Learn how to solve a Rubik's Cube using a Tutoring System

An algorithmic implementation of different strategies with a step-by-step approach based on human reasoning

---

Author:
Floris van Voorst tot Voorst

Student Number:
5990076

Supervisor:
Dr. Tomas Klos

Second Reader:
Dr. Benjamin Rin

A 7.5 ECTS Thesis submitted for the degree of Bachelor of Science

Artificial Intelligence
Faculty of Humanities
Utrecht University

30 April 2021

Abstract

Learn how to solve a Rubik's Cube using a Tutoring System

Floris van Voorst tot Voorst

The Rubik's cube is a popular logical puzzle with one simple goal: Start with some randomized, shuffled, messy configuration of the cube and, by rotating the faces, get back to the original solved pattern with each side being one single color. But this goal is not as easy as it might seem.
A lot of solvers that exist, are based on efficiency and finding a optimal solution in the least possible moves. These solutions do not take human reasoning and strategies into account and are therefore useless for people who are trying to learn to solve it themselves. These solvers can not logically explain their steps, while this is something which would aid humans in their solving process.
Therefore the aim of this thesis is to answer whether it is possible to create a tutoring system, based on human reasoning strategies, which could aid people to learn how to solve the Rubik's cube on different difficulties? And if this is the case, what would such a tutoring system look like?
In this thesis I will formalize three different solving strategies, which include different algorithms based on human reasoning. These different strategies vary in difficulty and are able to explain each step along the way. They are implemented into a Tutoring System in order to aid people in learning how to solve a Rubik's Cube.

Contents

Chapter 1

Introduction

The Rubik's cube is a popular logical puzzle with one simple goal: Start with some randomized, shuffled, messy configuration of the cube and, by rotating the faces, get back to the original solved pattern with each side being one single color. But this goal is not as easy as it might seem. A standard Rubik's cube of 3 by 3 has an immense number of $4.3 * 10^{19}$ different starting positions.[1] The complexity of these starting positions may vary between easy variations, consisting of only one move away from the solved position, to complex variations which require a lot more in depth analysis to solve quickly. This makes the challenge of solving the cube very interesting and that is why it is reckoned that about a billion people have at least tried to solve the cube, including myself.[2] The math behind the cube allows that there are always several different ways to solve the cube, ranging from efficient to inefficient methods. This makes the cube fun for casual players and for people attempting to get the world record solving time.[3]

Due to the logic and math nature behind a Rubik's cube, a lot of research has been conducted over the years in finding optimal and efficient algorithms for solving the Rubik's cube. A team of researchers combined their knowledge with the computing power of Google, to find that every permutation of the Rubik's Cube can be solved in twenty moves or fewer, where a move is defined as any twist of any face. Twenty is called the Rubik's cube 'Gods number'.[4,1]

Typically it is possible to divide the solving algorithms in two groups. First there is the set of algorithms which a human is not able to execute and usually require a enormous amount of computing power to calculate. Examples for such algorithms are: Korf's algorithm, which is an iterative-deepening-A* with a heuristic function based on pattern databases.[5] and algorithms which use deep reinforcement learning[6] or learned guidance functions.[7] The second set of algorithms are those for which a human is able to execute or calculate it. Examples of such algorithms would be in the CFOP[8] method, used by speed cubers around the world.

The main difference between the two sets of algorithms is the efficiency of the solution, or in other words, the number of moves that are required to get to the solved permutation. While the computer based algorithms are very successful at efficiently solving a Rubik's Cube, they give humans no insight into how to efficiently tackle the Rubik's Cube themselves and are therefore useless in helping a human learn how to solve it. If a human wants to learn how to solve a Rubik's Cube, it has to rely on algorithms that are understandable and based on the way a human approaches this logical puzzle.

This paper aims to investigate these algorithms that are understandable and based on human reasoning. There are a lot of different strategies a human is able to use in order to solve the Rubik's Cube, ranging from easier to learn inefficient strategies to harder to learn, but more efficient strategies. While these strategies use algorithms that are not as efficient as the computer based algorithms, they can be used to help humans learn how to solve a Rubik's Cube themselves.

Therefore, my goal is to create and understandable and transparent tutoring system which could help someone learn these different strategies. I will be trying the answer the research question whether it is possible to create a tutoring system, based on human reasoning strategies, which could aid people to learn how to solve the Rubik's cube on different difficulties? And if this is the case, what would such a tutoring system look like?

In this thesis I will propose three different strategies, which include different algorithms based on human reasoning. I will gather these different algorithms from different sources where people come to discuss the possibilities for solving the Rubik's cube. From these sources I will create strategies with increasing level of difficulty and analyse how these different difficulties compare to each other. These strategies will not only give the correct solutions, but should also show how to achieve these solutions and could therefore be used in creating a tutoring system to help people learn how the solve a Rubik's Cube themselves.

1.1 Artificial Intelligence and Social Relevance

Many of the solving algorithms for the Rubik's cube are designed to be as efficient as possible, but how the algorithm actually solves the problem is left in the dark. The system that is proposed in this paper aims at actually explaining what the steps are for reaching the goal state. This is relevant for the research of Explainable AI, which is an important topic in the field of Artificial Intelligence. The aim is to surpass the "Black box" and actually provide information about the steps that should be taken.[9] This paper attempts to combine human reasoning strategies with transparent algorithms on a logical puzzle, thus it may be helpful for further research on Explainable AI.

In addition, the research of the proposed tutoring system could also benefit other fields. The goal of the created system in this paper is to aid people into achieving a certain solution for a logical puzzle and learning from it. This could also be applicable for other fields where human reasoning is required to solve certain logical problems. Learning how to program could be such an example.

1.2 Structure of this Paper

In Chapter 2 the terminology of the Rubik's cube will be explained, followed by the formalisation of the rules of the Rubik's cube and it will discuss the different strategies and their difficulties. In Chapter 3 it is discussed how from the theoretical background a tutoring system is implemented and how it operates. Chapter 4 will go over the testing phase of the different strategies and analyze how their difficulties compare to each other. Finally in Chapter 5, the results of this paper will be discussed.

Chapter 2

Theoretical Background

The aim of this Chapter is to provide the necessary background information about the formalisation of the Rubik's Cube (2.1) and the different strategies humans take in solving the Rubik's Cube (2.2).
This chapter serves as the backbone for the implementation of the tutoring system in Chapter 3.

2.1 Rubik's Cube Formalisation

This section will provide the basic understanding of structure of the Rubik's Cube (2.1.1), the notation of the rotations of the Rubik's Cube (2.1.2) and the mathematics and logic behind the Rubik's Cube (2.1.3).

2.1.1 Structure

The classic Rubik's cube has the dimensions of 3 x 3 x 3 with 6 faces, where each face is differentiated by one of the six colors {blue, green, orange, red, white, yellow}. The model used in this thesis is called the Western Color Scheme, which indicate that white is on the opposite face of yellow, red is on the opposite face of orange and blue is on the opposite face of green.[10]
Each of the faces is split into 9 smaller faces which are called **facelets**. The cube has a total of $6 * 9 = 54$ facets and each of these facelets are colored in one of the six cube colors, such that there are 9 facelets per color. See Figure 2.1.

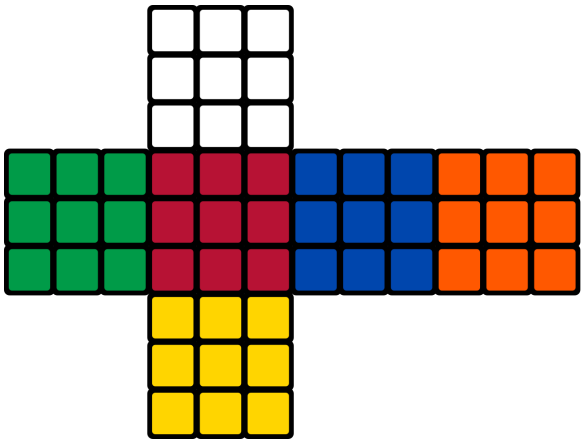
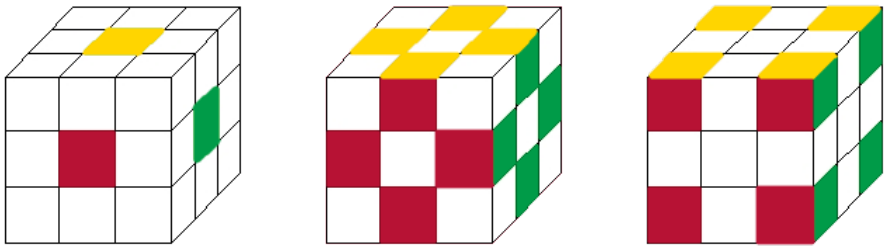


Figure 2.1: Two-dimensional representation of faces with their facelets using the Western Color Scheme.[10]

In order to transform this two-dimensional representation of the faces and the facelets into a three-dimensional cube, we need to define a model which connects these facelets together. To create this model, the Rubik's cube is divided into 26 smaller cubes, called **cubies**. There are three types of cubies: a center cubie, a edge cubie and a corner cubie.[11]
The center cubie is a single colored cubie, thus consisting of only 1 facelet. This cubie is always located in the center of a face. In total there are 6 center cubies, one for each face. See Figure 2.2a.
The edge cubie is a double colored cubie, thus consisting of 2 facelets. This cubie is always located on the edge of two faces. In total there are 12 edge cubies. See Figure 2.2b.
The corner cubie is a tri-colored cubie, thus consisting of 3 facelets. This cubie is always located on the corner of three faces. In total there are 8 corner cubies. See Figure 2.2c.



(a) Center Cubies (b) Edge Cubies (c) Corner Cubies

Figure 2.2: Three-dimensional representation of the different types of Cubies.

It is important to note that the center cubies have an important additional property, they can not move. The center cubies will always stay in the same position as they started in, which results in the fact that the opposite colors from the Western Color Scheme will always hold. The other two types of cubies can be moved, mirrored and twisted.

2.1.2 Notation

In order to describe the sequence of moves an algorithm provides, it is necessary to have an standard notation for the Rubik's Cube. In this paper we will use the standard notation, created by mathematician David Singmaster, which is agreed upon by most of the different sources that create algorithms for solving the Rubik's Cube.[12,11,13,14]

We start by assigning letters to the 6 different faces a Rubik's cube has, relative to how the cube is facing you:

- F (Front): the face currently facing you.
- B (Back): the face opposite of the front side.
- U (Up): the face above the front side.
- D (Down): the face below the front side.
- L (Left): the face left of the front side.
- R (Right): the face right of the front side.

In a sequence of moves, the annotation of one of these letters means to turn that face 90 degrees clockwise, or in other words, a quarter turn. To indicate a move counter-clockwise, the lowercase letter i will be added after the letter. A letter followed by a number x means that the same move has to repeated x times.

An example of such a sequence would be $F\ U\ Fi\ Ui\ B2$, meaning you would start with turning F and U clockwise, than F and U counter-clockwise and ending with turning B two times. A visual representation of the different moves can be see in Figure 2.3.

In order for an algorithm to solve a Rubik's Cube, it is sometimes necessary that the Rubik's Cube itself has to be rotated entirely. This will result in different colors facing you, which can be helpful in a quicker solving solution or a better understanding of the sequence. The following letters are assigned to these cube rotating moves:

- X : rotating the entire cube on R. (do an R move without holding the two other layers)
- Y : rotating the entire cube on U. (do an U move without holding the two other layers)

- Z : rotating the entire cube on F. (do an F move without holding the two other layers)

Facelets and cubies can also be denoted using letters. An important note is that these annotations will never appear in a sequence of moves, but are rather used to indicate which pieces are the focus of a move.

In the case of cubies we use the uppercase letters of the faces, combined as a string. For example an corner cubie could be the string 'ULF' which means we are talking about the cubie which resides in the corner between the Up, Left and Front face. An edge cubie will have a string of two letters 'UF', this would be the cubie on the edge of the Up and Front face. An center cubie will be a string of only one letter 'F', thus this would be the center cubie on the front face.

An similar style will be used for the facelets, but it will make use of lowercase letters of the faces. Thus an facelet on the corner of a face would be the string 'ulf', where u is the face on which the facelet is located and l and f are the faces it borders to. An edge facelet will be the string 'uf', where u is the face on which the facelet is located and f is the face it borders to. and a center facelet will be a single letter string 'f', the face which contains the facelet.

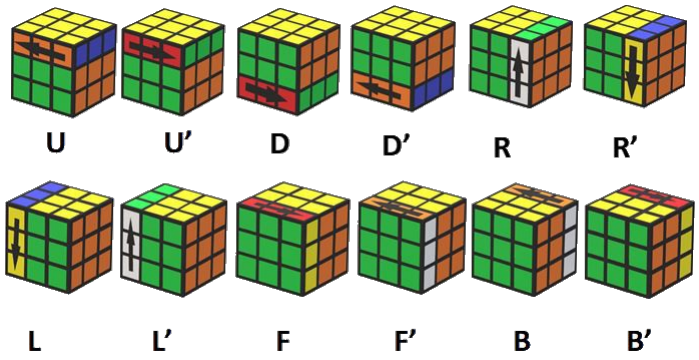


Figure 2.3: Visual representation of the different faces and their corresponding moves.[15]

2.1.3 Mathematics and Logic

Mathematically the Rubik's cube is a permutation group, consisting of 6 colours which are repeated 9 times. Therefore the Rubik's cube could be considered a ordered list with 54 elements, on which we can apply the basic permutation operations as described in the previous subsection 2.1.2. These permutations will reorient the cube in an certain way, thus reorienting the ordered list in a certain

way. A specific combination of these permutations can lead from a unsolved to a solved pattern. An example of a permutation operation on the ordered list can be seen in Figure 2.4.

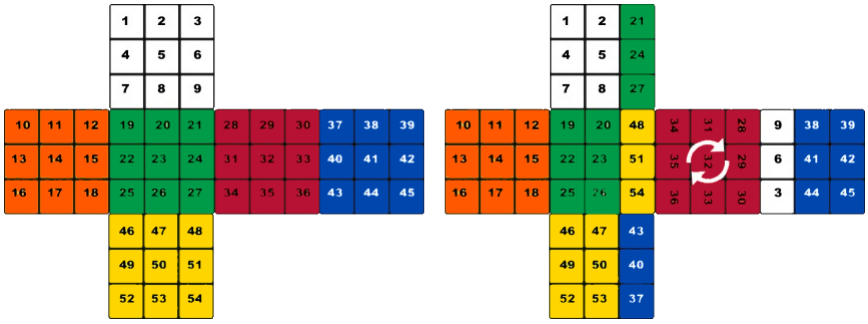


Figure 2.4: Visual representation of a permutation operation on the cube as a ordered list.[1]

Due to the logic of the Rubik's cube, there are certain restrictions in place for the number of different permutations that are possible. As described in the subsection 2.1.1, the center cubies can not be moved. The edge and corner cubies have fixed colors on each of their facelets, thus making it impossible to have a permutation which consists of a edge piece with the colors white and yellow or a corner piece with white, yellow and blue. This is proven in Figure 2.4 above, a move of the face R changes the permutation, but the edge and corner cubies still have the same facelets as they had before. Without these restrictions, the number of possible permutations would be $5.19{*}10^{20}$, but with these restrictions it is decreased to $4.3 * 10^{19}$.[1]

Another important mathematical fact about the permutation operations, is that every operation has a certain degree. A degree is a finite number that shows how many times we have to execute the operation to return to its initial permutation.[16] For example, if we would use the F operator four times in a row, it would result in the same permutation as the starting permutation. This can be used to optimize the solving algorithms. Since the finite number for the standard permutation operations is 4, we can conclude that F3 equals F'. Therefore we can reduce the total number of moves needed to reach the a certain permutation by 2, if we replace F3 with F'. A more in depth optimization will be discussed in Chapter 3.

2.2 Rubik's Cube Strategies

This section will explain the different strategies humans use to solve the Rubik's cube. It will start with the beginner method (2.2.1), then the intermediate method (2.2.2) and finally the expert method (2.2.3).

2.2.1 Beginner

Due to the immense number of different permutations of the Rubik's Cube, it is impossible to memorize every single sequence of moves to solve each permutation individually. Therefore the algorithm for beginners divides the cube into stages, with each having a set number of algorithms to solve the current stage. The algorithms keep track of all previous solved stages and they make sure that they stay solved. For simplicity, it is commonly used to start with the white face of the cube and end with the yellow face. Therefore my algorithms will be based on that notion.[17]

2.2.1.1 White Cross

The first stage is to make a white cross on the white face of the cube. You must find the four different edge cubies which contain a white facelet and set these in their correct positions. It is important that you take into account that the other colors of the edge cubies should also match their respective side center cubies as seen in Figure 2.5b.
Since there is not yet anything solved on the cube, the rest of the cube can be disregarded while you try to solve the cross. If the edge cubies are on the B face of the cube, there are two fixed algorithms in order to set the edge cubie in its correct position.
If the edge cubies' white facelet is already on the Z-axis, you only need to rotate the corresponding side twice in order to get the correct position of the edge cubie.
If the edge cubies' white facelet is on the X-axis, it requires the edge cubie to be mirrored after it has been put in its correct position. Mirroring a edge cubie can be done with the following sequence of moves: $F\ Ui\ R\ U$.

(a) Incorrect permutation.

(b) Correct permutation.

Figure 2.5: White cross solutions.[17]

2.2.1.2 White Corners

The second stage is finishing the white face by placing the corner cubies containing a white facelet in their correct position, with taking the side face colors into account.
Rotate the cube in such a way that the white face is on the U face of the cube.

The next step is to rotate the D face until the corner cubie on the D face, which contains the three correct facelets, is directly below the corner piece of the white U face where it is supposed to be.
There are three different algorithms, based on the orientation of the corner cubie, that can lead to the correct orientation:
If the corner cubies' white facelet is on the X-axis, the sequence of moves is: $Ri\ Bi\ R$.
If the corner cubies' white facelet is on the Y-axis, the sequence of moves is: $D\ B\ Di$.
If the corner cubies' white facelet is on the Z-axis, the sequence of moves is: $Ri\ B\ B\ R\ B\ Ri\ Bi\ R$.
Repeat this for each of the corners until the white face is solved, see Figure 2.6.

Figure 2.6: White face solution.

Now that the white face is solved, rotate the cube with the following sequence: $X\ X$, resulting in the white face being at the bottom.

2.2.1.3 Second Layer

The third stage is to finish the second layer. Find the correct edge cubie (one with two matching facelets colors for F) on the U layer of the cube and turn the U face in such a way that the edge cubie is above the F face and place the cubie in his respective place on the F face of the cube.
There are two different algorithms the achieve this goal, depending on the orientation of the edge cubie:
If the edge cubies' facelet on the Y-axis is the front color of the cube, the sequence of moves is: $Ui\ Fi\ U\ F\ U\ R\ Ui\ Ri$.
If the edge cubies' facelet on the Y-axis is the right color of the cube, the sequence of moves is: $U\ U\ R\ Ui\ Ri\ Ui\ Fi\ U\ F$.
Always make sure the D layer of the cube stays intact.
If both edge cubies on the F face are correct, rotate the cube using the sequence: $Y\ Y$ and repeat the process. If this stage is solved, it looks like Figure 2.7.

Figure 2.7: First two layers solution.

2.2.1.4 Yellow Cross

The fourth stage is to create the yellow cross on the U face of the cube.
There are four possible patterns at the start of this stage. One of these possible pattern is already a yellow cross, which means you can skip this stage and go to the fifth stage. The other three patterns, as can be seen in Figure 2.8, require the use of two algorithms, depending on the current situation:
If the pattern of the edge cubies' yellow facelets on the Y-axis of the cube make a line shape (see Figure 2.8c), the sequence of moves is: $F\ R\ U\ Ri\ Ui\ Fi$.
If the pattern of the edge cubies' yellow facelets on the Y-axis of the cube make an L shape (see Figure 2.8b), the sequence of moves is: $F\ U\ R\ Ui\ Ri\ Fi$.
If none of the edge cubies' yellow facelets are on the Y-axis of the cube (see Figure 2.8a), the combination of the first and second algorithm is the required sequence.

(a) Pattern 1. (b) Pattern 2. (c) Pattern 3.

Figure 2.8: Three different starting patterns.[17]

In this stage it does not matter if the edge cubies are not in their correct spot regarding the side face colors, the only goal here is to make sure all the yellow facelets of the edge cubies are visible on the U face. If this stage is solved, it looks like Figure 2.9.

Figure 2.9: Yellow cross solution.

2.2.1.5 Yellow Edges

The fifth stage is switching the yellow edge cubies to their correct position, respective to the center cubies colors on each of the cube side faces. It is possible that all the edge cubies color already match with each respective center cubie color, in that case move on to stage six.
In case that they do not all match, there is one algorithm to fix the edge cubies locations which are not correct, while keeping the rest of the yellow cross and cube intact:
If one edge cubie is already in its correct position, a algorithm that switches three edges clockwise is repeated until they are all in the position they are supposed to be in. The sequence of moves is: $Ri\ U\ U\ R\ U\ Ri\ U\ R\ U$.
If all the edge cubies are not in their correct position, rotate the U face until one of them is in its correct position. After this, it is similar to the previous case.
If this stage is solved, it looks like Figure 2.10.

Figure 2.10: Yellow cross solution, with matching side colors.

2.2.1.6 Yellow Corners Placement

The sixth stage is getting the last four corner cubies in their correct position, respective to the sides face colors of the cube.
The orientation of the corners does not matter in this stage, the only goal is getting them in their correct position. A algorithm that swaps the position of three of the four corners clockwise can be used to get all four corners correct, while making sure the rest of the yellow cross and cube stay intact:
If one corner cubie is already in its correct position, the algorithm is repeated

until they are all in the position they are supposed to be in. The sequence of moves is: *Ri U L Ui R U Li Ui*.
If all the corner cubies are not in their correct position, the algorithm is repeated on three of the four corner cubies until one is in its correct position. The algorithm will then rotate the remaining three until they all are in their correct position.
If this stage is solved, it will look like Figure 2.14, although the orientation of the corner cubes can be different.

Figure 2.11: Yellow corners placement solution.

2.2.1.7 Yellow Corners Orientation

The seventh and last stage is about the last four corner cubies orientation. All the cubies are in their correct location, thus orienting the last four corner cubies will solve the Rubik's Cube.
A fixed algorithm will be repeated on each corner until the yellow facelet is on the Y-axis of the cube. The sequence of moves: *Ri Di R D Ri Di R D*.
It is important that you keep the same face of the cube on the F side of the cube, while only doing permutations to the R and D layers of the cube. Once you have orientated all of the four corner cubies, the Rubik's Cube is solved and looks like Figure 2.16.

Figure 2.12: Solved Rubik's Cube.

2.2.2 Intermediate

The Intermediate strategy builds upon the Beginner strategy and the goal of the Intermediate strategy is to decrease the total number of moves required to solve the Rubik's Cube.[18]
The Intermediate strategy combines some of the stages from the Beginner subsection 2.2.1 and optimizes them. The Intermediate method requires more and different algorithms, but this will decrease the total number of moves, thus solving the Cube faster. This section will only go over the changes between the Beginner and the Intermediate strategy, leaving out the stages that are still the same.

2.2.2.1 First Two Layers

The first new stage combines stages 2 and 3 from the Beginner section 2.2.1. Instead of solving these two stages individually, the aim is to solve the stages simultaneously and as a result reducing the number of moves considerably.
In order to do this, the goal is to pair up the white corner cubie from stage 2 with the corresponding edge cubie from stage 3, before putting them in their correct position. There are 41 different possible permutations the corner and edge cubie can be in and each of these permutations has a fixed algorithm to solve that permutation.
All the different permutations with their corresponding algorithms can be found in Appendix A.
In Figure 2.13a you can see an example of a paired corner cubie with its respective edge cubie and in Figure 2.13b the permutation move Ri puts them in the correct place simultaneously.

(a) Combining the corner- and edge cubie.

(b) Rotating them in the correct position.

Figure 2.13: F2L example.

If you do this for all four corner- and edge cubies, you have successfully reached the state of Figure 2.7 by combining stages 2 and 3.

2.2.2.2 Simple Permutation of Last Layer

The second new stage improves stages 5 and 6 from the Beginner section 2.2.1. The Beginners strategy uses two algorithms, one for the corner cubies and one for the edge cubies, to get them in their correct positions. This is not the most efficient way, since the algorithms may have to be repeated several times to get the correct permutation. In the intermediate strategy, the goal is to add several algorithms. With these new algorithms, there is no need for repetition, thus less moves are required to reach the same result.
For the edges cubies there are two new algorithms:
It adds a algorithm that swaps three edges cubies anti-clockwise, the sequence of moves is: $Ri\ Ui\ R\ Ui\ Ri\ U\ U\ R$.
It adds a algorithm that swaps two edges with each other, the sequence of moves is: $R\ U\ Ri\ U\ R\ U\ U\ Ri\ U$.
For the corner cubies there is one new algorithm:
It adds a algorithm that swaps three corner cubies anti-clockwise, the sequence of moves is: $U\ L\ Ui\ Ri\ U\ Li\ Ui\ R$.

Keep in mind that the corner- and edge cubies will still be moved separately, in the Expert method they will be combined into one algorithm. See Paragraph 2.2.3.2.
If this new stage is solved, it will look like Figure 2.14, although the orientation of the corner cubes can be different.

Figure 2.14: Yellow corners solution.

2.2.3 Expert

The Expert strategy builds upon the Intermediate and Beginner strategies and the goal of the Expert strategy is to decrease the total number of moves required to solve the Rubik's Cube even further.[18] The Expert strategy combines stages from the Beginner and Intermediate strategies. The Expert method requires more and different algorithms, but this will decrease the total number of moves, thus solving the Cube faster. This section will only go over the changes between the previous two strategies and the Expert strategy, leaving out the stages that are still the same.

2.2.3.1 Orientation of Last Layer

The first new stage combines stages 4 and 7 from the Beginner section 2.2.1.
The goal of this stage is to orientate all the cubies on the U face of the cube in such a way that the entire U face is yellow. The side colors and the positions of the corner- and edge cubies do not matter, these will be fixed in a later stage. There are 57 different possible permutations that the cube can be in and each of these permutation has a fixed algorithm to solve that permutation.
All the different permutations with their corresponding algorithms can be found in Appendix B.
If this stage is solved, it will look like Figure 2.15, although the colors of the corner- and edge cubies may be a different combination.

Figure 2.15: OLL solved.

2.2.3.2 Permutation of Last Layer

As mentioned in Paragraph 2.2.2.2, the Expert strategy will combine moving the corner- and edge cubies simultaneously.
To achieve this goal, there are a number of new algorithms to learn. The first step is to align the U face of the cube to match as many of the correct positions as possible. All the cubies that are not in their correct positions, needs to be swapped. There are 21 different possible permutations that the cube can be in and each of these permutation has a fixed algorithm to solve that permutation.
All the different permutations with their corresponding algorithms can be found in Appendix C.

The orientation of the cubies is already fixed in the previous Paragraph 2.2.3.1, thus the Rubik's Cube should be solved now. See Figure 2.16.

Figure 2.16: Solved Rubik's Cube.

Chapter 3

Algorithmic Implementation

The aim of this Chapter is to implement the theoretical background information given in Chapter 2, into a working tutoring system. It starts with explaining the algorithmic implementation of the Rubik's Cube (3.1), followed by the implementation of the strategies (3.2) and finally how these are combined into a tutoring system (3.3).
This Chapter will only explain how everything is implemented, the full code base of the implementation is available on GitHub.

3.1 Rubik's Cube Implementation

This section will provide the algorithmic implementation of the structure of the Rubik's Cube (3.1.1) and the algorithmic implementation of the mathematics and logic behind the Rubik's Cube (3.1.2).

3.1.1 Structure

The backbone of the Rubik's Cube structure implementation is the Cubie class.[19] As explained in Subsection 2.1.1, there are 3 different types of cubies and in total there are 26 cubies that make up a Rubik's Cube. The Cubie class is the representation of these cubies and stores two pieces of information about each Cubie.
First it stores a position vector (x, y, z), where x, y and z have a integer value of {-1, 0, 1}. A positive x-axis points to the Right face, a positive y-axis points to the Up face and a positive z-axis points to the Front face. This way is it possible to distinguish 27 different cubies. The cubie (0,0,0) lies at the centre of the Rubik's cube, which is not visible and can therefore be disregarded. This leaves the Rubik's cube with the positions of the 26 visible cubies.
The second information it stores is a color vector (cx, cy, cz), where cx, cy and

cz are the different facelets a cubie can have with a string value of {'White', 'Orange', 'Yellow', 'Green', 'Red', 'Blue', None} The reason the value can be None, is because the center and edge cubies consist of only one or two facelets respectively. An example of such an edge piece would be ('Orange', None, 'White'), with the orange facelet on the x-axis and the white facelet on the z-axis. The position of this edge piece would in this case be (1, 0, 1). See Figure 3.1.
The combination of the Position vector and the Color vector makes it possible to identify any of the cubies by its absolute position or by its unique combination of colors.

Figure 3.1: Visual representation of the Position vector and the Color vector of a edge cubie.

The Cube class is built on top of this Cubie class and stores a list of the different Cubies and provides the methods for the different moves that can be used on the Rubik's Cube in order to change the permutation.[19] See Subsection 3.1.2 for the in depth explanation of how a move is executed.
The Cube class has a two-dimensional representation of the Rubik's Cube and requires an input string of all the different positions of the facelets in order to get the starting permutation. The facelets should be ordered from the number 1 to 54, as shown in Figure 3.2.

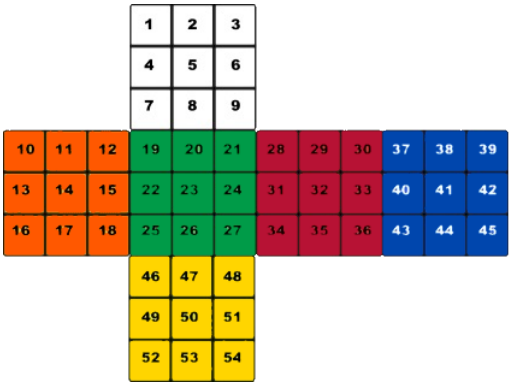


Figure 3.2: Visual representation of a solved two dimensional Rubik's Cube as a ordered list.[1]

3.1.2 Mathematics and Logic

The Cubie class has a rotation function in order to update the Position vector and the Color vector on a individual level. The Position vector is updated via a vector-matrix multiplication with a rotation matrix and the color vector is updated by swapping the colors in the color vector to make sure they face the correct way. Keep in mind that a rotation on a cubie can never change the orientation of all three color values of a cubie. One of the color values will always have the same orientation, while the other two color values have to be swapped. In case of an edge cubie, one of these colors has value None, but it still holds that only two values of the color vector have to be swapped.
The Cube class handles the logic behind a move on the Rubik's cube. Since the Cubie class rotation function takes controls of the rotation on an individual level, the Cube class only has to make sure that the rotation function of the Cubie class is applied on the cubies involved in a certain move. In order to do this, the Cube class constructs the appropriate rotation matrix for a rotation on the axis that is moved and selects all the cubies which lay on this axis, resulting in the new permutation of the Rubik's Cube.

3.2 Strategy Implementation

This section will discuss the implementation of the strategies and stages (3.2.1) and the optimization of the different output sequences (3.2.2).

3.2.1 Strategies and Stages

As explained in Section 2.2, each of the three different strategies are divided into smaller stages. The Beginner strategy has 7 stages, the Intermediate strategy has 5 stages and the Expert strategy has 4 stages. Since some stages are used in all three of the strategies, the total number of different stages is 12.
These 12 different strategies are implemented as individual functions. This makes it possible for each of the stages to be invoked separately, thus only giving the output for that specific stage. The strategies themselves are implemented as calling functions, which means that the only thing they do is making sure that the correct stage functions are invoked. If a strategy function is invoked, the output will be the combined output for all the stages in that strategy.
The output of these stages and strategies, is a sequence of moves. A sequence of moves is a list of the required moves to go from the start permutation to the goal permutation. This means that if a stage function is invoked individually, it will be a sequence of moves to reach the next stage and if the strategy function is invoked, it will be a sequence of moves that leads to the solved permutation.

3.2.2 Optimization

The output of a stage or strategy is not always a optimized sequence of moves. There are two reasons why this is the case. The first reason is that in order for a stage to use one of the fixed algorithms, it has to do some redundant moves first to find the correct starting permutation for those fixed algorithms. The other reason is that sometimes a stage requires several of these fixed algorithms in a row. While the fixed algorithms themselves have a optimized sequence, this does not mean that combining them leads to a optimized sequence. Therefore there are several improvements that can be made on these outputs and the aim of this subsection is to go over the optimizations that are implemented into the system.[19]

The first optimization method searches the output for moves that are directly followed by its reversed counterpart. These moves can be eliminated, because the net effect will remain the same.

An example of a part of the sequence that will be eliminated by this is: $R\ R\ Ri\ Ri$.

The second optimization method searches the output sequence for moves that are repeated more than two times in a row. If a move is repeated three times in a row it can be changed to a anti-clockwise move and if a move is repeated four times in a row it has no net effect and can thus be eliminated. Take for example the sequence $U\ U\ U$, this will be turned into Ui. Another example is the sequence $U\ U\ U\ U$, which will be eliminated entirely.

3.3 Tutoring System

Intelligent tutoring systems (ITS) are computer programs which are designed to incorporate techniques for solving certain problems, in order to provide a system that knows how to solve these problems and how to teach to solve these problems. The general structure of an ITS, as can be seen in Figure 3.3, consists of four elements.[20]

The first element is the Expert knowledge module, which includes all the knowledge to solve the problems and all the rules and steps to get to these solutions. These Expert knowledge modules are fully transparent, therefore each reasoning step can be inspected and interpreted. The second element is the Student model module, this refers to the representation of the knowledge and skill of the student. In order for the tutoring system to work, it requires a certain understanding of the problems prior to the tutoring. The third and the fourth modules are the actual tutoring system. These are the different features that the system has in order to teach the user and the interface to access these different features.

The next part of this section will go over the algorithmic implementation of the tutoring system that I designed for this thesis. The user should understand the basic notation and logic of the Rubik's Cube in order for the tutoring system to work.

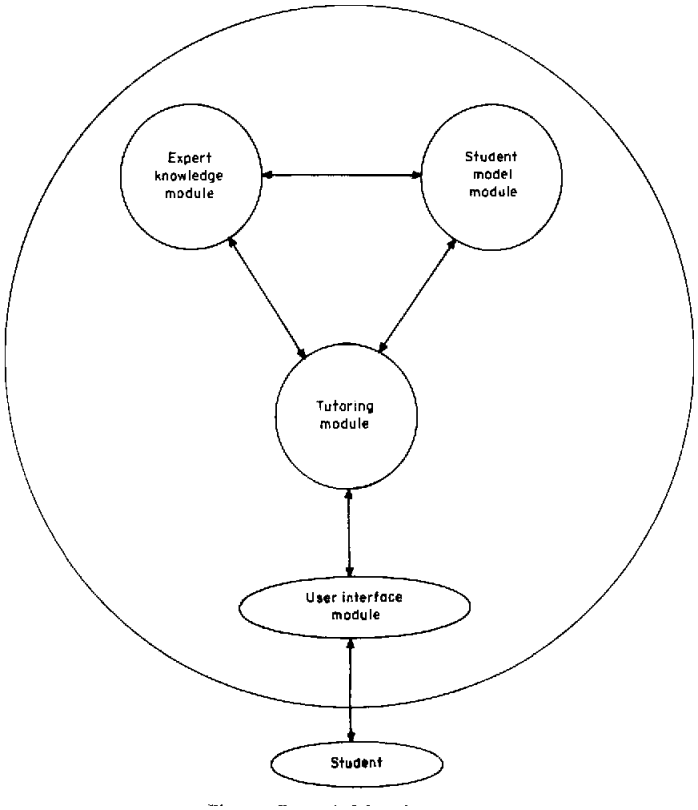


Figure 3.3: General ITS architecture.[20]

3.3.1 Visual Application

The Tutoring system is a Graphical User Interface, created with the inbuilt Python library TKInter.[21] The full interface can be seen in Figure 3.4.
The different features of the interface are presented in the Figures 3.5, 3.6 and 3.7. In short, the interface contains an input for the Rubik's Cube string, an output string, a 2D-visualisation of the Rubik's Cube, multiple strategy selection buttons, a description of the selected stage and a input for a sequence of moves.
The interface works as follows: A user can insert his own Rubik's Cube string or choose a randomly generated starting permutation. The next step is deciding which of the strategies the user wants to learn. Once a strategy is chosen, the user can click on the different stages of that strategy to get a description of the stage, the goal of the stage and some hints for solving the stage. The user can manually add a sequence of moves to get to the goal permutation, or choose that the system will automatically fill in the recommended sequence of moves.

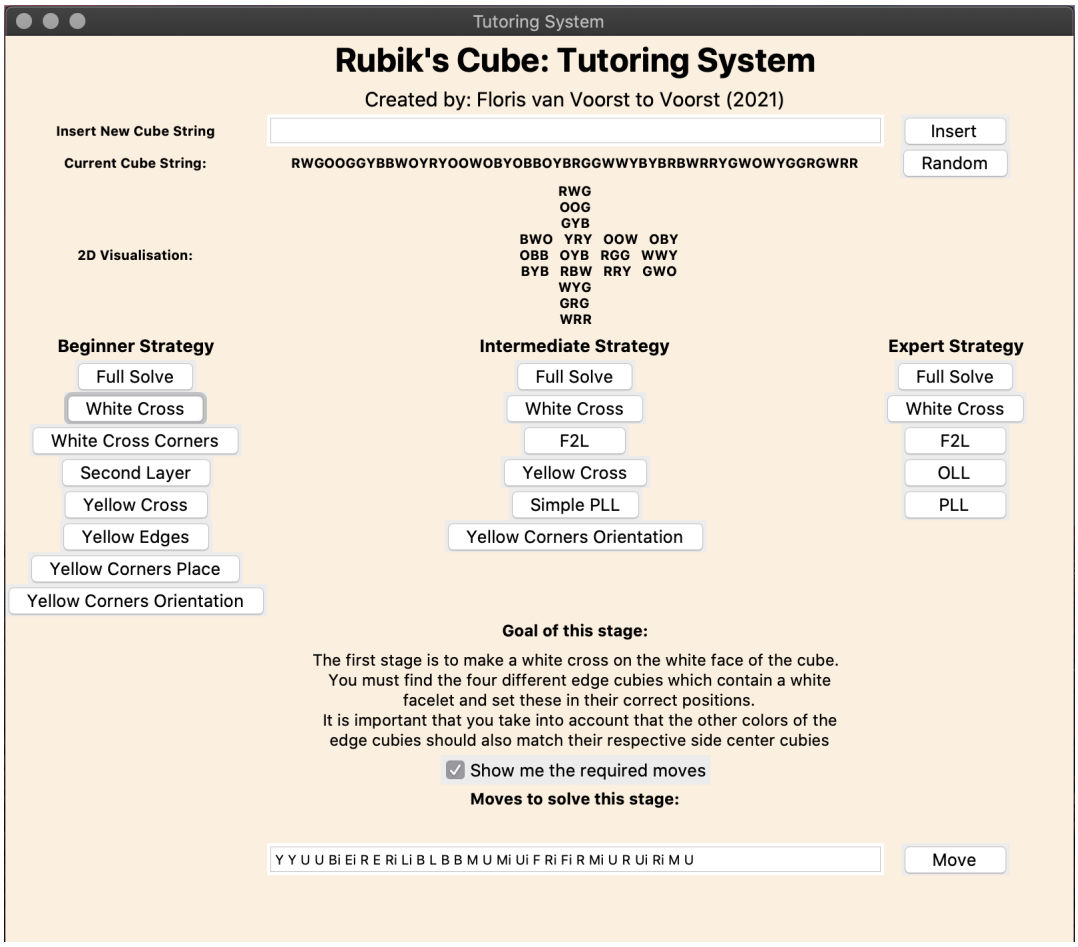


Figure 3.4: The Tutoring System's visual interface.

Figure 3.5 shows the first set of features. These are the features that represent the current Cube string. The user has the ability to insert a new string into the system, which will automatically change the current Cube string output and the 2D-visualisation. The user can also choose to import a random cube string with the random button. This randomly generated starting permutation is created by doing 300 random moves on a solved Rubik's Cube, thus it always results in a valid permutation.

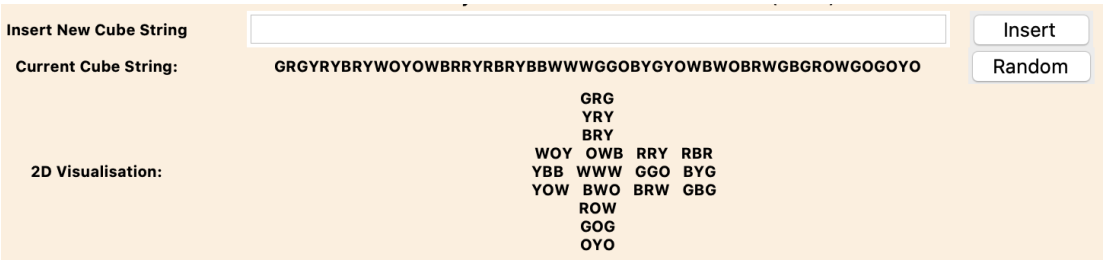


Figure 3.5: The first set of features, the representation of the cube.

Figure 3.6 shows the second set of features. These are the strategy features. The user has the ability to choose among the three different difficulties and their corresponding stages. The user can either choose to get a full solve, or go with a step-by-step approach. A full solve will generate the sequence of moves from the start permutation to the solved permutation. The user will get a warning that this is not the correct way to learn how to solve a Rubik's Cube, but that this feature rather exist for testing whether a given input string was valid, or in other words, whether the given input string was solve able.
If the user decides to take the step-by-step approach, he starts at the first stage and works his way down until he has solved his starting permutation. Ideally, the tutoring system would have been able to tell the user what stage his permutation requires. Due to the lack of time to implement this feature, the user has to manually check which of the stages are already solved for his given permutation. In Chapter 5, I will go in more detail about the shortcomings of this implementation.

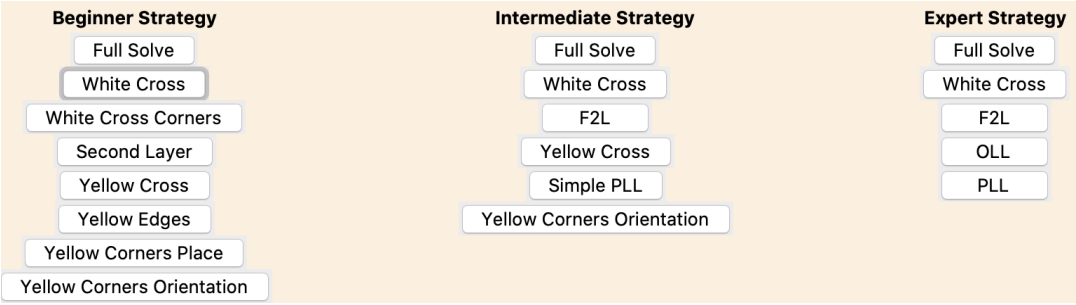


Figure 3.6: The second set of features, the different strategy options.

Figure 3.7 shows the third set of features. These are the information and move features. When the user chooses a particular strategy or stage, the information about this stage is displayed here. It will give a short introduction of the stage, it will tell the user what the goal of a stage is, which pieces are the main objective and some hints to make the solving easier.

The user has two different approaches in order to solve a stage. The first approach the user can take, is manually inserting a sequence of moves. The moves will then be applied onto the current Rubik's Cube string and thus change its permutation. The user can use the given information about the stage to help him in the process of solving it. It is not required to give one single sequence of moves, the user can keep adding different sequences of moves until the stage is solved. After each of the sequences, the current permutation will be updated. The second approach is meant for when the user gets stuck. If the user can not find the right sequence of moves in order to get to the goal permutation, he can ask the tutoring system to add the required sequence of moves into the input area. The tutoring system will calculate this sequence of moves based on the current permutation. The user can then decide to execute this given sequence or remove it from the input and try to replicate a similar sequence themselves. Once the current Cube has reached the goal permutation of a particular stage, the user can move on to the next stage and repeat the whole process.

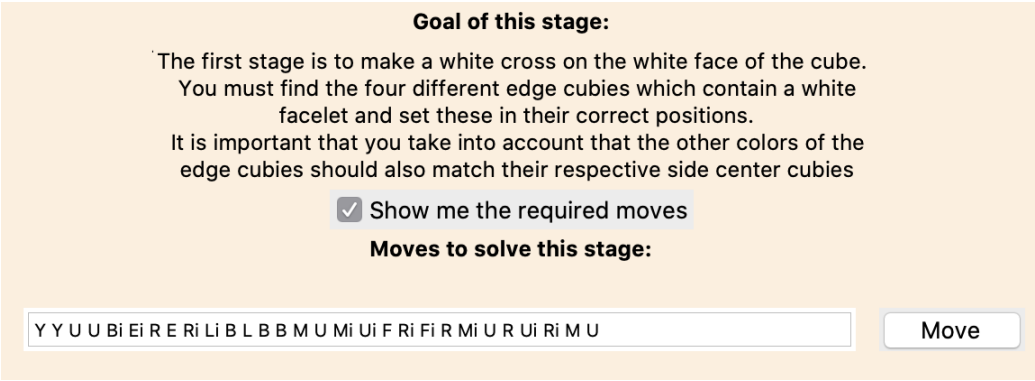


Figure 3.7: The third set of features, the provided information from the system.

Chapter 4

Testing and Analysis

The aim of this chapter is to go over the testing phase of the algorithms (4.1) and to analyze the different algorithms and how their difficulties compare to each other (4.2).

4.1 Testing

Testing is an important part of designing algorithms, because testing is necessary in order to make sure that the algorithms work as intended. Due to the immense number of different permutations a Rubik's Cube can have, as discussed in section 2.1.3, it is very important that all three strategies are tested on a large data set. The large size of the data set ensures that the algorithms can handle all the different starting permutations of a Rubik's Cube.
A data set entry is created by taking a completely solved Rubik's Cube as input and performing 300 randomly chosen moves on it. The reason that these moves are randomly chosen, is in order to create unique data set entries.
The entire data set used for testing the algorithms consisted of 10.000 randomly generated Rubik's Cubes. The number of different entries is chosen arbitrarily, but it should be big enough to represent all of the different permutations a Rubik's Cube can have. The entire data set was divided in 10 smaller subsets which were each tested separately. I chose to divide the data set into smaller subsets, because that way I could easier isolate where a problem would be in the case of an infinite loop or another error from one of the algorithms. A subset of 1000 uniquely generated entries of the entire data set can be found on GitHub in the data_set.txt file. (see link in Chapter 3)
After each subset was tested, the results were that all of the three strategies solved each data set entry correctly, thus we can confirm that the strategies work as intended.

4.2 Analysis

The best way to analyse the efficiency of each strategy and the difference between each of the strategies difficulty, is to analyse the number of moves it takes them to solve a Rubik's Cube. I have implemented three different difficulties, therefore the Beginner difficulty solution should result in more moves than the Intermediate difficulty. The same goes for the relation between the Intermediate difficulty and the Expert difficulty. The reason that there should be a difference in difficulty, is the amount of fixed algorithms that are needed to solve each difficulty. The Beginner strategy has 12 different algorithms, the Intermediate strategy has 51 different algorithms and the Expert strategy has 121 different algorithms.
Each of the strategies were given the same randomly generated starting cube as input and gave the total number of moves required to solve it as output. As discussed in subsection 3.2.2, the total number of moves can be optimized, therefore the output will be two numbers, the total number of moves and the total number of moves optimized.
The number of moves a strategy needs to solve different Rubik's Cubes can vary a lot, depending on the start permutation a particular Rubik's Cube has. To eliminate this variation, each strategy solved the same set of 1000 randomly generated Rubik's Cube. In order to see the difference between the number of moves on different Rubik's Cubes, the average number of moves is computed after 1 solve, 10 solves, 100 solves and 1000 solves. The results are shown in Figure 4.1 and Figure 4.2.

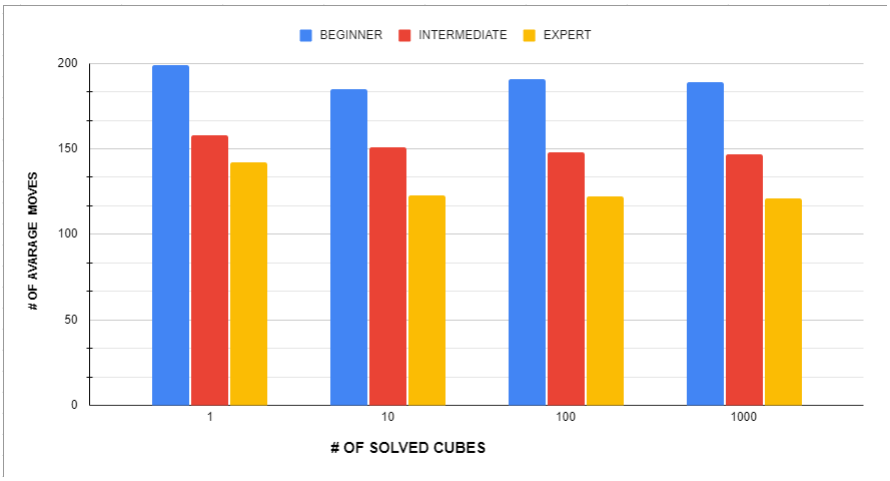


Figure 4.1: Grouped Bar Chart visualising the difference in average moves for each Strategy.

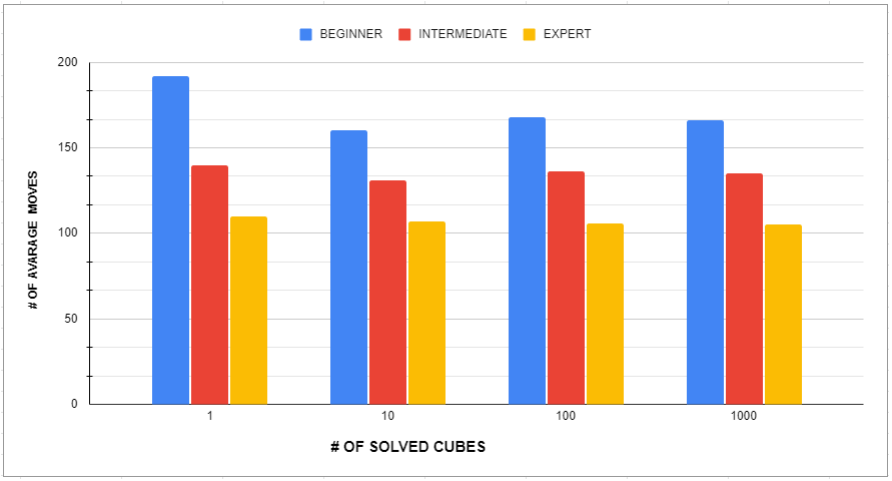


Figure 4.2: Grouped Bar Chart visualising the difference in average optimized moves for each Strategy.

First and the most important thing to notice over all, is that both Figure 4.1 and Figure 4.2 confirm that the number of average moves and the number of average optimized declines from Beginner to Intermediate and from Intermediate to Expert. This is great news, because this confirms that the strategies do have a difference in difficulty!
The second important thing to notice is the efficiency of the different strategies. One method for measuring the efficiency of a strategy is to compare the decrease in moves after optimizing. The intermediate and expert strategies show an average decrease of 15 moves after optimizing, while the beginner strategy shows an average decrease of 30 moves after optimizing. This shows that the Beginner strategy is less efficient compared to the other two strategies. Another method for measuring efficiency of a strategy is to compare the difference in the average number of optimized moves between the different number of solved cubes. The Beginner strategy fluctuates a lot more, while the Intermediate and Expert method seem more stable, thus meaning that they are more efficient overall.
The last important thing to notice is that taking the average of moves over a larger randomly generated cube set gives a way better perspective than solving only one cube. You can clearly see that the number of average moves over 10, 100 and 1000 solved cubes are pretty close, while the numbers after only one cube are very different. This confirms that taking the average of moves over a large data set was the best choice for comparison.

Chapter 5

Discussion

The aim of this final Chapter is to discuss the strengths and shortcomings of the algorithmic implementation, give some insights for further research, discuss the results of this paper and give an answer on the research question.

The Strategies
When creating the three different strategies, I designed them in a way that there should be a difference in difficulty and efficiency between them. The results from the analysis confirm that this was the case.
To assist human users in solving their permutation of the Rubik's Cube, the strategies take a human reasoning based approach. This means that the way the strategies solve the Rubik's Cube is understandable for human users. Still, I do believe that there is more room for improvement in the algorithmic implementation of these strategies. There are several cases where the algorithm needs to do a lot of repeated moves or entire cube rotations in order to find the correct permutation to continue. This is a problem that could be optimized. A human could easily see that turning the cube anti-clockwise is the right permutation. The algorithm does not see this and turns the cube clockwise, checks if it can be solved or continues to turn the cube clockwise until it hits the anti-clockwise position that the human could see from the start. I tried to make up for these redundant moves by optimising the required sequence of moves once the solution was found. This does eliminate several of these redundant moves, but it can not find them all. Improving these problems in the code itself, by adding some kind of self learning, could result in a more optimized outcome, while also reducing the time the system needs in order to solve the cube. For the purpose of teaching the user how to solve a Rubik's Cube, these problems are only minor inconveniences and do not effect the goal of the tutoring system.

The Tutoring System
The Tutoring system works as intended and is able to explain to the user what the different stages' goals are and to give them hints for trying to solve these stages. However, there are some additions that could be made to the tutoring

system to improve the user experience.
As briefly mentioned in section 3.3.1, the tutoring system requires the user to manually check which of the stages are already solved for their given permutation. An improvement to the tutoring system would be a feature that can calculate the current stage of any given permutation. This way the tutoring system could aid the user by telling them what the required next stage would be, depending on which of the strategies the user wants to learn.
Another addition for the tutoring system would be an automatic error detection. If the user tried a particular sequence of moves to solve a stage and it failed, the tutoring system could analyse this sequences of moves and give tips back about what went wrong or give them advice for a different approach.
A final possible addition for the tutoring system could be the implementation of different kinds of Rubik's Cubes. This thesis only focuses on the standard 3 by 3 by Rubik's Cube, but there are a lot of different variations that could be included in further research.[22] The logic behind the tutoring system is open for the expanding of different cubes and different strategies.

Conclusion
To answer the research question, I believe that I have achieved to create a tutoring system, based on human reasoning strategies, that can aid people to learn how to solve the Rubik's cube on different difficulties. I created three different strategies with increasing difficulties and these strategies are able to explain the steps that are necessary in order to achieve a solved pattern. By explaining the necessary steps and giving hints about what these steps should achieve, the tutoring system is able to help a human user to learn how to solve a Rubik's Cube with the three different difficulties.

To reflect on the artificial intelligence and social relevance, I would not classify the current tutoring system as a Explainable AI, due to the shortcomings as explained earlier. However, the implementation of the suggested self learning in the strategies could be a stepping stone to becoming a real Explainable AI.
I believe that this research could prove valuable in different fields that use logic as a basis. While this research only focuses on Rubik's Cubes, there are many other logical problems which can use a tutoring system, based on a human reasoning strategies, in order for humans to learn how to solve these problems themselves.

Bibliography

[1] Mathematics of the Rubik's Cube - Ruwix.com. Accessed: 17-2-2021. URL: https://ruwix.com/the-rubiks-cube/mathematics-of-the-rubiks-cube-permutation-group/.

[2] O. Pekonen. "Cubed: The Puzzle of Us All by Ernő Rubik". In: *The Mathematical Intelligencer* (Mar. 2021), pp. 1–2.

[3] Records - World Cube Association. Accessed: 8-3-2021. URL: https://www.worldcubeassociation.org/results/records.

[4] T. Rokicki et al. "The Diameter of the Rubik's Cube Group Is Twenty". In: *SIAM Rev.* 56 (2014), pp. 645–670.

[5] Richard E. Korf. "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases". In: AAAI'97/IAAI'97 (1997), pp. 700–705.

[6] F. Agostinelli et al. "Solving the Rubik's cube with deep reinforcement learning and search". In: *Nat Mach Intell 1* (2019), pp. 356–363.

[7] C. Johnson. "Solving the Rubik's Cube with Learned Guidance Functions". In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)* (2018), pp. 2082–2089.

[8] CFOP method - Speedsolving.com. Accessed: 10-3-2021. URL: https://www.speedsolving.com/wiki/index.php/CFOP_method.

[9] Riccardo Guidotti et al. "A Survey of Methods for Explaining Black Box Models". In: *ACM Comput. Surv.* 51.5 (Aug. 2018).

[10] Western Color Scheme - Speedsolving.com. Accessed: 2-4-2021. URL: https://www.speedsolving.com/wiki/index.php/Western_Color_Scheme.

[11] Erin McManus. "Mathematical Understandings Of A Rubik's Cube". In: *Honors Theses* (2018), p. 183.

[12] Notation - Rubiks.fandom.com. Accessed: 2-4-2021. URL: https://rubiks.fandom.com/wiki/Notation.

[13] Rubik's Cube Notation - Ruwix.com. Accessed: 2-4-2021. URL: https://ruwix.com/the-rubiks-cube/notation/.

[14] Rubik's Cube Move Notations - Rubiksplace.com. Accessed: 4-4-2021. URL: http://www.rubiksplace.com/move-notations/.

[15] How to solve the rubiks cube - Thatspeedcubergirl.home.blog. Accessed: 4-2-2021. URL: https://thatspeedcubergirl.home.blog/2019/08/06/how-to-solve-the-rubiks-cube.

[16] Rubik's Cube Algorithms - Ruwix.com. Accessed: 3-2-2021. URL: https://ruwix.com/the-rubiks-cube/algorithm/.

[17] How to solve the Rubik's Cube? - Ruwix.com. Accessed: 6-2-2021. URL: https://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/.

[18] Intermediate and Expert method - Ozcubegirl.com. Accessed: 6-2-2021. URL: http://ozcubegirl.com/rubikscubesolution.html.

[19] Rubik-cube 0.0.1 - Paul Glass. Accessed: 10-4-2021. URL: https://pypi.org/project/rubik-cube/.

[20] H. Nwana. "Intelligent tutoring systems: an overview". In: *Artificial Intelligence Review* 4 (2004), pp. 251–277.

[21] Tkinter - Python.org. Accessed: 20-4-2021. URL: https://docs.python.org/3/library/tkinter.html.

[22] Twisty puzzles - Ruwix.com. Accessed: 20-4-2021. URL: https://ruwix.com/twisty-puzzles/.

Appendix A

F2L Algorithms

Source:
ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/first-two-layers-f2l/


R U R'


F' U' F


U' F' U F


U R U' R'


(U R U' R') (U' F' U F)


(U' F' U F) (U R U' R')


(F' U F) (U' F' U F)


(R U R') (U' R U R')


(R U' R') (U R U' R')


(F' U' F) (U F' U' F)


(R U R' U') (R U R' U') (R U R')


(U F' U F) (U F' U2 F)


(U' R U' R') (U' R U2 R')


(R U' R' U) (d R' U' R)


(U F' U2 F) (U F' U2 F)


(R U' R') (d R' U R)


(U F' U' F) (d' F U F')


(U' R U R') (d R' U' R)


(F' U F U') (d' F U F')


(U' R U2 R') (U' R U2 R')


(U F' U' F) (U F' U2 F)


(U' R U' R' U) (R U R')


(U' R U R' U) (R U R')


(U F' U2 F U') (R U R')


(R U R' U') U' (R U R' U') (R U R')


(U' R U R') (U' R U2 R')


(U F' U F U') (F' U' F)


(U F' U' F U') (F' U' F)


(U' R U2 R' U) (F' U' F)


y' (R' U' R U) U (R' U' R U) (R' U' R)


(U2 R U R') (U R U' R')


(U2 F' U' F) (U' F' U F)


(U R U2 R') (U R U' R')


(U' F' U2 F) (U' F' U F)


(R U2 R') (U' R U R')


(F' U2 F) (U F' U' F)


(R U' R' d R' U2 R) (U R' U2 R)


(R U' R' U R U2 R') (U R U' R')


(R U' R' U' R U R') (U' R U2 R')

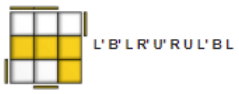

(R U R' U' R U' R') (U d R' U' R)


(R U' R' d R' U' R) (U' R' U' R)

Appendix B

OLL Algorithms

Source:
ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/orient-the-last-layer-oll/




R U B' I U I2' x' U' R' F R F'


y L' R2 B R' B L U2' L' B M'


(R U R' U) R' F R F' U2 R' F R F'


R' U2 F (R U R' U') y' R2 U2 x' R U


R' U' y L' U L' y' L F L' F R


F U R U' R' U R U' R' F'


L U' R' U L' U (R U R' U) R


L' U R U' L U R'


R' F R F' U2 R' F R y' R2 U2 R


R' U2 x R' U R U' y R' U' R' U R' F


M' U2 M U2 M' U M U2 M' U2 M


F (R U R' U) y' R' U2 (R' F R F')


R U' y R2 D R' U2 R D' R2 d R'


L' B' L U' R' U R U' R' U R L' B L


(R U R' U) R U' R' U R U2 R'


R' U2 (R U R' U) R


R' F' L F R F' L' F


R2 D R' U2 R D' R' U2 R'


R' F' L' F R F' L F


M' U' M U2' M' U' M


L' (R U R' U') L R' F R F'


L F R' F R F2 L'


F R' F' R U R U' R'


R' U' R y' x' R U' R' F R U R'


U' R U2' R' U' R U' R2 y' R' U' R U B


F (R U R' U') (R U R' U') F'


L F' L' F U2 L2 y' L F L' F


U' R' U2 (R U R' U) R2 y (R U R' U') F'


r U2 R' U' R U' r'


R' U2 l R U' R' U l' U2 R


F' L' U' L U L' U' L U F


R' F R' F' R2 U2 x' U' R U R'


R U R' y R' F R U' R' F' R


U2 r R2' U' R U' R' U2 R U' M


L U' y' R' U2' R' U R U' R U2 R d' L'


R2' U R' B' R U' R2' U l U l'


R U x' R U' R' U x U' R'


R' F R U R' F' R y L U' L'


R' F R F' U2 R2 y R' F' R F'


L' B' L U' R' U R L' B L


x' U' R U' R2' F x (R U R' U') R B2


U2 l' L2 U L' U L U2 L' U M


r' U2 (R U R' U) r


(R U R' U') x D' R' U R E'


L F' L' U' L F L' y' R' U R

Appendix C

PLL Algorithms

Source:
ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/permutate-the-last-layer-pll/

A 1:

x [(R' U R') D2] [(R U' R') D2] R2

A 2:

x' [(R U' R) D2] [(R' U R) D2] R2

U 1:

R2 U [R U R' U'] (R' U') (R' U R')

U 2:

[R U'] [R U] [R U] [R U'] R' U' R2

H:

M2 U M2 U2 M2 U M2

T:

[R U R' U'] [R' F] [R2 U' R'] U' [R U R' F']

J 1:

[R' U L'] [U2 R U' R' U2] [R L U']

J 2:

[R U R' F'] {[R U R' U'] [R' F] [R2 U' R'] U'}

R 1:

[L U2' L' U2'] [L F'] [L' U' L U] [L F] L2' U

R 2:

[R' U2 R U2] [R' F] [R U R' U'] [R' F'] R2 U'

V:

[R' U R' d'] [R' F'] [R2 U' R' U] [R' F R F]

G 1:

R2 u R' U R' U' R u' R2 [y' R' U R]

G 2:

[R' U' R] y R2 u R' U R U' R u' R2

G 4:

[R U R'] y' R2 u' R U' R' U R' u R2

Z:

M2 U M2 U M' U2 M2 U2 M' U2

N 1:

{(L U' R) U2 (L' U R')} {(L U' R) U2 (L' U R')} U

E:

X' (R U' R') D (R U R') u2 (R' U R) D (R' U' R)

G 3:

R2 u' R U' R U R' u R2 [y R U' R']

F:

[R' U2 R' d'] [R' F'] [R2 U' R' U] [R' F R U' F]

Y:

F R U' R' U' [R U R' F'] {[R U R' U'] [R' F R F']}

N 2:

{(R' U L') U2 (R U' L)} {(R' U L') U2 (R U' L)} U'