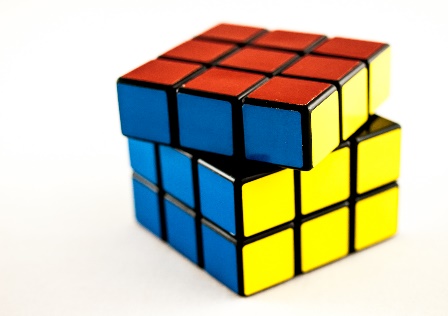
**Rubik’s cube- A\* application:**

The Rubik's Cube can indeed be approached using techniques similar to the A\* algorithm, a popular search algorithm in artificial intelligence. This connection primarily revolves around the use of heuristics to guide the solving process efficiently.



[This Photo](https://www.flickr.com/photos/wwarby/11913436786) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

**Understanding the A\* Algorithm**

The A\* algorithm is designed to find the shortest path from a starting point to a goal by evaluating paths based on a cost function, typically represented as:

**f(n) = g(n) + h(n)**

**Where:**

- f(n) is the total estimated cost of the cheapest solution through node n.

- g(n) is the cost from the start node to node n.

- h(n) is a heuristic that estimates the cost from node n to the goal.

**Heuristic Functions for Rubik's Cube**

In the context of solving a Rubik's Cube, heuristics are crucial for estimating how far a given configuration is from being solved. Common heuristic methods include:

1. Manhattan Distance: This calculates the total distance each piece (corner or edge) must move to reach its goal position, treating each move as a step in a grid.

2. Pattern Databases: These involve precomputing solutions for subsets of pieces and storing them in a hash table. When evaluating a state, one can quickly retrieve how many moves are needed to solve that subset, thus providing an efficient heuristic.

**Example of A\* (a-star) in Rubik's Cube Solving**

To illustrate how A\* can be applied to the Rubik's Cube, consider the following steps:

1. Initial State: Start with a scrambled cube configuration.

2. Generate Successors: From the current configuration, generate all possible configurations that can be reached with one move (e.g., rotating any face).

3. Evaluate Costs:

- For each successor state, calculate g(n) as the number of moves taken to reach that state.

- Calculate h(n) using Manhattan Distance or another heuristic.

- Compute f(n) for each successor.

4. Select Next State: Choose the successor with the lowest f(n) value and repeat until reaching the solved state.

By systematically exploring configurations based on these heuristics, A\* efficiently narrows down potential paths to solve the cube, leveraging both actual moves made and estimated future moves required.

**Conclusion**

Using A\* or similar algorithms to solve a Rubik's Cube involves defining an effective heuristic that guides the search process. Techniques like Manhattan Distance and Pattern Databases enhance efficiency by allowing quick estimates of how close a configuration is to being solved. This structured approach illustrates how computational algorithms can apply to classic puzzles like the Rubik's Cube, making them solvable within practical time frames even from complex starting positions.

**Citations:**

[1] https://www.wired.com/story/how-to-solve-a-rubiks-cube-step-by-step/

[2] https://www.youtube.com/watch?v=iohUekiQRJk

[3] https://stackoverflow.com/questions/60130124/heuristic-function-for-rubiks-cube-in-a-algorithm-artificial-intelligence

[4] https://www.diva-portal.org/smash/get/diva2:816583/FULLTEXT01.pdf

[5] https://www.youtube.com/watch?v=3NYzeX8eE\_4

[6] https://www.reddit.com/r/algorithms/comments/wgfloz/heuristic\_function\_for\_a\_rubiks\_cube/

[7] https://www.speedcube.us/pages/how-to-solve-a-rubiks-cube?shpxid=9d42c340-63fa-401d-8027-9bae5282414f

[8] https://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/step-4-yellow-cross/