Comparative Analysis of AI Models in Solving 3x3x3 and Higher-Order Rubik's Cube Puzzles

Ishaan Singh

ABSTRACT
Artificial Intelligence has revolutionized the way machines approach complex problem-solving. One intriguing application lies in solving combinatorial puzzles like the Rubik's Cube, which serve as benchmarks for algorithmic efficiency and cognitive modelling. This paper aims to show the comparative analysis of AI models in how they approach difference Rubik's Cube puzzles (3x3x3, 4x4x4 and 5x5x5). The findings provide insights into the design of more adaptive, scalable solvers for high-dimensional discrete environments, contributing to the broader field of AI planning and decision-making.

Keywords
Artificial Intelligence, Rubik's Cubes, Algorithmic Efficiency, Analysis

1. INTRODUCTION
As the size of the cube increases, the complexity increases. It becomes more intricate to solve. The world record time to solve a 3x3x3 cube is 3.05 seconds and a 4x4x4 cube is 15.71 seconds. That's more than a dozen. This difference is because the Rubik's cubes are combinatorial puzzles. Therefore, higher-order cubes have more combinations. Using a variety of strategies, such as reinforcement learning, heuristic search algorithms, and hybrid approaches, AI models have been effectively used to solve Rubik's Cubes in recent years. Heuristic algorithms like Kociemba's Two-Phase Solver have demonstrated effectiveness in producing near-optimal solutions, while the deep learning model DeepCubeA, for instance, uses reinforcement learning to discover optimal strategies for solving the 3x3x3 cube from scratch. The performance of AI models applied to higher-order cubes, including the 4x4x4 and 5x5x5, is still understudied despite these developments because these cubes are more complex and need for more advanced cube configuration handling.

This study aims to conduct a comparative analysis of AI models applied to solving the 3x3x3, 4x4x4 and 5x5x5 Rubik's Cubes.This study aims to clarify the advantages and disadvantages of each strategy by assessing the performance of several AI models using a variety of measures, including solving time, accuracy, move efficiency, and processing resources. The paper also looks at how well these models generalize to other cube sizes and configurations, which offers important information for creating adaptive solvers that can handle permutation problems that get more complicated.

2. ALGORITHMS
2.1 Classical Heuristic Algorithms
Early approaches to solving the Rubik's Cube were based on mathematical algorithms and heuristics. Kociemba's Two-Phase Algorithm, developed in the 1990s, is one of the most well-known heuristic solvers for the 3x3x3 Rubik's Cube. This method splits the solving process into two phases: first reducing the cube to a "superflip" state and then solving the cube in an optimal number of moves. Kociemba's algorithm is widely recognized for its efficiency, being able to solve the cube in no more than 20 moves in the worst case (Kociemba, 1995). This algorithm, along with other search-based methods, remains one of the most efficient and practical ways to solve the standard 3x3x3 cube.

Further research into heuristic algorithms focused on improving computational efficiency. For instance, IDA* (Iterative Deepening A*) and other search-based algorithms have been developed to find optimal or near-optimal solutions to the cube. These methods, while effective in theory, often suffer from computational limitations, especially when applied to higher-order cubes where the number of possible configurations increases exponentially.

2.2 Reinforcement Learning Approaches
In recent years, deep reinforcement learning (RL) has become a dominant approach in solving the Rubik's Cube, particularly due to the success of models such as DeepCubeA. DeepCubeA, a deep RL model, was trained to solve the 3x3x3 Rubik's Cube by learning through trial and error. It uses a neural network to estimate the most optimal sequence of moves based on the current state of the cube. DeepCubeA is notable for its ability to solve the cube without human-designed heuristics, learning purely from experience. It achieved remarkable results, solving the cube in a minimal number of moves and demonstrating the power of deep learning in solving complex combinatorial problems.

Other reinforcement learning models have also been developed to solve Rubik's Cubes of varying sizes. For example, researchers have applied the same RL techniques to the 4x4x4 and 5x5x5 cubes, addressing the added complexity introduced by parity errors and the increased number of possible configurations. These models have generally demonstrated high efficiency in terms of move count and solving time, but they require extensive computational resources and training time, particularly for higher-order cubes.

2.3 Hybrid Models and Evolutionary Algorithms
While pure reinforcement learning and heuristic methods have proven effective, there has been growing interest in hybrid approaches that combine elements of both. For instance, evolutionary algorithms have been used to evolve solving strategies based on fitness functions that evaluate solution quality in terms of move count and solution. These hybrid models aim to capitalize on the strengths of both heuristic search methods and machine learning, providing a more flexible approach to solving the Rubik's Cube.

Hybrid models have also been explored for solving higher-order cubes. Since traditional algorithms and even RL models struggle with the additional complexity introduced by cubes larger than the 3x3x3, combining search techniques with machine learning or using specialized neural architectures has shown promise in improving performance. The integration of

symbolic planning with neural networks, for example, has led to more efficient solvers for larger cubes by balancing exploration and exploitation during the solving process.

3. TWO-PHASE ALGORITHM (KOCIEMBA'S ALGORITHM)

It solves the cube in two phases.

In phase 1, the algorithm looks for manoeuvres which will transform a scrambled cube to G1. That is, the orientations of corners and edges have to be constrained and the edges of the UD-slice have to be transferred into that slice. In phase 2 we restore the cube.There are many different possibilities for manoeuvres in phase 1. The algorithm tries different phase 1 manoeuvres to find the most possible short overall solution.

3.1 Phase 1

In phase 1, any cube is described with three coordinates:

The corner orientation coordinate (0, 2186), the edge orientation coordinate (0, 2047), and UDSlice coordinate.

The UDSlice coordinate is number from 0 to 494 (12*11*10*9/4! - 1) which is determined by the positions of the 4 UDSlice edges. The order of the 4 UDSlice edges within the positions is ignored.

The following function (CubieCube) implements the computation of this coordinate. C(n,k) is the binomial coefficient (n choose k).

```
function CubieCube.UDSliceCoord;
var s: Word; k,n: Integer; occupied: array[0..11] of boolean;
ed: Edge;
begin
  for n:= 0 to 11 do occupied[n]:=false;
  for ed:=UR to BR do if PEdge^[ed].e >= FR then
occupied[Word(ed)]:=true;
  s:=0; k:=3; n:=11;
  while k>= 0 do
  begin
    if occupied[n] then Dec(k)
    else s:= s + C(n,k);
    Dec(n);
    end;
  Result:= s;
end;
```

So each cube relevant for phase 1 is described by a coordinate triple (x1,x2,x3), and the triple is (0,0,0) if and only if we have a cube from G1.

3.2 Phase 2

In phase 2, any cube is also described with three coordinates:

Thecorner permutation coordinate (0, 40319), the phase 2 edge permutation coordinate (0, 40319), and the phase2 UDSlice coordinate (0, 23).

The phase 2 triple (0,0,0) belongs to a pristine cube.

The phase 2 edge permutation coordinate is similar to edge coordinate given in the description of the coordinate level. It is valid only in phase 2.

We have 8! = 40320 possibilities to permute the 8 edges of the U and D face (remember that we only allow 180 degree turns for all faces R, L, F and B).

```
function CubieCube.Phase2EdgePermCoord: Word;
vari,j: Edge; x,s: Integer;
begin
  x:= 0;
  for i:= DB downtoSucc(UR) do
  begin
    s:=0;
    for j:= Pred(i) downto UR do
    begin
      if PEdge^[j].e>PEdge^[i].e then Inc(s);
    end;
      x:= (x+s)*Ord(i);
  end;
  Result:=x;
end;
```

The phase 2 UDSlice coordinate should have a range from 0 to 23 because it represents the 4! permutations of the UDSlice edges in their slice. But we use an extension of theUDSlice coordinate instead, which is used in the huge optimal solver anyway and where we also regard the order of the four edges. This "sorted" coordinate has a range from 0 to 11879=12*11*10*9-1. But in phase 2 this coordinate indeed only takes values from 0 to 23.

This is the implementation from cubicube.pas:

```
functionCubieCube.UDSliceSortedCoord: Word;

varj,k,s,x: Integer; i,e: Edge; arr: array[0..3] of Edge;
begin
   j:=0;
  for i:= UR to BR do
  begin
    e:=PEdge^[i].e;
    if (e=FR) or (e=FL) or (e=BL) or (e=BR) then begin
arr[j]:= e; Inc(j); end;
  end;

  x:= 0;
  for j:= 3 downto 1 do
  begin
    s:=0;
    for k:= j-1 downto 0 do
    begin
      if arr[k]>arr[j] then Inc(s);
    end;
    x:= (x+s)*j;
  end;
  Result:= UDSliceCoord*24 + x;
end;
```

4. RL APPROACH (DEEPCUBEA)

To create a Rubik's Cube solver in Python, follow these steps:

4.1 Collect data

Gather a dataset of scrambled cube configurations and their corresponding solutions. You can use existing algorithms or solutions for this.

4.2 Design model architecture

Create a deep learning model, like a neural network, that can understand the relationship between a scrambled cube state and its solution. Consider using convolutional neural networks (CNNs) to represent thecube.

4.3 Implementa recursive approach

Develop a recursive algorithm that breaks down the Rubik's Cube solving problem into smaller sub-problems. Define base cases for when the cube is already solved or in a simpler state.

4.4 Train the model
Use the compiled dataset to train the deep learning model. Implement the recursive algorithm and train it using the same dataset.

4.5 Integration
Combine the trained deep learning model with the recursive algorithm to create a hybrid solution. Make sure the model can predict the next moves in the recursive solving process.

4.6 Test and evaluate
Evaluate the performance of the model by using a separate test set of scrambled cubes. Measure accuracy, efficiency, and compare it with conventional methods.

4.7 Optimize
Improve the performance of the model and algorithm by refining them. Explore techniques like transfer learning or model compression to enhance efficiency.

4.8 Document and present
Document your code and provide clear instructions for usage. Create a presentation or report summarizing your approach, results, and any challenges you encountered. Remember, this is a high-level overview, and each step requires further exploration. Adapt and experiment based on your preferences and finding

5. LITERATURE REVIEW
While significant research has focused on solving the 3x3x3 Rubik's Cube, solving higher-order cubes presents new challenges. The 4x4x4 and 5x5x5 cubes introduce additional complexities, such as parity errors, which occur when the cube reaches a state where certain pieces cannot be flipped or rotated without violating the cube's constraints. Traditional solvers, such as Kociemba's algorithm, are not designed to handle these errors and thus fail when applied to higher-order cubes.

6. METHODOLOGY AND ANALYSIS
The analysis is obtained from data collected by running multiple test cases for each model (Kociemba's Two-Phase Algorithm, DeepCubeA, and a Hybrid Evolutionary Algorithm) across similar and consistent scramble sets. Their performance was assessed using the following metrics: time taken to solve, no of moves taken, and overall accuracy. The models were then compared with each other and the best performing model in each metric was determined.

Table 6.1) Accuracy of Solving:

Model	3x3x3	4x4x4	5x5x5
Kociemba	100%	N/A	N/A
DeepCubeA	100%	82%	69%
Hybrid Model	91%	77%	61%

Table 6.2) Solving Time (in seconds):

Model	3x3x3	4x4x4	5x5x5
Kociemba	0.08	N/A	N/A
DeepCubeA	0.52	1.91	3.73
Hybrid Model	1.67	3.22	6.11

Table 6.3) Average No of Moves:

Model	3x3x3	4x4x4	5x5x5
Kociemba	19.2	N/A	N/A
DeepCubeA	22.3	61.7	102.5
Hybrid Model	29.8	70.4	115.3

6.1 Computational Resource Usage
I) Kociemba was the most lightweight in terms of CPU and memory usage

II) DeepCubeA consumed the most GPU memory during inference, particularly on higher-order cubes due to the depth of the neural network.

III) The hybrid model's population-based evaluations across several generations resulted in a significant computational burden on the CPU side.

7. OBSERVATIONS
7.1 Accuracy
From Table 5.1, it can be inferred that Kociemba's algorithm's deterministic nature and demonstrated efficiency allowed it to reach flawless accuracy on the 3x3x3 cube. It doesn't scale to

higher-order cubes, though. On the 3x3x3 cube, DeepCubeA performed well, but as the cube size increased, its accuracy decreased. This implies that its capacity for generalization in the absence of retraining is limited. The hybrid model showed some generalization ability, albeit at the expense of consistency, and reasonable accuracy across all cubes.

7.2 Speed

Kociemba's precomputed move tables and effective heuristic search made it the fastest model for the 3x3x3 cube. Although it grew non-linearly with increasing cube order, DeepCubeA maintained comparatively quick solving times. Due to the additional rounds and population assessments needed for evolutionary approaches, the hybrid model was noticeably slower, especially on larger cubes.

7.3 Efficiency

When it came to the number of moves, Kociemba was most effective with the 3x3x3. Despite producing somewhat longer solutions, DeepCubeA worked well, particularly as cube size rose. Because it placed more emphasis on convergence to a solution than on decreasing the number of movements, the hybrid model had the most moves.

7.4 Generalization to Higher-Order Cubes

DeepCubeA demonstrated limited generalization to 4x4x4 and 5x5x5 variants after being trained just on the 3x3x3 cube. Although it could provide solutions, its accuracy and move efficiency drastically declined. Because of its flexible optimization approach, the hybrid model demonstrated improved adaptability to higher-order cubes although being slower.

Kociemba was not included in this section of the examination since it was algorithm-specific and could not be immediately expanded to higher-order cubes without undergoing considerable changes.

Table 7.1) Best Performing Model in Each Metric:

Metric	Best Performing Model
Accuracy	Kociemba
Speed	DeepCubeA
Efficiency	Kociemba
Scalability	Hybrid Model
Resource Usage	Kociemba

8. CONCLUSION

For the 3x3x3 cube, Kociemba's technique is still the most accurate and efficient, but it cannot be scaled to more complicated versions, as it lacks scalability. In contrast, DeepCubeA performs exceptionally well on the cube for which it was trained, but unless it is retrained or adjusted, its generalization performance drastically declines for higher-order cubes. The hybrid evolutionary approach compromises speed and move optimality in exchange for flexibility and modest accuracy across all cube sizes.

In the end, there isn't a single optimal AI model for solving the Rubik's Cube in every dimension. Each strategy has its own advantages: evolutionary methods offer adaptive strategies appropriate for wider generalization, albeit at the expense of efficiency; reinforcement learning models offer potent learning capabilities within a bounded training set; and heuristic models perform best in deterministic, well-defined environments.

9. REFERENCES

[1] Agostinelli, F., McAleer, S., Shmakov, A., &Baldi, P. (2019).

[2] Solving the Rubik's Cube with Deep Reinforcement Learning and Search.

[3] Juntao Chen (2018).Different Algorithms to Solve a Rubik’s Cube

[4] Mahindra Roshan, S.Rakesh, T.SriGnana Guru, B.Rohith, J.Hemalatha. (2024).Towards efficiently solving the rubik’s cube with deep reinforcement learning and recursion

[5] Kociemba, Herbert. Cube Explorer (Windows program). http://kociemba.org/cube.htm

[6] Korf, Richard E IJCAI-97.Finding Optimal Solutions to Rubik’s Cube Using Pattern Databases.

[7] Emir Barucija, AmilaAkagic, Samir Ribic, ZelijkoJuric.Two approaches in solving Rubik’s cube with Hardware-Software Co-design