Solving Rubik’s Cube Using Evolutionary Strategy

**Delivered to**

Prof. Dr. Magda Fayek

**Team Members**

1. Ayman Mohamed Abo El-Maaty
2. Mahmoud Ahmed Ismael
3. Mohamed Magdy Sedeek

# Abstract

The Rubik’s Cube is one of the most challenging problems for its multi-objective optimization problem. In this report, we have proposed and implemented an ES approach based on Michael Herdy’s ES using predefined swapping and flipping algorithms. We discuss in detail the problem definition to map it to ES regarding individual representation, selection method, fitness function and mutation operations. Finally, the results are presented and discussed for more understanding of the Rubik’s Cube problem and our proposed approach.

# Table of Contents

[Abstract 2](#_Toc419884570)

[Table of Contents 3](#_Toc419884571)

[Table of Figures 4](#_Toc419884572)

[1. Introduction 5](#_Toc419884573)

[1.1 Rubik's Cube Structure 5](#_Toc419884574)

[1.2 Notation 5](#_Toc419884575)

[1.3 Characteristics 6](#_Toc419884576)

[2. Related Work 6](#_Toc419884577)

[3. Methodology 6](#_Toc419884578)

[3.1 Individual Representation 7](#_Toc419884579)

[3.2 Mutation 7](#_Toc419884580)

[4. Implementation & Interface 8](#_Toc419884581)

[4.1 Fitness calculation 8](#_Toc419884582)

[4.2 Mutation operators 8](#_Toc419884583)

[4.3 Selection method 9](#_Toc419884584)

[4.4 Implementation 9](#_Toc419884585)

[5. Experimental Results 12](#_Toc419884586)

[5.1 Dataset 12](#_Toc419884587)

[5.2 Using (1, λ) Approach 12](#_Toc419884588)

[5.3 Using (μ, λ) Approach 15](#_Toc419884589)

[6. Discussion 16](#_Toc419884590)

[6.1 Cube Difficulty 16](#_Toc419884591)

[6.2 Population Size and Number of Parents 16](#_Toc419884592)

[6.3 Result’s Distribution 16](#_Toc419884593)

[7. Future Work 17](#_Toc419884594)

[7.1 Mutation Operations 17](#_Toc419884595)

[7.2 Fitness Function 17](#_Toc419884596)

[7.3 CMA-ES 17](#_Toc419884597)

[7.4 Result’s Distribution 17](#_Toc419884598)

[8. Conclusion 18](#_Toc419884599)

[References 19](#_Toc419884600)

[Appendix A: Matlab Code 20](#_Toc419884601)

# Table of Figures

[Figure 1: The application to solve the rubik cube using Herdy ES method 9](#_Toc419884620)

[Figure 2: Sample of scrambled cube with face description 10](#_Toc419884621)

[Figure 3: Program after loading sample Cube 10](#_Toc419884622)

[Figure 4: Program output after solving Cube 11](#_Toc419884623)

[Figure 5: Average number of generations per each sample using (1, λ) approach 12](#_Toc419884624)

[Figure 6: Cube 1 Output generations using population size = 50 13](#_Toc419884625)

[Figure 7: Cube 1 Output generations using population size = 100 13](#_Toc419884626)

[Figure 8: Cube 1 Output generations using population size = 200 14](#_Toc419884627)

[Figure 9: Cube 2 Output generations using population size = 200 14](#_Toc419884628)

[Figure 10: Average number of generations per each sample using (μ, λ) approach 15](#_Toc419884629)

# Introduction

Rubik's Cube was invented by Erno Rubik in 1974 and since its commercialization started in 1980 it has been the interest of both hobbyists and scientific researchers. Primarily mathematicians found themselves working on the Rubik's Cube as a discrete optimization problem trying to find efficient ways to solve it. With its simple structure the classic Rubik's Cube can reach a total number of 4.3 x 1019 different configurations which induces an underlying complex optimization problem. To this day it is impossible to calculate all of those configurations. Also, the shortest length of sequences to reach any of those configurations (God's Number) is still unknown and subject to ongoing research.

## Rubik's Cube Structure

The classic 33 Rubik's Cube is widely known and it’s the one subject to our technique. It consists of 26 pieces: 8 corner pieces, 12 edge pieces and 6 center pieces distributed equally on the six sides of the Cube. Each side of the Cube will be called face; each 2-dimensional square on a face will be referred to as **facelet**. Corners, edges and centers are all **cubies** - representing the physical object. A corner shows 3 facelets, an edge 2 and a center 1. Each side of the Rubik's Cube can be rotated clockwise (CW) and counterclockwise (CCW). Every such single move changes the position of 4 edges and 4 corners - **note that the center facelet on every of the Cube's faces always stays in the same position**. That’s why the color of a solved face is always determined by its center color. For each edge and corner it is of great importance to distinguish between position and orientation: i.e. an edge can be in its right position (defined by the two adjacent center colors) but in the wrong orientation (flipped).

## Notation

There are several known notations for applying single moves on the Rubik's Cube. We will use (F;R; U;B; L;D) to denote a clockwise quarter-turn of the front, right, up, back, left, down face and (Fi;Ri; Ui;Bi; Li;Di) for a counterclockwise quarter-turn. Every such turn is a single move. In Cube related research half-turns (F2;R2; U2;B2; L2;D2) are also counted as single move, we did so as well. This notation is dependent on the user viewpoint to the cube rather than the center facelets' colors. However, as a convention used for this work we assume the classic Rubik's Cube color configuration which is white: yellow; red: orange; blue: green where “:” denotes opposite of. The starting orientation for the scrambles will be F = white; R = red; U = blue; B = yellow; L = orange; D = green.

## Characteristics

The number of all attainable states 4:3 \_ 1019 depicts the order of the Cube group GC =< F;R;U;B;L;D >. All configuration of the Rubik's Cube can be reached by using combinations of single moves in this group, thus the single moves generate GC. Further,

* There is always a neutral element, i.e. F.FFFF = FFFFF = F and F4 = 1.
* There always exists an inverse: Fi .F = 1 and Fi = FFF

The inverse of any operation is quickly calculated by reversing the order of single moves and their direction. For example the inverse of (FRiDLBUi) would be (UBiLiDiRFi).

We can define subgroups of the group GC. Let H =< R;U > be a such a subgroup. If we only use moves from H there are just 26 x 38 x 52 x 7 = 73483200 different configurations we can attain. This significantly reduces the number of possible states a Cube can reach, but induces certain constraints like unchangeable edge orientations.

# Related Work

A few evolutionary approaches dedicated to solve the Rubik's Cube exist. In 1994 **Herdy** devised a method which successfully solves the Cube using pre-defined sequences as mutation operators that only alter few cubies, resulting in very long solutions. Another approach by Castella could not be verified due to the lack of documentation.

Another interesting evolutionary approach is using a common method to give ES the slightest possible degree of restriction is to break very complex optimization problems down into smaller parts which are solved independently, at best returning a solution for each sub-problem that can finally be joined to get a solution for the original problem. This is exactly how the Algorithm devised by Thistlethwaite in 1981 works. In the original algorithm each sub-problem is solved by finding a solution in pre-calculated lookup-tables. But his ES does not use those lookup-tables to solve those sub-problems, instead it evolves Cubes solving each problem by using a dedicated fitness function.

# Methodology

When designing an ES to solve the Rubik's Cube a few things come to mind. While ES turn out to be very useful problem-solving algorithms for complex optimization problems, typically those problems are described in a system of continuous variables. Small variations of these variables usually lead to small changes of the value of the fitness function (called strong causality). This is a fundamental behavior needed in ES to ensure a steady search for better solutions in the solution space. It is of high importance to transfer this behavior to discrete optimization problems. This can be done by adapting suitable mutation operators that work well with the fitness function used.

Obviously, the individuals that will be evolved are actual representations of a Rubik's Cube. Starting from the solved state, a certain configuration of a Cube is needed through a sequence of moves applied. A distinct state most certainly has multiple sequences leading to it, the state itself however is unique and one of the possible states. A scrambled Cube can be evolved by applying moves that hopefully near it to the solved state. This will be the ground principle of the two forthcoming ES.

## Individual Representation

Each face is implemented as a 3X3 2-dimensional matrix containing values from 1 to 6 where each value depicts one color. Thus, one individual is described by 6 matrices, describing each facelet color configuration.

Every quarter- and half-turn can be applied to this representation, yielding a total of 18 different single moves while leaving the Rubik's Cube integrity intact. Additionally the whole Cube can be rotated clockwise and counter clockwise. This guarantees easy human verification with a real physical Cube.

## Mutation

Mutation being the primary search operator of ES is easily realized by not modifying a single facelet’s color but applying a sequence of moves to the Cube. This guarantees that the Cube’s integrity stays intact at all times and makes a separate integrity test superfluous. Allowing mutations which operate on single facelets and change their color could yield non-existent Cube states, due to its structure.

Every Cube saves the mutations it has undergone, i.e. a list of moves that have been applied. To keep this list as small as possible, redundant moves are removed automatically. To clarify: we assume the above Cube where only F has been applied. Let the next mutation be FRRiB. This will automatically yield in F.FRRiB = F2B. We will go into further detail when describing each of the Evolution Strategies.

# Implementation & Interface

In 1994 Michael Herdy presented an (1,50) Evolution Strategy solving the Rubik's Cube in a mean of 38.7 generations, calculating only a mean of 1935 of all possible configurations. This algorithm was implemented as follows.

## Fitness calculation

To calculate the fitness of an individual the standard fitness function proposed by Michael Herdy was used. Three qualities are defined:

increases by 1 for each facelet whose color differs from the centre facelet on the same face.

increases by 4 for each wrong positioned edge, orientation is not considered.

increases by 6 for each wrong positioned corner, orientation is not considered.

As we see, each of those qualities can reach a maximum of 48, leading us to . Obviously the Cube is in a solved state when the sum's value reaches 0.

## Mutation operators

Herdy proposed the following mutation operators which change the fitness of each individual just slightly: two-edge-flip, two-corner-flip, three-edge-swap, two-edge/two-corner-swap, three-corner-swap, two-corner-swap and two-edge-swap - each in both directions (meaning the inverse sequence). Those were slightly extended using mirrors and adding three-inslice-edge-swap. Note that each such mutation operator is depicted by a sequence of single moves.

|  |  |  |
| --- | --- | --- |
| **Mutation** | **Sequence** | **Length** |
| Two edge flip cw | FRBLULiUBiRiFiLiUiLUi | 14 |
| Two edge flip ccw | FiLiBiRiUiRUiBLFRURiU | 14 |
| Two corner flip cw | LDiLiFiDiFUFiDFLDLiUi | 14 |
| Two corner flip ccw | RiDRFDFiUiFDiFiRiDiRU | 14 |
| Three edge swap cw | UF2UiRiDiLiF2LDR | 10 |
| Three edge swap ccw | UiF2ULDRF2RiDiLi | 10 |
| Two edge / corner swap cw | RiURUiRiUFRBiRBRFiR2 | 14 |
| Two edge / corner swap ccw | LUiLiULUiFiLiBLiBiLiFL2 | 14 |
| Three corner swap cw | FiUBUiFUBiUi | 8 |
| Three corner swap ccw | FUiBiUFiUiBU | 8 |
| Three inslice edge swap cw | RLiU2RiLF2 | 6 |
| Three inslice edge swap cw | LiRU2LRiF2 | 6 |

Table 1. Herdy ES Mutation Operators (omitting Mirrors)

The above operators change the state of a Cube just slightly by only affecting 2-4 positions while leaving the rest of the Cube intact. This guarantees a slow and steady exploration of the solution space, slightly improving the population per generation. To fully utilize the above operators’ potential, the face being the physical front of the Cube has to be randomly chosen before each mutation as well as the orientation of the Cube. Thus an actual mutation step looks like this:

1. Choose a random face to become the new front, this involves the entire Cube to be rotated, there are 6 faces to choose from.
2. Choose a random orientation of the front by rotating the whole Cube cw/ccw 1 or 2 times.
3. Choose a random mutation operation to be performed on the Cube.

## Selection method

After having applied random mutations to every individual in the population, they are sorted by their fitness in ascending order (remember 0 = solved). Now, the best individuals are selected for duplication.

## Implementation

The implemented method is Herdy’s evolutionary strategy.

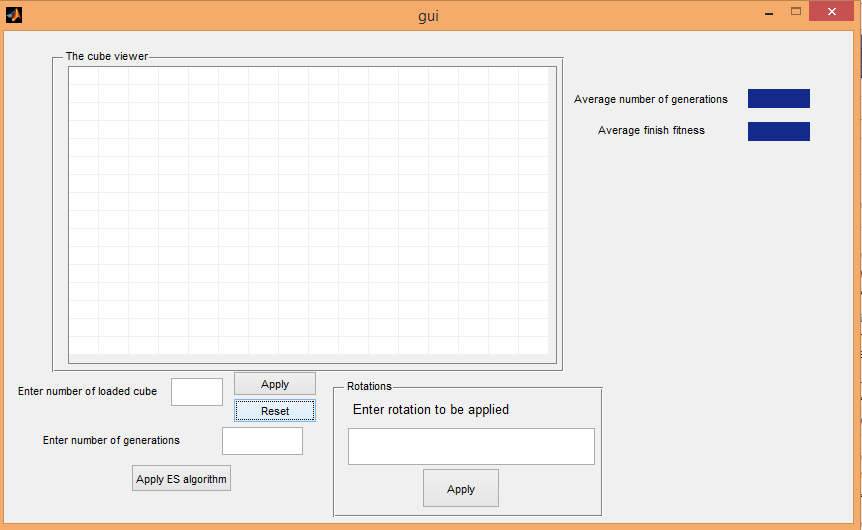


Figure 1: The application to solve the rubik cube using Herdy ES method

1. Enter the number of the cube from 1 to 20 to load a cube.
2. If wanted to apply rotations (F, F-, B, B-, R, R-, L, L-, U, U-, D, D-) on this cube write the rotations separated with a space character.
3. Enter number of generated mutants from the current cube that will be generated using the randomly from the operations discussed above.

|  |  |  |
| --- | --- | --- |
|  | Top |  |
| Left | Front | Right |
|  | Down |  |
| Back |

1. The cube viewer window is where the current cube will be viewed as follows.  
   

Figure 2: Sample of scrambled cube with face description

1. Then press the “Apply ES algorithm” to apply the Herdy method and the average number of generations, average of the fitness of the last mutants and the last cube is solved or not will be viewed.

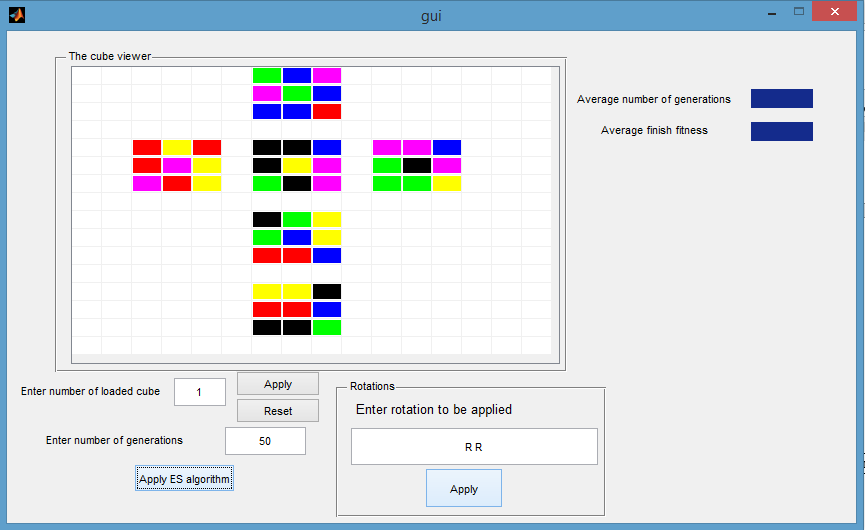


Figure 3: Program after loading sample Cube

1. Sample output:

* for cube number 1 and with 50 generations
* The average number of generations to solve the cube = 139.31, the average  
  finish fitness = 0.0 and the cube is solved.

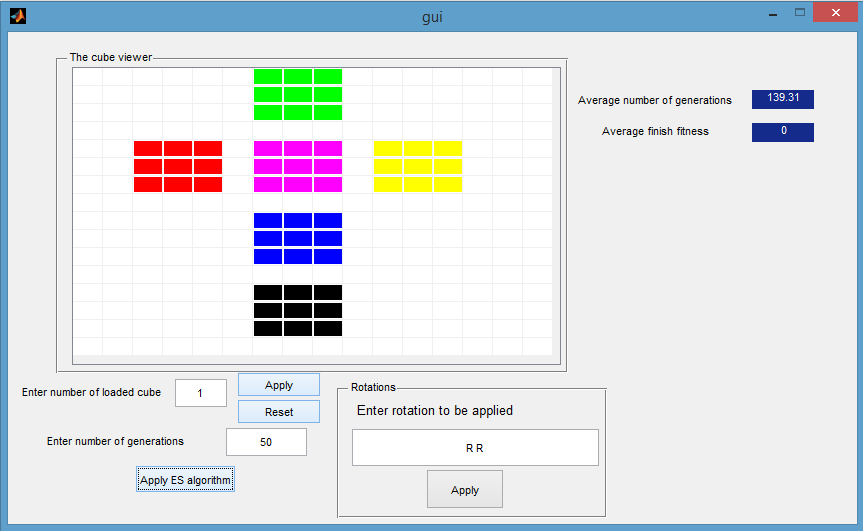


Figure 4: Program output after solving Cube

# Experimental Results

In this section we will present our experiments and results in details.

## Dataset

In order to define our Test cases, we generated 23 random cubes with different difficulties to measure our approach performance. The cube difficulty is defined by the number of random moves applied to a solved cube to scramble its faces.  
We defined 6 levels of difficulties as follows:

1. Simple Level: Cubes scrambled by 1~4 random moves.
2. Level 1: Cubes scrambled by 20 random moves.
3. Level 2: Cubes scrambled by 30 random moves.
4. Level 3: Cubes scrambled by 40 random moves.
5. Level 4: Cubes scrambled by 50 random moves.

The dataset cubes are uniformly distributed on the different difficulty levels. The distribution is set as 3 cubes for simple level and 5 cubes for each other level.

## Using (1, λ) Approach

We applied the 23 dataset cubes on different population sizes 50, 100 and 200 respectively. 100 runs are performed for each cube to calculate the number of generations required to find a solution. Then average number of generations is calculated to observe the difficulty’s effect and the expected output for new test cases.

Figure 5: Average number of generations per each sample using (1, λ) approach

Histograms for output generations are drawn for selected sample cubes to observe the randomness effect and study the approach’s efficiency. The histograms are drawn for cube 1 applying population sizes 50, 100, 200 respectively.

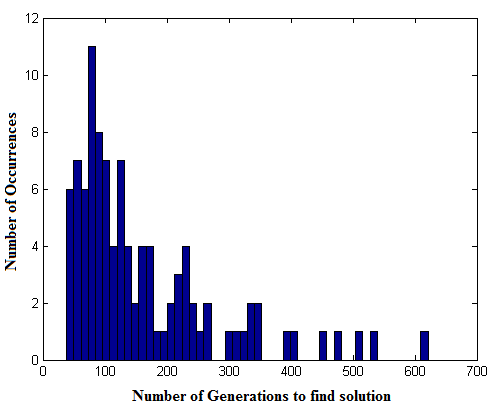


Figure 6: Cube 1 Output generations using population size = 50

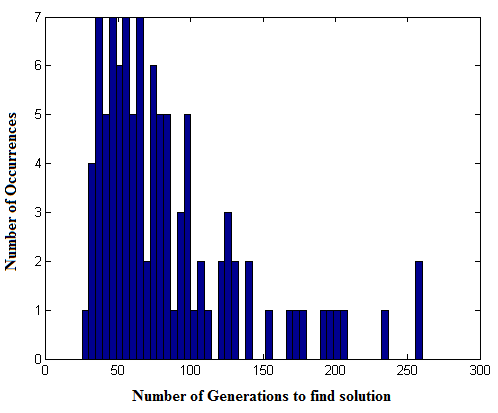


Figure 7: Cube 1 Output generations using population size = 100

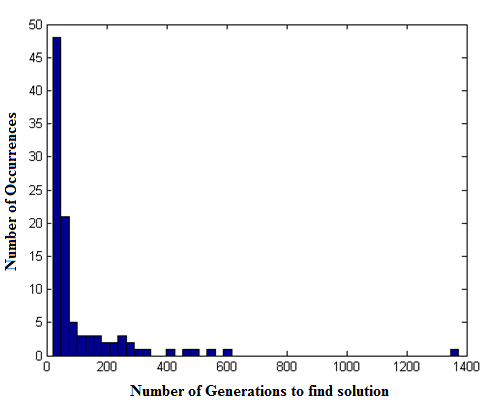


Figure 8: Cube 1 Output generations using population size = 200

Another Histogram is drawn for sample cube 2 with population size 200 to compare the results with cube 1 histogram.

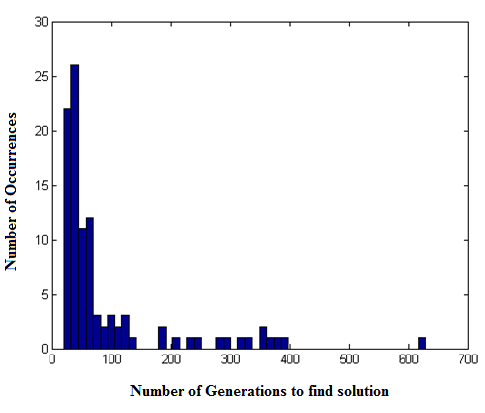


Figure 9: Cube 2 Output generations using population size = 200

## Using (μ, λ) Approach

We applied the first 5 cubes of the dataset on different population sizes and different number of parents selected (10, 100), (10, 200), (10,500), (20, 200), (20, 500),  
(50, 500), and (50, 1000) respectively. 5 runs are performed for each cube to calculate the number of generations required to find a solution.

Figure 10: Average number of generations per each sample using (μ, λ) approach

# Discussion

In this section we will discuss the results from different point of views.

## Cube Difficulty

The cube difficulty did not affect the average number of generations needed to solve the cube. However, Simple level cubes also needed a relatively large number of generations to find a solution. The mutation operations’ nature of long complex operations makes solving simple cubes harder than in traditional approaches.

## Population Size and Number of Parents

The size of population defines the width of states examined using the ES. Obviously the increase in the population size enhances the outputs. But the relation between the number of parents and the population size is more remarkable. For example, the average number of generations using (10, 200) ES was smaller than   
(20, 200) ES. Although (1, 50) approach was able to reach the solved cube,  
the (10, 50) approach didn’t reach a solution within the stopping criteria limits.

From our results, we defined an inversely proportional relation between the number of generations needed to solve a cube; with both the population size used, and the ratio between the population size and number of parents selected in each generation (λ/μ).

## Result’s Distribution

Finally, the histogram of the output generations for each cube describes the operation’s randomness effect on the performance. The distribution graph shows that it is more likely to find the required solution in the first few set of generations, although the mutation operations are selected by a uniform random distribution.

# Future Work

Several approaches could be developed using the observations discussed in the last section.

## Mutation Operations

The definition of limited set of complex operations in the mutation operations was used in Herdy’s approach to steadily explore the search space. This approach performed well for difficult test cases. However, the simple level tests needed almost the same number of operations. Adding more simple operations to the set of mutation operations would have a great effect on the number of generations to solve the cube.

## Fitness Function

The number of generations is not the only goal for the researchers. Minimizing the number of moves to solve the cube is a greater goal. The number of moves could be decoded and added to the fitness function to enforce the ES to select the best semi-solved cubes having small number or moves.

## CMA-ES

The cube orientation and pose preparation before mutation is done using uniform random distribution. Directing the ES to select the faces with more scrambled facelets will set the exploration direction towards fixing the cube more than scrambling it. Covariance Matrix Adaptation (CMA) had performed well in multiple problems, and it would increase the performance.

## Result’s Distribution

The distribution of resulted number of generations could be studied in detail to define the relation between the randomness of the mutation operations, the population size, and the ratio between population size and number of parents (λ/μ).

# Conclusion

ES is very powerful in massive search space problems as Rubik’s Cube problem. The problem encoding using group theory allowed researchers to build high performance adaptable programs solving any scrambled cube. Herdy’s approach provides fast solutions for any scrambled cube using predefined mutation operations. Even with small (μ, λ) and a very simple fitness function.

More tests and techniques will have to be conducted for further examination of the Herdy’s ES behavior with different parameters producing smaller number of moves.

# References

1. Herdy, M., Patone, G.: Evolution Strategy in Action, 10 ES-Demonstrations. Technical Report, International Conference on Evolutionary Computation (1994)
2. C. Di Chio et al. (Eds.): EvoApplications 2010, Part I, LNCS 6024, pp. 80–89, 2010.
3. Nail El-Sourani, Markus Borschbach, Design and Comparison of two Evolutionary Approaches for Solving the Rubik's Cube, Parallel Problem Solving from Nature, PPSN XI. 11th International Conference, Kraków, Poland, September 11-15, 2010, Proceedings, Part II, pp 442-451.

# Appendix A: Matlab Code

Attached in the Email