**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 Astar, and Iterative Deepening. 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.

**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. 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 1. 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\* 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, 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.



Figure I. Example of a Eight-puzzle problem state chart

To test this problem the simplest approach is to test each move of the space. For example, in the above figure ….

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 where [3] N-Puzzle is solved using the distributed approach in which a problem is decomposed into sub-goals that can be treated independently, these sub goals are further divided into agents that assure those sub-goals. This method gives potential solutions and can solve large N-Puzzles. In the other work presented by Vipin Kumar, K. Ramesh and V. Nageshwara Rao Kumar [4], 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 [5]. They suggest that the Depth First iterative-deepening algorithm is capable of finding an optimal solution for randomly generated 15 puzzles. Korf, R. E. presented a Linear Best First Algorithm, exploring nodes in the best first order, and expanding fewer nodes. This works on the sliding puzzle with reduced computation time, but with a trade-off on solution cost.

Alexander Reinefeld mentions in his paper [2] 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 [6] 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] and A\* and Iterative Deepening A\* seemed to work best for longer solutions and randomly generated instances as mentioned in paper [2] and [6]. We took Iterative Deepening A\* search, Uniform Cost Search (UCS) and Iterative Deepening Depth First Search (IDDFS) as implementation for 8-Puzzle problem to test if we could find out the best out of best algorithms stated in the above research.

**3 Problem Definitions 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.

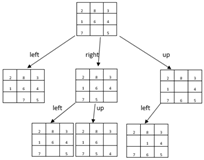


Fig.2 A typical approach to 8 puzzle game formed by sliding any consecutive number on blank square horizontally/vertically.

**3.2 Algorithms**

To address the N Puzzle problem, 3 very familiar search algorithms which are Uniform Cost Search (UCS), Iterative Deepening Depth First Search (IDDFS) and 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\* uses some heuristic measure. For the comparative analysis of the above-mentioned algorithms, a common goal state is being used for all the test cases and demonstrations. Goal state is:

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

While the goal state is the same for all the test cases, 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.

Pseudo code for UCS:

1. Insert the start state into the queue.

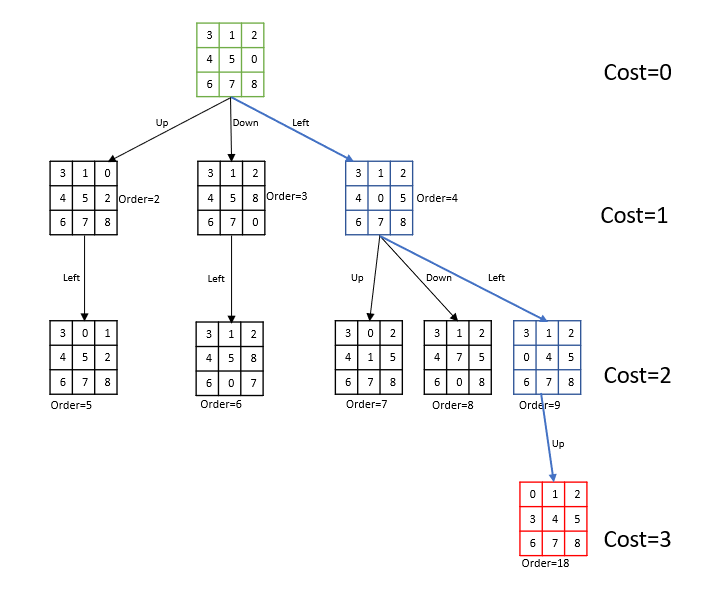
Until the queue is not empty repeat the following steps:

1. 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.
2. If the state is goal state then print the path and Terminate.

Else

1. Find the successor states for the dequeued state.
2. 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.

An example is shown below:



**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). The algorithm works as follows: First, perform a depth-first search to depth one. Then, discarding the nodes generated in the first search, start over and do a depth-first search to level two. Next, start over again and do a depth-first search to depth three, etc., continuing this process until a goal state is reached. Since DFID expands all nodes at a given depth before expanding any nodes at a greater depth, it is guaranteed to find a shortest-length solution. 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).

**3.2.3 A\* search**

The most commonly user algorithm called is A Start 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 start 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.

Pseudo code for A\*:

1. Insert the start state into the queue.

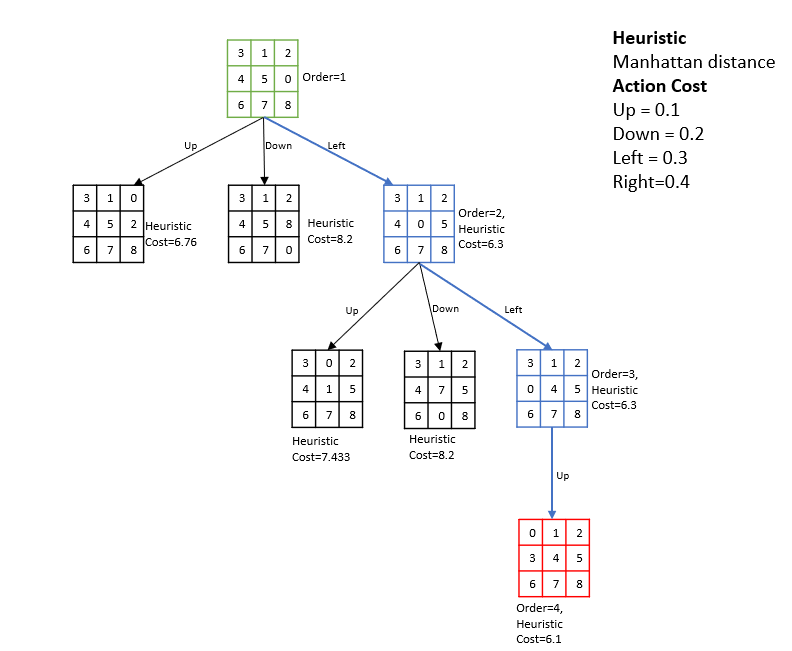
Until the queue is not empty repeat the following steps:

1. 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.
2. If the state is goal state then print the path and Terminate.

Else

1. Find the successor states for the dequeued state.
2. 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.

An example is shown below:



**3.3 Experimental Setup**

We have performed 4 use cases and captured the results in form of below parameter for algorithms A Star, Uniform Cost Search, and Iterative Deepening Depth First 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 algal 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\_ru\_maxrss attribute in the resource module, reported in megabytes. |

**4 Experimental Results + (Abhinav)**

For the comparative analysis of the different search algorithms, 4 test cases 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 different algorithms for the experiments have been detailed here:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1,2,5,3,4,0,6,7,8** | **A STAR** | **UCS** | **IDDFS** |  | **3,1,2,0,4,5,6,7,8** | **A STAR** | **UCS** | **IDDFS** |
| **path\_to\_goal** | ['Up','Left', 'Left'] | ['Up', 'Left', 'Left'] | ['Up', 'Left', 'Left'] |  | **path\_to\_goal** | ['Up'] | ['Up'] | ['Up'] |
| **cost\_of\_path** | 3 | 3 | 3 |  | **cost\_of\_path** | 1 | 1 | 1 |
| **nodes\_expanded** | 3 | 16 | 5 |  | **nodes\_expanded** | 1 | 1 | 1 |
| **search\_depth** | 3 | 3 | 3 |  | **search\_depth** | 1 | 1 | 1 |
| **max\_search\_depth** | 3 | 4 | 3 |  | **max\_search\_depth** | 1 | 1 | 1 |
| **running\_time** | 0.00022197 | 0.00037551 | 0.00016546 |  | **running\_time** | 0.00011802 | 0.00004768 | 0.00003910 |
| **max\_ram\_usage** | 10.00000000 | 9.91015625 | 9.84375000 |  | **max\_ram\_usage** | 10.00000000 | 9.87890625 | 9.89062500 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **3,1,2,4,5,0,6,7,8** | **A STAR** | **UCS** | **IDDFS** |  | **1,2,5,0,3,4,6,7,8** | **A STAR** | **UCS** | **IDDFS** |
| **path\_to\_goal** | ['Left', 'Left', 'Up'] | ['Left', 'Left', 'Up'] | ['Left', 'Left', 'Up'] |  | **path\_to\_goal** | ['Right', 'Right', 'Up', 'Left', 'Left'] | ['Right', 'Right', 'Up', 'Left', 'Left'] | ['Right', 'Right', 'Up', 'Left', 'Left'] |
| **cost\_of\_path** | 3 | 3 | 3 |  | **cost\_of\_path** | 5 | 5 | 5 |
| **nodes\_expanded** | 3 | 30 | 11 |  | **nodes\_expanded** | 5 | 272 | 92 |
| **search\_depth** | 3 | 3 | 3 |  | **search\_depth** | 5 | 5 | 5 |
| **max\_search\_depth** | 3 | 4 | 3 |  | **max\_search\_depth** | 5 | 6 | 5 |
| **running\_time** | 0.00026393 | 0.00074506 | 0.00023556 |  | **running\_time** | 0.00036287 | 0.00594044 | 0.00223231 |
| **max\_ram\_usage** | 10.00000000 | 9.86328125 | 9.84375000 |  | **max\_ram\_usage** | 10.00000000 | 11.60156250 | 9.89453125 |

**5 Conclusions**

**References**

[1] Pickard S. The puzzle king: Sam Loyd’s chess problems and selected mathematical puzzles. Pickard & Son Pub; 1996.

[2] Reinefeld, A. 2006, Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA\*, Paderborn Center for Parallel Computing, Germany.

[3] Drogoul, A., and Dubreuil, C. "A distributed approach to n-puzzle solving." Proceedings of the Distributed Artificial Intelligence Workshop. 1993

[4] Kumar. V. Ramesh, K. and Rao, V. N. "Parallel Best-First Search of State-Space Graphs: A Summary of Results." AAAI. Vol. 88. 1988.

[5] Richard E. Korf, “Depth-First Iterative-Deepening: i z An Optimal Admissible Tree Search”, Department of Computer Science, Columbia University

[6] Kuruvilla Mathew and Mujahid Tabassum “Experimental Comparison of Uninformed and Heuristic AI Algorithms for N Puzzle and 8 Queen Puzzle Solution”; 2014.

[7] Stuart J. Russell and Peter Norvig, Artificial Intelligence A Modern Approach Third Edition.