An Empirical Study of X and X Rubik's Cube Solving Algorithms

Noah Greski
nj06gres@siena.edu

April 29, 2024

Abstract

In this study, I present and analyze timings and operation counts for two rubik's cube solving algorithms. These algorithms include X1 and X2. These algorithms are programmed for 2x2 cubes rather than a more standard 3x3 cube. This was done to simplify the process of this analysis and remain in the scope of this project. Both solving algorithms and cube scrambling methods are done within several Java programs.

1 Introduction

This study focuses on two rubik's cube solving algorithms in a few different ways. The computing environment section gives information about external factors that may affect the timing results of these algorithms. Both algorithms have different techniques for scrambling and solving these cubes. Testing will be done to compare which algorithm is more efficient overall and why that is. The testing will calculate the number of moves done to solve the cube and the elapsed time for the program to execute the solving algorithm. Tests, results and conclusions will be discussed on why one is more efficient than the other or vice versa.

2 Computing Environment

For this study, a Java program is used to run these two algorithms. There are no inputs by the user, cube scrambling methods and solving algorithms are done by the Java program. The elapsed time, number of moves, and a simple visual representation of the cube on several steps of the solving process are output by the program.

All runs are done using the following Java version:

```
openjdk version "17.0.8" 2023-07-18
OpenJDK Runtime Environment Temurin-17.0.8+7 (build 17.0.8+7)
OpenJDK 64-Bit Server VM Temurin-17.0.8+7 (build 17.0.8+7, mixed
mode, sharing)
```

The computing environment is a Dell PC with a Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz running on Windows 10 Enterprise. The L2 Cache size is 2048 KB and the L3 Cache size is 16384 KB.

3 X1

X1 is a Rubik's Cube solving algorithm that was created by Nabil-1999 and can be found on Github at https://github.com/Nabil-1999/rubik-s-cube. This algorithm was implemented into my own Java program and can be found under the "rubiksSolver1" folder in my repository. I have slightly altered this program for this empirical study, changes can be found in the code paired with "//added by Noah" comment.

X1 is a complex algorithm and is not worth adding the code to this study. The X1 solving method solves the cube layer by layer. First by solving the bottom layer, middle then top layer. Different algorithms are used to solve each layer, combining into one algorithm to solve the cube. First corner placement, edge placement and edge pairing are done on the first two layers. Then OLL (Orientation of Last Layer) and PBL (Permutation of Last Layer). These orient the pieces of the top layer and place the pieces of the top layer into the correct positions to solve the cube.

3.1 Tests

To test the X1 rubik's cube solving algorithm, the program was run 20 different times with three different amounts of cubes being solved. The program loops through solving one rubik's cube, for testing purposes I did a loop with 10, 100 and 1000 solves for the cubes, 20 times each. 60 total program executions. The reason for increasing the amount of cubes being solved per execution was to decrease the weight of the outliers on the data. The average number of moves and elapsed time for each program execution were output.

3.2 Expectations

There are a few expectations I have for both of these two Rubik's cube solving algorithms. The X1 algorithm solves layer by layer using several different algorithms. My

expectation is that this allows for X1 to be more efficient than X2. As X2 uses the same algorithm over and over to solve the cube. X1 being more efficient meaning its elapsed time and numbers of moves will be overall less than X2. On the other hand, X2, I predict will have less variance in the results, as it always uses the same algorithm. X1 will have more variance in the number of moves because it can skip certain steps when it sees different patterns using different algorithms, but sometimes cannot skip these steps, resulting in a worse number of moves. The way I solve a Rubik's cube is layer by layer like in the X1 algorithm. When solving I am able to skip steps depending on the original scramble of the cube. The X2 algorithm has no steps, the same moves are repeated over and over. Overall, I predict X1 will be more efficient than X2.

3.3 Results

The raw results for tests can be found in the files X1_10, X1_100, X1_1000, in the repository. The format is `(number of cubes) (number of moves) (elapsed time)`. The number in the file names indicate the number of cubes being solved per execution. The first number is the number of cubes solved per execution. The second number is the average number of moves made to solve the X number of cubes. The third number is the average elapsed time of solving each cube, in milliseconds.

Below will display graphs and charts to analyze the results

3.4 Discussion

4 X2

X2 is a Rubik's Cube solving algorithm that was created by ljz112 and can be found on Github at https://github.com/ljz112/rubiks-cube. This algorithm was implemented into my own Java program and can be found under the "rubiksSolver2" folder in my repository. I have slightly altered this program for this empirical study, changes can be found in the code paired with "//added by Noah" comment.

X2 is a complex algorithm and is not worth adding the code to this study. The X2 solving method is known as the Old Pochmann Method. This method is mainly used for blindfolded solves. It uses one specific algorithm to swap corners until they are all in the correct position. In the case of a 2x2 cube, this solves the cube, all pieces on the cube are corners.

4.1 Tests

The exact same tests were done on X2 that were done on X1. A loop was added to the X2 code to have the same functionality as X1. To test the X2 rubik's cube solving algorithm, the program was run 20 different times with three different amounts of cubes being solved. The program loops through solving one rubik's cube, for testing purposes I did a loop with 10, 100 and 1000 solves for the cubes, 20 times each. 60 total program executions. The reason for increasing the amount of cubes being solved per execution was to decrease the weight of the outliers on the data. The average number of moves and elapsed time for each program execution were output.

4.2 Expectations

The expectations from the 3.2 section for X1 are the same for this section. Refer to section 3.2 for this. Both algorithms were discussed in that section.

4.3 Results

The raw results for tests can be found in the files X2_10, X2_100, X2_1000, in the repository. The format is (number of cubes) (number of moves) (elapsed time). The number in the file names indicate the number of cubes being solved per execution. The first number is the number of cubes solved per execution. The second number is the average number of moves made to solve the X number of cubes. The third number is the average elapsed time of solving each cube, in milliseconds.

Below will display graphs and charts to analyze the results

4.4 Discussion

6 Conclusions