**Sudoku Cube (Assignment 1)**

**L14-4347 Ali Asghar**

**L14-4013 Asad Mahmood**

**L14-4149 Hamza Shah**

**States**

To represent a single state we used a 3D array (12x3x3). 6 for the faces i.e.

Up, Down, Left, Right, Front, Back. 5 is kept fixed in the center of all the states as per the rule.

As for the total states.

No. of Faces \* No. of Different Orientations = 6 \* 8! (since 5 is always fixed)

**Initial State**

It is give in a file named “InitialState.txt”. 3 consecutive lines represent a face of Sudoku cube. And there are 6 faces separated by an empty line given in the order Up, Left, Front, Right, Back, Down.

**Successor Function**

The Successor function takes in the current state of cube and the rotation that is to be applied and it gives the cube with that rotation applied on it. Following is the Format

“F” 🡪 Front **(Clockwise)** “F-” 🡪 Back **(Anticlockwise)**

“B” 🡪 Back “B-” 🡪 Back

“L” 🡪 Left “L” 🡪 Back

“R” 🡪 Right “R” 🡪 Back

“U” 🡪 Up “U” 🡪 Back

“D” 🡪 Down “D” 🡪 Back

**Goal State**

It is give in a file named “GoalState.txt”. 3 consecutive lines represent a face of Sudoku cube. And there are 6 faces separated by an empty line and they are all solved having numbers 1, 2, 3, 4, 5, 6, 7, 8, 9.

**Path Cost**

Path cost for every rotation is set to be 1.

**Algorithms**

**Breadth First Search (BFS) (Complete)**

We have applied BFS on it. Since the path cost of every rotation is constant so the BFS returns the most optimal path to the goal.

The Algorithm Returns a list of steps that we should take in order to solve the cube.

The orientation provided with this submission can be solved in 39 (s).

**Second Approach**

**Preprocessing**

An array of 6 x 18 is read from initial txt file where each row represents each face and contain all 9 digits along with their orientation. This array then gets mapped to 12 2-faced edges and 8 3-faced cubies where each edge is represented by 2 values and their orientations and each cubie is represented by 3 values and their orientations. The same is done for final txt file.

**Initial State**

The initial state holds the position of each cubie and edge giving array of 20 according to initial text file configuration.

**States**

Each state is represented by the positions of cubies and edges giving array of 20

**Successor Function**

Function takes in positions of cubies and edges and perform one of 18 possible actions resulting in new positions of cubies and edges.

**Path Cost**

Cost of each step is 1. Cost of complete path is equal to the number of actions performed to solve the cube.

**Algorithm**

**Iterative Deepening A\*** is the algorithm used to solve the cube with pruning branches whose cost exceed threshold. Threshold is initialized according to step cost which then increases iteratively.

* **Space Complexity**

Since IDA\* is essentially Depth first search space complexity is not an issue here. For algorithms like BFS, A\* space complexity becomes a serious issue when number of steps increases.

* **Time Complexity**

Time Complexity of IDA\* is resolved using well defined 3D Manhattan distance heuristic.

I

**Heuristic**

**3D Manhattan Distance of Cubies and Edges**

Manhattan distance of each cubie and edge is computed based on its position in current state and its position in final state on the cube. After calculating that distance, number of out of placed face values of that correctly placed cubie is computed which is then added to Manhattan Distance and form the heuristic value for that state.