8-Puzzle Problem Solver

CS7IS2 Project (2019-20)

Abhinav Gandhi, Shikher Singh, Tanvi Bagla, Vishal Kumar

gandhia@tcd.ie, ssingh2@tcd.ie, tbagla@tcd.ie, Kumarv1@tcd.ie

Abstract. This paper approaches to solve the 8-puzzle problem, a 3x3 board with 8 tiles marked 1 to 8 and a blank square and the goal is to arrange them in order, which is a typical artificial intelligence problem. To treat this problem optimally we take four search algorithms – A-Star, Uniformed Cost, Iterative Deepening Search, and Iterative Deepening A-Star search. This paper derives and tests various heuristics using the above algorithms to compare and provide the results in the form stating which algorithm can give most optimal and accurate solution to goal state. As this is a traditional problem that gives insight to several other AI problems like solving N queen puzzle, NxN puzzle and many more, solving this and empirically comparing space and time performance may give several useful insights and results that can be used in wider AI areas which motivates us to take this as an exploration.

Keywords: A-Star, UCS, BFS, Iterative Deepening, IDA

1 Introduction

8-Puzzle is a game invented by Sam Loyd [1] consists of 9 tiles numbered from 1 to 8 and one blank square arranged randomly. The tiles can slide horizontally or vertically in a 3x3 frame to rearrange itself in the correct order of numbers. Note that the goal state can be any fixed state that needs to be achieved **(Figure 1)**. Before moving deeper let's check the problem statement.

- **Initial state-** Initial arrangement of tiles
- **Goal state-** Final destination of the tiles
- **Actions-** Up, Down, Left or Right w.r.t the blank square
- **Path Cost-** Cost of each step. Therefore, node depth is equal to node cost
- **Uninformed search algorithms examined-** Uniform Cost Search (UCS), Iterative Deepening Depth First Search (IDDFS)
- **Informed Search algorithms examined-** A* and IDA* search

This game is a subject of continuous research where numerous algorithms can be devised to make the transition from one state to another until it reaches the goal state. Complexity is when the goal state has to be achieved in the real time providing the solution to be optimal. In this paper we study and analyze several parameters of the puzzle mentioned as follows- path_to_goal, cost_of_path,

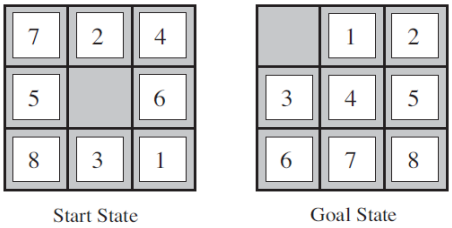

Fig. 1. Example of a Eight-puzzle problem state chart [2]

.

nodes_expanded, fringe_size, max_fringe_size, search_depth, max_search_depth, running_time and max_ram_usage. Depending on this search algorithms strong estimation can be made about which search algorithm performs better.

This paper is organized as follows. Section II looks at related work in the areas of research. Section III outlines the problem and our approach towards the solution. Section IV details the experiment and results whereas section V presents the summary and conclusion based on results observed in section IV.

2 Related Work

There is one research [4] where comparative analysis was done on search algorithms like BFS, DFS, UCS, A*, and Best First Search that specifies that UCS takes less space and is usually applied when there is no need of intelligence like that in solving maze problems and path finding. The paper highlights the time complexity, space complexity, optimality and completeness. Author concludes with the efficiency of heuristic search over blind search. In their experiment where UCS took 11 nodes to expand and took more time; A* took only 6 nodes to reach the goal state. In the other work presented by Vipin Kumar, K. Ramesh and V. Nageshwara Rao Kumar [5], best first search was applied on state space graphs displaying the summary of results. Various modulations of A* Best First algorithm was implemented, stating the better performing modulations. Korf, R. E. discussed a study on Depth First search as asymptotically optimal exponential tree searches [6]. He suggests that the Depth First iterative-deepening algorithm is capable of finding an optimal solution for randomly generated 15 puzzles. Alexander Reinefeld mentions in his paper [3] how IDA* is beneficial in node ordering for the 8-Puzzle problem. Authors concluded that the longest path heuristic node ordering system was most effective and fixed operator sequence worked worst. Kuruvilla Mathew and Mujahid Tabassum [7] used Breadth First Search, Depth First Search, A* Search, Best First Search and Hill Climbing Search to find out the optimal solution for N Puzzle game and highlighted that where Greedy BFS is memory efficient for shorter solutions, A* is suitable for longer solutions.

Noting from aforementioned researches, we picked a few algorithms that we wanted to compute parallel and generate comparison between them. Iterative

Deepening Depth First Search was highlighted to be space and time efficient as compared to breadth first search that consumes too much space and depth first search that takes a lot more time by Korf R.E in [5]. A* and Iterative Deepening A* seemed to work best for longer solutions and randomly generated instances as mentioned in paper [3] and [6]. Also as it is mentioned in [4] that UCS takes less space while computation it is worthwhile to compare UCS with IDDFS for different problem.

We took A* search, Uniform Cost Search (UCS) and Iterative Deepening Depth First Search (IDDFS) and Iterative Deepening A* search as implementation for 8-Puzzle problem to test if we could find out the best out of best algorithms stated in the above research. Where A* comes is the informed search, UCS and IDDFS come under uninformed searchstu. Keeping the hypothesis that A* will outperform the other two algorithms as it is the heuristic (Difference between Goal State and Current State) approach we will first apply A*. In the 8 puzzle it will select nodes according to how close it is to the target state and how far it is from the root node. On the other hand the concept of UCS will be applied that works on a priority queue assigning highest priority to minimum cost.

3 Problem Definition and Algorithm

3.1 Problem statement

Our problem statement is to find out the solution for the N-Puzzle problem by applying Artificial Intelligence algorithms. Our prospect is not only to solve the problem but rather to display new paradigms of solution using different search algorithms that can be applied to solve efficiently instead of using traditional techniques.

3.2 Algorithms

To address the N Puzzle problem, 4 search algorithms which are Uniform Cost Search (UCS), Iterative Deepening Depth First Search (IDDFS), A* search, and Iterative Deepening A* search are being used. The search algorithms aim to discover the shortest path to the goal state if it exists. The cost of the path is measured differently in different algorithms like A* and IDA* uses some heuristic measures. For the comparative analysis of the above-mentioned algorithms, a common goal state (**Figure 1**) is being used for all the testcases and demonstrations.

While the goal state is the same for all the testcases, the start state is different for all of them.

3.2.1 Uniform Cost Search (UCS)

Uniform Cost Search is an algorithm best known for its searching techniques as it does not involve the usage of heuristics. It can solve any general graph for its optimal cost. Uniform Cost Search as it sounds searches in branches that are the same in cost. The algorithm uses the priority queue. An example is shown in **Figure 2**.

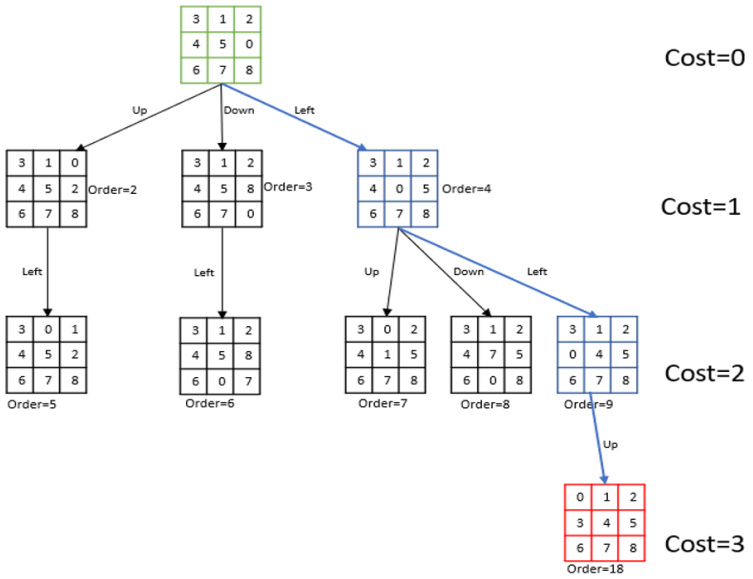

Fig. 2. Example of states in UCS

Pseudo code for UCS:

1. Insert the start state into the queue.
   Until the queue is not empty repeat the following steps:
2. Dequeue the maximum priority element (minimum cost in N puzzle) from the queue. For N puzzle, if the elements have the same cost, then the order of actions, i.e., "Up", "Down", "Left", "Right" is taken into consideration.
3. If the state is goal state then print the path and Terminate. Else
4. Find the successor states for the dequeued state.
5. Add the successor states which we have not visited yet to the priority queue. If a state is already present, update the cost to the lowest cost possible.

3.2.2 Iterative Deepening Depth first Search (IDDFS)

A search algorithm which suffers neither the drawbacks of breadth-first nor depth-first search on trees is depth-first iterative-deepening (DFID). This algorithm does not endure the disadvantages of breadth-first or depth-first search. The formula operates as follows: run DFS to depth one. Then, discard the nodes created in the first iteration, reiterate and run DFS for second level. Continue further for next levels continuing this process until a goal state is reached. DFID extends all nodes to a specified depth until extending some nodes to a greater length, it is expected to find the shortest-length option. Also, since at any given time it is performing a depth-first search, and never searches deeper than depth d, the space it uses is O(d).

Pseudo code for IDDFS:
Keep repeating until the max depth (predefined depth) is reached:

1. Insert the start state into the stack.
   Until the stack is not empty repeat the following steps:
2. Pop the state from the stack.
3. If the state is goal state then print the path and Terminate. Else
4. Find the successor states for the popped state if the depth of the state is less than the current depth.
5. Add the successor states to the stack.

If the solution is not found till the max depth, search is stopped. As it can be

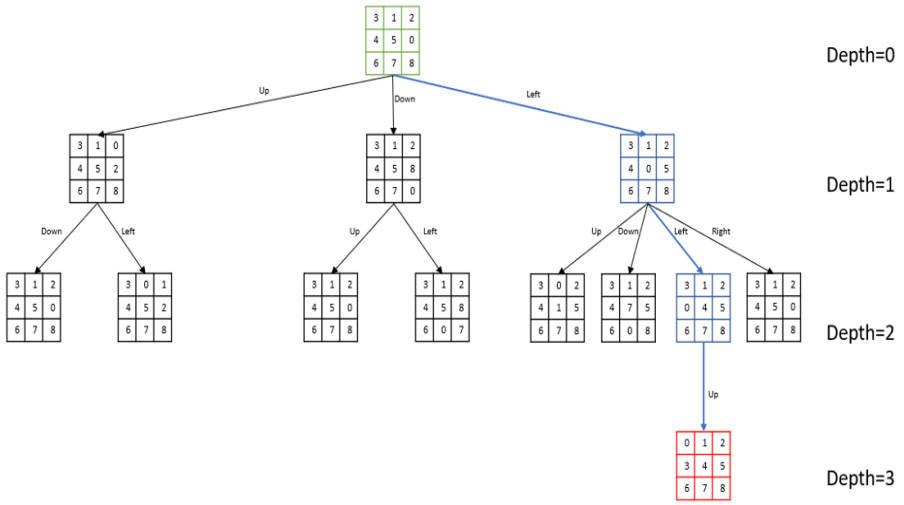

Fig. 3. Example of states in IDDFS

seen from the example in **Figure 3** the same nodes can be visited again in this algorithm, so the number of nodes expanded can be higher.

3.2.3 A* search

The most commonly used algorithm called is A Star search. It evaluates node by combining g(n), the cost to reach the node, and h(n), the cost to get from node to the goal. F(n) = g(n) + h(n)

Since g(n) giving the path cost from starting node to node n, and h(n) is the estimated cost of cheapest path from n to the goal. Hence, we are after to find the cheapest solution. That's a reason A star is considered the better because it is complete and optimal. For the N puzzle problem, we are using Manhattan Distance as the Heuristic measure and cost of the action which consists of 4

moves: “Up” with cost 0.1, “Down” with cost 0.2, “Left” with cost 0.3, “Right” with cost 0.4 is also taken into consideration.An example is shown in **Figure 4**.

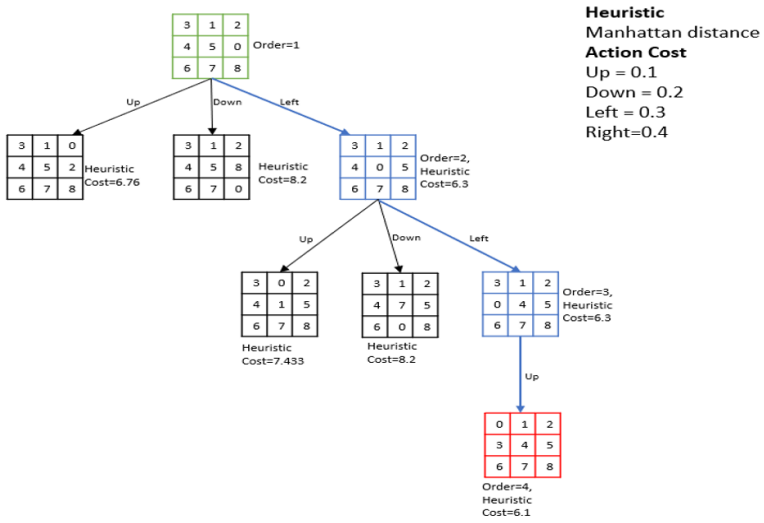

Fig. 4. Example of states in A*

Pseudo code for A*:

1. Insert the start state into the queue.
   Until the queue is not empty repeat the following steps:
2. Dequeue the maximum priority element (minimum heuristic value and cost in N puzzle) from the queue. For N puzzle, if the elements have the same cost, then the order of actions, i.e., “Up”, “Down”, “Left”, “Right” is taken into consideration.
3. If the state is goal state then print the path and Terminate. Else
4. Find the successor states for the dequeued state.
5. Add the successor states which we have not visited yet to the priority queue. If a state is already present, update the cost to the lowest cost possible.

3.2.4 IDA* search

IDA* is a variation of IDDFS, where iterative deepening depth first is combined with A* search. The theory behind IDA* is that the iterations rather than keeping track of the increasing depth, keeps track of the increase in total cost, where overall cost is calculated by adding cost incurred so far in reaching the node and the estimated cost that could be involved in moving from the node to

a goal state. It makes use of Manhattan distance as heuristic estimate. At each current iteration, the threshold for the following iteration is the minimum cost of all the costs which exceeds the existing threshold. An example is shown in **Figure 5**.

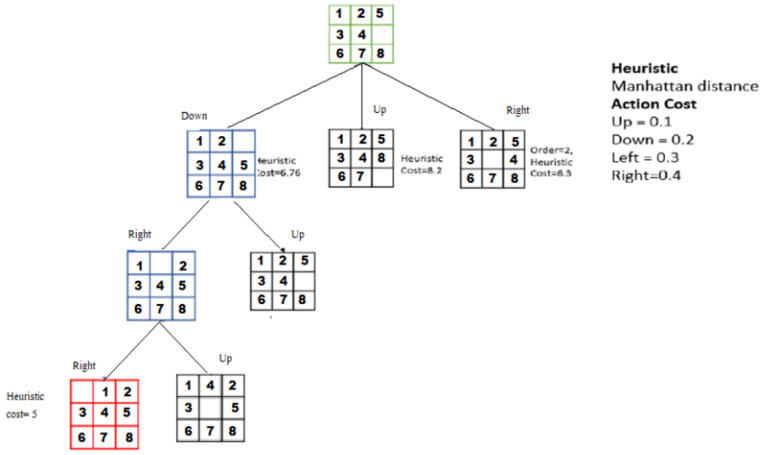

Fig. 5. Example of states in IDA*

Pseudo code for IDA*:

1. In the current node take each child
2. Check if it is the goal state, return
3. Else calculate the total cost (present cost incurred plus estimated cost from node to goal state) if total cost exceeds the current threshold, return this exceeding threshold
4. Set the current node to this node and go back to 1.
5. After going through all children of the current node, traverse to parent's next child (the next sibling)
6. Traverse all children of the start node; update the threshold to the smallest of the exceeding thresholds.
7. If all leaf nodes traversed, the goal node doesn't exist.

3.3 Experimental Setup

Initially we set-up 8 puzzle problem for our base algorithm in BFS and reported the results (**Table 2**) in terms of parameters in **Table 1** to compare. Then, we have performed 4 use cases and captured the results in form of below parameter for algorithms: A-Star, Uniform Cost Search, Iterative Deepening Depth First Search, and Iterative Deepening A* search.

path_to_goal	The sequence of moves taken to reach the goal
cost_of_path	The number of moves taken to reach the goal
nodes_expanded	The number of nodes that have been expanded
search_depth	The Depth within the search tree when the goal node is found
max_search_depth	The Maximum depth of the search tree in the lifetime of the algorithm.
running_time	The Total running time of the search instance, reported in seconds
max_ram_usage	The Maximum RAM usage in the lifetime of the process as measured by the ru_maxrss attribute in the resource module, reported in megabytes.

Table 1. Experimental setup

BFS (Base algorithm)				
Testcases	1,2,5,3,4,0,6,7,8	3,1,2,0,4,5,6,7,8	3,1,2,4,5,0,6,7,8	1,2,5,0,3,4,6,7,8
path_to_goal	['Up', 'Left', 'Left']	['Up']	['Left', 'Left', 'Up']	['Right', 'Right', 'Up', 'Left', 'Left']
cost_of_path	3	1	3	5
nodes_expanded	10	1	17	58
search_depth	3	1	3	5
max_search_depth	4	1	4	6
running_time	0.00043607	0.00007987	0.00067616	0.00219297
max_ram_usage	10	10	10	10

Table 2. Testcase result for Base algorithm

Testcase 1 - 1,2,5,3,4,0,6,7,8				
Algorithms	**UCS**	**IDS**	**A***	**IDA***
path_to_goal	['Up', 'Left', 'Left']	['Up', 'Left', 'Left']	['Up','Left', 'Left']	['Up', 'Left', 'Left']
cost_of_path	3	3	3	3
nodes_expanded	16	5	3	3
search_depth	3	3	3	3
max_search_depth	4	3	3	3
running_time	0.00037551	0.00016546	0.00022197	0.00019634
max_ram_usage	9.91015625	9.84375	10	9.24218

Table 3. Experimental result for testcase 1

4 Experimental Results

For the comparative analysis of the different search algorithms, 4 testcases have been considered and their performance in terms of time, memory, nodes explored, and maximum search depth have been used as measures. The results for the

Testcase 2 - 3,1,2,0,4,5,6,7,8				
Algorithms	**UCS**	**IDS**	**A***	**IDA***
path_to_goal	['Up']	['Up']	['Up']	['Up']
cost_of_path	1	1	1	1
nodes_expanded	1	1	1	1
search_depth	1	1	1	1
max_search_depth	1	1	1	1
running_time	0.00004768	0.0000391	0.000118	0.0001046
max_ram_usage	9.87890625	9.890625	10	8.72656250

Table 4. Experimental result for testcase 2

Testcase 3 - 3,1,2,4,5,0,6,7,8				
Algorithms	**UCS**	**IDS**	**A***	**IDA***
path_to_goal	['Left', 'Left', 'Up']	['Left', 'Left', 'Up']	['Left', 'Left', 'Up']	['Left', 'Left', 'Up']
cost_of_path	3	3	3	3
nodes_expanded	30	11	3	9
search_depth	3	3	3	3
max_search_depth	4	3	3	3
running_time	0.00074506	0.00023556	0.00026393	0.00046801
max_ram_usage	9.86328125	9.84375	10	8.734375

Table 5. Experimental result for testcase 3

Testcase 4 - 1,2,5,0,3,4,6,7,8				
Algorithms	**UCS**	**IDS**	**A***	**IDA***
path_to_goal	['Right', 'Right', 'Up', 'Left', 'Left']	['Right', 'Right', 'Up', 'Left', 'Left']	['Right', 'Right', 'Up', 'Left', 'Left']	['Right', 'Right', 'Up', 'Left', 'Left']
cost_of_path	5	5	5	5
nodes_expanded	272	92	5	31
search_depth	5	5	5	5
max_search_depth	6	5	5	5
running_time	0.00594044	0.00223231	0.00036287	0.00134102
max_ram_usage	11.6015625	9.89453125	10	8.73828

Table 6. Experimental result for testcase 4

different algorithms for the experiments have been detailed in **Table 3, 4, 5 and 6**.

As it is clearly evident that Iterative Deepening A-Star performs best in solving the 8-puzzle problem. Let's take testcase 1 where the nodes expanded is same in both the heuristic algorithms (A* and IDA*) but still the running_time and max_ram_usage is comparatively lesser in IDA*. In other scenario (testcase 3) the nodes expanded in case of IDA* is 9 which is greater than that of A* but still IDA* performs better in terms of max_ram_usage. In all the cases UCS performs on the lowest side, whereas IDDFS still performs better as it includes iterative deepening feature instead of expanding a complete search tree. A* has a little extra RAM usage in some cases but the difference is negligible to even consider. Iterative Deepening Depth First Search is an algorithm with a guaranteed optimal solution if the heuristics are not given as all the possible paths are explored and the shortest path is taken as a solution if it exists.

5 Conclusion

The research and the implementation of the four algorithms A*, IDA*, UCS and IDS have provided us with an insight of how different algorithms can be compared and what factors make one algorithm better than the others. Through the experimental results, the assumption that having knowledge about the heuristic (A* and IDA*) measures can help an algorithm perform better in most cases has been proven true. Also if we take a close look into the results of both the heuristic algorithms it can be noticed that IDA* performs more optimally than A* in terms of time taken and space used. Unlike other algorithms IDA doesn't keep track of visited node hence consuming less memory. It is mostly similar to IDDFS search but it performs better as it uses the cost as a heuristic measure. In future, there is a scope of improvement in these algorithms using better heuristic measures and a better node expansion strategy which could reduce the number of nodes expanded and could result in better execution time. Further research can be worked upon to calculate the rate of convergence and compare the algorithms of puzzles with high cardinality.

References

1. Pickard S. The puzzle king: Sam Loyd's chess problems and selected mathematical puzzles. Pickard & Son Pub (1996)
2. Stuart J. Russell and Peter Norvig: Artificial Intelligence A Modern Approach Third Edition
3. Reinefeld, A.: Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA*, Paderborn Center for Parallel Computing, Germany (2006)
4. Maharshi J. Pathak, Ronit L. Patel, Sonal P. Rami: Comparative Analysis of Search Algorithms. International Journal of Computer Applications (2018)
5. Kumar. V. Ramesh, K. and Rao, V. N.: Parallel Best-First Search of State-Space Graphs: A Summary of Results. AAAI. Vol. 88. (1988)
6. Richard E. Korf: Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, Department of Computer Science, Columbia University
7. Kuruvilla Mathew and Mujahid Tabassum: Experimental Comparison of Uninformed and Heuristic AI Algorithms for N Puzzle and 8 Queen Puzzle Solution (2014)