Department of Computer Science

Submitted in part fulfilment for the degree of BEng

Investigating Rubik's Cube Solvers

Lloyd Banner

2021

Supervisor: Steven Wright

ACKNOWLEDGEMENTS

I would like to thank Steven Wright who has been a supportive and helpful supervisor throughout this project.

STATEMENT OF ETHICS

No legal, social, ethical, professional or commercial issues were identified for this project. All information from external sources that have been used have been referenced.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Executive summary

This project aims to find the quickest software solving method for the Rubik's Cube. Software methods for solving the cube have been implemented in the past but not many have had the goal of solving the cube quickly. Trying different methods will give a comparison of solve times and may show a pattern in which types of methods are best for solving the cube in software. Without the limitations of the physical world, a software solver relies on the software implementation of the cube and the solving algorithms it uses to solve the cube quickly. This allows Rubik's cube methods to be compared without other factors such as motor skills of the solver needing to be considered. This means a direct comparison of methods instead of a comparison between the people using them.

In this project, the Beginner's Method, Corners First method and Roux method have been compared. The Beginner's method was the previous fastest method for solving the cube in software implementation so provides a good baseline for comparison with the other methods. The Corners First Method seemed like it could be a good solving method as it was one of the first solving methods developed suggesting some simplicity like the Beginner's Method. The Roux method is a speedcubing method that prioritises solving in a low number of moves, which seemed like it could be more promising than other methods due to a lower number of algorithms and moves being needed for a solve.

These methods were compared with different configurations using the same implementation of the Rubik's Cube and its moves. It was found that in all cases the Corners First method was the slowest solving method with its quickest average being 0.631ms to solve. Then the Roux and Beginner's Methods performed better depending on their configuration, but the Beginner's Method had the quickest average solve time out of any implementation when parallelised. The quickest average for the Beginner's method was 0.431ms to solve and the quickest was 0.510ms for the Roux method. It has been concluded that the parallelised Beginner's method was the quickest but that without parallelisation the Roux method is likely slightly better. All of these times are better than the previous quickest average of 39.1ms [1] so the goal of the project has been achieved.

To improve on this further different methods to solve the cube may need to be considered or the implementations of the solvers could be

rewritten or parallelised further. Rewriting the solvers should probably prioritise some move reduction as this was found to reduce solve time during the project. This was mostly done by optimising loops and finding ways to prevent anomalous results and will likely only be beneficial to a certain extent. Looking for other methods will likely be the best approach as many methods have yet to be tried with a software implementation.

1 Introduction

Since the Rubik's Cube initially branded the "Magic Cube" was released in 1977 by Erno Rubik, many people have investigated this 3x3x3 cube. Initially, each face is a different colour but the cube is made up of 26 smaller cubes, known as cubies, and a mechanical centre piece. This centre piece allows the rotation of 3x3x1 sections of the cube. After rotations the cube can be scrambled into 43,252,003,274,489,856,000 different possible states [2]. As a result of the complexity of the problem of restoring a Rubik's Cube into its initial state, it grew massively in popularity as people attempted to solve the Rubik's cube. Since its creation, many have tried to find the quickest methods of solving a Rubik's Cube, as a result, methods known as speedcubing developed which allow speedcubers to solve the Cube quickly with Yusheng Du holding the record of 3.47 seconds [3]. Many robots have also been designed, the quickest able to solve the cube in under 0.38 seconds [4].

The goal of the project is to find the quickest method to find a solution given a purely software implementation. This will remove much of the limitations of the physical movement time. Instead, the limiting factor will be the speed of the software implementation of the solve, this includes the speed to execute a movement algorithm and the time taken for the software to determine the next move given each state it is in. It appears that the component that will take the most computational time is determining the next move at any given state. There will likely be a trade-off between the number of algorithms that are used in a given solve and the number of moves needed to solve the cube.

To find a quick method the current quickest method to solve the cube in software will be implemented. Then other promising methods will be implemented that have the potential to be quicker than this method. This will allow for easy comparison between the times these methods take on one machine and in the same environment. After finding the quickest methods, the best methods can then be attempted to be improved further and maybe even parallelised. This may give more of an insight into what methods and implementations cause a solve to be fast.

2 Literature Review

2.1 The Rubik's Cube

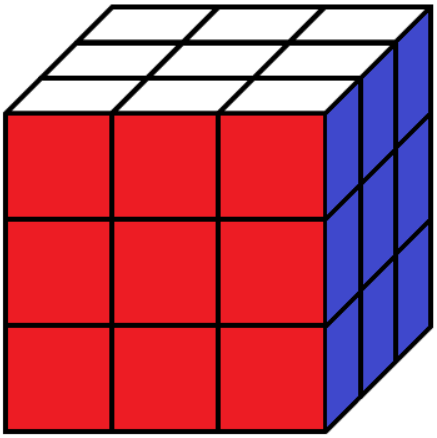

***Figure 2.1**: Solved Rubik's Cube [16]*

The Cube is made up of 12 edge cubies, 8 corner cubies, 6 centre cubies and the rotational component in the middle. Movements are made in 90 degree turns of a 3x3x1 section of the cube. This section can be from the middle or sides of the cube on any face and in any orientation. A cube can be seen above in **Figure 2.1**. There is a common notation for moves which can be seen below in **Table 2.1** and **Figure 2.2**. A letter denotes a move and the prime of that letter is a move in the opposite direction. This notation is centred on a side of the cube that is facing the solver. The notation is listed in the table below.

Notation	Movement	Prime	Prime Movement
R	Right side up	R'	Right side down
L	Left side down	L'	Left side up
F	Front face clockwise	F'	Front face anticlockwise
B	Back face anticlockwise	B'	Back face clockwise
U	Top side left	U'	Top side right
D	Bottom side right	D'	Bottom side left

***Table 2.1**: Rubik's Cube Notation*

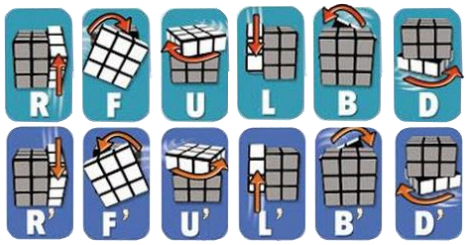

***Figure 2.2**: Rubik's Cube Notation [8]*

With $\sim 43 \times 10^{18}$ possible configurations solving the Rubik's Cube is undoubtedly a complex problem. Having many movement options in each state makes this problem particularly difficult. In fact, it was defined as NP-Complete in 2018 [5], which means it needs at least a non-deterministic Turing Machine to compute and so is a reasonably difficult problem. However, since God's number is 20 [6], the difficulty of a Rubik's cube plateaus after around 20 moves in a scramble. Since the worst case, optimal solve for a Rubik's Cube takes 20 moves, doing more than 20 moves in a scramble should not make the cube any harder to solve.

As the size of the Rubik's Cube changes to an NxNxN cube from the original 3x3x3 the complexity also changes. The Corners always have a similar number of possible states as these can only be swapped with each other and there are always 8 corners. Fraser suggests this gives a possible $(8!)3^7$ States for the corners on an odd cube and $(7!)3^6$ on an even cube [7]. The change for the even cubes is as there is no central middle point so more symmetry is possible from different sides. Fraser has defined a formula to calculate the number of positions using this and the number of edge cubies based on the size of the cube. As a result, for each increase in cube size after 3x3x3 the possible states of the cube increase by over 30 orders of magnitude. This is a substantial increase in complexity but some solving methods should be transferable due to the corners remaining the same.

2.2 Methods

There are many methods and algorithms for solving Rubik's Cubes. These vary from the Beginner's Method [8], the easiest method that the Rubik's Website recommends when first starting, to speedcubing methods like CFOP or ZZ [9]. Generally, speedsolving methods require the solver to know many more movement algorithms and remember more patterns, however, this can allow these methods to enable the solver to find a solution to the cube far quicker by hand. This may not be the case with a software solver though as Svyetov found that on average their implementation of the Beginner's Method was far quicker than the CFOP speedcubing method [1]. On average it took 39.1ms for the beginner's method and 95.5ms for the CFOP method in this implementation. A big difference between software solving and solving by hand is that in competition a solver is allowed to look at the cube for up to 15 seconds before the solve begins to be timed. This means that a solver can begin solving the cube in their head before they start to be timed, whereas a software solver is timed as soon as it begins to inspect the cube. As a result, simpler methods may be put at a disadvantage for speedcubers as there are fewer possible moves to consider in the 15 seconds so this time will not be used effectively. Also, some methods may make the physical movement of the cube easier which is not an issue for software that

will do each move near instantly. It appears the most effective methods for humans may not be the best for a software solver.

Comparing the Beginner's Method and CFOP it appears that, the Beginner's Method requires about 10 movement algorithms to be known and takes around 140 moves [1] and CFOP takes around 55 moves and requires at least 78 algorithms to be known [10]. Assuming that every move considers all algorithms, take ***a*** as the algorithms considered, ***m*** as the average number of moves, and ***t*** as the maximum time to consider an algorithm given the current state of the cube. Then this gives us O(***mnt***) for the solve time. This means ***m***, ***n*** and ***t*** need to be minimised to arrive at the quickest solve time. There is little information on t, but almost every Rubik's cube algorithm online has a recorded number of algorithms to know and an average number of moves to solve. If there are fewer things to consider for each move ***t*** will be lower, so it is also worth considering methods that do not require as much inspection. With this information, it is possible to start to consider methods that might minimise solve time.

Looking at just minimising the number of moves this takes us to God's Number [6] and Korf's algorithm [2]. God's Number is the worst case minimum moves that any Rubik's Cube can be solved in and was proved to be 20 in 2010 using 35 CPU-years of computation donated by Google. This meant that every possible state of the $\sim 43 * 10^{18}$ positions were solved in less than 20 moves. A brute force approach was used to test every possible move until a method with under 20 moves was found. As a result, there is not a definitive God's number algorithm, however, Korf used Iterative Deepening A* to try to find optimal solves in 1997. He approximates this method to take 4 weeks to solve a single cube of depth 18 for a scramble with 42 megabytes of memory and suggests the speed will increase linearly with memory. Even with quite a high end computer today with 32 gigabytes of memory, which would be around 762 (3s.f) times quicker assuming this linear improvement, it would likely still take close to an hour to solve. This time would increase considerably with depth 19 and 20 which would be required in some cases as God's Number is 20. This was parallelised and still took a minimum of 36.7 seconds to solve on average [11]. Just minimising moves will likely not provide a quick enough solution.

On the other end of the spectrum, minimising the number of movement algorithms required can be considered. A single movement algorithm that sequences through every possible state of the Rubik's Cube is called the Devil's Algorithm [12] and the number of moves that this requires to solve is the Devil's Number. Although this would only require a single algorithm, that algorithm would have to sequence through every single possible state of the Rubik's Cube. This means

that it could take ***x*** moves where $34 \times 10^{15} < x \leq 43 \times 10^{18}$ in Mike's estimate [12]. A method that requires such a large number of moves will not provide a quick solve time and no Devil's algorithm has been found yet. Aside from the Devil's algorithm, there is a common well known method that can solve a Rubik's Cube with a very low number of algorithms. This method is called Phillip Marshall's method or "The Ultimate Solution" [13] and requires the solver to know 2 algorithms with an average of 65 moves to solve. This method depends heavily on inspection of the cube, however, so, ***t***, the time to analyse an algorithm in the current state of the cube will likely be very high. This method may be worth considering but other options should probably be explored first.

Looking at methods that both minimise moves and movement algorithms at the same time Speedcubing methods may be worth looking at. The Roux Method seemed to have the lowest maximum number of algorithms and the lowest average moves other than some methods that expected intuitive user movements [9] which may be difficult to implement in software. With 42 algorithms and 48 moves on average, this could potentially be another good method. Another potential method is ZZ. This requires 40 algorithms, if a version of the method with fewer algorithms is used, and takes around 53 moves.

Since the Beginner's Method has been the quickest in the past other simple or beginner methods may be worth considering. The simplicity of the method may have allowed the Beginner's Method to be quickest so far. If there is less to consider or a definitive next move at each stage that requires less intuition and knowledge this could reduce ***t*** considerably and provide a quick method. The first ever solution proposed was by Erno Rubik himself and was the Corners First Method [9]. There are many variations of the Corners First Method, but some require few algorithms and most require less than 100 moves on average. Although there is a higher number of moves and algorithms than other methods mentioned, the simplicity of this method and the simple approach may make it a quicker solving method. In addition, since this was the first method used to solve the Rubik's cube, the Corners First Method may be more intuitive to implement and could be a more understandable method than some of the others listed. Many other beginner methods require the solver to work out a lot more for themselves, so they will likely be harder to implement and require a larger ***t*** than the Corners First and Rubik's website Beginner's Methods.

3 Implementation

C++ has been used for the project to implement the cube and methods. This language was chosen as it is a higher performance language with the ability to manage resource allocation if needed. This could be useful to speed up the solving of the cube. Also, C++ can use classes and OOP which may be useful for abstraction and making the solutions easier to break down and understand, however, this could introduce some overhead if overused. C++ also allows for parallel programming which may be significant in reducing solve time.

Git and Github have been used for version control of the project to allow code to easily be reverted to previous states and so that the code can be easily accessed when needed. Visual Studio was chosen as an IDE as this has autocomplete features and good class management with some git integration when the right plugins are installed.

3.1 The Cube and its Moves

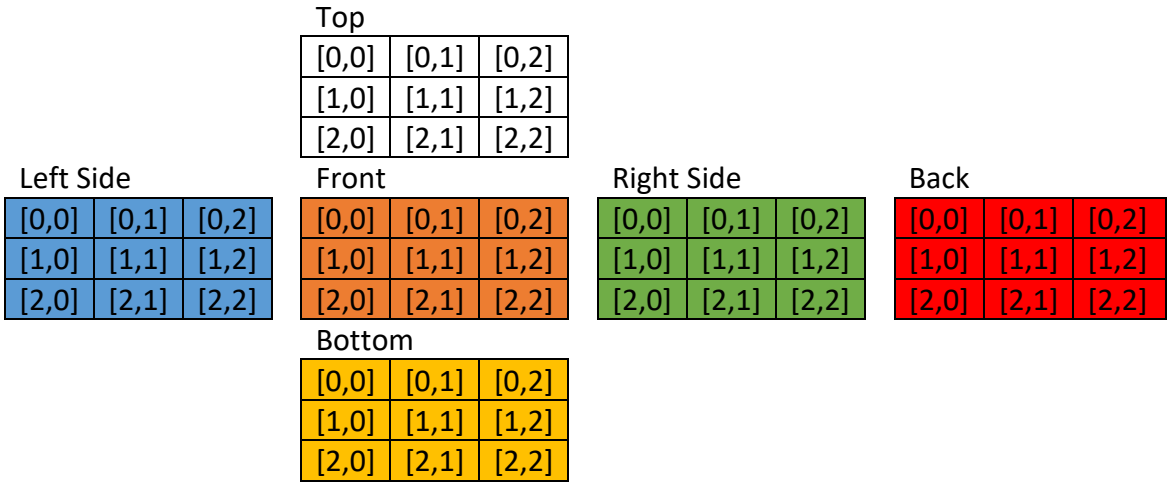

***Figure 3.1**: Rubik's Cube Representation*

To begin with, the different sides of the cube were implemented as arrays as seen in **Figure 3.1**. To do this each position was indexed by its height and width from 0 to 2. In each of these positions was a letter to represent the colour of the face which was the first letter of the colour of that face. Using the name of each face, positions can be accessed using functions of the cube object. A UML diagram of how the cube has been represented can be seen in **Figure 3.2**.

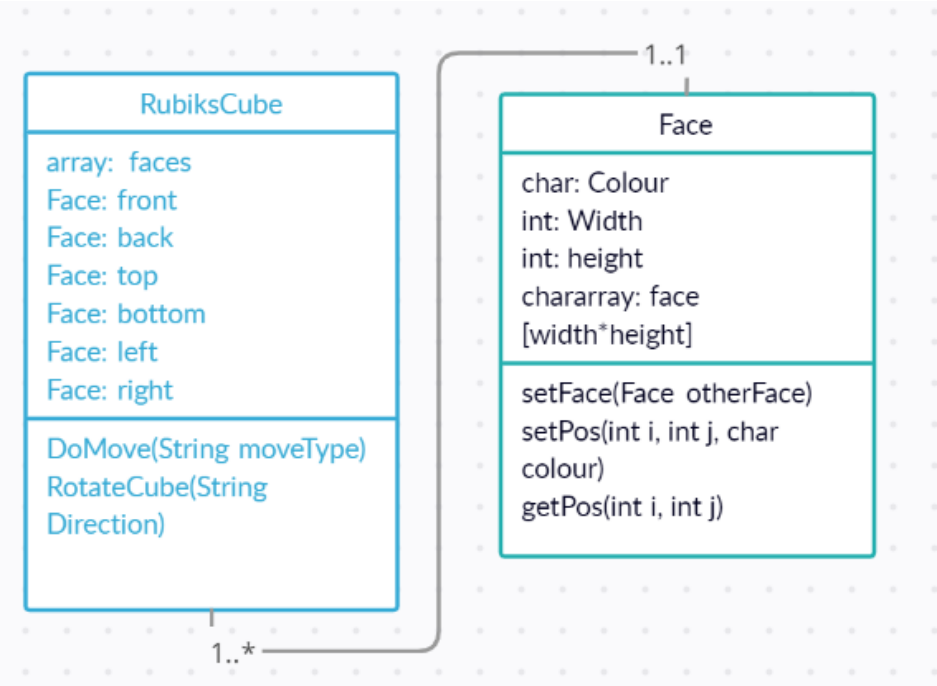

***Figure 3.2**: UML Diagram for Cube Class*

After the cube had been represented it was important to be able to rotate the faces of the cube so that it could function. This required implementation of translation functions that replicated each move from Rubik's Cube notation. This allowed all 3x3x1 rotations of the cube. For each move, four 3x3x1 strips had to be rotated to faces 90 degrees from their original positions and another face perpendicular to the 3x3x1 strips had to be entirely rotated. This required parts of arrays to be copied into adjacent arrays and other arrays to have letters within them repositioned. The option to specify the number of times to do a move was also added, which may make the function more powerful. With these implemented all possible moves of the cube can be done and it may be easier to translate solving methods to code when using the same notation as is commonly used. An example of the implementation of a move can be seen in **Figure 3.3**.

```
void RubiksCube::doR(int num) {
    for (int n = 0; n < num; n++) {
        //Copying 3x3x1 sections on to the correct face and right positions
        for (int i = 0; i < height; i++) {
            temp.setPos(i, width - 1, front.getPos(i, width - 1));
            front.setPos(i, width - 1, bottom.getPos(i, width - 1));
            bottom.setPos(i, width - 1, back.getPos(height - 1 - i, 0));
            back.setPos(height - 1 - i, 0, top.getPos(i, width - 1));
            top.setPos(i, width - 1, temp.getPos(i, width - 1));
        }
        //Roating right side face
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                temp.setPos(j, width - 1 - i, rightSide.getPos(i, j));
            }
        }
        rightSide.setFace(temp.getFace());

        //Counting and listing moves
        moves.append("R, ");
        numMoves += 1;
    }
}
```

***Figure 3.3**: Example of a move*

In addition to the moves, the ability to rotate the whole cube right or up any number of times was added. This will allow the cube to be viewed from different angles and may make some things easier. This just required whole arrays to be copied across or rotated and can also have the number of rotations specified like each move.

3.2 Scrambling the Cube

Due to God's Number [6] there should be no need to scramble the cube any more than twenty moves as after this the complexity of the solve should not increase. As a result, when scrambling it will be by twenty moves generally. To implement the scramble a class was created that takes a cube and then allows functions to be applied to the cube. One of these functions is a random scramble. For each move a random number between 0 and 11 is selected each corresponding to one of the moves in the Rubik's Cube notation. The number of moves to randomly select is an input to the function. The cube can then be taken from the scrambler in its scrambled state.

These scrambled cubes will need to be used to test different solving methods. The randomisation is seeded based on the time of the execution so should be different every time which will allow each solving method to be sufficiently tested on multiple different scrambles. There is the possibility that a move is done and then directly after the prime of that move is done during the scramble. This would effectively reduce the number of moves in the scramble. There is only a 1/144

chance of this happening, with an even lower chance of this happening multiple times in a row, and if enough tests are done anomalous data will be identifiable so this is acceptable.

3.3 Solving Using the Beginner's Method

To start with the Beginner's Method on the Rubik's website [8] requires the white cross to be solved. This was expanded so it begins by solving the top face rather than just a white face as this may make parallelisation easier later on. Other than this the site says that this section should be done by "practice and by trial and error" [8] which is much more difficult to program than to do yourself. Instead, a hardcoded movement for each edge piece was defined if that piece contained a square with the top colour. This creates the top cross.

After the white cross, all later steps had defined algorithms. These were programmed based on sides rather than colours like the first step. These algorithms provided a series of moves to perform to get the cube into the correct state. The algorithms consisted of a list of moves from the Rubik's notation. To perform the algorithms the cube or faces of the cube needed to be rotated until there was a move that would help to achieve the goal of the current step. This was repeated until the goal of the step was achieved. When a step was completed the next step would be performed until all steps were completed. An example of how each step of the method was performed can be seen in **Figure 3.4**.

```
1 while stepCompleted == False
2     count++
3     if algorithmCanBePerformed
4         doAlgorithm
5     if count > threshold
6         performLoopBreakingAlgorithm
7
8     stepCompleted = checkIfStepIsCompleted
```

***Figure 3.4**: Pseudocode Example of a Step*

Generally copying the steps from the method was enough, but there were a few issues. Initially, on some edge cases, some steps would get stuck in loops as the step was never completed. This was usually due to cubies being positioned in ways where the algorithm would never reach them due to the way the method had been programmed. Loop breaking algorithms were performed periodically to prevent this.

3.4 Solving Using the Corners First Method

There are many different Corners First Methods for Rubik's Cube solving, a simpler method was used to follow the intuition that simpler methods with less inspection may be quicker. As a result, a beginner version of the Corners First Method is being used [14].

To begin with, this method required the bottom corners to be solved intuitively. Instead, the same algorithm was used as for the second step of the Beginner's Method. This solved the top corners so after completing the step the cube was flipped upside down. This may mean that the method is not as quick as it could be if the corners were solved on their own, but for now, this will be good enough.

After this, a similar procedure to how the Beginner's Method was solved was used. Algorithms and states to solve the cube from using the algorithms were provided. This meant that the cube had to be moved in a way that did not break the solved sections, but would cycle through all the cubies that could have pieces that would need to be moved by the algorithm.

A few steps would get stuck in loops again so the same loop counting method was used to avoid this in most cases. Occasionally, there was an issue where cubies that should have been solved on a step were scrambled again. This was due to the wrong faces of the cube being moved when cycling through states to apply the algorithms and only a minor change was needed.

This method introduced some new moves and move notation for the middle layers of the cube. These moves were M, M' and E. Using the previously defined moves these were emulated, but the performance of the solve could be impacted by this. Since the movement time is only a fraction of the solve compared to the inspection time this probably will not have too large an impact. If the method proves to be quick optimisations of these moves can be made later. How these moves have been implemented can be seen in **Figure 3.5** and **3.6**.

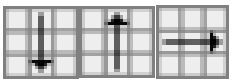

***Figure 3.5**: M, M' and E from left to right [14]*

```
//M
cubeToSolve.doLPrime(1);
cubeToSolve.doR(1);
cubeToSolve.rotateCubeUp(3);
//end M
```

***Figure 3.6**: M Implemented with other Rubik's Moves*

3.5 Solving Using the Roux Method

Gilles Roux' Roux tutorial [15] will be used as this seemed to be one of the better documented Roux methods with clearer steps and Gilles is the creator of this method. Other versions generally require more algorithms which will add complexity. Like the Corner's First Method middle sections are moved. Moves S and lower r are listed in addition to the moves from the Corners First Method. r is R and then M' and S is a middle rotation to the right. These will be implemented with the previously defined moves for now as was done in the last Method. Roux moves can be seen in **Figure 3.7**.

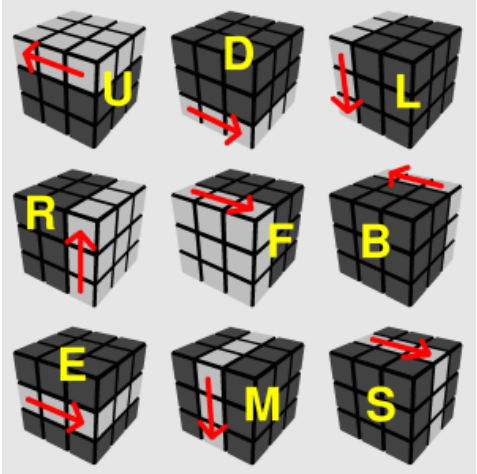

***Figure 3.7**: Moves required for Roux Method [15]*

It is worth noting that, Gilles has made the first step of this method to utilise the 15 seconds at the start of a speedsolving competition efficiently and to make the cube easier to view for a human. The first step positions the six cubies in the bottom left so a human solver can see all the rest of the unsolved pieces at once. A person is expected to work out how to position these pieces intuitively in the 15 seconds before they start solving. This means that a software solver may be put at a disadvantage with some parts of this method.

For the first step, more intuitive movements are expected to create a 3x2 block in the bottom left. Instead, hard coded moves from the top cross from the Beginner's Method have been used and then two adjacent corners have been solved using part of the top corners step from the Beginner's Method. Some extra moves have been stripped out of the corners section, but the full top cross is still solved. This is not much of a sacrifice since most of the same cubies would still have been checked for the cross so only a few extra moves than necessary have been added, and little extra inspection should be required. If the Roux Method proves to be reasonably quick the edges can be improved later. The cube is then rotated until these pieces are on the left hand side.

The Roux method uses a lot of hardcoded sequences of moves from many different positions if you use a more complex version. This has been limited to allow a more algorithmic approach making the method easier to program and meaning that fewer movement algorithms are needed. To do this, methods to move the cube into positions where a particular algorithm could be used were performed. This effectively allowed possible pieces needed for the next step to sequence through a few positions that the chosen algorithm could be performed from. This is a very similar method to **Figure 3.4** if you consider the sequencing moves the loop breaking algorithm.

3.6 Adding Middle Movements

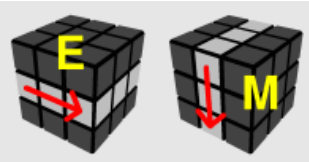

***Figure 3.8**: Moves to Be Added [15]*

The moves shown in **Figure 3.8** and their primes were added since they could improve the solve time of both the Corners First Method and the Roux Method. It was found that the Roux was a relatively quick software solving method so adding these moves seemed worthwhile. S will not be added as I have not used it in my implementation of the Roux method already. After consideration adding r will not provide an improvement on doing R and then M' as the exact same pieces are moved. In addition to adding the above moves, rotating the cube down and left were also implemented. This removed the need to do multiple of a different rotation instead of just rotating the way you wanted to. For example, before you would have to rotate the cube right three times instead of rotating the cube left once. This will increase the computational time since three times as much work is done. This may not be too much of an improvement as no less inspection is done but it seemed worthwhile to try to reduce the moves needed.

After these moves were added new solving classes were created where the previous use of multiple moves to do one of these moves was replaced with the new move. This will allow a time comparison between the use of these new moves and using other moves to simulate them.

3.7 Reducing Moves Further

Even after the implementation of the middle movements, the moves to solve were considerably higher than the usual average for these methods. This is partially due to the way these methods have been implemented, however, it seemed like this would be a good area for

improvement. The Corners First method has an unknown average number of moves [9] but the average from the implementation had some very high outliers. The average moves expected for the Beginner's Method is 110 which is much lower than has been observed so far. 48 moves is the average expected for the Roux Method [9] which is also much lower than has been seen so far.

To reduce moves sections of loops were looked at for where the implementation did not appear optimal. For many steps, the edge cases had poor implementations or were not considered often enough. In addition, some loops did not consider some cases which meant that many iterations of other algorithms were performed instead of one different algorithm that could be used to complete the step much more quickly.

Some sections require more moves as they sometimes turn one side many times to put the cubie in a specific position to change so that one algorithm can be performed instead of many. To change these sections would require much of the methods to be rewritten so this has not been done. If the methods were to be rewritten this could be worth considering but an alternate implementation may require more inspection which could increase solve time.

3.8 Parallelising the Methods

After trying different ways of implementing the methods they were parallelised to see how this affected the solve time. This could possibly decrease moves to solve further and bring the moves closer to the expected averages. Also if multiple cubes are solved at once some outliers should be eliminated. This may allow a reduction in solve times as well.

In this case, parallelisation is done by running the reduced moves methods above, as these seemed to work best for each method, on each of the six sides of the cube at the same time. This is quite a simple parallelisation and is effectively just running six solves at the same time. The time and moves to solve are taken for the cube that has the quickest solve time out of these cubes. This is repeated for each reduced moves method to see how this affects these different implementations.

To do this the scrambler had to be modified to also accept a sequence of inputs so that the same scrambling could be reused for each thread. An initial cube is scrambled and the moves to scramble are recorded. This is then used to scramble six separate cubes which are all rotated to different faces. Each thread is then given a different cube to solve in the same way as in the reduced moves methods above.

Further parallelisation may be difficult. Forking at each decision would create a very large state space which would likely be unmanageable. Running the six solvers in parallel reduces the state space considerably. In addition, taking the best first step and continuing from there may also be difficult as certain steps can only be done in one way based on what has been done before. This would mean that parallelisation would have little effect on these steps. For further parallelisation, some steps would likely need to be implemented again to take advantage of that type of implementation. This could lead to a significant deviation from the solving methods that have been implemented.

4 Results

The results in this selection have been collected using a timer that starts before the first step of the solver is started and ends after the last step has been finished. The validity of the solve, if the cube has actually been solved, time of the solve and number of moves of the solve for the cube are all recorded. This is then repeated for one thousand cubes for each method. This allows the data below to be generated. One thousand cubes should be a sufficient number to prevent anomalies from affecting the data too much and allows for analysis of the data. All cubes solved in each of the tests of one thousand cubes were valid solutions. The recorded results were collected in a Windows 10 Environment with no other programs running. All were done on the same computer. This should ensure as much consistency as possible, but uncontrollable background tasks may have slowed elements of the code for some methods. The effect of this should be reduced since one thousand cubes are tested with a one second interval between each which spreads all the cubes of each method out over around twenty minutes.

4.1 Beginner's Method Results

4.1.1 Initial Results

For the Beginner's Method, the average time to solve was 1.54ms (3s.f) which is quicker than any other software implementation that has been seen by the writer so far. This implementation improved upon Svyetov's Beginner's Method Implementation that took 39.1ms (3s.f) to solve the cube on average [1]. As a result, the quickest average solve time of a software implementation has been achieved which was the goal of this project. However, this method is variable and out of the total one thousand cubes, there were over ten outliers with the longest solve taking 25.0ms (3s.f) which is over ten times the average solve time. This is still below the 39.1ms solve time of Svyetov's implementation but is much closer to that time. This does show that the worst case solve for this implementation is quicker than the previous quickest average though. This method is therefore definitively quicker than this previous implementation when it solves the cube.

The average moves for this method were 345 (3s.f). This is higher than many other observed methods but may have improved the solve time. Many of the moves are used to sequence through positions that could put a piece in the correct place for an algorithm to be performed. This means that fewer different inspection methods are required and the same method can be utilised repeatedly for each step. If move numbers were also being minimised these methods would not have been viable.

The variance of this method was 1.09ms (3s.f) which shows that most solve times were very close to the average and that this method was reasonably stable.

4.1.2 New Moves Results

For the Beginner's method, only a few lines of code were changed with the move improvements from the newly added moves so a substantial difference was not expected. Rotating the cube up 3 times was replaced with rotating it down once and the same with rotating right 3 times to left. The average solve time was 1.24ms (3s.f), which is quicker than before. This is only a 0.3ms difference though so is close enough to the previous average, considering the variance, that this result is not substantially different. The average moves are also very close with an average of 355 moves (3s.f) to solve.

The variance here was much lower, but this must just be due to the sample of cubes solved as the implementation for this method is virtually unchanged.

4.1.3 Reduced Moves Results

After reducing the moves on the Beginner's Method the average time to solve reduced to 0.553ms (3s.f) which is less than half the previous solve time. The changes to the Beginner's Method reduced the number of iterations needed for many of the loops so prevented time from being wasted repeating tasks that were not as necessary. This improved the solve time far more than was expected. The worst solve time was 1.97ms (3s.f) which is also considerably lower. This improvement appears to have removed many edge cases that would cause loops to increase both time and moves to solve considerably.

The average moves to solve also reduced by nearly 100 to 265 moves (3s.f) which is also a large improvement. This is likely due to the removal of some edge cases as mentioned earlier. Also, some more inspection has prevented the need to iterate through loops as many times. This has probably caused the reduction in both solve time and moves.

Also, the variance of solve time reduced to 0.0362ms (3s.f) which is incredibly low. This suggests a far more consistent solve time and method. As a result, fewer anomalies are seen and generally, the solve time is far closer to the average.

4.1.4 Comparison of Beginner's Methods

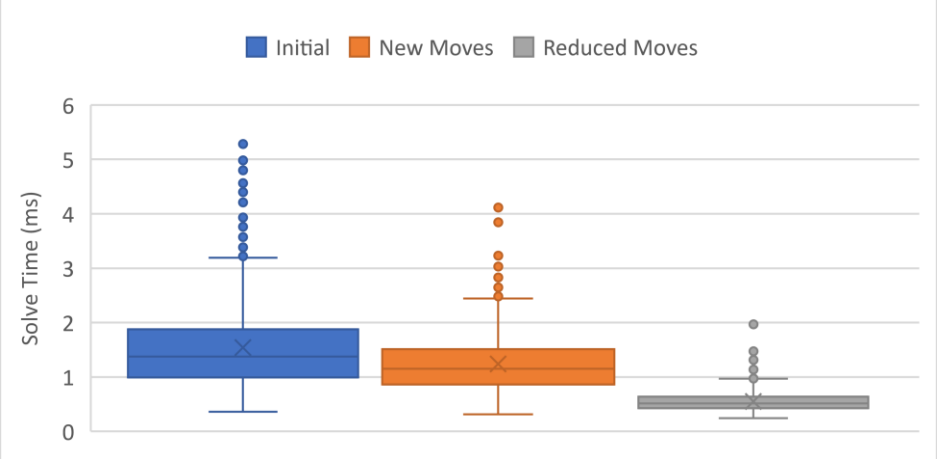

***Figure 4.1**: Box Plot to Compare the Solve Time of Beginner's Method Implementations (anomalous result for initial of 25.0ms (3s.f) excluded)*

The Beginner's method showed little change when the middle moves and extra rotational moves were added. There were few of these used in the Beginner's Method so it makes sense that they had little effect on the results. Slight changes were seen but this was likely due more to the randomisation of the cubes that were solved rather than an improvement in the solver.

Considerable changes were seen when the moves were reduced though. Average solve time, variance and moves to solve all decreased considerably. But the average moves is still quite a bit higher than the expected moves for the Beginner's method of just over 100 moves. It is hard to say if reducing moves further would improve solve times more but doing so would likely have diminishing returns.

The reduction in all areas was likely down to better avoidance of edge cases. Having fewer and lower anomalous results shows us that edge cases with higher solve times have been reduced. The data can be seen in **Figure 4.1** and **4.2** and **Appendix A Table A.1**.

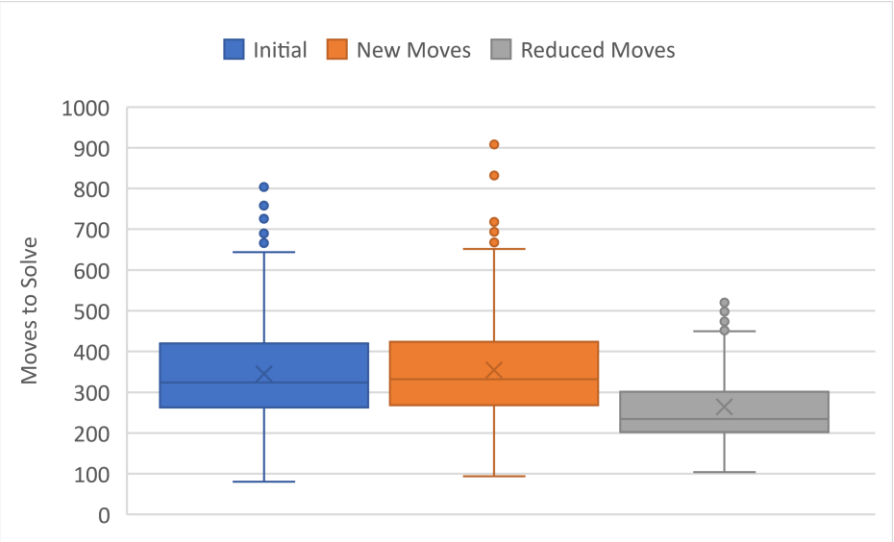

***Figure 4.2**: Box Plot to Compare the Moves to Solve Beginner's Method*

4.2 Corners First Method Results

4.2.1 Initial Results

The average Solve time for the initial implementation of the Corners First method was 2.46ms (3s.f) which is almost 1ms greater than the initial average for the Beginner's Method solve time. This method had over fifteen outliers with the greatest solve time taking 18.2ms (3s.f). This is in the worst case quicker than the Beginner's Method.

This method took 533 moves (3s.f) to solve on average which is much higher than the Beginner's method. This is likely due to the reliance on M, M' and E moves. These have been implemented using multiple other moves. This move number could be reduced considerably by implementing these other moves as the original moves were implemented. This other implementation would likely not have reduced solve time enough for this method to be the quickest software solver for the Rubik's cube though.

For this method, the variance was 3.45ms (3s.f) which is much higher than the Beginner's Method. As a result, it appears the Corners First Method may be much less stable. This is likely the cause of the greater average solve time for this method.

4.2.2 New Moves Results

The results for adding the new moves to the Corners First Method show an increase in average solve time to 4.46ms (3s.f) which is almost double the previous average. This is still close enough, considering the variance, that this is not a substantial difference. Even with the average moves reduced by close to 100 moves with an

average of 434 moves (3s.f) the solve time did not decrease significantly.

The variance is very similar to the previous Corners First implementation at 3.95ms (3s.f) which is to be expected. The same algorithms and method are used, the moves were just compressed.

4.2.3 Reduced Moves Results

The Corners First method showed the biggest improvement with the moves reductions. The average solve time reduced to 0.798ms (3s.f) which is multiple times quicker than both of its previous implementations. This method previously had some step implementations that appeared to be reliant on loop breaking methods to be solved. Some extra inspection and algorithms were added which was enough to prevent much of this. The greatest solve time was 2.98ms (3s.f) which is also much lower than before and suggests more consistency with the average solve time than before.

The average moves for this method have almost halved to 241 moves (3s.f) on average. This is much lower suggesting some issues with the previous implementation. Again the number of loops has been reduced also reducing the number of moves and the solve time.

The variance of solve time is now 0.0832ms (3s.f) which is also very low like with the Beginner's method. It appears that this method is also far more consistent than its previous implementation.

4.2.4 Comparison of Corners First Method Results

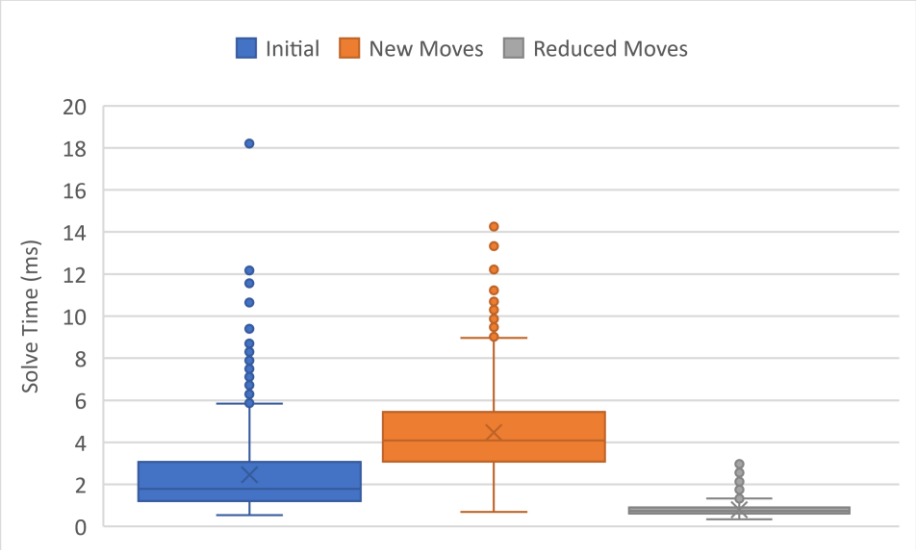

***Figure 4.3**: Box Plot to Compare the Solve Time of Corners First Method Implementations*

With the addition of the new moves, the Corners First Method showed a considerable decrease in average moves to solve but the average time to solve increased. The move reduction was expected but the increased solve time was not. This change in solve time is not too significant a change given the variance which did not change significantly. This increase in solve time is likely due to the variance in cubes used. This shows us that reducing the moves to solve may not always reduce the time to solve.

The Corners First Method showed the biggest improvement when the moves were reduced. Average solve time reduced to 0.798ms (3s.f) which is the quickest the Corners First Method has been. Variance also reduced substantially showing that this is the most stable version of the Corners First Method. The almost halving in moves to solve shows a correlation between a reduction in moves and time to solve in this case. So although a reduction in moves does not always reduce solve times, it appears some move improvements can.

As with the Beginner's Method, we have seen many fewer and lower anomalous results with the best method. This shows that stability is important for a good solver. In addition, it appears edge cases that would have increased the average and took much longer have mostly been avoided. Data can be seen in **Figure 4.3** and **4.4** and **Appendix A Table A.2**.

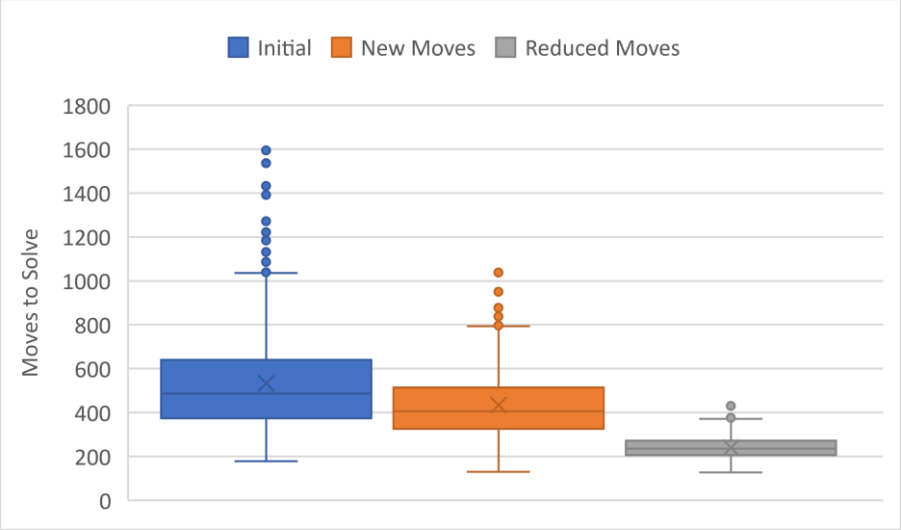

***Figure 4.4**: Box Plot to Compare the Moves to Solve Corners First Method*

4.3 Roux Method Results

4.3.1 Initial Results

For the Roux Method, the average solve time is 1.73ms (3s.f) which is only slightly higher than the 1.54ms (3s.f) to solve on average using

the Beginner's Method initially. In addition, the worst case solve time for this method is 4.96ms (3s.f) which the lowest of the three methods compared. This method gives the quickest initial worst case solve time. It is surprising how well this method performs as it was designed to best utilise the 15 seconds provided to humans in speedcubing for the first step. This meant that it appeared this method would not perform so well in a software implementation without this extra time.

This method took an average of 306 moves (3s.f) which is lower than the other two methods presented even without the use of r, M and M' moves. If these were implemented like the other moves were we would likely see this solve taking below 300 moves on average. In addition, implementing these moves properly could see this method finding solutions in a closer average time to the Beginner's Method. It could possibly even be a quicker method given how close the two methods average solve times are.

This method required more moves than the average suggested on the Speedsolving wiki which was around 50 moves [9]. There are likely a few reasons for this. The solve always starts from the same position when checking for moves and does the first possible move. This reduces inspection but means that methods that require fewer moves may be missed. Also, some sections of the implementation sequenced through cubies moving them into a spot where only one or two algorithms would be needed instead of using all of the algorithms provided on Gilles' website as there were so many [15]. This means that movements are increased but the amount of inspection should stay the same. In addition, the extra moves to perform the middle moves will also increase the overall moves.

The variance for this method was 0.417ms (3s.f) which is the lowest seen so far showing that this implementation of the Roux method is the most stable software solver. This has likely allowed this method to have the quickest worst case solve time.

4.3.2 New Moves Results

A considerable change has been seen when adding the new moves with the Roux Method. The solve time has reduced to 1.13ms (3s.f) which is the quickest average solve time for the new moves implementations. This is a substantial improvement and is far enough away from the original result to be considered a significant change. The worst solve time was recorded at 4.49ms (3s.f) which is also quicker than the previously recorded worst value in a test. This is enough to say that this implementation of the Roux Method could be considered definitively better than the other two new moves implementations.

The average moves decreased to 263 moves (3s.f), which is not as significant a change as with the Corners First Method but it is still substantial. This decrease in moves has likely allowed the improved average solve time since this was the only change made to the method.

The variance of this new data was 0.269ms (3s.f) which is very similar to the previously recorded data. This is to be expected as the method has not substantially changed as with the other two methods.

4.3.3 Reduced Moves Results

For the reduced moves Roux Method, the average solve time is now very close to the Beginner's Method with 0.551ms (3s.f) to solve on average. This is close to half the previous implementation of the Roux. However, the worst case solve time has not changed considerably with 4.04ms (3s.f). This suggests that there may still be some more room for improvement to prevent these anomalies with the Roux method.

For this method, the average moves have not changed as much either with an average of 230 moves (3s.f) to solve. This is still lower than before but is not as significant an improvement as the other methods. This method already had a lower solve time so it may be harder to reduce the number of moves without making more significant changes to the implementation.

The variance for this method has reduced to 0.0648ms (3s.f) which is lower than before but since this method already had a low variance, also is not as large an improvement as with the other methods.

4.3.4 Comparison of Roux Method Results

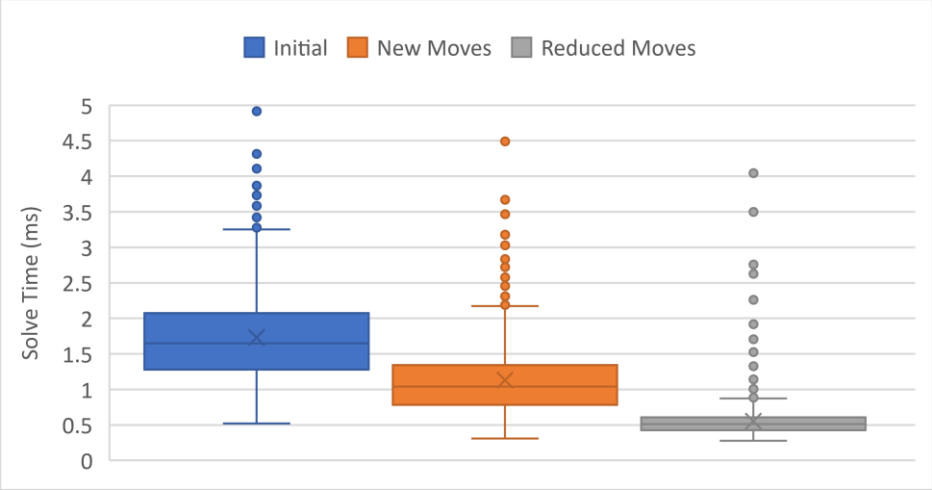

***Figure 4.5**: Box Plot to Compare the Solve Time of Roux Method Implementations*

The Roux Method Showed a steady improvement with each implementation. The initial implementation of this method was already good with quite a low variance and worst case solve time. The addition of the new moves reduced solve time quite a lot but was not really a substantial difference. The reduction in moves was more considerable but not as great as the reduction in moves for the Corner's First Method.

The reduced moves implementation halved the average time to solve and reduced moves a little. Variance also showed a massive decrease suggesting more stability as with the other methods. The worst case solve time did not reduce much though, unlike other methods. This was already quite low with the Roux Method though. Reducing moves appears to have improved solve time in all cases for the Roux Method.

Anomalous data did not improve much with the Roux method, but the average solve time and moves showed a steady improvement with each change. Variance also decreased showing more stability. Edge cases may not have been prevented as well as with the other two methods as the worst case solve time was the highest after the implementation of reduced moves. Data can be seen in **Figure 4.5** and **4.6** and **Appendix A Table A.3**.

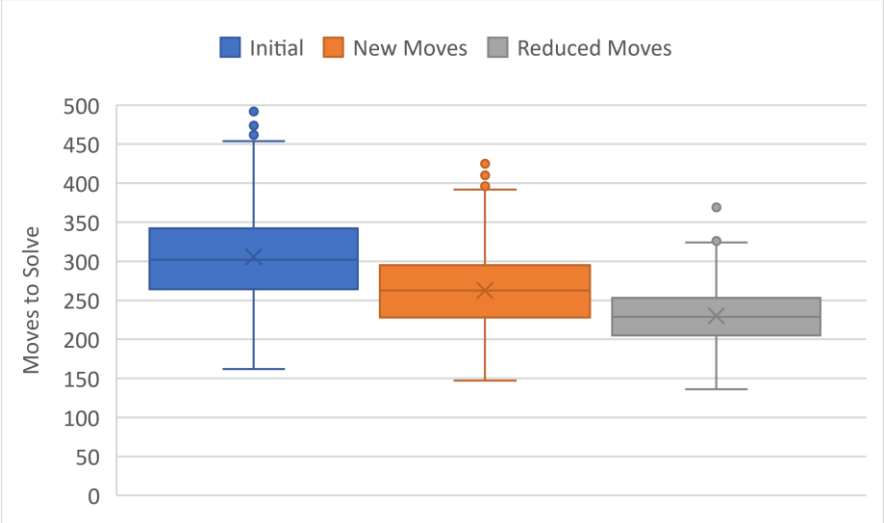

***Figure 4.6**: Box Plot to Compare the Moves to Solve Roux Method*

4.4 Comparison of Different Methods

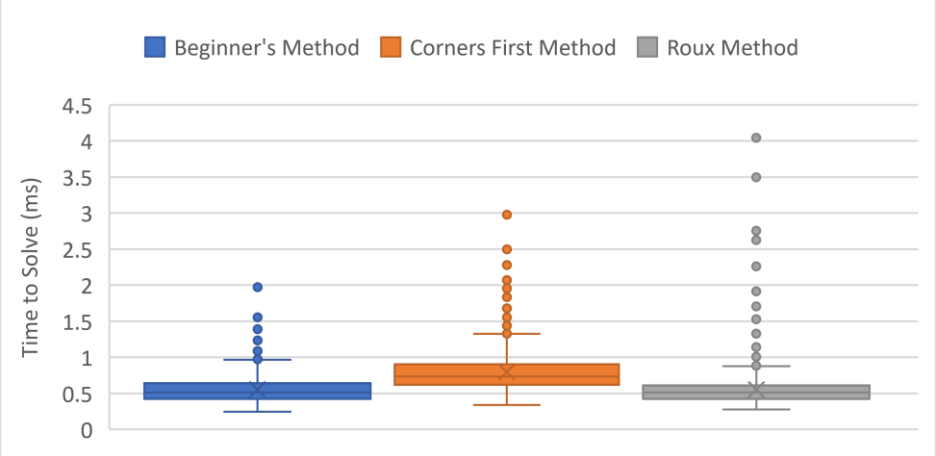

***Figure 4.7**: Box Plot to Compare the Solve Time of Reduced Moves Methods*

In this section, the reduced moves implementations of each method have been compared as these were the best implementation for each method. In almost every way the reduced move implementation was an improvement for all methods so this is probably the best comparison point.

The Beginner's and Roux Methods always had close average solve times at each step. The average solve time of the reduced moves Beginner's Method and Roux Method are closer than any before with 0.553ms and 0.551ms respectively. These are now the quickest methods seen. Corners First was always behind but showed a huge improvement now taking an average solve time of 0.798ms. All of these methods improved greatly suggesting that reducing the moves was an effective way to reduce solve time. This may have diminishing returns as the moves are reduced further though and will eventually start to increase solve time as we saw the optimal 20 moves solve taking 36.7 seconds when parallelised.

The Roux method has the longest worst case solve time of 4.04ms while the Beginner's method worst case solve time is the quickest at 1.97ms. This shows that the consistency of the Beginner's Method has been improved considerably. Given how close the average solve times of the Roux and Beginner's methods are this suggests the Beginner's Method may be the better solver with these implementations but it is still hard to definitively say which is best. All methods showed a considerable improvement in variance suggesting all methods are now much more stable than their initial implementations. The method used to reduce moves meant that edge cases were considered more efficiently preventing as many loops; as a result, it makes sense that

these methods are now more consistent. A comparison of the implementations can be seen in **Figure 4.7**.

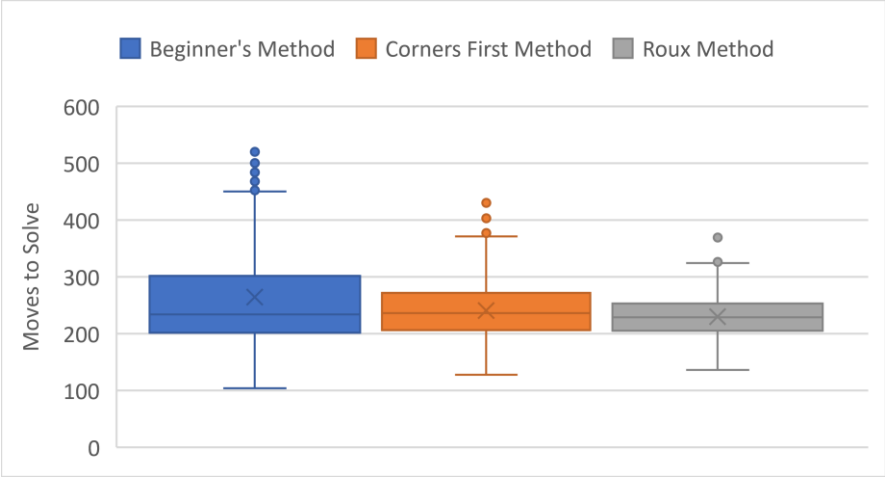

***Figure 4.8**: Box Plot to Compare the Moves to Solve with different Reduced Moves Methods*

The worst case moves to solve have decreased considerably for the Corners First method and Beginner's Method. The Beginner's Method had a worst case of 520 moves and the Corners First 430 which is much lower than the worst seen for either before. The Roux methods worst case moves to solve has improved but not as significantly as the other two methods but this method already had quite a worst case. It looks like the Beginner's method may have the biggest room for improvement in this area as there are still a few anomalous results and this method has the widest spread of moves to solve.

Average moves to solve are still higher than the expected averages for these methods. There may be a few reasons for this. All algorithms were not implemented for some methods to reduce complexity. If these had been implemented solve times may have taken longer, and the implementations would have taken much longer to write. In addition, the anomalous results suggest all situations are not handled as well as perhaps they could be. These situations may be hard to find though and by adding further inspection to identify them solve times could increase. In a software implementation, there will be a trade off between complexity and solve time. more testing of different implementations would be required to find the best point in this trade off to get the best solve time. More data can be seen in **Appendix A Table A.4** and **Figure 4.8**.

4.5 Comparison of Parallelisation

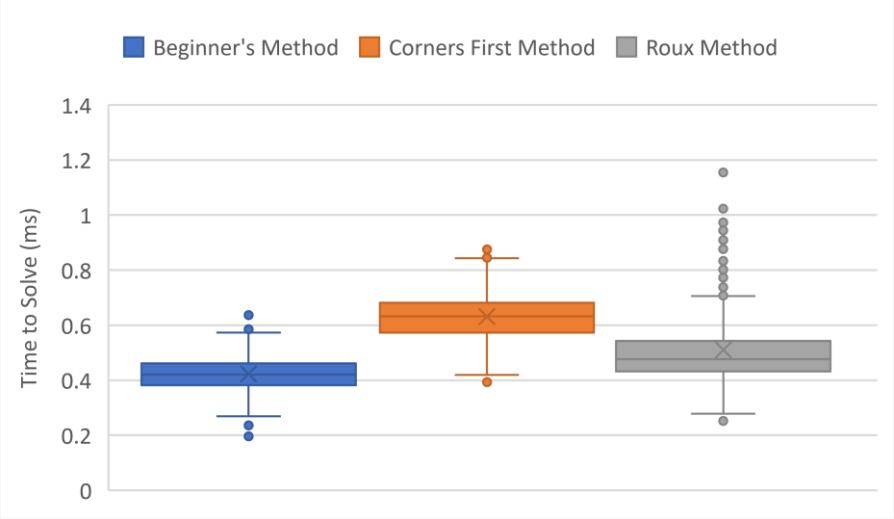

***Figure 4.9**: Box Plot to Compare the Solve Time of Parallelised Methods*

After parallelisation, all methods showed a reduction in time to solve and a considerable reduction in variance. This suggests an increased stability which would be expected given the best solve of six is used for each cube, however, anomalous results do not seem to have been reduced for the Roux Method. This could have been by chance as for the other two methods the anomalous results are less and the ones seen are also much closer to the average. The Beginner's method showed an improvement of 0.13ms allowing it to have a considerably quicker solve than the Roux method which only showed an improvement of 0.041ms. The Corners First Method showed the greatest improvement in solve time of 0.166ms. A comparison of the solve times can be seen in **Figure 4.9**.

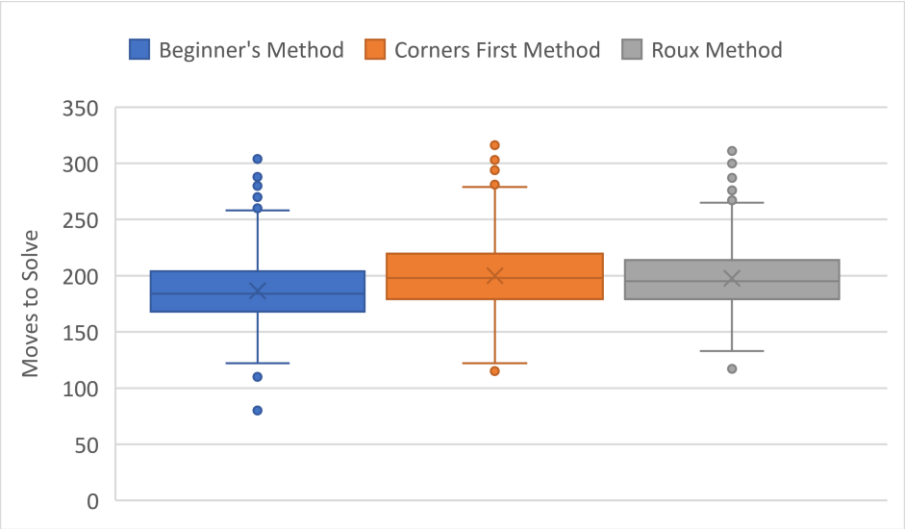

***Figure 4.10**: Box Plot to Compare the Moves to Solve of Parallelised Methods*

Moves to solve also all decreased significantly. The average moves to solve for the Beginner's Method decreased by 96 moves, the Corners First and Roux Method showed smaller improvements of 40 and 32 respectively. This shows the increased stability of the parallelised implementation and brings the methods closer to the expected average moves; although still a bit higher. The parallelisation only takes the best solve of six so may reduce the effect of the anomalies of the implementation but these flaws still increase the average above the usual. Anomalous results for all methods are now much closer to the average and are all not far out from the box plot also suggesting stability. It appears the suggestion that the Beginner's Method had the biggest room for improvement may have been correct as parallelisation was the most effective for that method. A comparison of moves can be seen in **Figure 4.10** and other data can be found in **Appendix A Table A.5**.

4.6 Understanding Why Cubes Took More Moves to Solve Than Their Expected Averages

Although the Beginner's Method begins to come close to its expected average when parallelised, it is still taking more moves than expected as is the Roux Method. No expected average move data could be found for the Corners First Method. The Roux Method is expected to take just over 50 moves [9] and the Beginner's Method is expected to take around 110 moves [1]. With parallelisation, the lowest averages were seen of 198 and 187 moves respectively and without parallelisation, the same methods gave 230 and 265 moves. This suggests that the software implementation of these methods may have some issues. To try and assess this problem the reduced

moves methods were rerun but with the average time and moves for each step recorded instead of for each solve, the data for this can be found in **Appendix A Table A.6**.

Initially, it seemed that the intuitive move sections may be the areas of issue as they did not have a defined algorithm. These areas were the top cross for the Beginner's Method, the bottom left of the Roux Method and the bottom corners and the Corners First Method. For the Beginner's Method, the top cross had the lowest average moves to solve of 14 moves suggesting this is likely not the area causing high moves for this method. For the Roux Method, the bottom left section took an average of 47 moves to solve alone which is close to the expected moves to solve for the whole cube. For this section several states are sequenced through until the expected position is achieved, an approach similar to the hard coding in the Beginner's Method would have been difficult to implement as there was less symmetry due to a rectangle being solved. In addition, the bottom corners section of the Corners First method took 54 moves to solve on average this is the joint second highest moves to solve for any step on the Corners first method suggesting that intuitive sections are probably a cause of inflated moves.

The bottom Corners section of the Corners First Method used the top corners method from the Beginner's Method as a base. This was also used in components of the bottom left step in the Roux Method. The top corners step in the Beginner's Method took an average of 53 moves to solve in the Beginner's Method and is the third highest moves for a step out of six. It appears this common link between the methods may be a cause of inflation in the moves between all three methods. However, other steps also have high moves to solve. For the Roux Method, the bottom right and top corners steps also have high moves to solve, for the Beginner's Method the middle layer and bottom face steps have high moves to solve, and for the Corners First Method the right edges and top corners steps have high moves to solve.

These sections have algorithms that loop through a series of states looking for cubies to move into the correct place mostly. This results in higher moves for these steps but it is hard to say if it increases solve times or not. These sections have higher solve times than sections with lower moves but have not been implemented using lower move methods. As a result, using lower move methods for these steps would be required to determine if solving in this way is beneficial.

5 Conclusion

A new quickest solve time method has been found with an average of 0.551ms to solve the Rubik's Cube when unparallelised. This was using the Roux Method as a basis for the implementation. The Beginner's Method also showed a very similar average solve time with 0.553ms on average to solve when unparallelised, but had much lower worst case solve times. Due to the similarity in the averages of these two methods and their variances, it was hard to definitively say which was better. After parallelisation solve times improved for all methods allowing for a quickest method to be seen which was the parallelised implementation of the Beginner's Method. This took 0.423ms to solve on average and the Roux method took 0.510ms when parallelised in the same way. This shows that parallelisation was not equally beneficial for all methods.

The Corners First Method also had a good solve time of 0.798ms before parallelisation on average. After parallelisation corners first took an average of 0.632ms to solve, but no matter the implementation always proved worse than the Beginner's and Roux Methods. As a result, the Corners First method is likely not worth considering further although it saw great improvements when its moves were reduced and after parallelisation. All methods were quicker than the previous best of 39.1ms [1] so the goal of this project has been achieved.

Reducing moves has had a considerable effect on reducing solve time throughout the project. This will likely not always be effective though. As moves decrease, the amount of inspection required increases which also takes time. Methods to solve the cube in an optimal number of moves often take longer than the quickest solving methods in practice and a parallelised implementation to solve in 20 moves in every case took 36.7 seconds [11]. With such low moves, a search is required instead of looking at algorithmic methods as has been done in this project. Lowering the moves without using a search could help but could still increase solve time when the moves are reduced too much. Further testing will likely be needed to determine how far moves can be reduced before it is no longer beneficial. Unfortunately, this would likely require considerable changes to the current implementations of the methods used in this project.

Parallelisation had a large impact on solve times and appears to have been universally beneficial. A more granular approach may be worth considering as this could allow the quickest solve for each step to be found. This would require steps to be modified to allow for parallelisation for it to be useful. This could be difficult to adapt and it may be easier to rewrite each step with the intention of parallelisation from the start. This would require some methods to be modified slightly

as some of the steps likely would not benefit much from parallelisation in the way they are currently written.

Other methods are worth considering. The evaluation used to determine which methods to use for this project may have allowed quicker methods to be found but it is hard to tell without testing other methods. With hundreds of different solving methods, there is likely the possibility of other quick or quicker methods. Trying to find better methods might prove an easier way to reduce Rubik's Cube solve times than improving the methods in this project further. It may be better to look for new methods as they could have the potential to be able to be improved to a point that could not be achieved with these methods. Without looking at other methods we will not know if this is the case or not.

Appendix A

	Initial	New Moves	Reduced Moves
Average Solve Time (ms)	1.541214	1.239899	0.552721
Variance of Solve Time (ms)	1.086192	0.289022	0.036235
Average Moves	345.274	354.97	264.58

Table A.1: Data for Each Change Beginner's Method

	Initial	New Moves	Reduced Moves
Average Solve Time (ms)	2.459585	4.463098	0.798042
Variance of Solve Time (ms)	3.448434	3.948306	0.083225
Average Moves	533.008	433.938	240.558

Table A.2: Data for Each Change Corners First Method

	Initial	New Moves	Reduced Moves
Average Solve Time (ms)	1.729832	1.128061	0.550917
Variance of Solve Time (ms)	0.417357	0.268733	0.064802
Average Moves	305.634	262.718	230.239

Table A.3: Data for Each Change Roux Method

	Beginner's Method	Corners First Method	Roux Method
Average Solve Time (ms)	0.552721	0.798042	0.550917
Variance of Solve Time (ms)	0.036235	0.083225	0.064802
Average Moves	264.58	240.558	230.239

Table A.4: Data for Each Reduced Moves Method

	Beginner's Method	Corners First Method	Roux Method
Average Solve Time (ms)	0.422657	0.631723	0.509863
Variance of Solve Time (ms)	0.003746	0.006952	0.016558
Average Moves	186.57	200.028	197.685

Table A.5: Data for Each Parallelised Method

	Beginner's Method	Corners First Method	Roux Method
Step 1	Top Cross	Bottom Corners	Bottom Left
Average Moves to Solve Step 1	14	54	47
Average Time to Solve Step 1 (ms)	0.0482362	0.155916	0.133089

Step 2	Top Corners	Top Corners	Bottom Right
Average Moves to Solve Step 2	53	54	64
Average Time to Solve Step 2 (ms)	0.127847	0.141196	0.106163
Step 3	Middle Layer	Three Left Edges	Top Corners
Average Moves to Solve Step 3	77	25	50
Average Time to Solve Step 3 (ms)	0.157623	0.098147	0.114641
Step 4	Bottom Face	Right Edges	Right and Left Edge
Average Moves to Solve Step 4	81	61	23
Average Time to Solve Step 4 (ms)	0.13195	0.187597	0.0338394
Step 5	Complete Corners	Last Left Edge	Middle Edges
Average Moves to Solve Step 5	16	8	20
Average Time to Solve Step 5 (ms)	0.0293477	0.0124797	0.0442527

Step 6	Complete Edges	Middle Edges	Finish Cube
Average Moves to Solve Step 6	17	21	14
Average Time to Solve Step 6 (ms)	0.0331985	0.0474772	0.044262
Step 7	-	Finish Cube	-
Average Moves to Solve Step 7	-	13	-
Average Time to Solve Step 7 (ms)	-	0.0428852	-

Table A.6: Data for Each Step of the Reduced Moves Methods

Bibliography

[1] N. Svyetov, “Rubik’s Cube Solver,” University of York, York, 2020.

[2] R. E. Korf, “Finding optimal solutions to Rubik's cube using pattern databases,” AAAI, Los Angeles, Ca, 1997.

[3] Rubik's, Speedcubing, [Online] Available: https://www.rubiks.com/en-us/speed-cubing [Accessed 20/01/2021]: Rubik's, 2018.

[4] J. Ben, The Rubik's Contraption, [Online] Available: http://build-its-inprogress.blogspot.com/2018/03/the-rubiks-contraption.html [Accessed: 20/01/2021], 2018.

[5] E. D. Demaine, S. Eisenstat and M. Rudoy, “Solving the Rubik’s Cube Optimally is NP-complete,” Cornell University, Ithaca, New York, 2018.

[6] T. Rokicki, P. Alto, H. Kociemba, M. Davidson and J. Dethridge, God's Number is 20, [Online] Available: https://www.cube20.org/ [Accessed: 20/01/2021], 2014.

[7] K. F. Fraser, “Rubik’s Cube Extended: Derivation of Number of States for Cubes of Any Size and Values for up to Size 25x25x25,” [Online] Available: https://www.kenblackbox.com/cube/maths/cubestates.pdf [Accessed: 12/02/2021], 2017.

[8] Rubik's, How to Solve the Rubik's Cube, [Online] Available: https://www.rubiks.com/en-us/blog/how-to-solve-the-rubiks-cube [Accessed: 20/01/2021]: Rubik's, 2018.

[9] Speedsolving.com, List of Methods, [Online] Available: https://www.speedsolving.com/wiki/index.php/List_of_methods [Accessed: 20/01/2021]: Speedsolving.com, 2020.

[10] Speedcubing.org, Guide To Choosing A Speedsolving Method, [Online] Available: https://speedcubing.org/pages/guide-to-choosing-a-speedsolving-method [Accessed: 21/01/2021]: Speedcubing.org, 2021.

[11] H. Hayakawa and H. Murao, “Optimal Rubik’s Cube Solver on GPU,” in *GPU Technology Conference*, 2013.

[12] M. Anttila, The Devil's Number and The Devils' Algorithm, [Online] Available: http://anttila.ca/michael/devilsalgorithm/ [Accessed: 20/01/2021], 2013.

[13] Speedsolving.com, Philip Marshall, [Online] Available: https://www.speedsolving.com/wiki/index.php/Philip_Marshall [Accessed: 20/01/2021]: Speedsolving.com, 2018.

[14] J. Jelinek, Corners-First Solution Method for Rubik's Cube - for Beginners, [Online] Available: http://rubikscube.info/beginner.php [Accessed: 20/01/2021]: rubikscube.info, 2015.

[15] G. Roux, Speedcubing Method, [Online] Available: http://grrroux.free.fr/method/Intro.html [Accessed: 21/01/2021]: Gilles Roux, 2005.

[16] O. Ryan, “Performance analysis of a simple Rubik’s Cube solver,” University of York, York, 2020.