Russ's Rubik Solver 1.0
User Manual

Version History:

1.0 – 2008-12-15 – Initial Release

Introduction

I created this program as an Undergraduate Honors Research Project as a part of the Kansas State University Department of Computer Science. This program was written using the Two-Phase algorithm as specified by Herbert Kociemba on his website http://kociemba.org/cube.htm (under the Mathematics section) in the Java programming language. I cannot claim any credit for the math behind the algorithm; all I did was take his information and my knowledge and implement it in the language I am most comfortable with.

~~Therefore, this program and accompanying source code are for academic use only, and not for public release. At some time in the future I may contact Mr. Kociemba and ask for his blessing upon this project, at which time I will make it available publicly as an open-source project.~~ Licensed under the MIT license 2015-07-30

If you have any questions or comments about any aspect of this program, or any of the mathematics or ideas behind the algorithm, please feel free to contact me. I am glad to answer them to the best of my knowledge.

Russell Feldhausen, KSU '08
russfeld@ksu.edu
russfeld_2166@yahoo.com

Using Russ's Rubik Solver 1.0

To run this program, double click on the file Run.bat in the dist folder. The first time you run the program, it will calculate all the data tables that are required for the program to operate. This calculation takes approximately 30 minutes on a Core2Duo processor, but could be significantly longer on older hardware. When it is complete, it will save 5 data files in the dist folder, the largest of which is nearly 250MB in size, so make sure there is plenty of room on the drive you are running this program from. When running, this program requires approximately 350MB of RAM. However, once you have done this calculation once and saved the files, you will not have to calculate them again.

- Create a New Cube: Click the "Create Solved Cube" button to create a new solved cube. The GUI display should be updated accordingly

- Input a Cube Layout: To input a layout by hand (from an existing cube, for example), click the "Edit Cube Layout" button. To edit the cube, click on the colored squares to cycle through all 6 possible colors. They follow the order of the colors on a rainbow (Red → Orange → Yellow → Green → Blue → White). You will notice that the colors on the center squares will not change. This is due to the fact that the algorithm will not be able to solve a cube if it is not input in that particular orientation. Simply orient your cube to match the display before you input it. Once you are done inputting a cube, click the "Save" button below the "Edit Cube Layout" button. If your cube is not valid, the program will warn you and stay in edit mode for you to correct the error.
- Solve a Cube: To solve a cube, simply click the "Solve" button. Most solutions should appear in under a second. To follow the solution graphically, you can click the "Step Forward" and "Step Back" buttons in the lower center of the GUI. For more information on the list of moves presented and how to read them, see the *Notation* section below.

- Apply a Move: To apply a move to a cube, simply click on one of the six buttons in the upper center of the GUI. They are labeled by the face that will be turned. By clicking a button, the selected face will be rotated clockwise one quarter turn.

- Apply a Symmetry: To apply a symmetry to the cube, simply select a symmetry index from list in the middle of the GUI, and click the button for the symmetry you would like to perform. For more information, see the Symmetry links on http://kociemba.org/cube.htm. There are four possible symmetry applications:
    - Symmetry – A simple application of the given symmetry
    - Sym Inverted – Applies the selected symmetry inverted (undoes the symmetry)
    - S then Sinv – Applies the symmetry to a solved cube, then applies the given permutation, then applies the inverse of the symmetry.
    - Sinv then S – Applies the inverse of the given symmetry to a solved cube, then applies the given permutation, then applies the symmetry.

  You will notice that the center squares do not change as you apply a symmetry. Since they are only for reference when inputting and solving a cube, they can be ignored. However, it may not be able to solve a cube if you have applied a symmetry to the cube in the program. Before solving the cube, make sure you undo any symmetries.

- View Coordinates: As you apply moves to a given cube, you will notice that the coordinates shown will change. Those coordinates are as given on Kociemba's site; for more information please refer to http://kociemba.org/cube.htm.

Notation

I am using the notation for moves as specified by Kociemba. It is as follows:

The first letter is the face to be turned. The number is the number of quarter turns to be applied. The faces are defined as follows:

- F – Front (White)
- B – Back (Yellow)
- R – Right (Blue)
- L – Left (Green)
- U – Up (Orange)
- D – Down (Red)

For example, “Rx1” would be a 90 degree clockwise rotation of the Right face, “Fx2” would be a 180 degree rotation of the Front face, and “Dx3” would be a 90 degree counterclockwise rotation of the Down face. All rotations are as if you are looking at the face.

My Algorithm

My algorithm is a slightly simplified version of Kociemba’s. The differences are as follows:

- In his articles, he compresses the pruning tables and uses the differential between two states. I use the absolute state.
- My Phase 1 simply finds the first solution. That means that as it applies moves to the cube, if it finds one at a lower pruning depth, it immediately takes that path. This speeds the algorithm along, since it always will find a solution with a minimal length for Phase 1.
- For Phase 2, it will add all lower pruning depths to the working queue, and address each in order to find a solution. If no solution can be found, it will go back and move laterally from the initial state and try again. If no solution is found then, it will move the initial state up one pruning level and try again. It has the ability to move a maximum of 18 pruning levels backward to find a solution, but rarely needs to move more than one.
- Kociemba’s algorithm has the ability to apply non-optimal Phase 1 solutions to Phase 2 in order to find a shorter overall solution. However, I feel that the extra work in order to remove 2 – 3 steps of a solution was not needed, especially since my solutions are very reasonable in length anyway.
- His algorithm also uses several other coordinates in Phase 1 to avoid restoring the cube manually before starting Phase 2. I simply ignored those coordinates and applied the Phase 1 solution to the cube before beginning with Phase 2.

If you run this program from the command line, you will notice that as you solve the program it will output certain debugging information. For the most part, the information is a listing of the coordinates that are checked, and a backtrace of the moves applied to reach that stage. For more information, refer to the source within the file RubikSolver2.java.

Please feel free to look through the source code for more information on exactly how the program works. If you receive this program without attached source, please let me know and I would be glad to send it to you.