**Project 6: Solving Rubik’s**

**Cube using genetic algorithms**

**and Wisdom of crowds**

Trevor Bright, Jackson Paul, Grant Campbell

Computer Engineering/Computer Science

Speed School of Engineering

University of Louisville, USA

**1.** **Introduction** (What did you do in this project and why?)

The Rubik’s cube was created in 1974 by Erno Rubik, with numerous prototypes of varying sizes such as 3x3x3, 4x4x4, and even 10x10x10. This project will focus on the 3x3x3 variant using a genetic algorithm and wisdom of crowds to find an optimal number of moves to complete a Rubik’s cube. Python through Visual Studio Code was used and the command line to run and compile the program.A Rubik’s cube consists of 3 types of blocks:

six center blocks which do not move relative to each other

twelve edge blocks with 2 colors on them

eight corner blocks with 3 colors on them.

These edge and corner pieces are referred to as ‘cubies’. The goal is to position all colors on a unique side in as few moves as possible. There are six basic moves on a Rubik’s cube, each with a counter-clockwise variant:

F = rotate the front face one quarter turn clockwise

B = rotate the back face one quarter turn clockwise

U = rotate the top face one quarter turn clockwise

D = rotate the bottom face one quarter turn clockwise

R = rotate the right face one quarter turn clockwise

L = rotate the left face one quarter turn clockwise

‘ = rotate the prefix face one quarter turn counterclockwise

2 = rotate the prefix face one half turn clockwise

Figure 1 shows an example scrambled cube, while Figure 2 shows a solved cube with a potential solution equation using the above notation..

Solution = R U D’ F L’ F2 B2 R’ B2 R’ F D2 L F U D L2 D’ R2 D2 F2 U2

**2.** **Approach** (Describe algorithm you are using for this project)

PyCuber has a built in function to make a random set of moves. This random algorithm was used to generate a scrambled cube. If multiple runs are specified by the user, then this same random algorithm is used for all runs. To solve the Rubik’s cube, sets of a random moves are generated with a max number set by the user. Instead of using PyCuber’s built in random function, since no max number of moves can be specified, all possible moves are held in an array and random elements are pulled from it to generate the list of moves. After a population is created, ranking a mating occur to produce offspring.

The fitness function only measures how many faces match a solved cube, the more matches the higher the ranking. Mating is done by choosing a random subset of moves, always including all moves leading up to the subset, since that matters for Rubik’s cube. The closer of the 2 subsets is inserted into the child and the rest of the moves from the other parent are appended to the child’s moves until the other parent is out of moves or the max moves limit is reached. Mutation then occurs, with a 1% chance of each person being mutated, and a move possibly being added, deleted or changed.

This was then repeated for 100 generations when the master process will check how all the populations are doing, see what the best moveset is, and restart all the subprocesses. If the best hasn’t improved in 300 generations, then the wisdom of the crowds function is called. Wisdom of the crowds looks at which 3 set of moves shows up the most and tries to recombine moves based upon leading and following moves. The best GA and Wisdom of the crowds are compared and the better of the 2 is returned.

If the number of runs is greater than 1 then the whole process is repeated as many times specified. All the best returned move sets are compared against each other, and the best is displayed along with the scramble move set.

**3.** **Results** (How well did the algorithm perform?)

The algorithm performed okay. There could be some more optimization in how the GA looks for a move set of a solved cube. Rubik’s cube by design have to become mixed up while solving it to solve it, and since this GA tries to get as close to solving as possible without mixing up the cube, there’s a limitation in how close the GA can get to a solution. With the wisdom of the crowds, the function looks for a set of moves to show up, 3 moves, and counts how many times it appears. Recombining the wisdom is difficult because each move relies on every move behind it to be worthwhile. Currently multiple separate arrays are created from recombining moves based upon its neighbors and the one returns the closest solved is returned.

**3.1** **Data**

The data used was a randomly generated scramble move set. This move set was randomly generated each time the python script was called.

**Results**

**1000 population, 5 runs, 100 moves**

**Min**: 29 **Max**: 34 **Avg**: 31.2 **Deviation**: 1.9390719429665317

**Random Alg:**

L F2 R' U F2 L2 B U F' L' U F2 R2 D2 L2 U' R2 B' D2 L2 F' D2 B R' D'

[y][y][o]

[r][y][w]

[b][g][y]

[b][w][w] [r]**[w]**[r] [g][o][g] **[w]**[r][o]

[b][r][g] [r][g][r] [b][o][y] [b][b][o]

[y][y][y] [r]**[w][w]** [o][y][g] **[w]**[g][g]

[b][b][b]

[o][w][g]

[o][r][o]

**Closest:**

R' U R2 D2 B D B F' U2 U' B2 L2 D' L F2 R B' L' F

[o][r][o]

[y][y][g]

[g][w][b]

[w][o][o] [y][b]**[w]** [r][y][w] [b]**[w]**[g]

[b][r][r] [g][g][b] [o][o][y] [r][b][y]

[y][w][y] [b][o][r] [g][g][r] [b][r][r]

[o][g][w]

[o][w][w]

[g][b][y]

**Distance:** 34

--- 31306.777089595795 seconds ---

**2000 population 2 runs**

**Min**: 32 **Max**: 34 **Avg**: 13.2 **Deviation**: 19.825236442474022

**Random Alg:**

U' F2 L2 U F' L D' F2 R D2 B' L2 F2 R' U B L' D' F L' B U2 R' F' U

[b][b][b]

[g][y][o]

[b][g][b]

[w][r][r] [y][o]**[w]** [o][g][w] [r][y][o]

[b][r][o] [y][g]**[w]** [g][o][r] [y][b][r]

[w][y][y] [g]**[w]**[o] [g][r][b] [o]**[w]**[g]

[r][o][y]

[g][w][w]

[r][b][y]

**Closest**:

F L2 R D2 F2 U R' U2 L B' R2 B F' D' D L' B2 U' D

[o][y][g]

[y][y][g]

[o][y][o]

[y][o][w] [b][r][y] [g][w][r] **[w]**[b][b]

[r][r][g] [o][g][o] [w][o][r] [g][b][b]

[r][b][b] [y]**[w]**[o] [w][y][r] **[w]**[b][g]

[r][r][g]

[o][w][g]

[y][w][b]

**Distance**: 34

--- 25703.793170452118 seconds ---

**4.** **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

There could be improvements done to this GA. Some include using the traditional algorithm of solving a Rubik's cube. Right now the GA just tries to get close to solved without mixing up the cube. I think taking the GA in stages, like solving 1 face, saving that move set, solving edges, combining that with the previous move set and so on, would allow the GA to get closer to completing the cube. Number of moves didn’t seem to affect runtime nearly as much as population size. The measurement function for a few runs is fast, but for 50 cubes takes a few seconds. This adds up quickly and makes population size the bottleneck.

**5.** **References** (If you used any sources in addition to lectures please include them here.)

PyCuber library and documentation:<https://github.com/adrianliaw/PyCuber>

The Mathematics of the Rubik’s Cube <https://math.berkeley.edu/~hutching/rubik.pdf>