# Rubik's Cube: Artificially Intelligent Solvers

W. O'Sullivan[1] B. Contovounesios[2] T. Ijaz[3] F. Ó Suibhne[1]

[1]High Performance Computing MSc
School of Maths

[2]Integrated Computer Science BA & MCS
School of Computer Science & Statistics

[3]Computer Science: Intelligent Systems MSc
School of Computer Science & Statistics

Trinity College Dublin

# The Rubik's Cube

- A Rubik's Cube is a $3 \times 3 \times 3$ set of 26 "cubies", the small constituent cubes on the face of the object. The cubies can pivot about the centre of the cube, and the faces can be manipulated by rotation.

Figure: A timeless classic!

# The Rubik's Cube

- The Rubik's Cube remains a popular puzzle to this day, and whilst the cube has been solved analytically, it still serves as a fascinating problem for those of us concerned with how artificially intelligent agents solve problems.
- Our research has specifically examined methods for solving Rubik's Cubes (and its simpler $2 \times 2 \times 2$ sibling the Pocket Cube) that do not rely on human knowledge. That means that we only establish the conditions of the state space and cost functions for the respective techniques, but we do not avail of analytical solutions to the cube.

# Deep Reinforcement Learning w/ A* Search

- Reinforcement Learning (RL) involves exploring state spaces in order to determine optimal policies for maximising rewards. Deep Reinforcement Learning (DRL) leverages Deep Learning to handle very large state spaces by training models to more efficiently approximate domain functions.
- Whilst DRL certainly reduces the number of states that an agent has to explore (the Rubik's Cube contains $> 10^{19}$ unique states, and only one solution state), the sparsity of the state space means reaching the goal state must be in some way guaranteed.
- Autodidactic Iteration is the process whereby a solved cube is scrambled, the according moves recorded, and the move list and scrambled state are used as an input for the DRL training. This ensures that the goal state can always be reached, ultimately resulting in the solver learning how to design policies which solve the cube faster.

# Deep Reinforcement Learning w/ A* Search

- The actual solution of the cube is then performed using a weighted A* Search, using the trained model as a heuristic for the distance of a given state from the goal state.
- At each node, the successors are examined to see which node presents the lowest total cost according to both the cost from the starting node and the cost given by the heuristic.
- This corresponds to building out a policy of the fewest moves, with different weightings determining whether approximate solutions are prioritised earlier or whether much closer solutions are favoured albeit further down the line.

# Genetic Algorithms

- Genetic Algorithms (GA) work on a process that mimics Natural Selection, whereby individuals (that together create a population) represent approximate solutions. Over generations, through reproduction, mutation or simply being "well-suited", individuals are able to pass along their characteristics.

**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**
      $x \leftarrow$ RANDOM-SELECTION(*population*, FITNESS-FN)
      $y \leftarrow$ 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

# Genetic Algorithms

- Reproduction accounts for 40% of the population. Individuals with a good fitness have a higher chance of reproducing, and this entails two parents coming together and each having half of their characteristics go on to form a new individual. In the context of solving Rubik's Cubes, this means mixing together their moves to form a new policy.
- Mutation accounts for 50% of the population whereby an individual can undergo a change in a small number of the moves in their policy, and in random locations.
- Finally, 10% of the population who are of the highest fitness persist into the next generation, to give them an additional chance to pass along their satisfactory characteristics.

# Simulated Annealing

- Simulated Annealing (SA) is an approach that combines a stochastic method with an iterative improvement algorithm. By moving from a higher randomness state to a lower randomness state over time, the algorithm converges to produce a solved cube.

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

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

# Simulated Annealing

- SA is an approach originally designed to model (you guessed it) annealing, whereby a material, typically a metal, is heated to relieve stress. It is then cooled gradually to allow new structures to form such that the ductility is increased—a preferable configuration.
- It is in this way that we treat the Rubik's Cube solver, whereby we start with a "high-temperature" system and as the system cools according to a specified cooling rate, configurations of moves which minimise the cost-function arise.
- If the system does not converge, it undergoes a reheating, whereby the cooling resumes from a temperature slightly above the end temperature. This cooling and reheating allows for the last few degenerate states to resolve and achieve a true global minimisation of the cost function.

# Findings

- The various methods were compared using the runtime and the number of moves required to reach a solution.

| Size | Metric | DRL + A* | GA | SA | Baseline |
|---|---|---|---|---|---|
| $n = 2$ | Runtime (s) | 3.2 | 7 | 2.9 | 10 |
| | # of Moves | 10 | 29 | 1270 | 17 |
| $n = 3$ | Runtime (s) | 13.6 | 255 | 204 | 15 |
| | # of Moves | 22 | 350 | 43623 | 32 |

# Findings

- In this table we can readily observe that DRL + A* consistently beat the baseline using domain knowledge—a very impressive feat.
- It consistently produced the most efficient set of moves across all of the methods, and it only lost the the SA in terms of runtime for $n = 2$.

| Size | Metric | DRL + A* | GA | SA | Baseline |
|---|---|---|---|---|---|
| $n = 2$ | Runtime (s) | 3.2 | 7 | 2.9 | 10 |
| | # of Moves | 10 | 29 | 1270 | 17 |
| $n = 3$ | Runtime (s) | 13.6 | 255 | 204 | 15 |
| | # of Moves | 22 | 350 | 43623 | 32 |

# Findings

- GA solved the $n = 2$ consistently, and only solved the $n = 3$ cubes 27% of the time.
- The difficulty was attributed to becoming stuck in the degenerate states, which were inherently difficult to select out between generations.

| Size | Metric | DRL + A* | GA | SA | Baseline |
|---|---|---|---|---|---|
| $n = 2$ | Runtime (s) | 3.2 | 7 | 2.9 | 10 |
| | # of Moves | 10 | 29 | 1270 | 17 |
| $n = 3$ | Runtime (s) | 13.6 | 255 | 204 | 15 |
| | # of Moves | 22 | 350 | 43623 | 32 |

# Findings

- SA solved all of the cubes it was presented with, though at a prohibitively high move count.
- This behaviour was anticipated however, as SA was the most stochastic method employed. Many unfavourable moves were selected probabilistically, leading to the bloated move count.

| Size | Metric | DRL + A* | GA | SA | Baseline |
|---|---|---|---|---|---|
| $n = 2$ | Runtime (s) | 3.2 | 7 | 2.9 | 10 |
| | # of Moves | 10 | 29 | 1270 | 17 |
| $n = 3$ | Runtime (s) | 13.6 | 255 | 204 | 15 |
| | # of Moves | 22 | 350 | 43623 | 32 |

# Conclusion

- All three approaches resulted in solved $n = 2$ and $n = 3$ cubes without human knowledge.
- Of the three, Deep Reinforcement Learning combined with an A$^*$ Search proved most consistently optimal.
- That said, this investigation served as proof of concept for the possibility of Genetic Algorithms and Simulated Annealing solving $n = 2$ and $n = 3$ cubes without domain knowledge—approaches that had not before been present in the literature.