBL we managed to decrease the number of moves by 10% and in case of CFOP by 13.1%. These are good results, whereas a little surprising, because as we said before, more steps were possible to optimize in the LBL method so we could expect a bigger decrease in this method, but actually a bigger decrease we achieved in CFOP method.

Comparing two optimizing approaches we can say in general that despite the fully optimized version is of course better, the results between two approaches are not that big in LBL, because they differ only by 2.46%. So we can say that it is not necessary to implement the fully optimized version in case of LBL, especially when we have limited computing power. Whereas in case of CFOP the decrease of number of moves is equal 6.73%, which is a much better result than in case of LBL. So we can say that in CFOP we should stick with fully optimized results.

Lastly comparing our results with results from the thesis of Duberg and Tideström (2015) we also achieved satisfying results. In the case of LBL my results are lower by 16.17% moves, which is a really big difference. In the case of CFOP I achieved slightly worse results, but the difference is at the level of 1.73%, so the results in this method are very similar, which I think is a good result.

We see that overall results of my work are satisfying, probably there is some room for improvement, but it is hard to implement any method perfectly.

Literature

Ken F. Fraser : Rubik's Cube Extended: *Derivation of Number of States for Cubes of Any Size and Values for up to Size 25x25x25*, 2017

W. D. Joyner : *Mathematics of the Rubik's cube*, 1996

https://en.wikipedia.org/wiki/Rubik%27s_family_cubes_of_all_sizes

Daniel Duberg and Jakob Tideström : *Comparison of Rubik's Cube Solving Methods Made for Humans*, 2015

Stefan Pochmann : *Analyzing Human Solving Methods for Rubik's Cube and similar Puzzles*, 2008

Madan, C. R. (2020) 'Considerations for Comparing Video Game AI Agents with Humans', *Challenges*, 11(2), p. 18. doi: 10.3390/challe11020018.

Petrus, L.: Solving Rubik's Cube for speed, http://lar5.com/Cube/

https://web.mit.edu/sp.268/www/rubik.pdf

El-Sourani N, Hauke S, Borschbach M. A*n evolutionary approach for solving the Rubik's cube incorporating exact methods*. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics). 2010;6024 LNCS(PART 1):80-89. doi:10.1007/978-3-642-12239-2-9

El-Sourani N, Borschbach M. *Design and comparison of two evolutionary approaches for solving the Rubik's cube*. Lect Notes Comput Sci (including Subser Lect Notes Artif Intell Lect Notes Bioinformatics). 2010;6239 LNCS(PART 2):442-451. doi:10.1007/978-3-642-15871-1_45

Zeng D, Li M, Wang J, Hou Y, Lu W, Huang Z. *Overview of Rubik's cube and reflections on its application in mechanism*. Chinese J Mech Eng (English Ed. 2018;31(4). doi:10.1186/s10033-018-0269-7

Travis M. *The Mathematics of the Rubik's Cube. MathUchicagoEdu*. 2009:1-20. http://math.uchicago.edu/~may/VIGRE/VIGRE2007/REUPapers/INCOMING/rubikscube.pdf

Bellman R. *Mathematics of the Rubik's Cube*. 1996:1-80.

Fogarassy, C. (2016) 'Comparison of Rubik ' s Cube Solution Softwares with SWOT Analyses for the Input-Output Process Modeling', *International Journal on Recent and Innovation Trends in Computing and Communication*, (December 2014).

Rubix cube solution. www.rubic.com

McAleer S, Shmakov A, Agostinelli F, Baldi P. *Solving the Rubik's cube with approximate policy iteration*. 7th Int Conf Learn Represent ICLR 2019. 2019:1-13.

Pochmann S. *Analyzing Human Solving Methods for Rubik's Cube and Similar Puzzles*. 2008:i-11. http://www.ke.informatik.tu-darmstadt.de/lehre/arbeiten/diplom/2008/Pochmann_Stefan.pdf.

Stitz H. *Visualization of Rubik ' s Cube Solution Algorithms*. 2019;(January 2009). doi:10.2312/eurova.20191119

Mcaleer S, Baldi P. *Solving the Rubik ' s Cube Without Human Knowledge*, 2018

Toshniwal MES. *Rubik's Cube Solver*: *A Review*. 2019.

Thistlethwaite, M.B.: *The 45-52 Move Strategy*. London CL VIII (1981)

Korf R. *Finding optimal solutions to Rubik's cube using pattern databases*. Proc Natl Conf Artif Intell. 1997:700-705.

Demaine M, Lubiw A. *Algorithms for Solving Rubik' s Cubes*. 2014;(June 2011):689-700.

Rokicki, T.: Twenty-Five Moves Suffice for Rubik's Cube, http://Cubezzz.homelinux.org/drupal/?q=node/view/121

Herdy, M., Patone, G.: Evolution Strategy in Action, 10 ES-Demonstrations. Tech- nical Report, International Conference on Evolutionary Computation (1994)

Borschbach, M., Grelle, C.: Empirical Benchmarks of a Genetic Algorithm Incor- porating Human Strategies. Technical Report, University of Applied Sciences, Ber- gisch Gladbach (2009)

Zemdegs, F. 'F2L Algorithms (First 2 Layers)', 2. Available at: http://cube.crider.co.uk/visualcube.php

Zemdegs, F. 'OLL Algorithms (Orientation of Last Layer)', 2. Available at: http://cube.crider.co.uk/visualcube.php

Zemdegs, F. 'PLL Algorithms (Permutation of Last Layer)', 2. Available at: http://cube.crider.co.uk/visualcube.php

https://artofproblemsolving.com/wiki/index.php/Rubiks_cube

https://www.cubeskills.com/uploads/pdf/tutorials/f2l.pdf

https://www.cubeskills.com/uploads/pdf/tutorials/oll-algorithms.pdf

https://www.cubeskills.com/uploads/pdf/tutorials/pll-algorithms.pdf

List of figures

List of tables

LIST OF APPENDICES

Code in R:
https://github.com/bryla121/Efficiency-of-methods-of-solving-Rubik-s-cube-implemented-in-R