Reykjavik University Spring 2023

T – 622 – ARTI Stefán Ólafsson

**Artificial Intelligence**

**Project 2**

**Group 1**

Logo, company name

Description automatically generated

Nina Carlson, [ninac22@ru.is](mailto:ninac22@ru.is)

Emma Shneidman, [emmas22@ru.is](mailto:emmas22@ru.is)

**Introduction**

The Rubik’s cube is a 3-dimensional puzzle invented in 1974 by Hungarian sculptor and professor of architecture, Erno Rubik. Typically, it consists of a 3x3x3 cube with six faces, each of which is divided into nine smaller squares of a single color. Although the Rubik’s cube may seem like a simple toy, it has proven to be a challenging puzzle for people of all ages. It has been estimated that there are over 43 quintillion possible combinations, which makes it difficult to solve by brute force alone. One way to solve the Rubik’s cube is to use search algorithms. These algorithms can be applied to find the shortest path from the starting configuration of the cube to the desired solution. The search methods we have implemented in our Python program, including Breadth-First Search, Iterative Deepening, and A\* Search, are fairly commonly used search algorithms for solving puzzles like the Rubik’s cube.

For the sake of simplicity, our program uses BFS, IDS, and A\* search to solve the 2x2x2 Rubik’s cube, rather than a 3x3x3 cube. The 2x2x2 cube has significantly fewer possible combinations than a 3x3x3 cube. A 2x2x2 cube has only 3,674,160 possible combinations, while a 3x3x3 cube has over 43 quintillion possible combinations. Therefore, our program requires far less computational power and time to solve compared to solving a 3x3x3 cube.

While conducting research on our project topic, we found many different implementations of Rubik’s cube solvers that inspired our own solution to the problem. Some people focused on the learning aspect and created a digital solver that people can fast forward and rewind to see each step in detail and learn how to solve their cube that way. Other people have focused on one specific algorithm to show how to solve the cube using only that way. This encouraged us to look into how different search algorithms can be compared while solving the cube in order to see which one works the best for different criteria.

It is important to understand each algorithm used in our program and how each algorithm can be applied to the Rubik’s cube problem:

1. Breadth First Search: this is a search algorithm that traverses the graph or tree level by level. It starts at the initial state of the Rubik’s cube and explores all possible moves that can be made from that state. Then, it moves to the next level of states, which are one move away from the initial state, and explores all possible moves from those states. This process continues until the solution is found. To apply BFS to the Rubik’s cube problem, we represented each state of the cube as a node in a graph. Each edge in the graph represents a move that can be made to transition from one state to another. BFS can then be used to find the shortest path from the initial state to the goal state.
2. Iterative Deepening Search: this is a variant of DFS that gradually increases the depth of the search until the solution is found. It starts with a depth limit of 1 and performs a DFS search up to that limit. If the solution is not found, the depth limit is increased and the search is performed again until the solution is found. To apply IDS to the Rubik’s cube problem, we started with a depth limit of 1 and performed DFS search up to that limit. If the solution has not been found, we increase the depth limit and perform the search again until the solution is found.
3. A\* Search: this is an algorithm that uses heuristics to guide the search. It combines the cost to reach a node and an estimate of the cost to get from the node to the goal state. A\* search expands the node that has the lowest estimated total cost. To apply A\* search to the Rubik’s cube problem, we used an admissible heuristic that estimates the distance between the current state and the goal state. This heuristic was used to estimate the cost of reaching the goal state from the initial state. A\* search was then used to find the shortest path from the initial state to the goal state, using the heuristic to guide the search.

**Research Question(s)**

How do different search algorithms (BFS, A\*, IDS) compare when solving a 2x2x2 Rubik’s cube?

1. Which algorithm solves the 2x2x2 cube in the least number of moves given an initial scramble pattern?
2. Which algorithm explores the greatest number of nodes?
3. Which algorithm explores the least number of nodes?
4. Which algorithm is most efficient in terms of time and nodes expanded?

The lettered questions allow us to take a look at different factors of the game in more detail. This is important when looking at the comparisons of the algorithms as we want to compare them in more ways than just length of the run time of the solution.

In our solution we implemented three search algorithms to solve a 2x2x2 Rubik’s cube. We kept track of the number of nodes expanded and run time of the solution. By looking at the results of each solution of the game with each algorithm we can see which one runs the fastest or takes the least amount of node expansions. This allowed us to compare the algorithms and see which one was better for each of the variables we kept track of.

**Methods**

*Describe the method(s) you used and how they were implemented.*

As we started our project we decided which algorithms we wanted to compare and what results we wanted to record. We decided on breadth-first search, iterative deepening search, and A\* search, and that we wanted to keep track of the number of nodes expanded and run time of the solution. We also keep track of the depth of the solution for the IDS. Once we knew what algorithms and results, we started on implementing our code. We wrote down all the possible moves as arrays so we knew which moves could be executed. Each array is a list of indices that correspond to one of the stickers on the Rubik’s cube. This way we can keep track of exactly which squares are being moved each time a move is made. We also created a visual representation for the cube so the user can see what the cube looks like scrambled (after the moves have been typed into the command line), and what the cube looks like once solved. We then implemented each search algorithm to solve the Rubik’s cube.

*For each experiment (search algorithm), Why was it conducted? What was the question you sought to answer? What were the settings of the system for each experiment?*

For our project, the experiments that we conducted were solving the Rubik’s cube with each different search algorithm. We tested each algorithm with the same sequence of moves trying many different sequences. We did this because it allowed us to collect and compare more data to answer the question of how the algorithms compare. We know that the data we collected is consistent between each algorithm because we used the same exact sequences for each search.

**How to run code:  
On a Mac:**

Step 1: Save project into a folder (in the example the folder is named AI)

Graphical user interface, application

Description automatically generated

Step 2: cd into the folder and find the project



Step 3: cd into project file until you see run.sh file



Step 4: Run the command sh run.sh bfs “R F L” or any of the commands below:

Start with one of the three search algorithms:

- bfs

- Ids

- A\*

Follow with a sequence of moves (these are the ones we used in our tests):

- “R F L”

**On Windows:**

1. Open Windows PowerShell or Git Bash
2. CD into the location where the project is saved.
3. Find the run.sh file.
4. Run the command “sh run.sh <search method> <scramble string>”

* Example: sh run.sh bfs "F R L"

Graphical user interface, text

Description automatically generated

**Results**

The results of the experiments that we ran are shown in the tables below. We chose several different scramble sequences for the Rubik’s cube. After each algorithm solved the scrambled cube we recorded the run time of the solution in seconds, the number of nodes expanded, and, for IDS, the depth of the solution.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scramble sequence** | **Search algorithm** | **Run-time (seconds)** | **Number of nodes expanded** | **Depth of solution (only for IDS)** |
| **“R F L”** | BFS | 0 | 391 |  |
| IDS | 0 | 492 | 3 |
| A\* | 0 | 64 |  |
| **“U R F”** | BFS | 0.01 | 550 |  |
| IDS | 0 | 731 | 3 |
| A\* | 0 | 61 |  |
| **“L D R F”** | BFS | 0.15 | 4374 |  |
| IDS | 0.03 | 7268 | 4 |
| A\* | 0.01 | 300 |  |
| **“U B B L”** | BFS | 0.07 | 2627 |  |
| IDS | 0.02 | 4744 | 4 |
| A\* | 0.01 | 328 |  |
| **“F R L B U”** | BFS | 0.82 | 10459 |  |
| IDS | 0.14 | 29637 | 5 |
| A\* | 0.19 | 1851 |  |
| **“U U R L B”** | BFS | 1.64 | 13517 |  |
| IDS | 0.32 | 68642 | 5 |
| A\* | 0.22 | 1999 |  |
| **“U D R L F D”** | BFS | 19.14 | 48631 |  |
| IDS | 1.27 | 27682 | 6 |
| A\* | 12.58 | 17017 |  |
| **“R L B U F L”** | BFS | 90.32 | 109280 |  |
| IDS | 2.19 | 478693 | 6 |
| A\* | 21.83 | 22351 |  |

For our experiments we decided to choose two scramble sequence of length 3, two scramble sequences of length 4, two scramble sequences of length 5, and two scramble sequences of length 6. Each scramble sequence shuffles the 2x2x2 cube based on the pattern of each sequence. After scrambling the cube each time, we ran our solver and collected results for each search algorithm (BFS, IDS, A\*).

**Interpretation of Results**

The run-time measures the time taken by an algorithm to find a solution. In all cases, the run-time is relatively low, suggesting that all the algorithms are fairly efficient at solving the cube. The run-time for BFS ranges from 0 to 90.32 seconds, for IDS ranges from 0 to 2.19 seconds, and for A\* ranges from 0 to 21.83 seconds. The runtime of A\* is generally higher than IDS but lower than BFS. The number of nodes expanded measures the number of possible states of the cube that the algorithm evaluates to find a solution. BFS explores the fewest number of nodes in all but one case, while IDS explores the most nodes in all but one case. A\* explores significantly fewer nodes than BFS and IDS in most cases, suggesting that it is the most efficient algorithm in terms of exploration of the search space. The number of nodes expanded by the algorithms ranges from 391 to 478693 nodes. This shows that the choice of algorithm can significantly affect the number of nodes that need to be explored to solve the 2x2x2 cube. The depth of the solution measures the number of moves required to solve the cube. The depth of the solution is only available for IDS, and it varies from 3 to 6. In most cases, IDS finds a solution of lower depth than BFS and A\*. This suggests that IDS is more effective and A\* at finding shorter solutions to the cube.

*Specific interpretations of results for each scramble pattern are as follows:*

**Sequence = “R F L”**

Length = 3

In this case, BFS explores the fewest number of nodes, while IDS takes slightly longer but still explores a relatively low number of nodes. A\* is the most efficient in terms of time and nodes expanded, finding a solution with the least number of moves.

**Sequence = “U R F”**

Length = 3

Once again, BFS explores the fewest number of nodes, while IDS takes longer but still explores a relatively low number of nodes. A\* is the most efficient in terms of time and nodes expanded, finding a solution with the least number of moves.

**Sequence = “L D R F”**

Length = 4

This scramble pattern results in a higher number of nodes being expanded. BFS still explores fewer nodes than IDS, but the difference is not as pronounced as in the previous cases. A\* is again the most efficient in terms of time and nodes expanded.

**Sequence = “U B B L”**

Length = 4

BFS and IDS explore a similar number of nodes, while A\* is the most efficient in terms of time and nodes expanded.

**Sequence = “F R L B U”**

Length = 5

This scramble pattern results in a significant increase in the number of nodes expanded. Once again, BFS explores fewer nodes than IDS, but IDS finds a solution with a lower depth. A\* is the most efficient in terms of time and nodes expanded.

**Sequence = “U U R L B”**

Length = 5

BFS and IDS explore a similar number of nodes, while A\* is again the most efficient in terms of time and nodes expanded.

**Sequence = “U D R L F D”**

Length = 6

This scramble pattern results in a much larger number of nodes being expanded, with BFS exploring the fewest number of nodes. IDS finds a solution with a higher depth than BFS and A\*. A\* is the most efficient in terms of time and nodes expanded.

**Sequence = “R L B U F L”**

Length = 6

This scramble pattern results in an even larger number of nodes being expanded, with BFS exploring the fewest number of nodes. IDS again finds a solution with a higher depth than BFS and A\*. While BFS and IDS explore more nodes, they may not necessarily find the optimal solution in the shortest amount of time. A\* is the most efficient in terms of time and nodes expanded. A\* is generally more efficient in terms of exploration of the search space, but it may not always find the optimal solution. The results suggest that a tradeoff exists between the efficiency of exploration of the search space and the optimality of the solution.

**Conclusions**

Based on the results collected, we can draw the following conclusions regarding the performance of different search algorithms (BFS, A\*, IDS) when solving a 2x2x2 Rubik’s cube:

1. BFS is not a good option for solving the 2x2x2 cube, as it takes a long time and expands a large number of nodes to find a solution for even relatively short initial scramble sequences.
2. IDS is a viable option for solving the 2x2x2 cube, as it generally finds solutions in a reasonable amount of time and with a reasonable number of nodes expanded.
3. A\* search is the most efficient algorithm for solving the 2x2x2 cube, as it typically finds solutions with the lowest runtime and the lowest number of nodes expanded.

BFS explores the greatest number of nodes, as it typically requires exploring a large portion of the search space to find a solution. A\* search explores the least number of nodes, as its informed search approach helps it avoid exploring unnecessary portions of the search space.

From our results we can see that for each sequence, the run-time and number of nodes expanded differed between the three algorithms. For every test we ran A\* search expanded the smallest number of nodes. This is good because we know that A\* should be the most optimal of the three searches we implemented. Therefore, our results for this specific criterion are what we expected them to be. We can also see that for each sequence IDS found the solution at the same number depth as the number of letters in the sequence. This makes sense as well because, as you can see when the code is run, we displayed the solution as a sequence of letters as well. For each scramble sequence the solution sequence is the same length of letters. Therefore, the solution should be found at depth 6 for a scramble sequence of 6 letters which is what our results confirm. Additionally, we can see that BFS has the longest run-time for every sequence we tested. This also makes sense with our tests because we would expect that BFS would be slower than both IDS and A\* in this situation. One more observation we have made in our results is that the solution sequence for each scramble is mostly the same for each search algorithm, however, sometimes A\* search finds a different solution sequence. This further confirms that our results are accurate because the follow the same solution sequence, so we can truly see how much faster IDS is than BFS for example.

Overall, our results tell us that we have correctly implemented each of the search algorithms to solve the Rubik’s cube. This also gives us the information we need to choose which algorithm we want to use based on what criteria is most important for our solution. A\* expands the least number of nodes, A\* and IDS are the fastest options, and if we want to see the depth then IDS is the best option.

In general, the results suggest that the choice of search algorithm can have a significant impact on the efficiency of the search process, and that different algorithms may be more or less effective depending on the characteristics of the search space and the specific problem at hand. Overall, informed search algorithms like A\* can be more efficient than uninformed search algorithms like BFS when a good heuristic is available, and iterative deepening search can be a good option when the depth of the solution is not known beforehand.

In the future we could expand on the comparisons by using more search algorithms to make sure we find the one that would work the best for all criteria. We could also move to the larger Rubik’s cubes such as the 3x3 or 4x4 cubes. With more data we could compare the results from the search algorithms for each size cube and see if they differ. Additionally, we could also look into more ways to display the Rubik’s cube as the solver goes through the solution. We could create a design that shows each square in color so that it is easier to follow along with. We could have graphics that show the moves as they happen and explanations with each move as well. Finally, another idea for future work could eventually be a simulator such as the ones we have looked at in class that allow the user to change the search algorithm being used and in this case also the size of the cube.