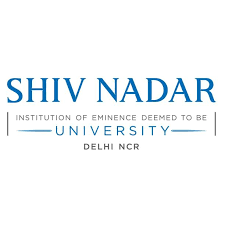
**Project Report***Artificial Intelligence (CSD311)*



**Project Title: *N x N Rubix Cube***

**Course Instructor: *Snehasis Mukherjee***

***Group No: 23***

*Group Members*

| **Name** | **Roll No.** |
| --- | --- |
| Aryan Sethia | 2110110754 |
| Chaitanya Tandon | 2110110171 |
| Tejansh Sachdeva | 2110110555 |
| Keshav Dubey | 2110110687 |
| Snehali Biswas | 2110110504 |
| Abhishek Soni | 2110110025 |
| Mitaali Singhal | 2110110883 |
| Vidushi Bist | 2110110049 |

*Abstract*

*This report details the creation of a Rubik's Cube solver utilizing a hybrid approach of Breadth-First Search (BFS) and A\* Search algorithms. The solver dynamically selects heuristics, including Hamming distance, Manhattan distance, and pattern database-based methods, based on the cube's size and scrambling complexity. The adaptive heuristic strategy aims to optimize computational efficiency and solution time, catering to varying cube sizes and complexities. Through systematic algorithmic exploration, the report provides valuable insights into the challenges and optimization techniques involved in Rubik's Cube solving.*

1. **INTRODUCTION**

*The Rubik's Cube is a famous puzzle that has fascinated people for a long time. It started as a 3x3x3 cube challenge. Even though it looks simple, solving it is more complicated. The main goal is to move the colored parts until each side is just one color. This might seem easy, but there's a lot going on inside the cube.*

*When you twist and turn the Rubik's Cube, you're creating different combinations of colored sides. This makes a big puzzle with many possibilities, and solving it requires a smart strategy. As enthusiasts venture into the realm of larger-dimensional Rubik's Cubes denoted as NxNxN, the complexity expands exponentially.*

1. **PROBLEM ANALYSIS**

* ***Enclosed Puzzle Space:***

*The Rubik's Cube is played within a three-dimensional, enclosed puzzle space. The challenge is to rearrange its colored faces within this confined structure.*

* ***Limited Set of Moves:***

*Players have a fixed set of six fundamental moves: top (T), down (D), left (L), right (R), front (F), and back (B). These moves involve rotating specific faces of the cube, providing the primary means of manipulating its configuration.*

* ***Fixed Move Constraints:***

*The cube's dynamics are defined by the limited set of moves, offering a predictable environment. No additional actions are possible beyond these six moves, setting clear constraints on the solving process.*

* ***Goal State Uniformity:***

*The objective is to achieve a goal state where each face of the Rubik's Cube displays a uniform color. Flexibility exists in how this uniformity is achieved, as long as each face attains a single color.*

* ***Complex Search States:***

*The solving process can lead the Rubik's Cube through various complex states resulting from different move combinations. Navigating this diverse solution space requires a sophisticated search algorithm capable of evaluating and exploring permutations effectively.*

* ***Strategic Move Sequences:***

*Solving the Rubik's Cube involves planning and executing strategic sequences of moves. Careful consideration is needed, as certain moves may have unintended consequences. Intelligent decision-making is crucial for optimizing the solving process.*

* ***Permutation Dynamics:***

*Each move introduces a unique permutation of the colored facets. Understanding and leveraging these permutation dynamics are essential for developing efficient solving algorithms. The solver must adapt to the changing state space created by permutations.*

* ***Computational Complexity:***

*As the cube's size increases, computational complexity escalates. Addressing the challenges posed by a larger solution space requires optimization strategies, heuristic evaluations, and efficient search algorithms to ensure practical solving times.*

1. **SOLVER IDEATION**

*We began by visualizing the workings of a Rubik's Cube, understanding its representations, and establishing conditions for valid moves. Progressing, we created a move function to determine how actions on one face impact others. The development extended to a display function and the implementation of rudimentary uninformed search algorithms such as DFS and BFS.*

*Advancing further, we explored informed algorithms like Greedy A\* with various heuristics, commencing with Hamming distance and Manhattan distance. Subsequently, we introduced a novel approach based on goal pull distance. To optimize running time, we implemented functions for generating pattern positions and cubie-to-numeric conversions.*

*Throughout this process, our focus remained on refining the solver's functionality and balancing sophistication with computational efficiency, ensuring adaptability across different cube sizes and configurations.*

***Data Structures Used***

***PriorityQueue***

* *Utilized in A\* search for state prioritization.*
* *Prioritizes states based on cost values.*

***Queue (Deque)***

* *Used in breadth-first search for state exploration.*
* *Maintains FIFO order.*

***Tuple***

* *Represents Rubik's Cube states.*
* *Concatenates face lists into a single tuple.*

***Sets***

* *Keep track of visited states.*
* *Prevents revisiting explored states.*

***Goal Ordering***

*The primary goal is to minimize the total cost, encompassing move cost, cube state distance, and a pattern database heuristic. The strategy involves prioritizing moves aligned with a predefined pattern, integrating the pattern database heuristic into the search process. This approach optimizes decision-making by combining heuristic guidance with considerations of move efficiency and cube state proximity, resulting in a more effective and informed exploration of the Rubik's Cube solution space.*

***Algorithm***

*In our Rubik's Cube solver, we employ a synergistic approach using Breadth-First Search (BFS) and A\* Search algorithms to efficiently navigate the intricate state space of the Rubik's Cube. The solver initiates with a scrambled cube and systematically explores potential states, guided by a combination of BFS for completeness and A\* Search for optimization. To enhance efficiency, the A\* Search algorithm utilizes various heuristics, including Hamming distance, Manhattan distance, and pattern database-based heuristics.*

*Initially, without heuristics, the solver exhibited significant delays or termination issues, particularly when the cube size (n) exceeded 5 or when the number of scrambles was increased. To address this, we experimented with the Manhattan distance heuristic to expedite the solving process. However, as we encountered challenges with increased values of n and scrambles, including execution timeouts, we explored the Hamming distance heuristic. Although this showed promise, the computational demands were often excessive, leading to incomplete solutions. Consequently, we pivoted to utilizing the Hamming distance in conjunction with a pattern database, yielding improved results. Seeking further optimization, we introduced the goal pull heuristic, which demonstrated comparable performance to the Hamming distance with a pattern database, yet proved superior in certain scenarios. This iterative refinement process aimed to strike a balance between computational efficiency and solution quality, ultimately contributing to the solver's adaptability across diverse cube configurations.*

***Code Flow***

1. ***Initialization:***

*User inputs the size of the Rubik's Cube (n) and the number of moves for scrambling.*

*Possible moves and a solved state are generated based on the cube size.*

*A solved cube and a randomly scrambled cube are created.*

1. ***Scrambling:***

*Random moves are applied to the solved cube to create a scrambled state.*

*The cube after each move is stored in a list for later analysis.*

1. ***Solving:***

*The solver applies the chosen algorithm to find a solution from the scrambled state to the solved state.*

*During the search, the cube's state and the sequence of moves are stored for analysis.*

1. ***Analysis and Output:***

*The solver outputs the sequence of moves required to solve the Rubik's Cube.*

*Performance metrics, such as the time taken for solving, are recorded.*

*Intermediate states of the cube during the solving process are displayed for visual analysis.*

1. ***User Interaction:***

*The user is prompted to play the solver again or exit the program.*

*The entire process, from initialization to solution, is encapsulated in a loop for multiple iterations.*

1. **TESTING**

* DFS without heuristics

| VALUE OF N | SCRAMBLES | TIME TAKEN |
| --- | --- | --- |
| 2 | 3 | 0.19s |
| 3 | 5 | 0.14s |
| 2 | 8 | 0.79s |
| 4 | 3 | 2.94s |
| 5 | 3 | 3.12s |
| 10 | 3 | 53s |
| 10 | 5 | Process getting killed |

* DFS with heuristic - Manhattan Distance

| VALUE OF N | SCRAMBLES | TIME |
| --- | --- | --- |
| 2 | 3 | 0.14s |
| 2 | 5 | 0.07s |
| 2 | 8 | 1.17s |
| 3 | 5 | 0.09s |
| 4 | 4 | 14s |
| 10 | 2 | 1.205s |
| 15 | 2 | 4.3s |
| 17 | 4 | Process getting killed |

* DFS with heuristic - Hamming Distance

| VALUE OF N | SCRAMBLES | TIME TAKEN |
| --- | --- | --- |
| 2 | 3 | 0.077s |
| 3 | 5 | 0.06s |
| 4 | 3 | 0.7s |
| 5 | 3 | 1.4s |
| 10 | 3 | 41s |
| 10 | 5 | 62.7s |
| 15 | 3 | 109s |

* A\* with heuristic (Hamming Distance) and Pattern Database

| VALUE OF N | SCRAMBLES | TIME |
| --- | --- | --- |
| 2 | 3 | 0.0039s |
| 3 | 3 | 0.007s |
| 4 | 4 | 0.022s |
| 7 | 5 | 0.029s |
| 7 | 8 | 0.04155s |
| 8 | 10 | 0.0644s |
| 15 | 5 | 0.1s |
| 15 | 9 | 0.24s |

\*Timing mentioned depends on the random Scrambling & varies according to how the cube is scrambled.

1. **REFERENCES**

* **Rubix Cube Simulato**r: <https://rubikscu.be/>
* <https://medium.com/@benjamin.botto/implementing-an-optimal-rubiks-cube-solver-using-korf-s-algorithm-bf750b332cf9>
* <https://medium.com/@benjamin.botto/implementing-an-optimal-rubiks-cube-solver-using-korf-s-algorithm-bf750b332cf9>
* <https://www.cs.princeton.edu/courses/archive/fall06/cos402/papers/korfrubik.pdf>
* <https://essay.utwente.nl/96688/1/Heij_BA_EEMCS.pdf>