Rubik's Cube: Artificially Intelligent Solvers
CS7IS2 Project (2020/2021)

William O'Sullivan, Basil Contovounesios, Talha Ijaz, and Fionntán Ó Suibhne

wosulliv@tcd.ie, contovob@tcd.ie, ijazm@tcd.ie, suibhnef@tcd.ie

Abstract An investigation into solving Rubik's Cubes by applying the following AI strategies was performed:
- Genetic Algorithms (GA)
- Deep Reinforcement Learning + A* Search (DRL + A*)
- Simulated Annealing (SA)

These approaches were used in order to determine solutions to a $2 \times 2 \times 2$ ($n = 2$) "Pocket Cube" and a classic $3 \times 3 \times 3$ ($n = 3$) Rubik's Cube. The techniques broadly met with success, with every technique achieving a solution for a randomised cube. Times taken to solve each configuration varied between each instance, as did the number of turns required by each method to reach a solution. The DRL + A* method required the most up-front time due to its training component, but it

Keywords: genetic algorithms, deep reinforcement learning, simulated annealing

1 Introduction

1.1 Motivation

1974 saw many historical events, including the election of the first female President of Argentina and ABBA's victory at the Eurovision with their iconic hit "Waterloo". 1974 also bore witness to the invention of the Rubik's Cube, a puzzle which has been in the minds of both pure mathematicians and computer scientists alike for decades. Whilst the puzzle has now been bested (even giving rise to the phenonmenon of Speedcubing!) it still serves as region of interest in fields where there is more concern for the method of solution.

Puzzles have long been a useful tool for examining the utility of artificial intelligence. Puzzles provide us with what would be considered an intellectual challenge, and creating agents that can produce solutions to puzzles can give us greater insight into both the limitations of such intelligences, as well as their strengths highlighting key areas where AI may find further use.

In the specific case of this investigation, we examine different agents for the problem of solving a Rubik's Cube, or indeed its smaller cousin the $2 \times 2 \times 2$ Pocket Cube, which will generally be described in the same way as it is simply a variant with a smaller state space. This particular puzzle sits at a cross-roads in terms

of analysis. The Rubik's Cube is by no means a trivial pathfinding algorithm owing to the importance of move order but neither is it a puzzle without a known solution. That said, the Rubik's Cube itself only represents one member of a family of similar puzzles, for example higher dimensional $2 \times 2 \times 2 \times 2$ objects, or longer sided $4 \times 4 \times 4$ cubes. A number of researchers have spent time investigating different methods of solving the cube via these intelligent agents only for some to determine that their approach does not suit the Rubik's Cube, whilst others meet with successful solutions.

At the outset, it was not expected that all of the approaches would yield perfect solutions to the Rubik's Cube. The heart of the investigation is built around understanding the methods and their applications in order to solve problems of this type, and this report will endeavour to share those findings accordingly.

NOTE: INCLUDE ONE_DRIVE LINK

2 Related Work

In keeping with the directive that the focus of the investigation should be on recent solutions of this field of problem, one group have been quite prolific in their publications on Rubik's Cube solvers in particular. The work of Stephen McAleer, Forest Agostinelli, Alexander Shmakov and Pierre Baldi [1, 2, 3] formed a consistent basis of understanding for the Rubik's Cube problem as a whole, whilst also drawing attention to the use of Deep Reinforcement Learning centred around their DeepCube solver. It was possible for the researchers to use the policies determined by DRL with different search approaches, including Monte Carlo Tree Search (MCTS) and A* Search.

Beyond the use of DRL, however, lay two other approaches which draw influence from biology and physics, in Genetic Algorithms and Simulated Annealing respectively.

Genetic Algorithms are a way of generating increasingly close approximations to a solution, to the point where a solution itself is achieved. The work of El-Sourani et al. [4] utilised GA to solve a Rubik's Cube by incorporating the analytical solution to a Rubik's Cube based on the original work by Thistlethwaite [5]. Whilst this technique resulted in solutions for any given cube configuration, Smith et al. [6] attempted to build on this by determining policies with their GA rather than determining configuration-unique solutions, ultimately meeting with a great degree of success.

Simulated Annealing is a kind of random search method not unlike the aforementioned MCTS, however it possesses an additional strength in that it is also an iterative improvement algorithm. This means that the SA approach is able to escape local extrema, whilst consistently improving performance. SA has seen use in solving Sudoku puzzles, as in the work of Lewis [7], and indeed SA could be applied to solving Rubik's Cubes by tweaking how the cost function is determined.

Between the applications of DRL + MCTS, DRL + A*, GA for values, GA for policies and SA, a decision was made to investigate the properties of DRL + A*, GA for values, and SA.

DRL + A* was selected as an investigation into the most modern approach to solving Rubik's Cubes. GA for values would allow for investigation of the differences between performance on $n = 2$ and $n = 3$ cubes without the added complexity of trying to determine policies for the different sized cubes. Finally, SA was selected in order to examine the effects of stochastic solvers in Rubik's Cubes, which had been scarcely investigated.

Overall, the proposed techniques comprise a decent and broad collection, and draw on a diverse set of approaches to solve one popular problem.

It should be noted that an implementation [8] based the work of Korf [9] was used as a baseline for comparison in terms of runtime. It is important to note that this algorithm does rely on human (domain) knowledge however, and so acts not as a baseline to beat necessarily, but a good comparison between the agents which do not have human knowledge, and those that do.

3 Problem Definition and Algorithm

The Rubik's Cube for both $n = 2$ and $n = 3$ is a sparse state space problem, meaning that only one solution exists in an otherwise very large state space. This presents unique challenges to finding suitable techniques for devising solutions to the cubes.

3.1 Deep Reinforcement Learning + A* Search

Reinforcement Learning (RL) involves exploring state spaces in order to determine optimal policies for maximising rewards. In the case of a Rubik's Cube, the state space is incredibly sparse, with $\sim 10^{19}$ possible states, and only one solution. As such, classical RL is insufficient to solve a Rubik's Cube, leading to the requirement of DRL. DRL utilises Deep Learning to more efficiently operate over a large state space by eliminating the need to store it directly. Instead, models are trained to compute effective outputs from unstructured inputs by iteratively approximating a function of the domain.

Unfortunately, even with these advanced techniques the Rubik's Cube proved unsuitable for solving with DRL alone. The sparse nature of the problem means that the learning element can often fail to propagate rewards from the goal state. To alleviate this problem, the way in which the DRL training occurs is reversed; rather than starting with a randomised cube and trying to solve from there, a solved cube is randomised with the agent knowing how the random state was reached. This is called Autodidactic Iteration, and ensures that every training instance can result in a solution. It is in this way that training samples for the DNN can be selected to yield a useful basis for training a 'cost-to-go' function J that estimates the number of moves of a given state s from the goal state. Training the DNN involves minimising the MSE between the current $J(s)$ and

its update $J'(s) = \min_a J(A(s,a))$, where $A(s,a)$ is the state resulting from taking action a in s, in a process named Deep Approximate Value Iteration [3].

The trained function can then be used to inform a search strategy for solving Rubik's Cubes. Greedy Best-First Search was found to perform badly with more scrambled cubes, so a weighted A* Search with the following cost function f is used instead:

$$f(x) = \lambda g(x) + h(x), \qquad \lambda \in [0,1]$$
$$h(x) = \begin{cases} 0 & \text{if } x \text{ is the goal node} \\ J(x) & \text{otherwise} \end{cases}$$

Controlling the weighting factor λ on the current path cost function g can afford better runtime characteristics at the expense of optimality. To further offset the DNN evaluation overhead, the frontier is expanded in parallel batches, giving rise to Batch Weighted A* Search (BWAS). This investigation examined the effects of varying λ on runtime efficiency and solution optimality, with empirical results presented in Figure 4.

3.2 Genetic Algorithms

function GENETIC-ALGORITHM(*population*, FITNESS-FN) **returns** an individual
  inputs: *population*, a set of individuals
    FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for i = 1 **to** SIZE(*population*) **do**
      x ← RANDOM-SELECTION(*population*, FITNESS-FN)
      y ← RANDOM-SELECTION(*population*, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *new_population*
    population ← *new_population*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to FITNESS-FN

Figure 1: Genetic Algorithms utilise reproduction, whereby a parent passes its characteristics onto its child. Over several generations an individual that better satisfies the fitness criteria emerges [10].

The applications of GA to solve Rubik's Cubes in the literature were based on an analytical solution. The analytical solution broke the method into four distinct parts, based on human knowledge, and so the GA approach utilised these conditions as waypoints between the evolution of population. GA define

populations as a set of approximate solutions, with each individual representing one such attempt. Certain individuals are better suited to meeting fitness criteria (i.e. how well a solution conforms to the required behaviour) and so upon the passage of a generation, individuals with better fitness have a higher probability to reproduce, creating a new individual with characteristics from two previous approximations. However, reproduction is not the only way in which the population evolves. Concepts of mutation and elitism allow for the propagation of individuals with desirable traits as well. Overall, the passage from one generation to the next is composed of 40% offspring, 50% by mutation and 10% by elitism. Offspring come about based on the previous notion of reproduction, whilst mutants are the result of lone individuals experiencing a number of small move alterations, and elitism allows for the propagation of the portion of the population that was judged to be most fit. The elites are selected from the best population that did not produce offspring, as to give them another chance to reproduce.

The idea is that these new individuals will possess a combination of attributes that allow them to satisfy the fitness function even better, in turn increasing the chance of their characteristics being passed along. In this manner that mimics natural selection, over a number of generations it becomes possible to arrive at a population that contains individuals which satisfy the fitness criteria perfectly, at which point a solution has been reached.

In this investigation, GA were used to solve a Rubik's Cube without human knowledge. The analytical solution modified the fitness criteria at different stages, in accordance with the Thistlethwaite methods, such that the direction of evolution was modified to become increasingly adept at solving the Rubik's Cube in a low number of steps. This particular implementation however did not alter the cost function at any point, and so the GA could only act to minimise the following cost function:

$$\text{All elements of a row} \begin{cases} +0 & \text{all colours uniform} \\ +1 & \text{all colours not uniform} \end{cases}$$

$$\text{All elements of a column} \begin{cases} +0 & \text{all colours uniform} \\ +1 & \text{all colours not uniform} \end{cases}$$

$$\text{All elements of a face} \begin{cases} +0 & \text{all colours uniform} \\ +1 & \text{all colours not uniform} \end{cases}$$

These cost conditions were used to calculate the cost on each face, which in turn were used to determine the cost of the overall cube. For example, whilst a cost 0 cube gives a completely solved system, a cost 12 cube implies that the cube may be at most a half-turn (180°) away from a solution.

3.3 Simulated Annealing

SA begins as a process with an initial random state with a corresponding initial "temperature". This temperature affects the probability of accepting unfavour-

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
inputs: *problem*, a problem
schedule, a mapping from time to "temperature"

current ← MAKE-NODE(*problem*.INITIAL-STATE)
for $t = 1$ **to** ∞ **do**
$T \leftarrow schedule(t)$
if $T = 0$ **then return** *current*
next ← a randomly selected successor of *current*
$\Delta E \leftarrow next$.VALUE – *current*.VALUE
if $\Delta E > 0$ **then** *current* ← *next*
else *current* ← *next* only with probability $e^{\Delta E/T}$

Figure 2: By slowly approaching a solution state using a controlled stochastic method, Simulated Annealing results in a solved system [10].

able moves that diminish the overall fitness of the current state. Whilst this may sound counter-productive, it is very important for preventing the Rubik's Cube from becoming stuck in a degenerate state, whereby it is unable to access moves that would diminish the cost function even further than the present state. This is a critical aspect of solving the puzzle on account of the fact that moves in Rubik's Cubes are non-commutative, that is to say, a move left followed by a move up is not identical to a move up followed by a move left. After a number of moves have occurred, the temperature begins to decrease, reducing the probabilities of accepting less favourable moves. This process continues until the system "cools" and either a solution has been reached, or the system is then "reheated" in a bid to shuffle around moves and solve the cube outright. Having specified parameters such as the initial temperature and cooling rate of the system, it was possible to investigate the effects of modifying these parameters to see whether a solution would successfully converge or not.

4 Experimental Results

4.1 Methodology

The solutions to both the Pocket Cube ($n = 2$) and Rubik's Cube ($n = 3$) were evaluated using two general criteria where they could be applied, counting the number of moves that a method took to reach a solution, and measuring the time taken for an individual solution to a newly randomised cube.

Whilst the DRL + A* solution for the $n = 2$ and $n = 3$ cubes was built upon the existing DeepCube environment, the GA and SA approaches required a custom test environment in which a cube could be evaluated in terms of cost and solution.

Shared Basis for GA and SA Before any algorithms could be applied a test environment had to be constructed. A `RubiksCube` object was constructed containing several features:

- A method for determining the current 'cost' of the cube. A solved Rubik's Cube would have a cost of 0, with displacement from a solved state resulting in an increased cost which scaled with n, the dimension of the cube. The specific way in which cost was determined was then unique for each technique.
- A method to generate a random move, and from that a method to randomise the cube itself. This would allow the cube to initialise into a state following 30 to 40 randomisation operations.
- A method to apply either the random moves or a set of moves specified by the agent. Moves would be entered as a combination of a row/column specifier and a direction declaration.

Figure 3: Taking the nearer (highlighted) face as the frame of reference, this particular transformation would be described as a $[2, u]$ move. Using index notation, the third column of the face is rotated upwards by 90° [11].

This cube could be manipulated, as in Figure 3 by an agent in order to reach a solution after a successful set of moves had been made.

4.2 Results

Table 1 presents the mean runtime and the mean number of moves for solving both the Pocket Cube and Rubik's Cube, over what were considered idealised parameters.

DRL + A*

GA

SA

Baseline

Size	Metric	DRL + A*	GA	SA	Baseline
$n = 2$	Runtime (s)	3.2	0	0	10
	# of Moves	9.8	0	0	17
$n = 3$	Runtime (s)	13.6	0	0	15
	# of Moves	22.1	0	0	32

Table 1: Mean empirical metrics across cube sizes

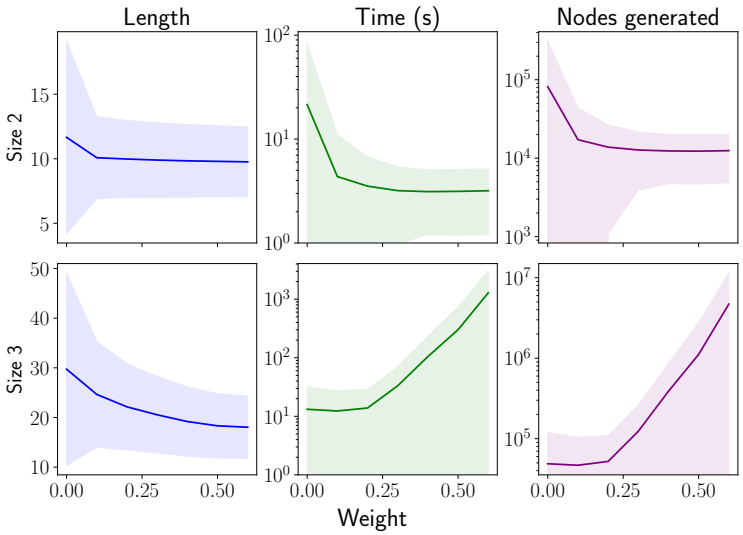

Figure 4: BWAS solutions ($\mu \pm \sigma$) with a batch size of 100 (the number of nodes that can be expanded simultaneously), for cubes up to 100 randomised moves away from the goal state. The length of a solution is also known as the number of moves required to reach a solved state.

4.3 Discussion

It should be noted that the common metric used by the models, that is runtime and number of moves, immediately runs into a dilemma—should the training time for the DRL + A* model included when measuring solution time? One could argue either way, but it is important to highlight that whilst the DRL + A* ran fastest, there was certainly a significant degree of overhead to its use.

Beyond that lay the parameters unique to each model, be it the architecture and hyperparameters of the DNN, the A* batch size and weightings, the rate of mutation in the GA or the cooling rates in the SA. Modifying these values has well understood implications when the problems are treated in tandem—it is possible to see that in the case of GA and SA high mutation rates and high starting temperatures increase the time taken to arrive at a solution state, and yet such an analogous relationship does not exist for the DRL + A*. However,

one could (inelegantly) compare the number of generations in a GA with the number of layers in the DNN used in the training process, arguing that many inputs are taken to ultimately return a unique item of use. According to present understanding, there is no such trait that can be described as truly common between the three methods employed in this investigation; where one analogy holds for two techniques, it either fails to capture the third outright or else makes the vaguest approximation to what is actually going on within the model.

5 Conclusions

Provide a final discussion of the main results and conclusions of the report. Comment on the lesson learnt and possible improvements.

A standard and well formatted bibliography of papers cited in the report. For example:

References

[1] Stephen McAleer et al. *Solving the Rubik's Cube Without Human Knowledge*. Version 1. 18th May 2018. arXiv: 1805.07470 [cs.AI].

[2] Stephen McAleer et al. 'Solving the Rubik's Cube with Approximate Policy Iteration'. In: *ICLR 2019*. Proceedings of the Seventh International Conference on Learning Representations (New Orleans, LA, USA). May 2019.

[3] Forest Agostinelli et al. 'Solving the Rubik's cube with deep reinforcement learning and search'. In: *Nature Machine Intelligence* 1.8 (2019), pp. 356–363.

[4] Nail El-Sourani, Sascha Hauke and Markus Borschbach. 'An Evolutionary Approach for Solving the Rubik's Cube Incorporating Exact Methods'. In: *Applications of Evolutionary Computation*. Ed. by Cecilia Di Chio et al. Germany: Springer, 2010, pp. 80–89. ISBN: 978-3-642-12239-2. DOI: 10.1007/978-3-642-12239-2_9.

[5] Morwen B. Thistlethwaite. 'The 45-52 Move Strategy'. In: *London CL VIII* (1981).

[6] Robert J. Smith, Stephen Kelly and Malcolm I. Heywood. 'Discovering Rubik's Cube Subgroups Using Coevolutionary GP: A Five Twist Experiment'. In: *GECCO '16*. Proceedings of the Genetic and Evolutionary Computation Conference 2016 (Denver, CO, USA). New York, NY, USA: Association for Computing Machinery, 2016, pp. 789–796. ISBN: 978-1-450-34206-3. DOI: 10.1145/2908812.2908887.

[7] Rhyd Lewis. 'Metaheuristics can solve Sudoku puzzles'. In: *Journal of Heuristics* 13.4 (July 2007), pp. 387–401. DOI: 10.1007/s10732-007-9012-8.

[8] Farhan Shoukat. *Rubik's Cube Solver*. Comp. software. Version a025187396. GitHub, 2019. URL: https://github.com/FarhanShoukat/Rubiks-Cube-Solver (visited on 22nd Apr. 2021).

[9] Richard E. Korf. 'Depth-first iterative-deepening: An optimal admissible tree search'. In: *Artificial Intelligence* 27.1 (Sept. 1985), pp. 97–109. ISSN: 0004-3702. DOI: 10.1016/0004-3702(85)90084-0.

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. USA: Pearson Education, 2010. ISBN: 978-0-136-04259-4.

[11] Le Thanh Hoang. 'Optimally Solving a Rubik's Cube Using Vision and Robotics'. MA thesis. Imperial College London, 15th June 2015. URL: https://www.doc.ic.ac.uk/teaching/distinguished-projects/2015/l.hoang.pdf (visited on 22nd Apr. 2021).