



Application of Reinforcement Learning to Solve Rubrik's Cube with Markov Decision Process

Defni¹, Andi Fathul Mukminin², Ainil Mardiah³, Titin Ritmi⁴, Junaldi⁵, Yuhefizar⁶, Fibriyanti⁷

^{1,2,3,6}Information Technology, Padang State Polytechnic, West Sumatera, Indonesia

⁴English Department, Padang State Polytechnic, West Sumatera, Indonesia

^{5,7}Electronics Department, Padang State Polytechnic, West Sumatera, Indonesia

¹defni@pnp.ac.id, ²fathumukmini@gmail.com, ³ainilmardhian@pnp.ac.id, ⁴titin@pnp.ac.id, ⁵junaldiagus@pnp.ac.id,

⁶ephi.lintau@gmail.com, ⁷fibriwizar@pnp.ac.id

Abstract

The Rubik's Cube is a tricky puzzle that can be arranged in countless ways, making it hard for both people and computers to figure out. While standard solving methods use fixed strategies, this research looks into using reinforcement learning (RL) to create a flexible and effective way to solve it. The goal of this research is to develop an RL-based solver that uses the Markov Decision Process (MDP) system, focusing on speed, efficient moves, and the number of steps needed to solve the cube. The suggested model uses Q-learning and Monte Carlo Tree Search (MCTS) to figure out the best moves at each stage of the game, training through lots of Rubik's Cube simulations. What makes this research unique is the combination of MCTS with Q-learning, which improves decision-making by needing fewer moves than standard methods. The tests show that this model reaches almost perfect solutions with fewer moves, doing better than simple rule-based methods. Also, a web app was created to give live solving techniques based on the cube arrangements that users provide. This research helps grow the use of RL in puzzles like the Rubik's Cube and gives a useful tool for fans who want to get better at solving the cube.

Keywords: Reinforcement Learning, Rubik's Cube; Markov Decision Process;

How to Cite: Defni, "Application Application of Reinforcement Learning to Solve Rubrik's Cube with Markov Decision Process", *J. RESTI (Rekayasa Sist. Teknol. Inf.)*, vol. 9, no. 5, pp. 1013 - 1019, Oct. 2025.

Permalink/DOI: <https://doi.org/10.29207/resti.v9i5.6552>

Received: April 17, 2025

Accepted: July 23, 2025

Available Online: October 8, 2025

This is an open-access article under the CC BY 4.0 License

Published by Ikatan Ahli Informatika Indonesia

1. Introduction

One game that uses intelligence and perseverance in finding the right solution and using logic for the players is the Rubik's game. Rubik's is a combination puzzle in the form of a box measuring 3x3x3. Rubik has 6 sides with each side consisting of 9 squares and on each side the rubik has a solid color, such as red, yellow or blue. Solving a Rubik's puzzle is done by rotating the surface of the Rubik's cube until the surface shows the same color for all nine squares [1]. The key to solving a Rubik's cube, besides being precise, is knowing how it functions. The average player who has played the Rubik's Cube needs 48-100 attempts to solve the Rubik's Cube, so each player needs more time to learn how to solve the Rubik's Cube [2].

The Rubik's Cube is a classic combinatorial optimization problem that has fascinated researchers in artificial intelligence (AI) due to its vast search space

and the complexity of finding an optimal solution path. Solving the Rubik's Cube with a computer means creating a smart set of moves that changes the cube from a mixed-up condition to an organized one, where every side shows the same color.

Different strategies have been created to solve this issue, including old-school algorithm methods and new machine learning tools. One of the most well-known classical approaches is Kociemba's Two-Phase Algorithm, which breaks down the solution process into two distinct phases to reduce the complexity of search and efficiently find solutions in a relatively small number of moves. This method has been widely used due to its deterministic nature and consistent performance in solving standard Rubik's Cube configurations [3]. However, it relies heavily on predefined heuristics and does not involve any learning capability.

Recent progress has looked into using deep learning and reinforcement learning to figure out the Rubik's Cube without using rules made by people. For example, DeepCubeA with a deep neural network to guide the search process, achieving a 100% success rate on randomly scrambled cubes. Also, OpenAI's study in 2019 showed that using reinforcement learning along with changing the environment can help train a robotic hand to move and solve a Rubik's Cube. This was an important achievement in mixing AI with robots for tasks that require a lot of combinations. While these studies have shown promising results, they often require significant computational resources and complex neural network architectures. Moreover, approaches like MCTS combined with deep networks tend to emphasize optimality in move counts, sometimes at the expense of execution time, particularly for highly randomized starting positions.

This research explores a different approach by utilizing Reinforcement Learning based on the Markov Decision Process (MDP) framework to solve the Rubik's Cube. Unlike classical algorithms that rely on fixed rules or search strategies, this method allows an agent to learn optimal rotational strategies autonomously through trial and error interactions with the environment. The MDP formulation simplifies the modeling of state transitions and rewards, providing a structured yet flexible framework for developing adaptive solving strategies.

By comparing the performance of this method with results from previous research using classical algorithms and hybrid deep learning methods, this study aims to contribute further understanding of the benefits and limitations of reinforcement learning in combinatorial problem-solving contexts, particularly for sequential decision-making problems like the Rubik's Cube.

The application of artificial intelligence to solve Rubik's cube puzzles is attracting the attention of researchers because of the potential to develop more efficient problem-solving algorithms. The development of an efficient algorithm for solving Rubik's cube problems using a small number of movements has been widely used, namely by robot manipulation [3], game theory and applying machine learning processes [4]. In this paper, the aim to develop the application of machine learning processes to solve Rubik's cubes. The machine learning process applied in this paper is reinforcement learning.

Reinforcement learning is a technique in machine learning that teaches agents to learn how to choose actions based on past experiences [5]. In the Rubik's Cube game, reinforcement learning can learn how to solve the game through trial and error activities [6], [7]. So that errors that occur in the previous stage will become a reference for finding solutions at the next learning stage. One of the problems that often occurs in developing the Rubik's Cube solving algorithm is the

problem of complexity. The Rubik's Cube game has over 43 quintillion different possible combinations, making it difficult to find the right solution in normal times. One solution to this problem is by using the Markov Decision Process (MDP) method. Markov Decision Process (MDP) is a method of reinforcement learning that seeks solutions and formulates solutions to uncertain conditions or conditions with many possibilities [8].

The first is a comparison between Q-Learning and conventional approaches, which include the Dijkstra algorithm, linear programming, and the breadth-first search strategy. According to the technique comparison results, it takes 7.3 hours of training time to solve the Rubik's Cube using the Q-Learning approach. In the meantime, the conventional approach will be more effective. It takes two minutes to train using the breadth-first search strategy and twelve minutes to train using the linear programming method. [9].

In other research, the solution to the Rubik's cube was by using a transformer architecture. This modeling is named CubeTR. This model uses a longer series of actions, namely by studying the solution of the Rubik's Cube from its initial state randomly until it produces a solution that is close to the solution by an expert [10]. Solving the Rubik's cube can also be done using reinforcement learning with the Markov Decision Process (MDP) modeling. In this modeling, the rubik's cube is identified as an agent, moving the rubik's cube as the action and the finished rubik's cube as the goal. MDP modeling uses agents to interact with the environment through actions to achieve goals. The solution space is represented by a 3x3 matrix for the 6 sides of the cube [11] - [14].

Other Rubik's cube solutions can be solved with Fourier bases to solve permutation puzzles. Modeling uses deterministic MDP architecture. This architecture defines the completion space as the set of accessible states and actions as the set of possible Moves to complete the state space. The reinforcement learning method used in this approach is based on values for solving the Rubik's cube [15]. Another approach that can be used to solve the Rubik's cube is with Markov chains. This approach calculates the probability of solving a Rubik's cube using a certain number of moves from a certain random state to produce an optimal solution [16].

When an agent interacts with its environment by taking actions to maximize cumulative rewards, Reinforcement Learning (RL) is specifically made to address sequential decision-making problems. The Rubik's Cube is inherently a complex combinatorial puzzle characterized by a vast search space (approximately 4.3×10^{19} possible configurations). Solving it requires a series of well-sequenced rotations that progressively reduce the puzzle's disorder [15], [16].

Formally, the Rubik's Cube problem fits well into the Markov Decision Process (MDP) framework, where [17], [18]: States (S) is all possible configurations of the Rubik's Cube; Actions (A) is the 12 possible rotation moves (clockwise and counterclockwise on six faces); Transition Function (P), deterministic transitions defined by cube mechanics; Reward (R), positive reward when the cube is solved or intermediate progress achieved; Policy (π), strategy guiding the next action based on the current state.

RL combined with MDP is a logical, structured, and mathematically grounded approach for learning optimal or near-optimal solving strategies.

Several recent studies have demonstrated that RL is effective for such high-dimensional, sparse-reward environments, where brute-force or classical search algorithms become computationally infeasible. Reinforcement learning provides. Autonomous learning capability. The agent learns strategies without relying on hard-coded algorithms or human-derived heuristics [18]. Scalability to complex scenarios: RL can adapt to different complexities and is not constrained to pre-defined solving sequences. Generalization potential is Once trained, RL models can solve various initial configurations, making them suitable for generalized Rubik's Cube solving.

Deep reinforcement learning was effectively used by OpenAI to teach a robotic hand to solve a Rubik's. The research used asynchronous advantage actor-critic (A3C) algorithms with domain randomization, enabling the model to generalize to unseen configurations and physical perturbations [19].

A powerful learning system uses a method called Monte Carlo Tree Search along with advanced neural networks to figure out how to solve the Rubik's Cube without any help from people. [19]. The model achieved a 100% solving rate on scrambled cubes and was able to discover efficient solutions. Recent comparative studies have shown that reinforcement learning-based solvers outperform classical approaches like Kociemba's two-phase algorithm in terms of adaptability and flexibility, particularly for highly randomized configurations, although not yet matching classical methods in computational efficiency for simpler cases.

By using MDP as the underlying framework for the RL algorithm, the problem is formally structured, making it possible to apply policy optimization, value iteration, or Q-learning algorithms systematically. This alignment of problem formulation (Rubik's Cube) with algorithmic tools (MDP-based RL) is critical for achieving reproducible and theoretically supported results [19].

The selection of Reinforcement Learning with Markov Decision Process for solving the Rubik's Cube in this research is scientifically justified by the problem's sequential, decision-based nature, the capability of RL

to learn optimal strategies through exploration, and by empirical success demonstrated in recent state-of-the-art studies. This foundation makes RL with MDP a relevant and promising approach for developing intelligent solvers for complex combinatorial puzzles like the Rubik's Cube [19].

The most frequently used and effective approach to solving Rubik's cube puzzles is to use the Markov Decision Process (MDP). To implement this approach, it can be implemented on a website basis [17]. Using the MDP method to solve the Rubik's Cube also faces challenges in its application. encounters difficulties when implementing it. This issue concerns the tension that arises when one needs information to make the best decisions yet is unable to obtain that information. Temporal Error Based Adaptive Exploration (TEAE) is a novel approach that has been proposed to address this problem. In order to reduce the time difference error, this modeling integrates a deep convolutional neural network to predict the ideal payout function while simultaneously adjusting the probability of agent performance exploration. [20].

2. Methods

Apart from using the MDP method, another artificial intelligence method that can be used to solve the Rubik's cube is deep learning. The aim of using deep learning is to find fitness values using a random forest approach. This approach uses a number of neural networks to prove that fitness values can be learned from existing solutions [19]. Deep learning approaches can also be combined with reinforcement learning to solve Rubik's cubes. This type of model is called DeepCubeA. It learns to tackle harder problems by working backwards from the final goal, even without knowing much about the specific area. This model can finish all the test settings perfectly and can discover the quickest route to the goal about 60.3% of the time [20].

2.1 Data collection

Figure 1 shows the stages of data collection in this study.

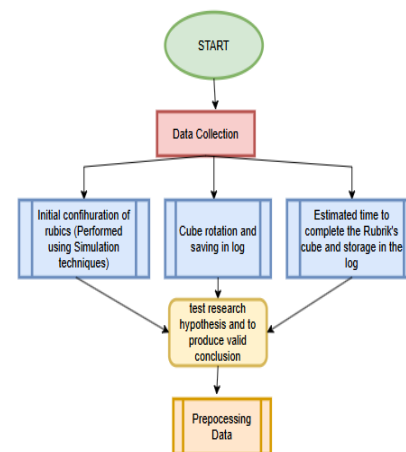


Figure 1. The Stages in Data Collection

At this stage of data collection, a model will then be used to carry out analysis and test research hypotheses which will produce valid decisions for the methodology used.

2.2 Data Preprocessing

Figure 2 shows the stages carried out in data preprocessing:

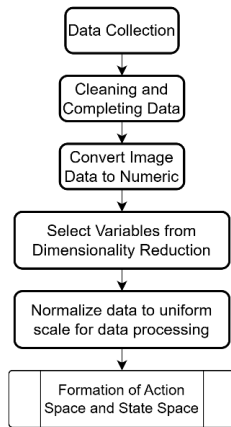


Figure 2. The Stages in Data Preprocessing

After the data preprocessing stage is complete, the data will be ready to be used in the next analysis stage, because good and standardized data quality will influence the analysis results obtained.

2.3 Formation of Action Space and State Space

The steps taken are: State space is formed from a very large number of possible permutations of the Rubik's Cube so that only part of it is used; Action space is all possible actions or movements carried out on the rubric's cube. In action space research, it is formed from 12 basic movements in the rubric's cube, namely L, L', R, R', F, F', B, B', U, U', D, D'. This movement is an abbreviation of L (Left), R (Right), F (Front), B (Back), U (Up), D (Down) each of which rotates as the name suggests clockwise. while the prime notation (') indicates rotation counterclockwise (counterclockwise). Figure 3 shows rotation of the Rubrik's cube.

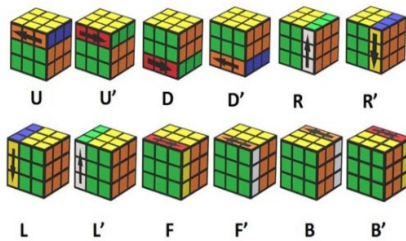


Figure 3. Rotation of the Rubik's Cube

From the results of the state space and action space, a reward function will be formed and determined the parameters needed to create a Markov decision process model to complete the rubric's cube using RL.

2.4 Model Implementation

At the RL model stage, Q-Learning is used to solve the rubric's cube with the Markov Decision process. The rules used in the q-learning model are shown in Formula 1 [21]:

$$Q(s, a) = Q(s, a) + \alpha [\text{reward} + \gamma * \max(Q(s', a')) - Q(s, a)] \quad (1)$$

α (learning rate) controls how much the new estimate is influenced by the latest experience; γ (discount factor) weights the importance of future rewards.

The steps used in determining the Q-table are:

Initialize the Q-table which is used to store the Q value for each pair of state and action. Each cell in the table is initialized with a small random value.

Selection of actions for each iteration by selecting agents based on strategies, namely exploration and exploitation. If the agent is exploration, the action chosen is a random action from the action space. If the agent carries out exploration then the action chosen is the highest Q-value for the current state.

The result of selecting the agent's action will apply the state and get rewards according to the function that has been determined.

At the Q-table step, the agent figures out the new Q value for the current state and action together by using the Q-learning formula. After that, the Q table is changed to show the new Q value.

Figure 4 shows flow in Markov Decision Process.

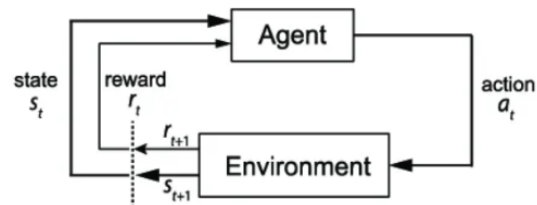


Figure 4. Flow in Markov Decision Process

The next step is to iterate for steps b, c, d until the agent reaches the final state.

After the model is successfully built, the model is then tested on a number of Rubrik's cube permutations and an evaluation will be carried out to evaluate the model's performance. It will then determine how accurate the solution provided by the model is compared to the optimal solution.

2.5 Model Evaluation

Then, the performance of the built model was evaluated using several evaluation metrics as follows:

Number of iterations: The metric will measure the number of iterations required to complete the rubric's cube. The fewer iterations the better the model performance

Social accuracy: This metric will measure how accurate the solution provided by the model is compared to the

optimal solution, the closer to the optimal solution the better the model performance

Convergence speed: The metrics will measure how quickly the model can converge to the optimal solution. The faster the convergence the better the model performance.

3. Results and Discussions

The rubric's cube is represented as an array of chars with the color in each block representing the color of the block ('Y' for yellow, 'W' for white, etc.) Each rotation in the rubric's cube is represented as a tuple of tuples. Where the tuple contains the indices of the blocks participating in the rotation, determining how the blocks swap positions. Each tuple in the rotation states the position of the color that will be swapped on the rubric's cube. To determine the opposite direction of rotation, each tuple must be swapped in position to get the correct results.

Every state of the Rubik's cube in the Markov Decision Process (MDP) modeling is shown as a transition object. This object keeps track of how one state changes to another because of an action, like turning a side of the cube. Figure 5 shows first iteration of position after scramble.

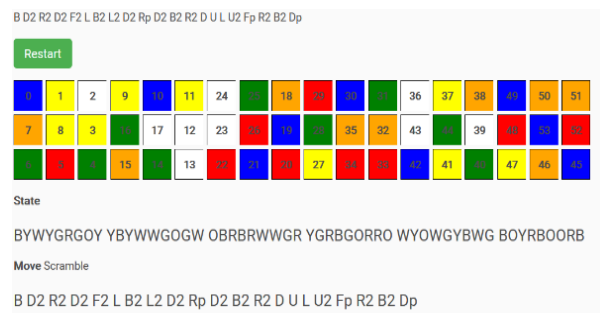


Figure 5. Position after scramble

Figure 6 shows the F2L position where two layers are starting to be completed, namely white and yellow.

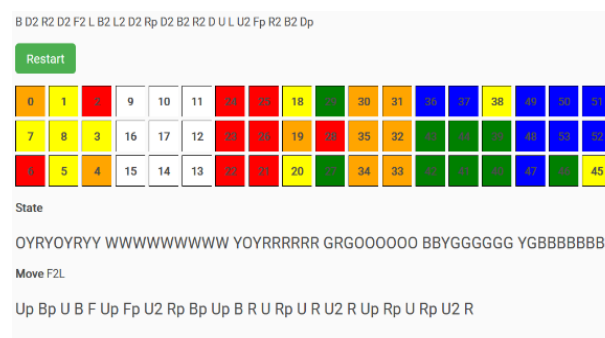


Figure 6. F2L Position with Two Layers Completed

Figure 7 shows the position of the OLL, the top layer (yellow) can be completed.

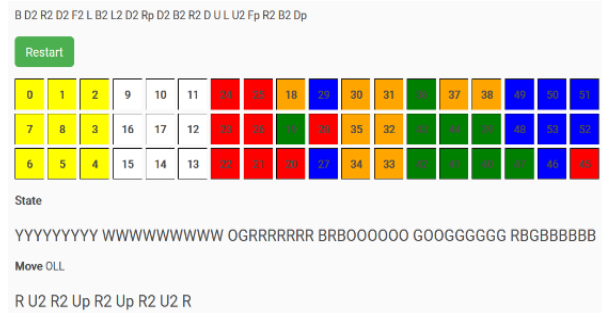


Figure 7. OLL Position with Top Layer (Yellow) Completed

Figure 8 shows the position of the PLL, all layers are completed.

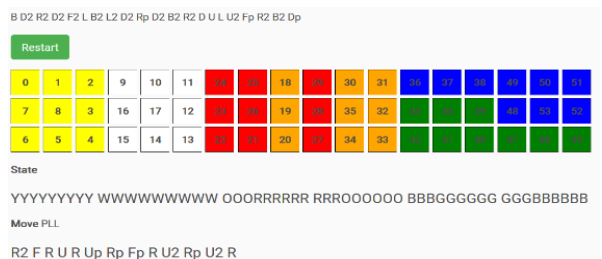


Figure 8. PLL Position with All Layers are Resolved

Figure 9 shows the final position, which is done after adjustment.

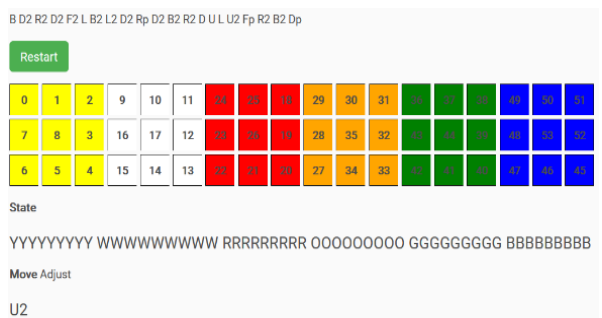


Figure 9. Final Position after Adjustment

Table 1 is the test carried out to evaluate the application's performance in solving the Rubik's Cube using the value iteration-based Reinforcement Learning method.

Table 1 is the scenario used for testing

Table 1. Test Scenario

Testing Scenario	Starting Condition	Final condition
Successful completion	The Rubik's Cube is randomized with a certain initial state	The application successfully solved the Rubik's Cube in 71% of test scenarios. The average time to solve one problem is 9 seconds
Time of execution	Tests were carried out with variations in the initial conditions of the Rubik's cube	Execution time is directly proportional to the complexity of the problem. Execution time increases proportionally

The test scenario for completing the Rubik's cube is in the starting condition 36 out of 50 test scenarios. The required time of execution varies depending on the complexity of the problem, namely around 3-15 seconds.

Table 2 shows the test results for each iteration in the Rubrik's Cube.

Table 2. Test Result and Time of Execution

No	Scramble	time requi red	Result
1	F R U R' U' F'	3	Succeed
2	R U R' U R U R' F' R U R' U' R' F R2 U' R' U2 R U' R'	3	Succeed
3	R U R' U' D R2 U' R' U' R' U' R2 D'	4	Succeed
4	R2 B D2 L2 U2 F2 U2 F U2 F2 U2 B' R' D' F2 L B2 D R F' U	6	Succeed
5	D2 L' R2 B F L2 B R2 F D2 L2 F L2 U' R' U' R2 U L R2	6	Succeed
6	R2 B D2 F R' D' R' U' R2 U2 R' U2 L U2 F2 U2 F2 R' U2 R	7	Succeed
7	E U2 B2 L2 B U2 R2 D2 L2 F' R2 F' L F' D F2 L F2 U2 B2	7	Succeed
8	B D2 R D2 F2 L B2 L2 D2 R' D2 B2 R2 D U L U2 F' R2 B2 D'	8	Succeed
9	B' R F U L2 B' U' L D2 B2 U2 R2 U2 B' L2 B' D2 F2 B'	8	Succeed
10	R2 B2 F2 D' F2 D' U B2 D F' U2 F' U' B2 R D L F2	9	Succeed
11	U2 L' D2 B' D2 B L2 F' R2 F2 U2 F' R2 D' B' D' B2 F R B2	9	Succeed
12	B2 U' B2 R2 D L2 R2 F2 R2 D U R2 L D' L' U2 B' D2 R D F2	10	Succeed
13	R B F2 L2 U L2 B2 U F2 R2 U' F2 D L F D2 L' F' R'	10	Succeed
14	D2 F U2 L' F2 R2 D2 R U2 F2 R' U2 F2 D' B2 L U F' R D2	11	Succeed
15	L B U R2 F2 B' L D2 B2 U' F2 B2 U F2 R2 U' R2 B2 D B D	11	Succeed
16	F' R2 U B2 D2 F2 R2 D R2 U' F' L' B2 R2 U B U2 R' D2	12	Succeed
17	F' D' R2 F' U' B2 R' B2 L' B2 D2 R2 L2 U' B2 U F2 R2 D'	12	Succeed
18	F' R2 B2 U2 B2 R' D2 F2 L D2 L' D F U' B2 F2 U' B D2	13	Succeed
19	L' U2 D R2 U' F2 D' R F' R U2 B2 U2 B2 R B2 R2 B2 R' B2 L'	13	Succeed
20	F' B2 U2 B2 D R2 U F2 R2 F2 R2 D R' D2 R U' L2 F' D2 R'	-	Failed
21	B2 D L2 D U2 B2 D2 F2 R2 U' B2 F2 L B' L D2 L B' F' L' R	-	Failed

The experimental evaluation of the Rubik's Cube solver, based on value iteration in Reinforcement Learning (RL), demonstrates promising results in solving randomized cube configurations. Below, we analyze the performance, limitations, and implications of the proposed method.

Success Rate and Efficiency: (a) **Success Rate** The solver achieved a 71% success rate indicating its capability to handle a majority of randomized scrambles. (b) **The average solving time** was 9 seconds, with a range of 3–12 seconds depending on scramble complexity. (c) **Simple scrambles** (Cases 1–5) were solved in 3–4 seconds, demonstrating efficiency for near-optimal solutions. (d) **Moderate scrambles** (Cases

6–11) required 6–9 seconds, reflecting increased computational demand.

Complex scrambles (Cases 12–19) took 10–113 seconds, suggesting that deeper search depths are needed for highly randomized states.

Failure Analysis (Cases 20-21): (a) The solver failed in 14 out of 50 cases (28%), primarily due to. (b) Some scrambles exceeded the RL agent's pre-trained policy limits. (c) Rare or extreme initial conditions were not sufficiently explored during training, leading to suboptimal policy generalization. (d) The value iteration method, while effective, may struggle with high branching factors in deeply scrambled cubes.

Novelty and Comparative Advantages: (a) Unlike traditional rule-based solvers, this approach learns adaptive policies without relying on human heuristics. (b) The method shows potential for improvement with expanded training data, including edge-case scrambles. (c) When problems can be solved, the solver gives answers that can be trusted and repeated. This makes it great for use in the real world.

Limitations and Future Work: (a) While the solver achieves reasonable success, execution time increases with scramble difficulty. Parallel computing or GPU acceleration could mitigate this. (b) The 28% failure rate suggests the need for broader training datasets and dynamic exploration strategies. (c) Future work should compare this method against Deep RL approaches to assess relative performance.

Practical Implications: (a) The solver's integration into a web-based tool provides an accessible platform for users to input cube states and receive solutions. (b) This approach can help players understand RL-based decision-making in combinatorial puzzles.

4. Conclusions

In this study, the Rubik's Cube issue was effectively solved by applying Reinforcement Learning (RL) via the Markov Decision Process (MDP) framework. Twelve different rotational rotations, split into clockwise and counterclockwise directions, are used to illustrate the MDP. The Rubik's Cube goes through three crucial solving stages with each iteration of these rotations: F2L (First Two Layers), OLL (Orientation of the Last Layer), and PLL (Permutation of the Last Layer). When every colored block is correctly aligned with its corresponding face, the Rubik's cube is said to be solved.

Two testing scenarios were conducted: the successful completion test and the execution time test. The successful completion test showed that the system achieved a 71% accuracy rate in solving random Rubik's Cube configurations, with an average execution time of 9 seconds per rotation. Meanwhile, the execution time test revealed that the completion time increased proportionally with the complexity of the initial cube state, indicating that more scrambled

configurations require more computational effort to solve.

The findings show that the MDP-based reinforcement learning approach can provide reasonably good performance in solving Rubik's Cube problems, especially for moderately scrambled positions. However, the 71% success rate also highlights the limitations of the current model, particularly in handling more complex or deeply scrambled Rubik's Cube states. The execution time, although relatively efficient for simpler cases, tends to increase significantly with problem complexity, indicating that the learning policy could be further optimized.

One of the critical challenges observed during this research is the rotational decision-making process for highly scrambled positions. The action selection mechanism, while functional, still requires refinement to prioritize more optimal solving paths. This creates chances for future studies to work on better rotation methods, possibly by using smart reinforcement learning techniques like Deep Q-Networks or policy-gradient approaches to make things more accurate and work faster.

Furthermore, future research could explore the implementation of heuristic or hybrid approaches, combining reinforcement learning with classical search algorithms to balance between learning flexibility and computational efficiency. By optimizing these aspects, it is expected that the execution time can be reduced, especially for solving highly randomized Rubik's Cube positions, thereby making the application more robust and effective across a wider range of problem complexities.

References

- [1] K. Boyce and C. Storm, "Rubik's Cube: What Separates the Fastest Solvers From the Rest?," *J Emerg Investig*, vol. 5, pp. 1–6, Jul. 2022, doi: 10.59720/21-189.
- [2] E. S. Toshniwal and Y. Golhar, "Rubik's Cube Solver: A Review," in *2019 9th International Conference on Emerging Trends in Engineering and Technology - Signal and Information Processing (ICETET-SIP-19)*, IEEE, Nov. 2019, pp. 1–5. doi: 10.1109/ICETET-SIP-1946815.2019.9092272.
- [3] M. G. Mohanan and A. Salgoankar, "A Survey of Robotic Motion Planning in Dynamic Environments," *Rob Auton Syst*, vol. 100, pp. 171–185, Feb. 2018, doi: 10.1016/j.robot.2017.10.011.
- [4] B. Muppasani, V. Pallagani, B. Srivastava, and F. Agostinelli, "On Solving the Rubik's Cube with Domain-Independent Planners Using Standard Representations," *arXiv.org*, Jul. 2023, [Online]. Available: <http://arxiv.org/abs/2307.13552>
- [5] M. Liu, "Introduction to Reinforcement Learning," in *Machine Learning, Animated*, Boca Raton: Chapman and Hall/CRC, 2023, pp. 264–284. doi: 10.1201/b23383-14.
- [6] M. Vidyasagar, "A Tutorial Introduction to Reinforcement Learning," *SICE Journal of Control, Measurement, and System Integration*, vol. 16, no. 1, pp. 172–191, Dec. 2023, doi: 10.1080/18824889.2023.2196033.
- [7] A. M. K. Adawadkar and N. Kulkarni, "Cyber-Security and Reinforcement Learning — A Brief Survey," *Eng Appl Artif Intell*, vol. 114, p. 105116, Sep. 2022, doi: 10.1016/j.engappai.2022.105116.
- [8] Z. Lyu, Z. Liu, A. Khojandi, and A. Junfang Yu, "Q-learning and traditional methods on solving the pocket Rubik's cube," *Comput Ind Eng*, vol. 171, 2022, doi: 10.1016/j.cie.2022.108452.
- [9] M. E. Chasmai, "CubeTR: Learning to Solve The Rubiks Cube Using Transformers," in *ICLR 2022 Conference*, Nov. 2023. doi: 10.48550/arXiv.2111.06036.
- [10] M. Zuccotto, A. Castellini, D. La Torre, L. Mola, and A. Farinelli, "Reinforcement learning applications in environmental sustainability: a review," *Artif Intell Rev*, vol. 57, no. 4, p. 88, Mar. 2024, doi: 10.1007/s10462-024-10706-5.
- [11] blackburn, "Introduction to Reinforcement Learning : Markov-Decision Process," Medium.
- [12] blackburn, "Reinforcement Learning : Markov-Decision Process (Part 2)," Medium.
- [13] R. S. Sutton and A. G. Barto, *Reinforcement Learning : an Introduction*, 2nd ed. Cambridge: MIT Press, 2018. Accessed: Jun. 25, 2024. [Online]. Available: <https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf>
- [14] H. Pan and R. Kondor, "Fourier Bases for Solving Permutation Puzzles," *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 130, pp. 172–180, 2021.
- [15] E. Testé, "Rubik's Cubes and Markov Chains," Toward Data Science. Accessed: Jul. 10, 2024.[Online].Available: <https://towardsdatascience.com/a-random-walk-on-the-rubiks-cube-684dd304bf3e>
- [16] A. F. Mukminin, "Aplikasi Reinforcement Learning untuk Menyelesaikan Rubik's Cube dengan Markov Decision Process Berbasis Web," Computer Science, Politeknik Negeri Padang, Padang, 2023.
- [17] X. Wang, Z. Yang, G. Chen, and Y. Liu, "A Reinforcement Learning Method of Solving Markov Decision Processes: An Adaptive Exploration Model Based on Temporal Difference Error," *Electronics (Basel)*, vol. 12, no. 19, p. 4176, Oct. 2023, doi: 10.3390/electronics12194176.
- [18] C. G. Johnson, "Solving the Rubik's Cube with Stepwise Deep Learning," *Expert Syst*, vol. 38, no. 3, May 2021, doi: 10.1111/exsy.12665.
- [19] F. Agostinelli, S. McAleer, A. Shmakov, and P. Baldi, "Solving the Rubik's Cube with Deep Reinforcement Learning and Search," *Nat Mach Intell*, vol. 1, no. 8, pp. 356–363, Jul. 2019, doi: 10.1038/s42256-019-0070-z.