

## Informatics College Pokhara



### Artificial Intelligence

### CU6051NI

### Individual Coursework-2

### Prototype for Rubik's cube solver

#### **Submitted By:**

Name: Bimal Khatri

London Met ID: 20048661

Group: C4

Date: 11<sup>th</sup> January, 2023

#### **Submitted To:**

Mr. Abhinav Dahal

Module Leader

Artificial Intelligence

### **Acknowledgement:**

I would like to express my deep gratitude to our lecturer Mr. Abhinav Dahal sir and Tutor Mr. Mahesh Dhungana sir for taking out his valuable time to hear, advice, encourage and help me complete this coursework in time. I would also like to thank all my teacher for their support and guidance. It would not be possible for me to complete this coursework without their help. Finally, I would like to extend my thanks to my friends and family who indirectly helped me in many ways to complete this report and coursework.

### **Abstract:**

This report provides the in-depth analysis of problem solving in artificial intelligence along with a working prototype of Rubik's cube solver, solving Rubik's cube with less maneuvers or moves based in Kociemba's algorithm which is also known as two-phase algorithm. The working of Kociemba's algorithm and the working mechanism of the solver is further explained in the report. The pseudocode, state transition diagram, and flow chart demonstrating how Kociemba's algorithm solves the scrambled Rubik's cube are also included in this report.

## Table of Contents:

1. Introduction: .....	1
1.1. Introduction to AI:.....	1
1.2. Introduction to Heuristic search and problem solving in AI: .....	3
1.2.1. Introduction to problem domain (Rubik's cube): .....	4
2. Background:.....	6
2.1. Journals:.....	6
2.1.1. Finding Optimal Solutions to Rubik's Cube using pattern Databases: .....	6
2.1.2. Algorithms for solving Rubik's cube: .....	7
2.1.3. 2 approaches in solving Rubik's cube with Hardware-Software Co-design: ..	8
2.1.4. The most efficient algorithm to solve a Rubik's cube: .....	9
2.2. Website: .....	10
2.2.1. Online Rubik's Cube Solver - 3D 3x3x3:.....	10
2.2.2. Cubesolver: .....	11
2.2.3. Rubik's Cube Explorer: .....	12
3. Solution:.....	13
3.1. Approach to solving problem:.....	13
3.2. Explanation of algorithms used: .....	14
3.2.1. Kociemba's algorithm: .....	14
3.2.2. Iterative deepening A* (IDA*): .....	16
3.3. Pseudocode: .....	17
3.4. Flow Chart:.....	21
3.5. State Transition Diagram:.....	22
3.6. Explanation of the development process: .....	23
3.7. Achieved results: .....	25
4. Conclusion: .....	27
5. Bibliography: .....	29

## Table of Figures:

Figure 1: Different state of Rubik's cube (scrambled / solved) .....	4
Figure 2: Finding Optimal Solutions to Rubik's Cube Using Pattern Databases.....	6
Figure 3: Algorithms for solving Rubik's cube .....	7
Figure 4: Two approaches in solving Rubik's cube with Hardware-Software Co-design. 8	
Figure 5: The most efficient algorithm to solve a Rubik's cube .....	9
Figure 6: Home page of online Rubik's Cube Solver - 3D 3x3x3 .....	10
Figure 7: Home page of Cubesolver .....	11
Figure 8: Home page of Rubik's Cube Explorer.....	12
Figure 9: Flowchart for solving Rubik's cube using Kociemba's algorithm .....	21
Figure 10: State transition diagram showing different state for solving cube. ....	22
Figure 11: Vs-code .....	23
Figure 12: Python .....	24
Figure 13: Creating files for storing maneuvers .....	25
Figure 14: Solving the randomly scramble cube. ....	25
Figure 15: loading the files and showing the interface .....	26
Figure 16: Showing error when invalid color string is inserted.....	26

## **1. Introduction:**

This is a complete report of the coursework 2 where different research and analysis of a particular AI topic( problem solving in AI) and a final prototype design of fully solvable Rubik's cube is made and done.

### **1.1. Introduction to AI:**

Artificial Intelligence (AI) is a modern engineering method for making machines or computers to think or use their intelligence like people do by copying qualities and by learning to take appropriate decisions and to perform assigned tasks properly. Simply, an artificial intelligence is the the replication of human intelligence processes by machines, especially computer systems. AI provides ways to make machines intelligent. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision. The companies like Facebook, Google, Microsoft, IBM, etc. have done remarkable work in the field of AI and are spending millions and billions of dollars in its development and research. (Verma, 2020)

The goal of AI is to explore the ways onto a machine that can behave, think, and reason just like a person. AI aids in making more intelligent decisions than humans, doing deeper data analyses and speeding up the work performance. (kelley, 2022)

There are several types of AI. However, each of them may be grouped according to the capabilities and functionalities of an AI.

#### **a. Based on capabilities:**

Based on capabilities there are following three types of AI:

I. Weak AI or Narrow AI:

Narrow AI is a type of AI which is able to perform a dedicated task with intelligence. The most common and currently available AI is Narrow AI in the world of Artificial Intelligence. Some Examples of Narrow AI are playing chess, purchasing suggestions on e-commerce site, self-driving cars, speech recognition, and image recognition.

II. General AI:

General AI is a type of intelligence which could perform any intellectual task with efficiency like a human. As systems with general AI are still under research, and it will take lots of efforts and time to develop such systems.

III. Super AI:

Super AI is a level of Intelligence of Systems at which machines could surpass human intelligence, and can perform any task better than human with cognitive properties. It is an outcome of general AI. Super AI is still a hypothetical concept of Artificial Intelligence. Development of such systems in real is still world changing task.

b. Based on functionality:

Based on capabilities there are following three types of AI:

I. Reactive Machines:

Purely reactive machines are the most basic types of Artificial Intelligence. This type of AI systems do not store memories or past experiences for future actions. IBM's Deep Blue system, Google's AlphaGo are some examples of reactive machines.

## II. Limited Memory

Limited memory machines can store past experiences or some data for a short period of time. Self-driving cars are the examples of Limited Memory systems.

## III. Theory of Mind

Theory of Mind AI should understand the human emotions, people, beliefs, and be able to interact socially like humans.

## IV. Self-Awareness

Self-awareness AI is the future of Artificial Intelligence. These machines will be super intelligent, and will have their own consciousness, sentiments, and self-awareness. Self-Awareness AI does not exist in reality still and it is a hypothetical concept. (JavaTPoint, 2021)

### **1.2. Introduction to Heuristic search and problem solving in AI:**

Problem solving and heuristic search was chosen as AI topic for this coursework. Problem-solving is commonly known as a strategy for achieving objectives or resolving particular situations. According to computer science, problem-solving refers to artificial intelligence techniques, including various techniques such as forming efficient algorithms, heuristics, and performing root cause analysis to find desirable solutions. In Artificial intelligence, problem-solving often involves investigating a solution to a problem using logical algorithms, making use of polynomial and differential equations, and executing them using modeling paradigms. A same problem can have several solutions, each of which is accomplished using a different heuristic. (pdfCo, 2022)

Search is a universal problem-solving mechanism in artificial intelligence (AI). The problems that have been addressed by AI search algorithms fall into three general classes: single-agent pathfinding problems, two-player games, and constraint-satisfaction problems. Rubik's cube, sliding puzzles, the travelling salesman problem, etc. comes under single-agent pathfinding problems. Chess,



checkers, Othello, etc. comes under two player game and at last the game like 8 queen problem comes under constraint-satisfaction problem. With the rapid development in the field of ai, different search algorithms are developed to solve different kinds of searching problems. Some of the searching algorithm includes brute force search( BFS, DFS, Bidirectional search), heuristic search(A\* algorithm, Iterative-deepening A\*, pure heuristic search, greedy search),etc.(Stefan Edelkamp, 2008)

In this coursework heuristic search is used to solve the problem (i.e., solving Rubik cube). A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time. Heuristics search also known informed search use basic domain knowledge and comprehend available information regarding a specified problem as a guideline for optimal solutions. In order to solve larger problems, domain-specific knowledge must be added to improve search efficiency. (pdfCo, 2022)

### 1.2.1. Introduction to problem domain (Rubik's cube):

For this coursework of Artificial intelligence, solving a Rubik's cube is taken as a problem domain. The Rubik's cube is a 3-D combination puzzle invented in 1974 by Ernő Rubik, a Hungarian sculptor and architecture professor. The Rubik's cube, sometimes referred to as the "Magic Cube," has gained popularity around the globe for over a generation. Individuals of all ages have spent hours, weeks or months trying to solve the cube. As a result, over 350 million Rubik's cubes have been sold worldwide, making it the world's most famous, bestselling and well-known puzzle to have a rich underlying logical, mathematical structure.

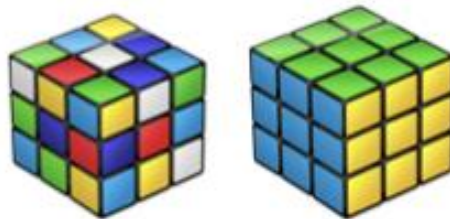


Figure 1: Different state of Rubik's cube (scrambled / solved)

The standard version of Rubik's cube(3x3x3) is a 3-D combination puzzle with six faces covered by nine stickers where each of the faces have one of six colors (i.e., white, red, blue, orange, green and yellow). The 3x3x3 Rubik's Cube is composed of twelve edge cubelets, eight corner cubelets, and six center cubelets making total of twenty-six cubelets. The puzzle is scrambled by making several random moves, where any face can be twisted 90,180 or 270 degrees. The task is to restore the cube to its goal state, where all the squares on each side of the cube are of the same color. An algorithm (a series of actions intended to get the puzzle closer to being solved) is required to solve a scrambled Rubik's cube. For a typical person, such sequences often need more than fifty to one hundred movements. Theoretically, every cube can be solved in fewer than twenty steps. Each corner piece has three visible faces, whereas each edge piece has two. Any given edge piece or corner piece can be rotated in any edge position. However, some permutation sets are not feasible. This is further explained on solution section.(Sher, 2014)

## 2. Background:

Solving a complex problem by a machine with minimum moves is not a simple task and to design such type of device requires a lot of research and dedication. To make research on the project different sources such as journals, website, etc. were used. Some of the journals and website that I used to research on solving Rubik's by computer as given below:

### 2.1. Journals:

The following journals were studied thoroughly after choosing the problem domain:

#### 2.1.1. Finding Optimal Solutions to Rubik's Cube using pattern Databases:

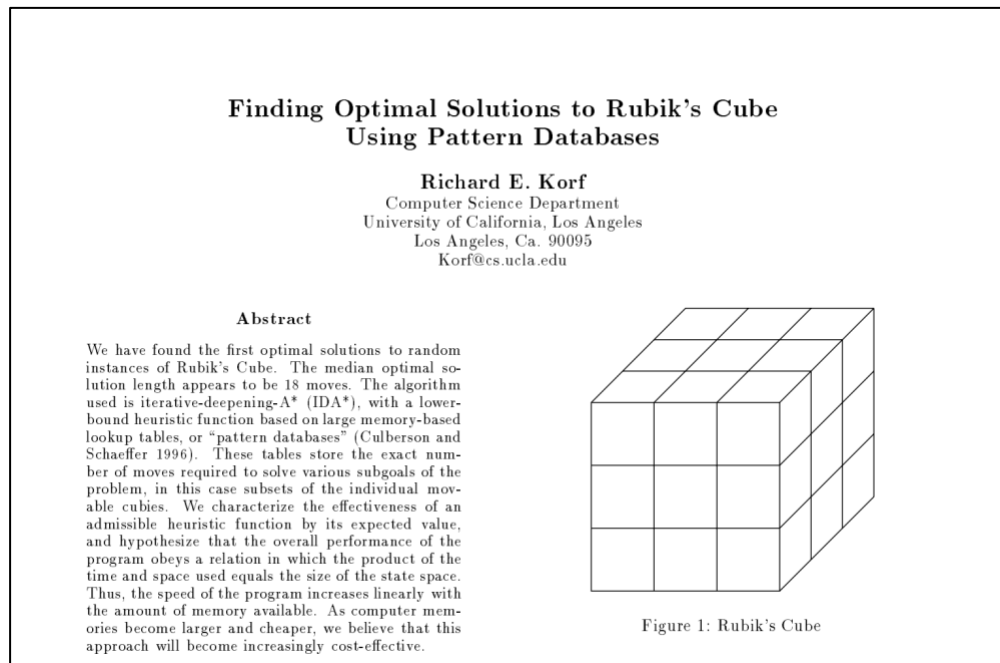


Figure 2: Finding Optimal Solutions to Rubik's Cube Using Pattern Databases

We have found first optimal solutions to random instances of Rubik's Cube. The median optimal solution length appears to be 18 moves. The algorithm used is iterative-deepening-A\* (IDA\*), with a lower- bound heuristic function based on large memory-based lookup tables, or "pattern databases". These tables store the exact number of moves required to solve various subgoals of the problem, in this case subsets of the individual movable cubies. We characterize the effectiveness of an admissible heuristic function by its expected value and hypothesize that the

overall performance of the program obeys a relation in which the product of the time and space used equals the size of the state space. Thus, the speed of the program increases linearly with the amount of memory available. As computer memories become larger and cheaper, we believe that this approach will become increasingly cost-effective. (Korf, 1997)

### 2.1.2. Algorithms for solving Rubik's cube:

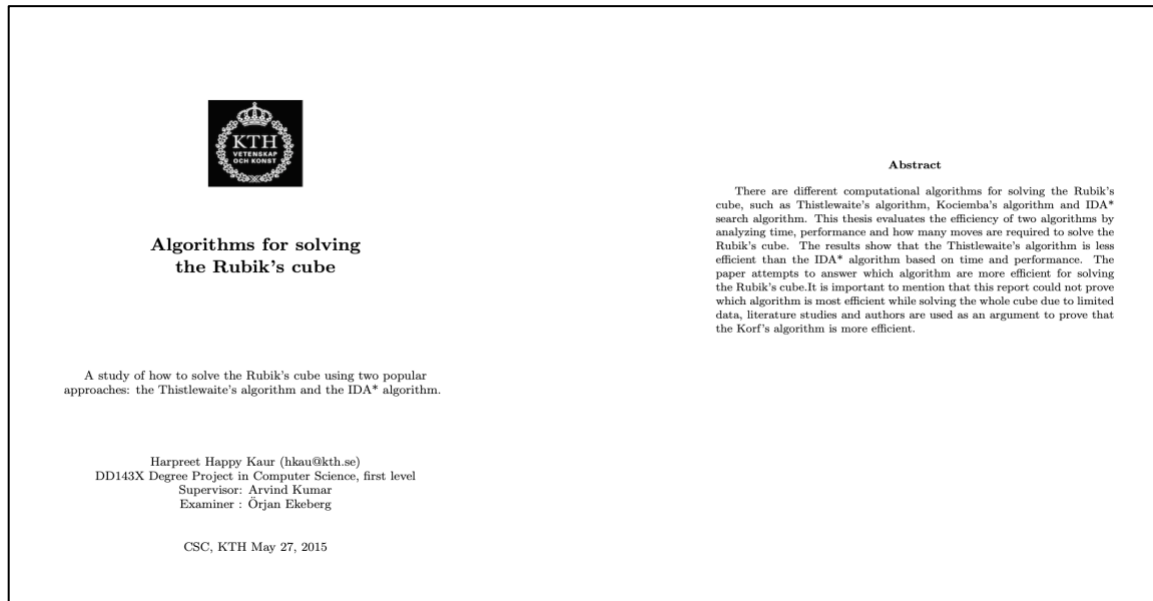


Figure 3: Algorithms for solving Rubik's cube

The Thistlewaite's method, Kociemba's algorithm, and IDA\* search algorithm are a few examples of computational algorithms for solving the Rubik's cube. The effectiveness of two algorithms is assessed in this thesis by looking at the amount of time, performance, and movements needed to solve the Rubik's cube. Based on time and performance, the results demonstrate that the Thistlewaite's approach is less effective than the IDA\* algorithm. The purpose of this study is to determine which method is more effective in solving the Rubik's cube. Due to the restricted data, it is crucial to note that this study was unable to demonstrate which method is more effective in solving the entire cube. Instead, literature reviews and authors are utilized to demonstrate the superiority of the Korf's approach. (Kaur, 2015)

### 2.1.3.2 approaches in solving Rubik's cube with Hardware-Software Co-design:

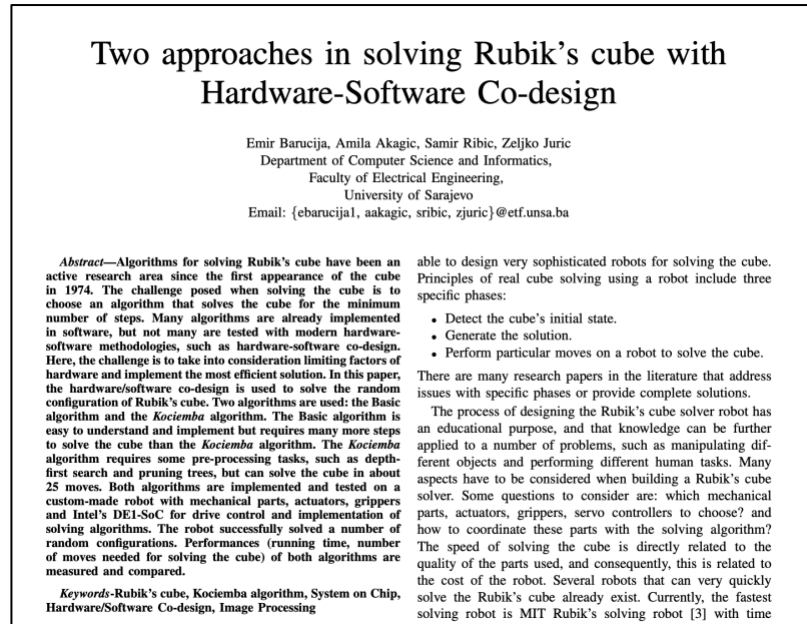


Figure 4: Two approaches in solving Rubik's cube with Hardware-Software Co-design

Since the Rubik's cube made its debut in 1974, research into algorithms for solving the puzzle has been vigorous. The difficulty in solving the cube is in selecting an algorithm that solves it in the fewest steps possible. Although many algorithms have already been implemented in software, few have been validated using contemporary hardware-software techniques, such as hardware-software co-design. Here, the problem is to develop the most effective solution while taking hardware limitations into account. In this study, the random configuration of the Rubik's cube is solved through hardware/software co-design. The Basic algorithm and the *Kociemba* algorithm are the two employed. The Basic approach is simple to comprehend and use, but it takes a lot longer than the *Kociemba* algorithm to solve the cube. The *Kociemba* method can solve the cube in roughly 25 moves, although various pre-processing activities like depth-first search and trimming trees are required. Both algorithms are implemented and tested on a custom-made robot with mechanical parts, actuators, grippers and Intel's DE1-SoC for drive control and implementation of solving algorithms. The robot successfully solved several random configurations. Performances (running time, number of

moves needed for solving the cube) of both algorithms are measured and compared. (Emir, et al., 2020)

#### 2.1.4. The most efficient algorithm to solve a Rubik's cube:

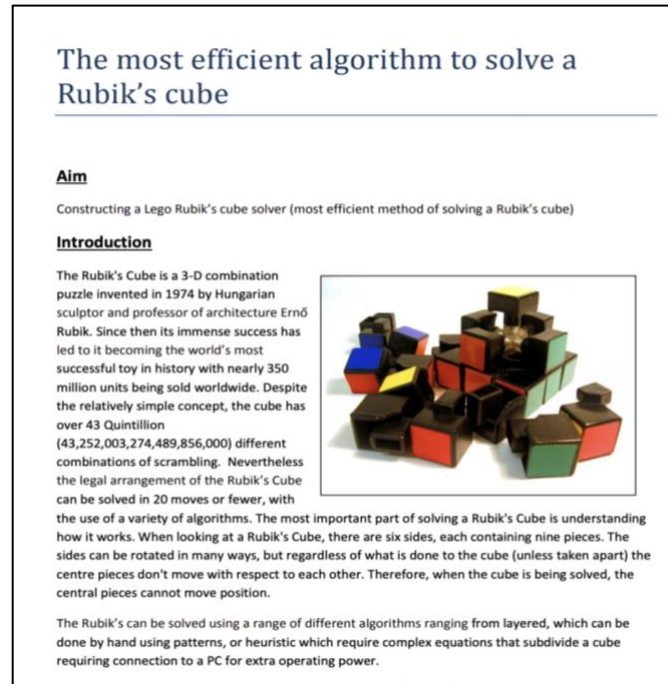


Figure 5: The most efficient algorithm to solve a Rubik's cube

The above report "The most efficient algorithm to solve a Rubik's cube" is a report of science Research project written by Justin Marcellienus. In the project a lego Rubik's cube solver was made and different algorithm that were developed to solve Rubik's cubes were tested and were analysed. The algorithm that were used in the project were Thistlewaite Algorithm, Kociemba Algorithm and Korf Algorithm. . Kociemba's Algorithm was an improvement on Thistlethwaite's algorithm. Korf's Algorithm was developed by Richard Korf in 1997. After completing the experiment, it was found that the Thistlewaite algorithm was the least effective method of solving the Rubik's cube requiring an average of 42 moves and 3 mins and 48 seconds, Kociemba algorithm proved to be the 2nd most effective algorithm, requiring an average of 28 moves and 2 mins and 32 seconds and Korf algorithm proved to be the most effective algorithm, requiring an average of 20 moves and 2 mins and 5 seconds. (Marcellienus, 2014)

## 2.2. Website:

The following are some of the websites that are used for solving the scrambled Rubik's cube.

### 2.2.1. Online Rubik's Cube Solver - 3D 3x3x3:

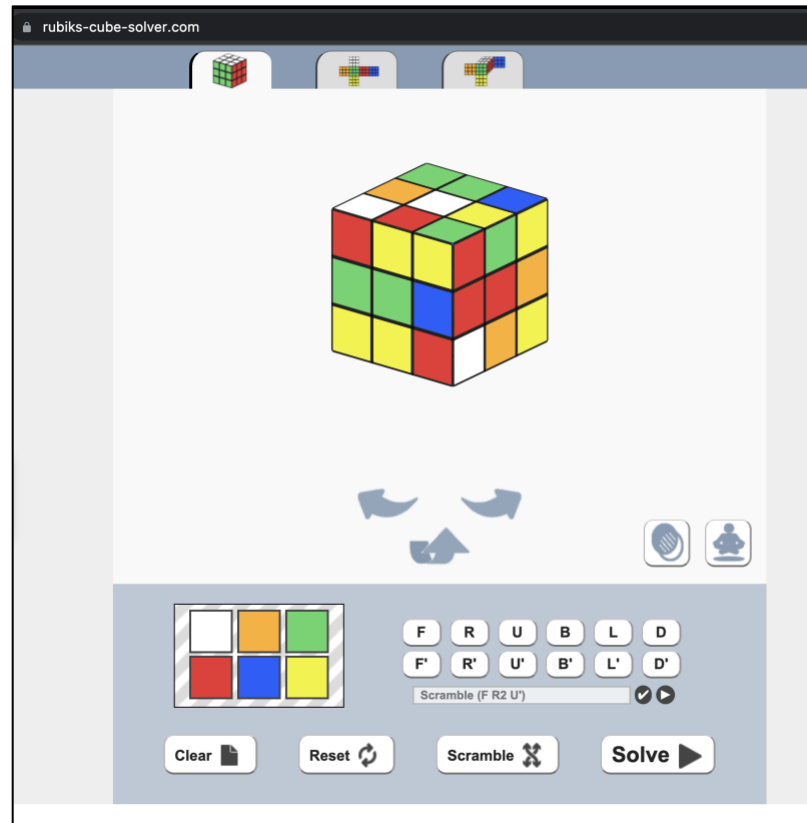


Figure 6: Home page of online Rubik's Cube Solver - 3D 3x3x3

Online Rubik's cube solver is a online platform that calculates the steps needed to solve a scrambled Rubik's Cube. To solve a scrambled cube one can enter the configuration of their real or virtual scrambled Rubik's cube. This platform is quicker and calculates optimal moves required to solve a scrambled Rubik's cube. The platform is user-friendly and determines the fewest possible steps for 2x2x2, 4x4x4, and Pyraminx twisty puzzles as well. Simply, pressing the solve button will solve the supplied jumbled cube. The app can be accessed through the link (<https://rubiks-cube-solver.com/>).

### 2.2.2. Cubesolver:

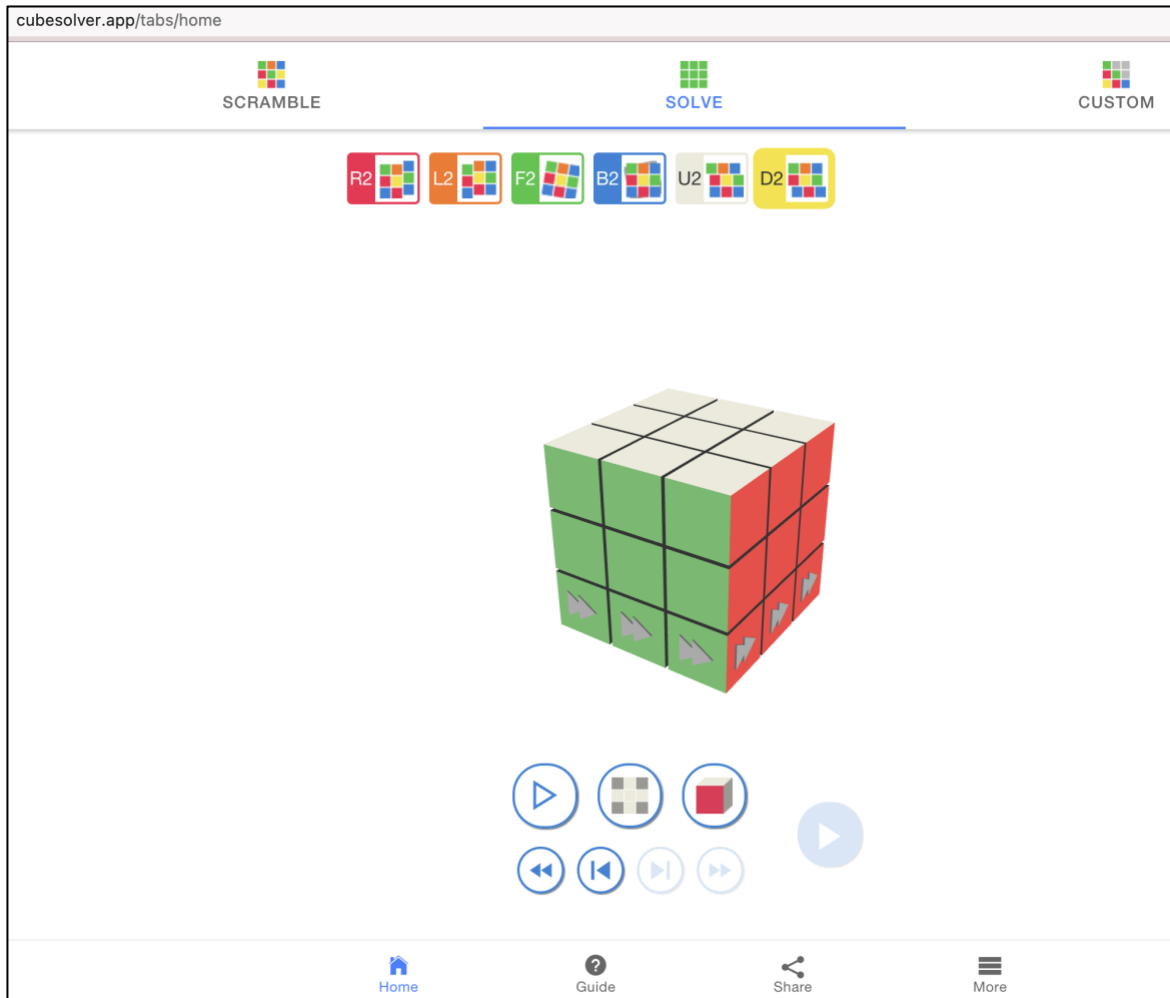


Figure 7: Home page of Cubesolver

Cubesolver is also an online platform to solve Rubik's cube. Like the above-mentioned app, Cubesolver also solves the 3x3x3 Rubik's cube. One can enter their custom Rubik's configuration to be solved. This app is also user-friendly and provides a full guide for new users. This app doesn't solve the Rubik's cube as fast as the above-mentioned app. The app can be accessed through the link (<https://cubesolver.app/tabs/home>)



### 2.2.3. Rubik's Cube Explorer:

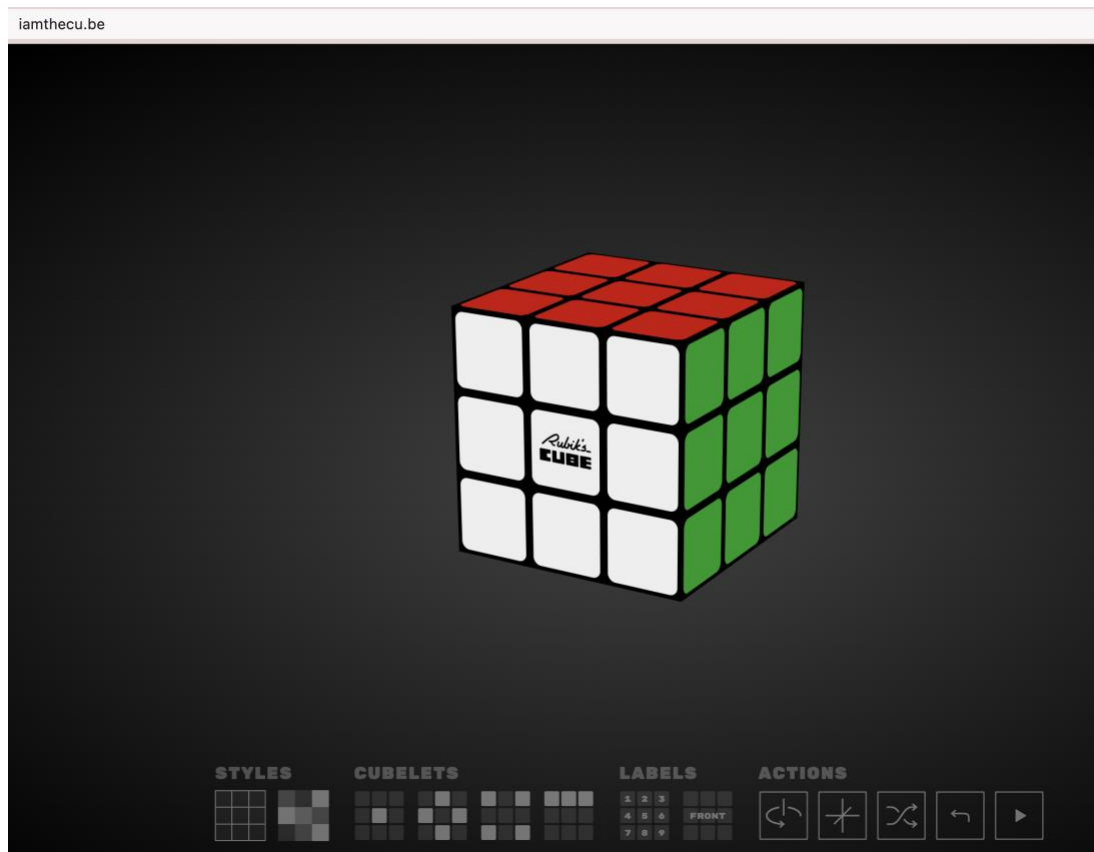


Figure 8: Home page of Rubik's Cube Explorer

Rubik's Cube Explorer is also an online platform for solving Rubik cube. Beside solving the cube, an interactive presentation helps understand the basics of Rubik's Cube. The app runs on Cuber framework. The app is attractive and free to use. Similar like above one can scramble the cube to solve it. The app can be accessed through (<https://iamthecu.be/>)

### 3. Solution:

#### 3.1. Approach to solving problem:

As we already know a standard 3x3x3 Rubik's cube consists of total of 8 edges and 12 center pieces. So, there are  $8!$  permutations of the eight corners and  $12!$  permutations of the twelve-center piece. Also, all the 8 edges pieces and 12 center pieces can be arranged in three and two possible orientations respectively. So each permutation of the edge pieces and center pieces have  $3^8$  and  $2^{12}$  arrangements respectively. From this the total number of states can be concludes as  $8! \times 3^8 \times 12! \times 2^{12}$ . However, these permutations consist of both valid and invalid configuration of cubies. Since from the three orientation of edge pieces only 1 is valid and similarly from two orientation of center pieces only 1 is valid. Also, both the edge piece and center piece can be swapped to different position. So, taking this into consideration, altogether there can be  $\{(8! * 3^8 * 12! * 2^{12})/12\}$  (i.e.,  $4.3 \times 10^{19}$ ) valid permutation of solvable Rubik's cube. So, altogether there can be  $4.3 \times 10^{19}$  states reachable from one state. As, the humans have limited memory, so it is not possible for any human being to physically count each permutation. And solving problems such as finding optimal solutions cannot be solved only by humans. Therefore, computers act as a utilitarian tool to solve problems such as Rubik's cube puzzle as computers are good at numeric calculations, memorizing large sets of data and can search quickly. And to do so, a computer needs certain algorithm to solve problem such as Rubik's cube. A Rubik's cube is said to be solved if all it's faces match the color of its center piece. (Kaur, 2015)

There are several computational approaches for solving the Rubik's cube. However, the standard approaches for solving the Rubik's cube are Thistlethwaite's algorithm, Kociemba's algorithm and Korf's algorithm.

The Thistlewaite's algorithm was found by Morwen Thistlethwaite. This algorithm works by restricting the positions of the cubes into groups of cube positions that can be solved using a certain set of moves. The groups are:

$$G_0 = \langle L, R, F, B, U, D \rangle$$

$$G1 = \langle L, R, F, B, U2, D2 \rangle$$

$$G2 = \langle L, R, F2, B2, U2, D2 \rangle$$

$$G3 = \langle L2, R2, F2, B2, U2, D2 \rangle$$

$$G4 = \{I\}$$

The L, R, F, B, U, D in the above groups represent left, right, front, back, up and down face turn of the Rubik's cube. All potential Rubik's Cube positions are contained in G0. G1 contains all positions that can be reached (from the solved state) with quarter turns of the left, right, front and back sides of the Rubik's Cube, but only double turns of the up and down sides. G3 contains the positions are restricted to ones that can be reached with only double turns of the front, back, up and down faces and quarter turns of the left and right faces. In G3, the positions can be solved using only double turns on all sides. The last group, G4, only has one position, which is the cube's solved state.

Similarly Korf's algorithm is another algorithm to solve the rubik's cube founded by Richard Korf. This algorithm make use of Iterative deepening A star(IDA\*) algorithm to find the optimal solution for Rubik's cube.

And at last Kociemba's algorithm is the improved version of Thistlewaite's algorithm. Unlike Thistlewaite's algorithm, Kociemba's algorithm contains only 2 groups. Kociemba's algorithm was found by Herbert Kociemba in 1992. (De, 2015)

In this coursework Kociemba's algorithm is used for finding the solution to solve the Rubik's cube.

### **3.2. Explanation of algorithms used:**

#### **3.2.1. Kociemba's algorithm:**

Kociemba's algorithm is one of the algorithm that is used to find the optimal solution for solving the scrambled Rubik's cube. Kociemba's algorithm is created by Herbert Kociemba in 1992. Kociemba's algorithm is improved version of Thistlethwait's algorithm. As we already know from the above section that

Thistlethwait's algorithm works by restricting the positions of the cubes into 4 different groups of cube positions. Just like that Kociemba's algorithm works by restricting the positions of the cube into 2 different groups of cube position. The groups are:

$$G_0 = \langle U, D, L, R, F, B \rangle$$

$$G_1 = \langle U, D, L^2, R^2, F^2, B^2 \rangle$$

$$G_2 = \{I\}$$

Similar like Thistlethwaite's algorithm this algorithm also search through the right coset space  $G_1/G_0$  to move the cube to group  $G_1$ . Next it searched the optimal solution for group  $G_1$ . This algorithm is also called two phase algorithm as it solves the cube in two phases. The first phase moves from  $G_0$  to  $G_1 = \langle U, D, R^2, L^2, F^2, B^2 \rangle$ , and the second phase from  $G_1$  to  $G_2 = I$ . It is obvious that these phases are too large to construct a whole table. However, creating pruning tables and using IDA\* to solve each phase are both quite simple. (Scherphuis, 2015)

### **Phase 1 of kociemba's algorithm:**

In phase 1, the algorithm looks for maneuvers which will transform a scrambled cube to  $G_1$ . That is, the orientations of corners and edges have to be constrained and the edges of the UD-slice((between the U-face and D-face)) have to be transferred into that slice. In this abstract space, a move just transforms a triple  $(x,y,z)$  into another triple  $(x',y',z')$ . All cubes of  $G_1$  have the same triple  $(x_0,y_0,z_0)$  and this is the goal state of phase 1.

To find this goal state the program uses a search algorithm which is called iterative deepening A\* with a lowerbound heuristic function (IDA\*). In the case of the Cube, this means that it iterates through all maneuvers of increasing length. The heuristic function  $h_1(x,y,z)$  estimates for each cube state  $(x,y,z)$  the number of moves that are necessary to reach the goal state. It is essential that the function never overestimates this number. The heuristic allows pruning while generating the maneuvers, which is essential to reduce wait time before the goal state is

reached. The heuristic function  $h_1$  is a memory based lookup table and allows pruning up to 12 moves in advance.

### **Phase 2 of kociemba's algorithm:**

In phase 2 the algorithm restores the cube in the subgroup  $G_1$ , using only moves of this subgroup. It restores the permutation of the 8 corners, the permutation of the 8 edges of the U-face and D-face and the permutation of the 4 UD-slice edges. The heuristic function  $h_2(a,b,c)$  only estimates the number of moves that are necessary to reach the goal state, because there are too many different elements in  $G_1$ .

The algorithm does not stop when a first solution is found but continues to search for shorter solutions by carrying out phase 2 from suboptimal solutions of phase 1. For example, if the first solution has 10 moves in phase 1 followed by 12 moves in phase 2, the second solution could have 11 moves in phase 1 and only 5 moves in phase 2. The length of the phase 1 maneuvers increase and the length of the phase 2 maneuvers decrease. If the phase 2 length reaches zero, the solution is optimal and the algorithm stops. (kociemba, 2013)

### **3.2.2. Iterative deepening A\* (IDA\*):**

Iterative deepening A\* (IDA\*) is a graph traversal and path-finding algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph. It is a variant of iterative deepening depth-first search that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the A\* search algorithm. So, simply IDA\* is iterative deepening combined with a heuristic. Instead of going through the solutions blindly, the algorithm tries to go down path that look like they might have a shorter solution. This helps find shorter solutions earlier, which cuts down on the additional searching. (Scherphuis, 2015)

### 3.3. Pseudocode:

Pseudo code is a methodology that allows the programmer to represent the implementation of an algorithm. The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer. (geeksforgeek, 2022)

Pseudo code for applying Kociemba algorithm to solve Rubik's cube:

kociemba():

    Import solver as sv

    Input the cube configuration

    SolutionBlock = True:

    when(solutionBlock):

        If the configuration is valid:

            sv.solve(configuration)

            responseBlock = True:

            while(responseBlock):

                input response as y/n for continuing:

                if response = y:

                    responseBlock = False

                else

                    responseBlock = False

                    solutionBlock = False

    else:

        print("invalid input")

```

solve(colorConfig, length= 20, time= 3):
    if the cube configuration is not valid:
        return invalid message
    solution = []
    sym = get symmetries from symmetereies()
    if some rotational symmetry = true:
        tr = [0,3]
    else:
        tr = range(6)
    if some antisymmetric = true:
        tr = list(filter(lambda x: x < 3, tr))
    for l in tr:
        solverThread(colorConfig, i % 3, i // 3, max_length, timeout, s_time,
            solutions, terminated)
    s = ''
    if len(solution) > 0:
        for m in solution[-1]:
            s = s+ m + ' '
    return s + len(s)

```

SolverThread():

```

function Run():
    cb = None
    if no rotation:
        cb = cubie.CubieCube(self.cb_cube.cp, self.cb_cube.co,
                             self.cb_cube.ep, self.cb_cube.eo)
    else if rotation = 120degree:
        cb = cubie.CubieCube(sy.symCube[32].cp, sy.symCube[32].co,
                             sy.symCube[32].ep, sy.symCube[32].eo)
        multiply cb with the cb_cube restricted to edge and corner
        cb.multiply(sy.symCube[16])
    else if rotation = 240 degree:
        cb = cubie.CubieCube(sy.symCube[16].cp, sy.symCube[16].co,
                             sy.symCube[16].ep, sy.symCube[16].eo)
        multiply cb with the cb_cube restricted to edge and corner
        cb.multiply(sy.symCube[32])
    else if invert cube:
        cb = store the inverse of (cb)
    co_cube = representation of cb in coordinate form
    dist = distance to the subgroup H where flip,twist, slice orientation is 0
    for togo1 in range(dist, 20):
        sofar_phase1 = []
        position = co_cube.flip, co_cube.twist, co_cube.slice_orientaion;
        search(position, dist)

function search( position p; depth d )
    if d=0 then
        if subgoal reached and last move was a quarter turn of R, L, F, or
        B then
            Phase2start( p )
    elseif d>0 then

```



```
        if prune1[p]<=d then
            for each available move m
                search( result of m applied to p; d-1 )
            endfor
        Endif
    Endif
Endfunction

function Phase2start ( position p)
    for depth d from 0 to maxLength – currentDepth
        search_phase2( p; d )
    Endfor
Endfunction

function search_phase2( position p; depth d )
    if d=0 then
        if solved then
            Found a solution!
            maxLength = currentDepth-1
        elseif d>0 then
            if prune2[p]<=d then
                for each available move m
                    search_phase2( result of m applied to p; d-1 )
                Endfor
            Endif
        Endif
    Endif
Endfunction
```

### 3.4. Flow Chart:

A flowchart is a diagram that shows how a system, computer algorithm, or process works. They are frequently used in many different disciplines to examine, organize, enhance, and convey complex processes in clear, easy-to-understand diagrams. (lucidchart, 2020)

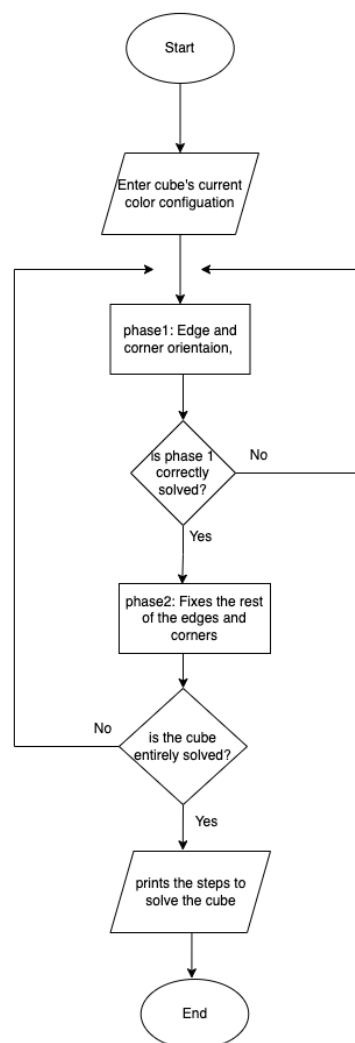


Figure 9: Flowchart for solving Rubik's cube using Kociemba's algorithm

### 3.5. State Transition Diagram:

State-transition diagrams describe all of the states that an object can have, the events under which an object changes state (transitions), the conditions that must be fulfilled before the transition will occur (guards), and the activities undertaken during the life of an object (actions). State-transition diagrams are highly helpful for explaining the behavior of individual objects over the full set of use cases that affect those objects. (Copeland, 2003)

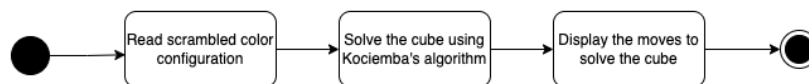


Figure 10: State transition diagram showing different state for solving cube.

### 3.6. Explanation of the development process:

To develop the prototype for solving Rubik's cube kociemba's algorithm is used. The coding part of the solver is based on the work done by Herbert Kociemba. To create the prototype VS code was used as IDE, and command terminal was used to run the solver. To start the solver main.py file is run in the command prompt. When the main.py file is run for the first time, all the files containing the maneuver or path to solve the Rubik's cube of random state is generated and saved in twophase folder. For generating the file it will take about 10 min. Once, the file is generated only for the first time. After that the files are only loaded in the solver. The whole solver is based on the symmetry and coordinate of the cube. For this the code already written by kociemba is used. The explanation of the tools used are given below:

#### A. VS-code:

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS. Debugging support, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git are among the features. The theme, keyboard shortcuts, options, and extensions that offer more functionality can all be changed by users. All of the code for creating the rubik's solver are written using VS-code. (Wikipedia, 2022)



Figure 11: Vs-code

**B. Python:**

Python is a high level language that uses garbage collection and has dynamic typing. It supports a variety of programming paradigms, including procedural, object-oriented, and functional programming as well as structured programming (especially this). Due to its extensive standard library, it is frequently referred to as a "batteries included" language. For creating the prototype, python language is used. Different libraries of python like Threading, time, enum, array and os are used in this project. The threading makes it possible to run many portions of the program simultaneously. The time library is used to get different time-related functions. Enum is used for creating enumerated constants. Array is used to create array of items and at last os is used to provides a portable way of using operating system dependent functionality. (Wikipedia, 2022)

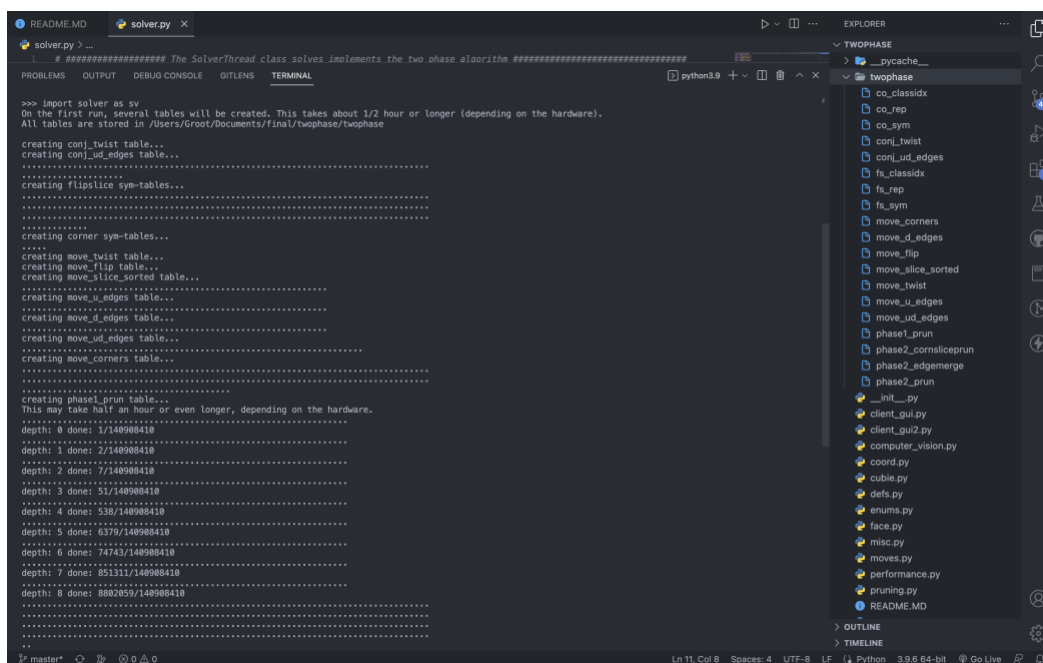


Figure 12: Python

### 3.7. Achieved results:

The results achieved are given below:

- A. When the main.py is loaded for the first time different tables containing the maneuver are created.



The screenshot shows a code editor with a file named `twophase.py` open. The code is in Python and is creating various tables for storing maneuvers. The code is as follows:

```

>>> import solver as sv
On the first run, several tables will be created. This takes about 1/2 hour or longer (depending on the hardware).
All tables are stored in /Users/Groot/Documents/final/twophase/twophase

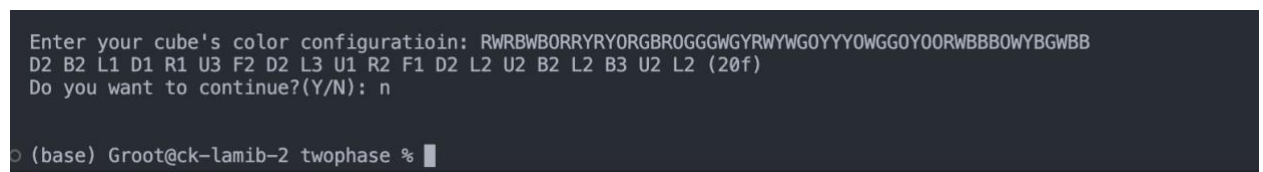
creating conj_twist table...
creating conj_ud_edges table...
.....
creating f1slice sym-tables...
.....
creating corner sym-tables...
.....
creating move_twist table...
creating move_flip table...
creating move_slice_sorted table...
.....
creating move_u_edges table...
.....
creating move_d_edges table...
.....
creating move_ud_edges table...
.....
creating move_corners table...
.....
creating phase1_prun table...
This may take half an hour or even longer, depending on the hardware.
depth: 0 done: 1/140908410
.....
depth: 1 done: 2/140908410
.....
depth: 2 done: 7/140908410
.....
depth: 3 done: 51/140908410
.....
depth: 4 done: 538/140908410
.....
depth: 5 done: 6379/140908410
.....
depth: 6 done: 74745/140908410
.....
depth: 7 done: 853211/140908410
.....
depth: 8 done: 8802059/140908410
.....
..
..

```

The right sidebar shows the Explorer view with a folder named `twophase` containing several files, including `co_classidx`, `co_rep`, `co_sym`, `conj_twist`, `conj_ud_edges`, `fs_classidx`, `fs_rep`, `fs_sym`, `move_corners`, `move_d_edges`, `move_flip`, `move_slice_sorted`, `move_twist`, `move_u_edges`, `move_ud_edges`, `phase1_prun`, `phase2_cornerprun`, `phase2_edgemerge`, `phase2_prun`, `init.py`, `client_gui.py`, `client_gui2.py`, `computer_vision.py`, `coord.py`, `cube.py`, `defs.py`, `enums.py`, `face.py`, `misc.py`, `moves.py`, `performance.py`, `pruning.py`, and `README.MD`.

Figure 13: Creating files for storing maneuvers

- B. A randomly scramble cube was solved.



The screenshot shows a terminal window with the following text:

```

Enter your cube's color configuration: RWRBWBORRYRYORGBROGGWGYRWYWG0YYYOWGG0Y0ORWBBBOWYBGWBB
D2 B2 L1 D1 R1 U3 F2 D2 L3 U1 R2 F1 D2 L2 U2 B2 L2 B3 U2 L2 (20f)
Do you want to continue?(Y/N): n

(base) Groot@ck-lamib-2 twophase %

```

Figure 14: Solving the randomly scramble cube.

C. After creating the files the files are loaded.

```
(base) Groot@ck-lamib-2 twophase % python main.py
loading conj_twist table...
loading conj_ud_edges table...
loading flipslice sym-tables...
loading corner sym-tables...
loading move_twist table...
loading move_flip table...
loading move_slice_sorted table...
loading move_u_edges table...
loading move_d_edges table...
loading move_ud_edges table...
loading move_corners table...
loading phase1_prun table...
loading phase2_prun table...
loading phase2_cornsliceprun table...
loading phase2_edgmerge table...

The facelet positions of the cube should be like this

*****
*W1**W2**W3*
*****
*W4**W5**W6*
*****
*W7**W8**W9*
*****
*****
|*****|*****|*****|*****|
|*01**02**03*|*G1**G2**G3*|*R1**R2**R3*|*B1**B2**B3*|
|*****|*****|*****|*****|
|*04**05**06*|*G4**G5**G6*|*R4**R5**R6*|*B4**B5**B6*|
|*****|*****|*****|*****|
|*07**08**09*|*G7**G8**G9*|*R7**R8**R9*|*B7**B8**B9*|
|*****|*****|*****|*****|
|*****|*****|*****|*****|
|*Y1**Y2**Y3*|
|*****|
|*Y4**Y5**Y6*|
|*****|
|*Y7**Y8**Y9*|
|*****|

Your face orientation should be:
W-color in upward face
R-color in right face
G-color in front face
Y-color in downward face
O-color in left face
B-color in backward face

The color configuration should be like:
WWWWWWMMMMMMMMRRRRRRRRGGGGGGGGYYYYYYYYY00000000BBBBBBBBB
Enter your cube's color configurationin: 
```

Figure 15: loading the files and showing the interface

D. When wrong color configuration is entered invalid message was shown.

```
Enter your cube's color configurationin: bimla
Invalid input

Enter your cube's color configurationin: 
```

Figure 16: Showing error when invalid color string is inserted

#### 4. Conclusion:

Finally, after much effort and commitment, coursework 2 for the artificial intelligence module was successfully finished with a fully developed prototype of Rubik's cube solver for solving Rubik's cube. In this coursework we had to make a fully functionable prototype of Rubik's cube solver that we had researched on the previous coursework. The coursework was not easy as it seemed. Though all the research needed to design the solver was already done on the previous coursework but creating a solver was way too hard than making the research. We had to do quite a lot of hard work to accomplish this coursework. We had to face many challenges and difficulties while doing the coursework.

In this coursework a fully solvable rubik's cube solver was created which can solve the scrambled cube in about 20 moves. Also, the research and the coding part of the solver is fully based on the work done by Herbert Kociemba in his Kociemba's algorithm to solve Rubik cube. Kociemba's algorithm is also known as two phase algorithms. Solving a rubik's cube with optimal solution is not an easy task as the cube contains 43 quintillion states. Professors and scientists around the world have often used the Rubik's cube as a case for studying optimal solutions, creating new algorithms (step-by-step guide to solve a Rubik's cube) or permutations (ordering of the faces of the cube). The Rubik's cube also has significant importance in the computer research field. It serves as an example of what the complex counting equations can be used for while studying the permutations. The Rubik' cube has also been a useful tool to test for the Artificial Intelligence to learn, develop new generic algorithms, and improve existing ones that can be applied to several other problems as well. Beyond the 3x3, other cube sizes such as the 2x2 and 4x4 cubes on up to NxN cube can also be solved using the algorithm used above. Additionally, other combinatorial puzzles or group theory problems can benefit from this research as well. After making proper adjustments, other puzzles can be solved in the same way. Puzzles such as the 15 Sliding Puzzle and Lights Out can be solved using similar approaches or in some cases even using the same model. IDA\* search algorithm which is used in the Kociemba's algorithm is also extremely beneficial when the problem is memory constrained as IDA\* does not remember any node except the ones



on the current path. So, as opposed to A\*, which keeps a large queue of unexplored nodes that can quickly fill up memory, this algorithm has an extremely small memory profile. This is especially helpful in society as this code can help simplify equations with little usage of memory. Because of this, it is often used in search engines and a range of phone apps.

At last a fully solvable Rubik's cube solver was created using the kociemba's algorithm. Though, the fully solvable solver was created but still there are different parts in the algorithm that I haven't fully understand. More research is still required in that part of the code. Furthermore, I will be still connected in the field of artificial intelligence after the coursework too. In summary, the coursework was challenging and difficult. Solving problem, researching and developing about new topics helps us to learn something different and new which helped us to develop our skills and broadened our knowledges. At last, I'm grateful I have the chance to broaden my knowledge and abilities.

## 5. Bibliography:

Korf, R. E., 1997. *Finding Optimal Solutions to Rubik's Cube Using Pattern Databases*. [Online]

Available at:

<https://www.cs.princeton.edu/courses/archive/fall06/cos402/papers/korfrubik.pdf>

[Accessed 05 12 2022].

Kaur, H. H., 2015. Algorithms for solving the Rubik's cube. p. 46.

Korf, R., 1996. Artificial Intelligence Search Algorithms.

JavaTPoint, 2021. *types-of-artificial-intelligence*. [Online]

Available at: <https://www.javatpoint.com/types-of-artificial-intelligence>

[Accessed 08 12 2022].

kelley, k., 2022. *What is Artificial Intelligence: Types, History, and Future*. [Online]

Available at: <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/what-is-artificial-intelligence>

[Accessed 08 12 2022].

Verma, P., 2020. *Artificial Intelligence based Recommendation System*. [Online]

Available at: <https://ieeexplore.ieee.org/document/9362962>

[Accessed 08 12 2022].

pdfCo, 2022. *Problem Solving Techniques in Artificial Intelligence (AI)*. [Online]

Available at: <https://pdf.co/blog/problem-solving-techniques-in-artificial-intelligence-ai#:~:text=Problem%2Dsolving%20in%20Artificial%20Intelligence,are%20achieved%20by%20different%20heuristics.>

[Accessed 08 12 2022].

Stefan Edelkamp, S. S., 2008. *Search: Theory and Applications*. [Online]

Available at: <https://www.nms.kcl.ac.uk/stefan.edelkamp/search.pdf>

[Accessed 08 12 2022].

Sher, S. T.-H., 2014. *The New Durable Rubik's Cube*. [Online]

Available at: [https://homes.luddy.indiana.edu/stsher/files/Rubiks\\_Cube.pdf](https://homes.luddy.indiana.edu/stsher/files/Rubiks_Cube.pdf)

[Accessed 08 12 2022].

Marcellienus, J., 2014. *The most efficient algorithm to solve a Rubik's cube*. [Online]

Available at: <http://www.youngscientist.com.au/wp-content/uploads/2015/02/Physics-10-12-Justin-Marcellienus-report.pdf>

[Accessed 08 12 2022].

Emir, B., Amila, A. & Samir, R., 2020. 3.3. *Two approaches in solving Rubik's cube with Hardware-Software Co-design*. Sarajevo,

<https://www.researchgate.net/publication/346784181>.

De, M., 2015. *SOLVING RUBIK'S CUBE*. [Online]

Available at:

[https://www.researchgate.net/publication/282133465\\_Group\\_Theory\\_to\\_Solve\\_Rubik\\_Cube](https://www.researchgate.net/publication/282133465_Group_Theory_to_Solve_Rubik_Cube)

[Accessed 08 12 2022].

Copeland, L., 2003. *State-Transition Diagrams*. [Online]

Available at: <https://www.stickyminds.com/article/state-transition-diagrams>

[Accessed 08 12 2022].

lucidchart, 2020. *What is a Flowchart*. [Online]

Available at: <https://www.lucidchart.com/pages/what-is-a-flowchart-tutorial>

[Accessed 08 12 2022].

geeksforgeek, 2022. *How to write a Pseudo Code?*. [Online]

Available at: <https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/>

[Accessed 08 12 2022].

Scherphuis, J., 2015. *Computer Puzzling*. [Online]

Available at: <https://www.jaapsch.net/puzzles/compcube.htm#tree>

[Accessed 08 12 2022].

kociemba, H., 2013. *The Two-Phase-Algorithm*. [Online]

Available at:

<https://www.cs.brandeis.edu/~storer/JimPuzzles/RUBIK/Rubik3x3x3/READING/KociembaPage.pdf>

[Accessed 08 12 2022].

Wikipedia, 2022. *Visual Studio Code*. [Online]

Available at: [https://en.wikipedia.org/wiki/Visual\\_Studio\\_Code](https://en.wikipedia.org/wiki/Visual_Studio_Code)

[Accessed 11 1 2023].

Wikipedia, 2022. *Python (programming language)*. [Online]

Available at: [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

[Accessed 11 1 2023].