Benchmarking Human Solving Methods for Rubik's cube

Duis autem vel eum iruire dolor in hendrerit in vulputate velit esse molestie consequat,
vel illum dolore eu feugiat null

ANDREAS NILSSON ANIL9@KTH.SE
ANTON SPÅNG ASPANG@KTH.SE

Abstract

This is a skeleton for KTH theses. More documentation regarding the KTH thesis class file can be found in the package documentation.

Sammanfattning

Denna fil ger ett avhandlingsskelett. Mer information om LaTeX-mallen finns i dokumentationen till paketet.

Contents

0.1 Terminology

Cubie a miniature cube

Scramble perfoming a amount of random operations on a solved cube to reach a non-solved state.

Layer contains one side of the cube and one row of the four neighbouring sides

Operation Rotating one layer of the cube

Notation a character that is a abbreviation of the operation-name.

Chapter 1

Introduction

The Rubik's cube is an 3-D combination puzzle, where each side of the cube is covered with nine squares in six possible colours: white, red, blue, orange, green and yellow. It was invented by the professor of architecture Ernő Rubik as a teaching tool to help his students to understand 3D objects. It was not until he scrambled his new cube and tried to restore it, that he realize that his creation was a puzzle. Originally the Rubik's cube was called the Magic Cube and was licensed to be sold by the american toy company Ideal Toy Company in 1980. [6].

When solving the cube the idea is to start with a scrambled cube, meaning that the colored miniature cubes (cubies) are randomly positioned by executing random operations on the cube (fig 1.1a). The goal is to obtain the unique solution where each side of the cube are covered with only one colour per side (fig 1.1b). Different methods have been developed to solve subproblems one at the time with a series of operations to reach the unique solution. Many methods are based on the idea to solve it one layer at the time.

If you were to randomly rotate the faces in an attempt to solve the cube, there is almost zero chance of achieving the solved state in your lifetime, because of all the possible permutations of the cube. There are $4.3 * 10^{19}$ (or 43 quintillion) [9]

(a) Scrambled cube   (b) Solved cube

Figure 1.1

different states. Assuming you get to a unique state every second it would take more than 130 billion years to test 10% of the cubes possible states [appendix A].

There are two major ways to compete in solving the Rubik's cube: the least amount of moves and solving the cube as fast as possible (speedcubing).

1.1 Problem

This thesis will explore two commonly known beginner methods for human solving of Rubik's cubes to find the differences. Both methods are based on the idea to solve the cube one layer at the time, which is the easiest way for the human mind to solve this kind of problem [8]. We will evaluate the methods operation variance and the number of operations to reach a solved state, to find the most move-efficient of them both.

The problem was stated as follows:

Which of the beginner algorithms would be more effective for speedcubing?

Which of the beginner algorithms solves the cube with the least amount of moves?

Which beginner algorithm is easiest to learn and execute to someone inexperienced with the cube?

1.2 Purpose

The purpose of testing the methods is to find out which uses the least amount of moves and to find out which method suits speedcubing best by analyzing the usage of different operations. The reason the operations shows if the method is suitable for speedcubing is because some of the operations are more time-consuming than others. The inexperienced user will find this as a guideline as to which algorithm to start his/her journey towards solving the Rubik's cube.

1.3 Structure

The second section will introduce the reader to concepts necessary to understand the algorithms implemented and benchmarked. The third section will explain the methods used in this thesis. The fourth sections will explain the implementations made in detail and the difficulties.
The fifth section will be for presenting the results and the sixth section for discussion regarding the results. After that will there be a conclusions section that completes the circle of the thesis, answering the problem statements. Lastly the references

used to this thesis will be listed followed by appendix containing computations and graphs.

Chapter 2

Background

Providing the reader with knowledge of how the cube works and operations that you can perform on it as well as the algorithms that are in focus for this thesis.

2.1 Competitions

There are two types of competitions regarding the cube.

Speedcubing When competing in an official event regulated by the World Cube Association (WCA), the competitor has at maximum 15 seconds of inspection time of the cube before the solve begins.[10] The time stops when the competitor have reached the unique solution.

Fewest moves The competitors have 60 minutes without any inspection time and the competitor should also be able to hand in a written solution with the notations used in the correct format [10].

2.2 Rubik's Cube

Explanation of how the cube is constructed and the different notations for the operations.

Description The cube consists of 26 cubies with three,two or one visible sides depending on the type of cubie. There is one core piece consisting of three axes which holds the center pieces together [1]. The corner and edge pieces (fig 2.1) are the cubies that are movable to different edge and corner positions, the center pieces (fig 2.1) can only be moved according to the axis.

Notation The notation describes the different move operations on the cube. This thesis uses the notation used by WCA. Explained below:

Clockwise 90 degrees:

F - Front face

B - Back face

R - Right face

L - Left face

U - Upper face

D - Bottom face

To denote the anti-clockwise 90 degrees rotation just put a single citation mark (')
after the letter. For example F' - move front face anti-clockwise 90 degrees.[11] To
denote clockwise 180 degrees rotation just put (2) after the letters described above.

2.3 Algorithms

The algorithms are physically methods for solving the cube by hand but matches
the mathematical definition of an algorithm. Unlike many of the "near-optimal
solvers" (some of simulates several moves ahead to find the optimal move[7]), these
algorithms can with little effort be taught to humans.

Layer-by-layer using daisy method The layer-by-layer (LBL) algorithms di-
vides the cube into layers and makes it possible to solve the subproblems without
breaking any pieces already made. The daisy method is to solve the white cross by
first make a cross with white edges and a yellow center and then turn the white
edges, completing the white cross in the bottom [4].

1. White cross

The goal here is to achieve a white cross, so that the white center-piece is aligned
with its 4 white edge-pieces in the bottom. For it to be a completed step, the second

Figure 2.1: Cubie types

(a) The white cross done   (b) First layer complete

Figure 2.2

color of the white edge-pieces must also align with it's center-piece counterpart as
shown in fig 2.2a. This is done by first making a cross on the top with the yellow
center-piece and then flip the edges over to the bottom one at the time using an
operation, to make sure the edge-pieces on the vertical sides are aligned with the
center-pieces.

2. White corners

With the white cross done the next step is to complete the first layer by positioning
the white corner-pieces correct between the cross edges (fig 2.2b). This is achieved
with three different operation-combinations depending on the colour positions, all
focusing on the upper-front-left corner of the cube.

3. Middle layer edges

The next step is to solve the middle layer by moving down a cubie from the middle
of the third layer to the correct position on the second layer (fig 2.3a).

4. Yellow cross

With the second layer complete (fig 2.3b) it is time to work on the yellow cross.
This is achieved by applying one of two operation-combinations or both depending
on how many white pieces that are correct positioned (fig 2.4).

5. Yellow corners

Positioning the yellow corners by applying different operations to move the edges
depending och how many corners that are in position already.

6. Last layer permutation

Now when the corners of the last layer are in position so that the top is all yellow,
the only thing left to do is to positioning the pieces of the last layer so they match
with the colours of the vertical sides. This is achived by applying three different

operation-combinations depending on if its edges or corners that should switch places.

(a) Technique for each side (b) Middle layer complete

Figure 2.3

Figure 2.4: Different states to achive the yellow cross

Dedmore algorithm   The dedmore algorithm is also solving the cube layer-by-layer, but focusing on a corners-first solution[5]. It was one of the first solution guides to the Rubik's cube[3].

1. Top corners (the X)

Start off by finding a starting corner. For example aim to solve the blue top face first. Rotate the cube to obtain the blue side of the corner in top position. Then move the blue center-piece into position without moving your corner. When this is done, rotate the whole cube to obtain the corner in your top left on the front side, as in the fig 2.5a.

Next you'll search for the next corner that should be positioned in the top right front side. Depending on the position on this corner, you will use different operation sequences to get it in position like the fig 2.5b.

Simply rotate the cube and continue on. The goal of this step is to form an 'X' on the top face as in fig 2.5c.

2. Top edges

The goal of this step is to simply get the edges in place. Get them in place one by one. When this step is finished the cube looks like fig 2.6a.

3. Middle layer

The next step is to solve the middle layer by moving down a cubie from the middle of the third layer to the correct position on the second layer (fig 2.3a).

(a) First corner in position (b) Second corner in position (c) Top edges in position

Figure 2.5

4. Bottom corners

Here the cube is turned upside down and the goal here is again to build a (green) X with the corners(fig 2.6b).

5. Bottom edges

In this step, at least one edge is already in the correct place (the color might be switched as in fig 2.6c). Find that edge and put it on the front side. Then position every other edge correctly by using a series of operations.

When every edge are correctly positioned, there are 2 possible states left. Each requires a different sequence of operations to solve. The 'fish' pattern (fig 2.7a) and the 'H' pattern (fig 2.7b). Execute the sequence for the pattern and you'll then have solved the cube.
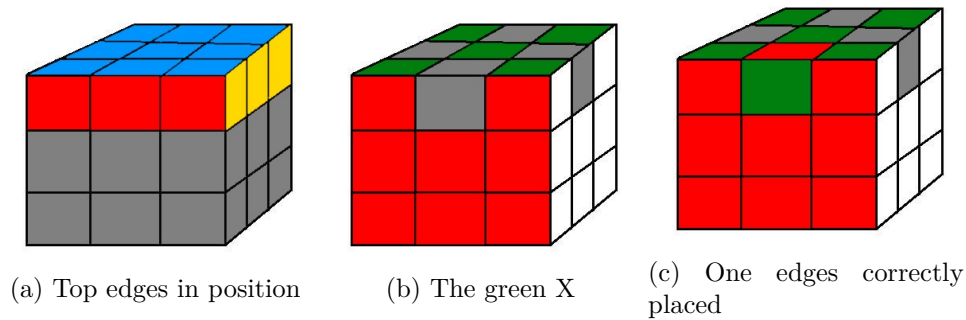

(a) Top edges in position   (b) The green X   (c) One edges correctly placed

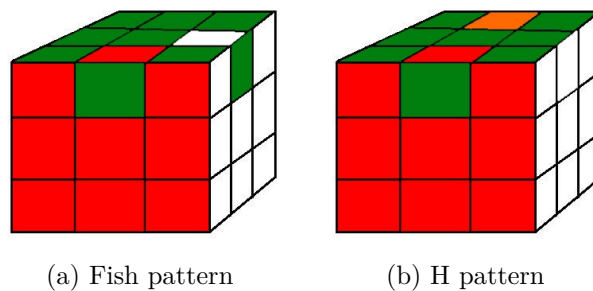Figure 2.6

(a) Fish pattern   (b) H pattern

Figure 2.7

Chapter 3

Method

For this thesis the two major methods used where implementation of methods and analyzis of the datarepresentation. Both of the algorithms will be implemented in the same language and with the same built-in functions and data structures to rule out unnecessary differences, to obtain a realistic comparison in the end. The algorithms needs to be executed a number of times to obtain a good set of data for the comparison. The data will be obtained during the executions by saving informations as number of moves used, statistics of moves used and number of times the method used least amount of moves to files. Based on the test data we will be able to evaluate each methods use of different operations, number of operations used and if suitable for speedcubing. The differences in used operations will provide information to evaluate if the times to solve are realistic for human solving. This depends on the usage of operations that are time-consuming for physical solving of the cube.

3.1 Cube representation

The cube is represented as six separate sides. Each side consists of nine positions each of which has a color (as in 3.1). There exists no relation between positions in different sides unlike the physical cube, where for example an edge has two colors. This means any correctly performed operation will need to move the colours to the correct side and location. The only way to keep the rules of the physical cube (correct colour-combination of a cubie) is to enforce them during the operations.

For example performing the 'F' operation (flip the front layer 90 degrees clockwise) on the cube shown in 3.1 would result in these internal assignments:

```
Side tempTop = new Side(top);
Side tempFront = new Side(front);
top.c7 = left.c9;
top.c8 = left.c6;
top.c9 = left.c3;
```

```
left.c3 = bot.c1;
left.c6 = bot.c2;
left.c9 = bot.c3;
bot.c1 = right.c7;
bot.c2 = right.c4;
bot.c3 = right.c1;
right.c1 = tempTop.c7;
right.c4 = tempTop.c8;
right.c7 = tempTop.c9;
front.c1 = tempFront.c7;
front.c2 = tempFront.c4;
front.c3 = tempFront.c1;
front.c4 = tempFront.c8;
front.c6 = tempFront.c2;
front.c7 = tempFront.c9;
front.c8 = tempFront.c6;
front.c9 = tempFront.c3;
```
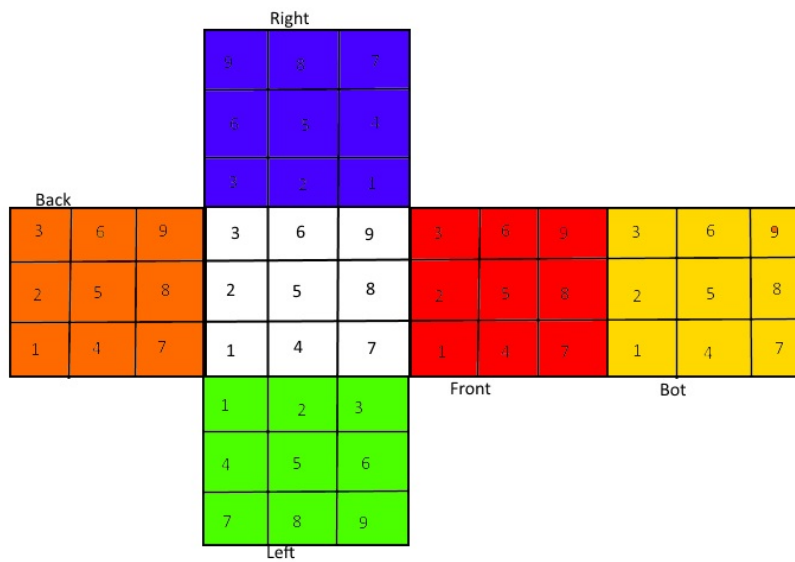

Figure 3.1: Side numberings

3.2 Scramble

The idea of the scrambler is to always start with a solved cube. The scramble are then simulated by making a large amount of operations in pseudo-random sequence (Algorithm 1). The scrambled cube is saved to a list awaiting the two solvers to pick and solve it.

Input: A cube, Number of operations to be made
Output: A scrambled cube
Let rand be a Random object.
for *Number of operations* **do**
    int op = rand.nextInt(Tot num of op)
    switch *op* **do**
        All operations have a number, performce the operation with the number op
    endsw
end

Algorithm 1: Scramble

Input: A cube
Output: A cube that has a subproblem solved
if *stepIsDone(cube)* **then**
    return cube
end
while *not(stepIsDone(cube))* **do**
    if *readyForOperations(cube)* **then**
        Perform the operations necessary to complete the step
    end
    else
        while *not(readyForOperations(cube))* **do**
            Operations to get the cube in a state to be able to perform the steps operations
        end
    end
end

Algorithm 2: Method steps general idea

Chapter 4

Results and Analyze

Containing visualisation of the data collected from the executions in form of graphs and description of what the graphs shows.

Fig 4.1 represents the sum of executions that used the same amount of moves, where the x-axis is amount of moves and the y-axis number of executions. As shown by the highest pillars for both methods, the Lbl with daisy method had a lower average (170) amount of moves than the dedmore method (188). Turning the whole cube in any direction was not counted for as a move.

Fig 4.2 represents a comparison of the number of executions the methods used least amount of moves, with the y-axis as number of executions. This shows that over ten thousand Scrambled cubes the Lbl with daisy method won 69% of the time, they got equal amount of moves 1% of the time and dedmore method had the fewest 30% of the time.

Fig 4.3 represents the avarage use of operations for both methods over all executions with the x-axis as the avarage number of times the operations where used. The data for both fig 4.2 and fig 4.3 can be seen in Appendix B.
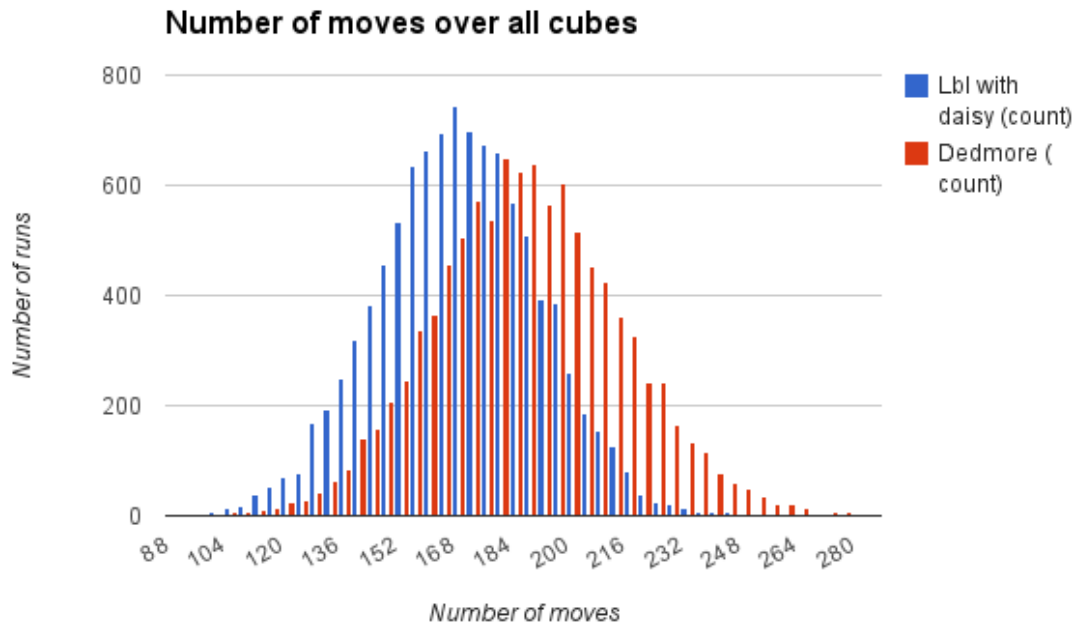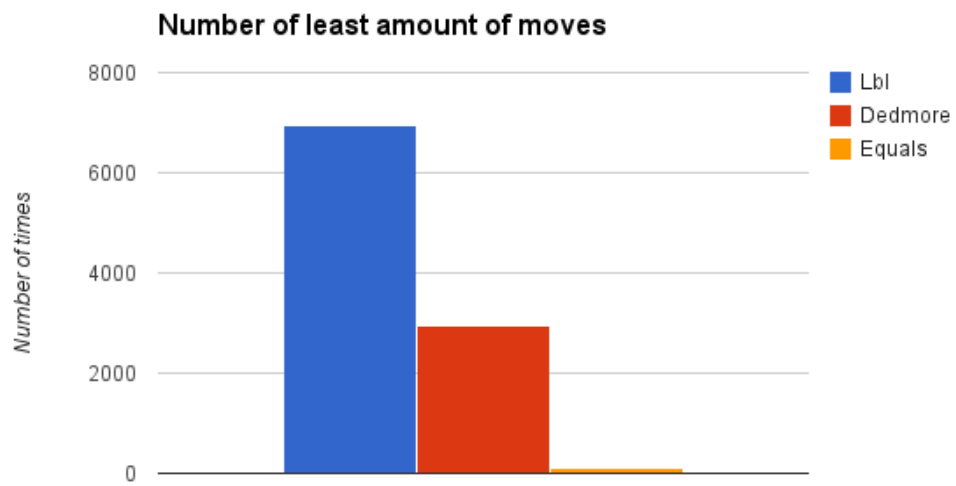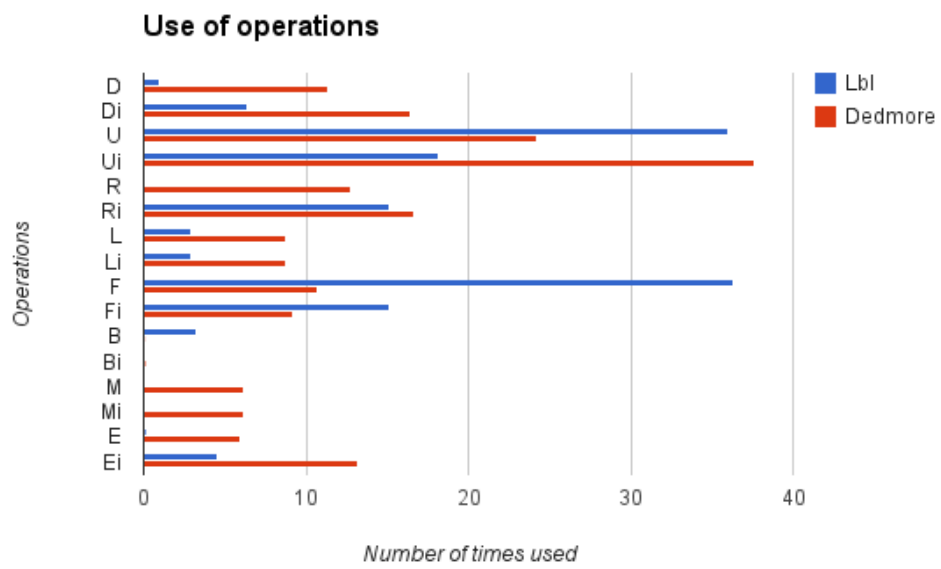
Figure 4.1

Figure 4.2

18

Figure 4.3: Avarage use of operations

Chapter 5

Discussion

5.1 Comparison

5.2 Difficulty

Our first idea was to represent the cube by 26 cubies in a layer-by-layer system, but this was problematic due to which colour-representation of the cubies should face which side when different operations where performed. A pro for this first idea was that the colors of a cubie had a relation which kept the colours to the correct cubie through all the operations, the only thing keeping the colours correct now is the operations implemented. But a pro with the representation now is that the operations only need to move colours around according to the different operations (as described in cube representation) and does not need to rotate individual cubies to get the colour in the right position.

Translating the methods for physical sovling of the cube to code was problematic with the intuative steps forsolving by hand, this made it difficult implement in a good way.

5.3 Errors

Chapter 6

Conclusion

[1]

References

[1] Advameg. How products are made, volume 7 - rubik's cube. `http://www.madehow.com/Volume-7/Rubik-s-Cube.html`, 2015. [Accessed 3 April 2015].

[2] Anon. Wolfram alpha. `http://www.wolframalpha.com/input/?i=4.3*10%5E18+seconds+to+years`, 2015. [Accessed 6 April 2015].

[3] Poonam Tyagi Ashutosh Tyagi. Gen-r: Genetic algorithm based model for rubik's cube solution generator. `http://www.ijsat.com/admin/download/[11-01-07-007].pdf`, September 2011. [Accessed 9 April 2015].

[4] Shelley Chang. How to solve the rubik's cube. `http://shellie.nfshost.com/cube/`, 2008. [Accessed 6 April 2015].

[5] Denny Dedmore. Rubik's cube solutions. `www.helm.lu/cube/solutions/rubikscube/`, March 1997. [Accessed 9 April 2015].

[6] Rubik's The home of Rubik's Cube. The history of the rubik's cube. `http://eu.rubiks.com/about/the-history-of-the-rubiks-cube`, 2015. [Accessed 6 April 2015].

[7] Herbert Kociemba. The two-phase-algorithm. `http://kociemba.org/twophase.htm`, 2014. [Accessed 6 April 2015].

[8] Lars Petrus. Solving rubik's cube for speed. `http://lar5.com/cube/`, May 1997. [Accessed 3 April 2015].

[9] Scott Vaughen. Counting the permutations of the rubik's cube. `http://faculty.mc3.edu/cvaughen/rubikscube/cube_counting.ppt`, no date. [Accessed 6 April 2015].

[10] WCA. Wca regulations article a: Speed cubing. `https://www.worldcubeassociation.org/regulations/#A3a1`, April 2014. [Accessed 3 April 2015].

[11] WCA. Wca regulations atricle 12: Notation. `https://www.worldcubeassociation.org/regulations/#article-12-notation`, April 2014. [Accessed 3 April 2015].

Appendix A

Calculation of randomly rotate faces

Assumption: Every second you get to a new state.
There are $4.3 * 10^{19}$ different states (rounded down) [9]. 10% of the different states:
$4.3 * 10^{1}9 * 0.1 = 4.3 * 10^{18}$
4.31018 seconds to years = 136.4109 years [2]

Appendix B

Data

Avarage use of operations.

Operations	Lbl	Dedmore
D	1.0365	11.3665
E	0.1103	6.0217
F	36.4211	10.7248
B	3.2228	0.0718
Ri	15.1558	16.7828
L	2.8949	8.8148
M	0.000	6.1479
Ui	18.122	37.7467
U	36.0845	24.2097
Bi	3.2228	0.0955
Mi	0.000	6.1479
Li	2.8949	8.7691
R	25.2278	12.7341
Ei	4.5369	13.2933
Fi	15.1163	9.2282
Di	6.3726	16.5212

Number of times the algorithms had least amount of moves.

Lbl	Dedmore	Equals
6935	2960	105