Implementing a Two-Phase Algorithm for Solving a Rubik's Cube

Russell Feldhausen

CIS 499 Honors Project

Fall 2008 – Dr. Howell

Introduction

- Create a computer program that will solve any given Rubik's Cube puzzle quickly and accurately
- Try to minimize the number of moves needed to solve the cube
- Understand more about the mechanics behind the puzzle

Why Solve a Rubik's Cube?

- Simple puzzle to understand
- Very Complex algorithms to find solutions
- Large number of possible states
- Have never solved one before
- Personal Pride

Initial Research

- Many Solutions available for manually solving a Rubik's Cube
- Most involve getting the cube into different "States", then using a defined set of moves to move to the next state
- Large number of moves required

Initial Research

- Some methods attempt to orient the corners and edges, then place them in the correct positions
- It has been (computationally) proven that any Rubik's Cube can be solved in at most 22 face turns

First Attempt

- Cube represented as 6 byte arrays containing the information for the six faces of the cube
- Moves implemented by hard coding the swapping of bytes between the various faces
- In theory should be very fast to simulate several moves

First Attempt

- Using Breadth-first search, attempt to solve a given cube by applying all possible moves, checking if a solution is found, then repeating on all generated cubes.
- Refine this algorithm into Branch and Bound, allowing pruning of branches

First Attempt

- Problem with exponential growth:
  - 18 possible moves (3 * 3 * 2)
    - $18^2 = 324$
    - $18^3 = 5{,}328$
    - $18^4 = 104{,}976$
    - $18^5 = 1{,}889{,}568$
    - $18^6 = 34{,}012{,}224$
    - $18^7 = 612{,}220{,}032$
    - $18^8 = 11{,}019{,}960{,}576$
    - $18^9 = 198{,}359{,}290{,}368$

First Attempt

- Took several days at depth of 9 moves
- Was able to solve nearly solved cubes
- Large amounts of memory used to store all created permutations

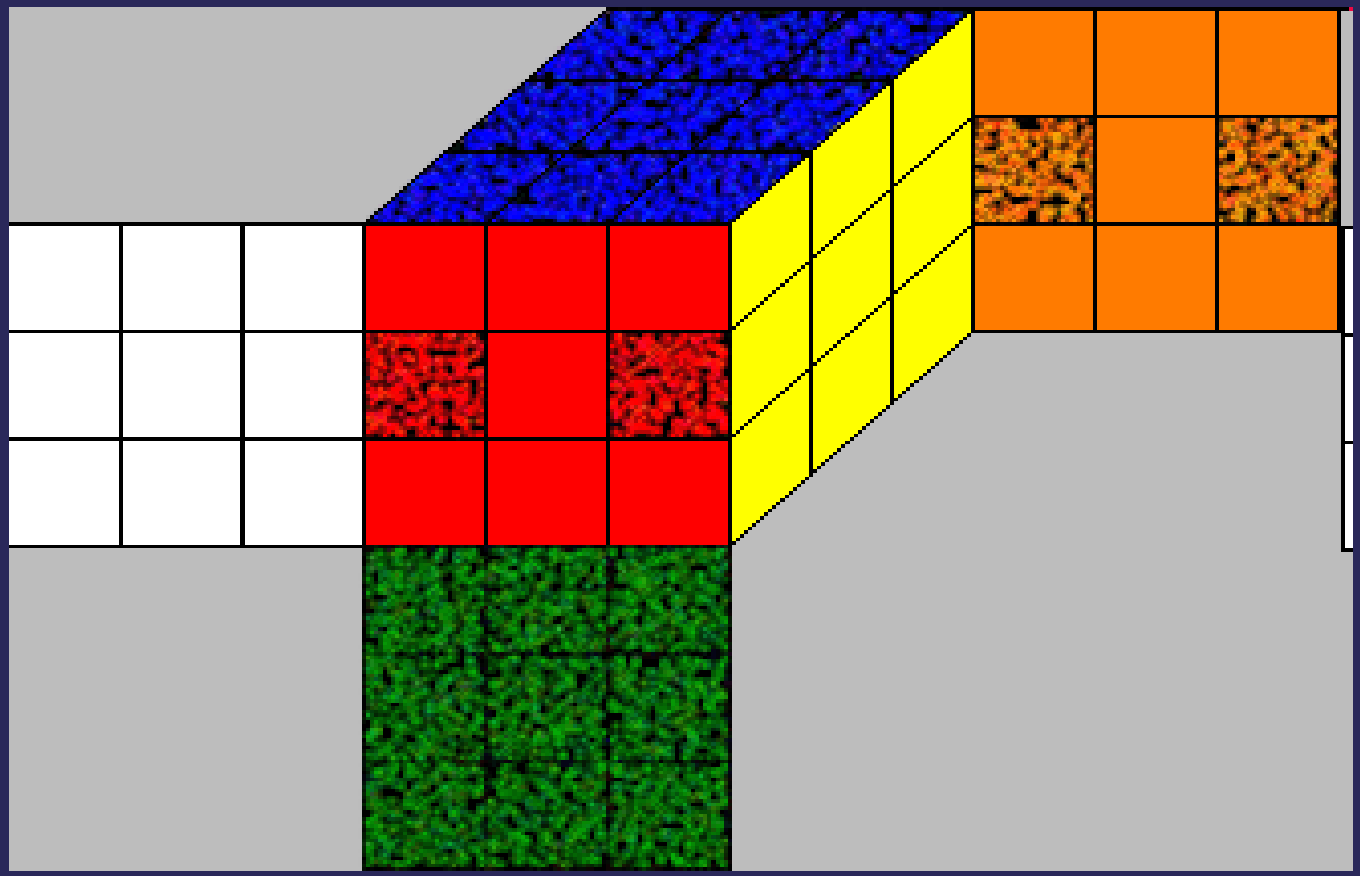
More Research

- Cube Explorer
  - www.kociemba.org/cube.htm
  - Herbert Kociemba
- Described an improvement to an earlier algorithm for solving a Rubik's Cube

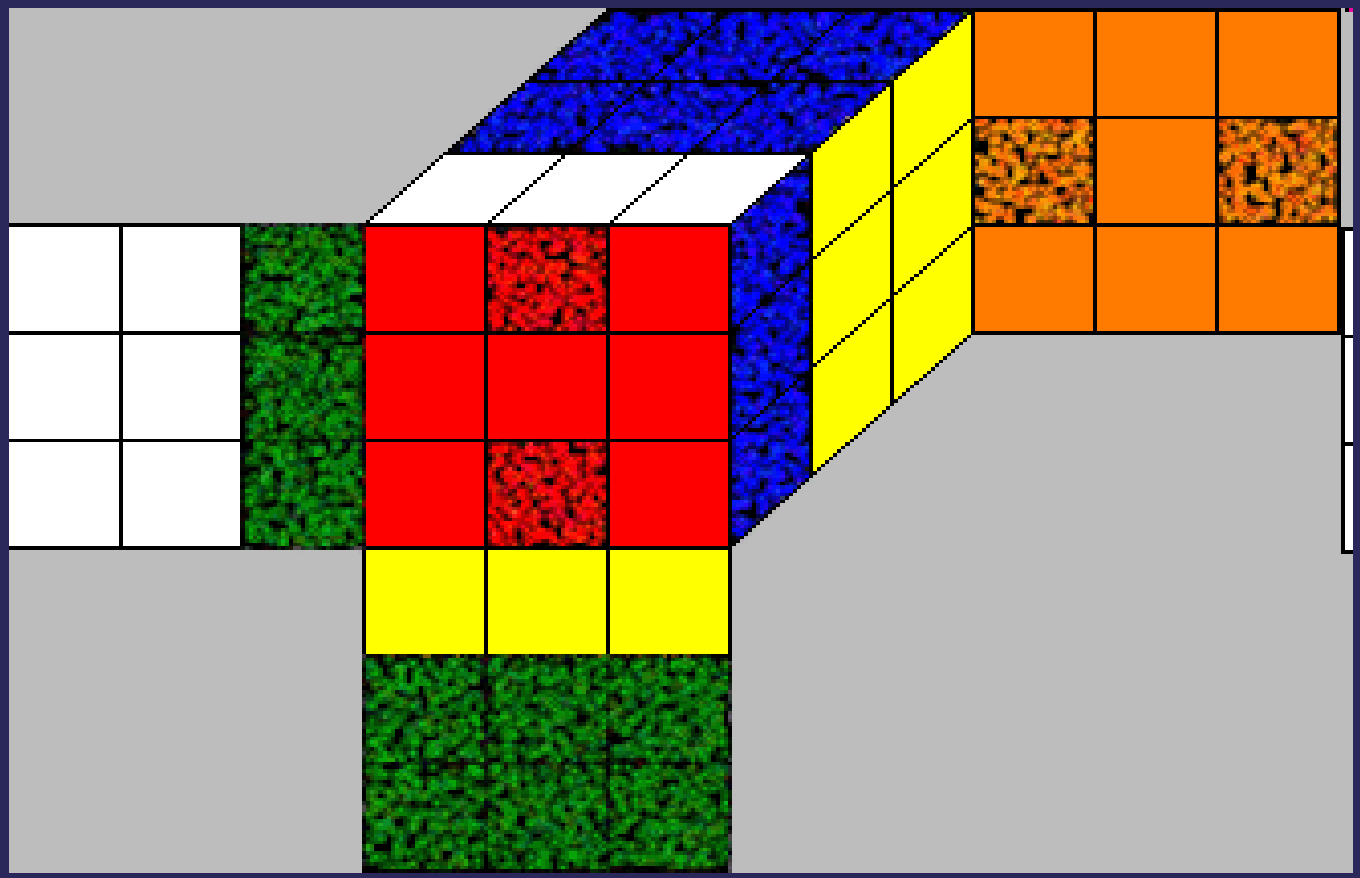
Background

- Describing a cube based on "facelets" is inefficient
- Uses "cubies" to describe the cube
  - Corner Cubies
  - Edge Cubies
- Each cube represented as a permutation of cubies

Background



Background



Corner Cubies

- Contain two major pieces of information
  - Original Position (name)
  - Orientation
- 8 Corner Cubies:

  URF, UFL, ULB, UBR,

  DFR, DLF, DBL, DRB

Edge Cubies

- Contain Similar information
  - Original Position (name)
  - Orientation
- 12 Edge Cubies:

  UR, UF, UL, UB

  DR, DF, DL, DB,

  FR, FL,

  BL, BR

Moves

- Moves are implemented by "multiplying" two permutations (i.e. applying one permutation to another)
- Very efficient system, implemented as arrays

Coordinate System

- Proposes a coordinate system to represent cubes
- Each permutation represented by a set of unique coordinates:
  - Corner Orientation
  - Corner Permutation
  - Edge Orientation
  - Edge Permutation

Coordinate System

- Example: Corner Orientation
  - Calculated by representing the orientations of the corner cubies in a base 3 system (i.e. 2001100)
  - Convert to a decimal system ($2*3^6 + 0*3^5 + 0*3^4 + 1*3^3 \ldots = 1494$)
  - Orientation of eighth cubie irrelevant, sum of all orientations must be divisible by 3 (in this case, it is 2)

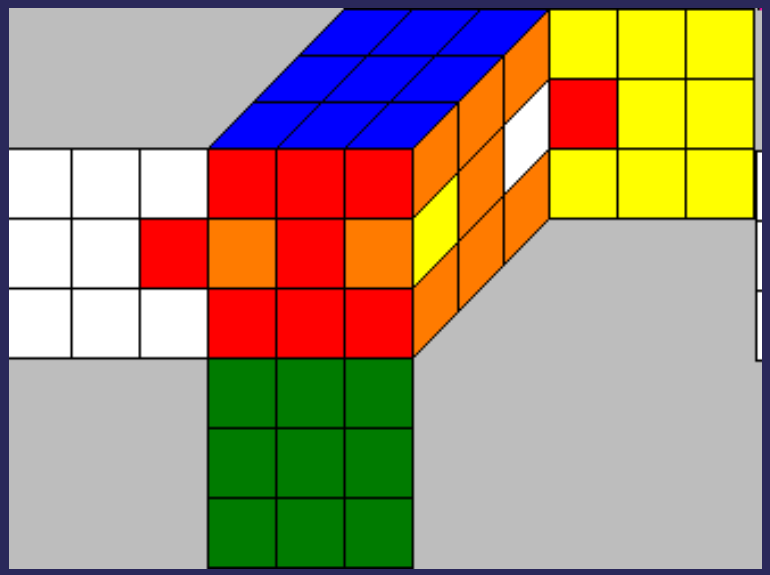
Coordinate System

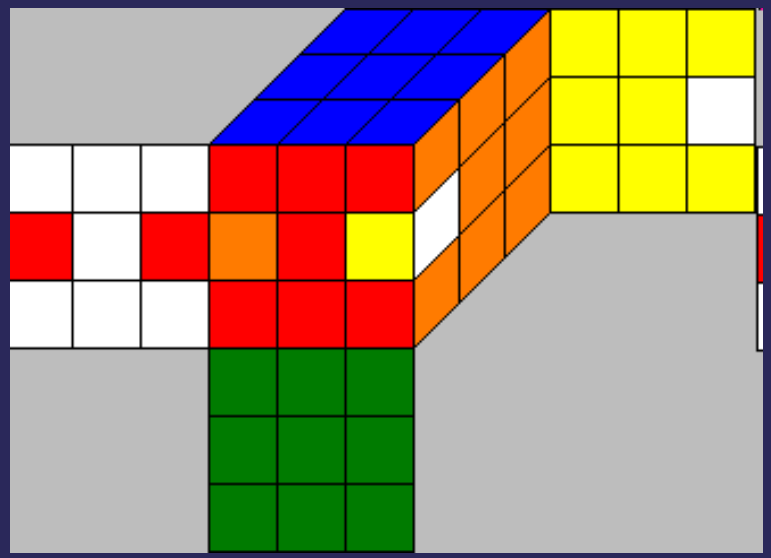
- Range of Values (Interval Notation)
  - Corner Orientation: [0, 2187)
  - Edge Orientation: [0, 2048)
  - Corner Permutation: [0, 40,320)
  - Edge Permutation: [0, 479,001,600)
- 43,252,003,274,489,856,000 possible permutations
- Other coordinates are used in algorithm, based on these four

Symmetries

- Number of possible combinations reduced by natural symmetries present in a cube
- 48 Possible Symmetries
- Symmetric cubes require the same set of moves to solve

Symmetries





Symmetries

- Raw coordinates transformed into Symmetry coordinates using these symmetries
- Smallest raw coordinate of a symmetric set used as representant
- Move applied to a representant can be transformed to match the cube

Two-Phase Algorithm

- Solves a cube in two phases:
  - Phase 1: Bring orientations of all corner and edge cubies to 0, and isolate the edge cubies that belong between the top and bottom faces in that slice
  - Phase 2: Using a limited set of possible moves, bring the locations of the corner and edge cubies back to their starting positions

Two-Phase Algorithm

- More Precisely:
  - Phase 2 solves any cube in the group created by applying only 180° turns to the Front, Back, Right and Left faces, and any turn to the Top and Bottom faces from a solved cube
  - Phase 1 transforms any cube into a member of that group

Two-Phase Algorithm

- Accomplished by three pre-calculated parts:
  - Symmetry Coordinate Tables
  - Move Tables
  - Pruning Tables

Symmetry Coordinate Tables

- Contains the information to transform a Symmetry Coordinate to a representant
- Also can take a raw coordinate and return the Symmetry Coordinate
- Calculation takes ~6 minutes

Move Tables

- Pre-calculated tables describing how each coordinate is transformed by each possible move
- Each coordinate is independent, so calculation is very simple
- Result of move on representant stored for Symmetry Coordinates
- Calculation takes ~8 minutes

Pruning Tables

- Provide a lower bound on the number of moves needed to reach a solution
  - In Phase 1, this number is absolute (i.e. a possible solution exists using this many moves)
  - In Phase 2, this number is a guide toward a solution, but not a guarantee

Pruning Tables

- Pre-calculated in reverse:
  - Take a solved cube, apply all possible moves, and store 1 in the positions for each resulting cube
  - For each cube containing 1, apply all possible moves and store 2 in the positions for each resulting cube if not already occupied
  - Repeat until finished

Pruning Tables

- Very Large Tables:
  - Phase 1: 140, 908,410 elements
  - Phase 2: 111,605,760 element
  - Approx 250 MB stored on disk
- Allow very quick lookups since stored as arrays (for speed)
- Calculation takes ~12 minutes

My Two-Phase Implementation

- Simplified his algorithm
- Phase 1:
  - Looks up the pruning depth of the cube
  - Applies each possible move in succession and checks if pruning depth is reduced
  - If so, repeat with the resulting cube
  - Since lower bound is absolute, solution found very quickly

My Two-Phase Implementation

- Phase 2:
  - Looks up the pruning depth of the cube
  - Applies each move possible in Phase 2 to the cube in succession
  - Adds each cube with a lower pruning depth to the queue
  - Repeats process with each cube in the queue until a solution is found

My Two-Phase Implementation

- One Problem:
  - Phase 2 algorithm ignores a coordinate (location of edge cubies in the slice between Top and Bottom faces)
  - Solution: Ignore it!
  - Simply check if it is correct, if not, discard solution and try again

My Two-Phase Implementation

- If no solution has been found and the queue is empty, backtrack from initial state and start again
  - Apply all possible moves to the cube, and add all resulting cubes (regardless of pruning depth) to the queue
  - Can backtrack as many as 18 moves to find a solution (maximum pruning depth is 18 in Phase 2)

My Two-Phase Implementation

- Difficulties
  - Mastering Symmetries
  - Understanding structure of Symmetry Coordinates
  - Figuring out how to handle ignored coordinate in Phase 2
  - Correctly adding entries to Pruning tables based on symmetries of cubes

My Two-Phase Implementation

- Difficulties
  - Optimizing pre-calculation of tables
    - Ordering pre-calculation to use one table to help calculate another
  - Restoring cube from Phase 1
    - He used other coordinates, but I found it simpler to run the solution on the cube itself

My Two-Phase Implementation

- Average Solution Time: <1 second
- Solution for Phase 1 is always optimal for that phase
- Solution for Phase 2 is not always optimal for that phase
- However, solution found in Phase 1 affects length of solution in Phase 2

My Two-Phase Implementation

- My algorithm does not account for this.
- Sub-optimal solutions for Phase 1 may generate shorter solutions overall
- My interest lies only in quickly reaching a short solution, but not an optimal one

Future Work

- Add interactivity to GUI, allow users to walk through a solution step by step
- Add ability to search for sub-optimal solutions to Phase 1 in order to find shorter overall solutions
- Possibly try to implement Kociemba's Optimal Solver

Demonstration of Project

Thank You!

Russell Feldhausen
russfeld@ksu.edu