N-Puzzle Problem

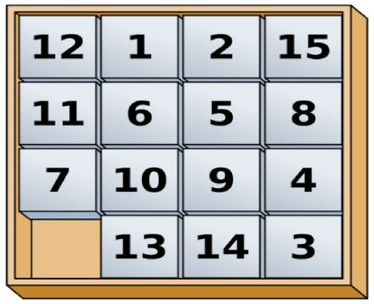
Shubhashis Roy Dipta [1] Shalima Binta Manir [1] Nilanjana Das [1]

[1]Computer Science, University of Maryland, Baltimore County

Abstract

The N-Puzzle problem is a classical sliding puzzle problem. To solve this problem, the number must be rearranged in order. The goal of this work is to explore and analyze different search strategies and compare the runtime, cost, completeness, and optimality. Three types of search strategies are implemented. These are Uninformed search, Informed Search, and Local Search. Under the Uninformed search, we have implemented Breadth-First Search(BFS), Depth-First Search (DFS), Iterative Deepening DFS, and Dijkstra's algorithm. Similarly, we have implemented Best-First Search, A* Algorithm, and Iterative Deepening A* as informed searches. Also, Hill Climbing Search is implemented as Local Search.

Figure 1. Example of a 15-Puzzle Problem



Backgrounds

The puzzle was invented by Noyes Palmer Chapman. There is an initial state and goal state of this puzzle. Different types of search strategies are used. These strategies are evaluated based on completeness, Admissibility, Time complexity, and Space complexity. To understand the problem and the solving techniques some background pieces of knowledge are described below:

Completeness: An algorithm is complete if it can find the solution when a solution exists.

Admissibility: If a solution is found, it is guaranteed to be optimal.

Time Complexity: It calculates the worst or average case. Usually measured by the number of nodes expanded.

Space Complexity: Usually measured by the maximum size of graph/tree during the search.

Methods:

A number of search strategies are used to implement the N-Puzzle problem. These are described below:

Breadth-First Search (BFS):

BFS is a graph traversing algorithm that starts with a selected node/starting node and visits all the nodes layer-wise until it reaches the goal node. It may take a long time to find a solution, and the solution will be optimal if it exists.

Depth-First Search (DFS):

DFS starts from a source node and tries to find a solution if it exists. In DFS, the graph traverses as far as possible along each branch before backtracking. One of the big disadvantages of DFS is it may not terminate, and the algorithm can fall into an infinite loop. This algorithm is not complete and optimal.

Iterative Deepening Depth-First Search (IDDFS):

A graph search approach called IDDFS involves continually running a depth-limited variant of depth-first search with increasing depth limitations until the desired result is discovered. It takes less memory than BFS, and it's optimal.

Dijkstra's algorithm:

This is one kind of greedy algorithm. It finds the shortest path from the source node to all other nodes of the graph. If the weight is non-negative, this graph traversal always gives the optimal solution.

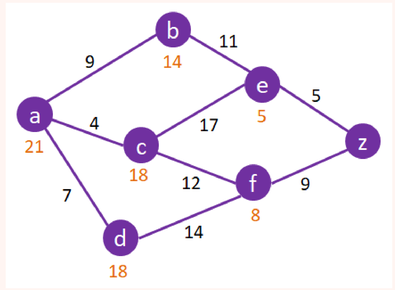
Greedy Best First Search:

It's a greedy algorithm that takes locally optimal choices to find a global optimum. An evaluation function $f(n) = h(n)$ is used. This algorithm selects node to expand based on the closeness to the goal node. More clearly, it chooses the node to expand that has the smallest f value. This algorithm is not complete.

A* Search Algorithm:

A* is the shortest path-searching algorithm. It selects the path that minimizes the evaluation function $f(n) = g(n)+h(n)$. Here n is the next node of the path and $g(n)$ is the path cost from starting node to n and $h(n)$ is a heuristic value that estimates the cheapest path from state n to the goal state. This algorithm is admissible.

Figure 2. A* Algorithm



Iterative Deepening A* (IDA*):

It is a variation of iterative deepening depth-first search that adopts the A* search algorithm's idea of using a heuristic function to assess the remaining cost to reach the goal.

Hill climbing Search:

It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution. Search evaluation function measures how far the current state is from a goal. This algorithm is not complete since the search may terminate at a local minima, plateau, or ridge.

Four types of heuristics have been used. These are Manhattan Distance, Euclidean Distance, Hamming Distance, and Linear Conflict.

Algorithm	Heuristic	Complete	Optimal	Median	Mean	Worst	Best	Standard Deviation
8 Puzzle Cost								
BFS		Yes	Yes	22	22.18	29	9	3.28
DFS		Yes	No	None				
Dijkstra		Yes	Yes	22	22.18	29	9	3.28
GBFS	Manhattan	Yes	No	22	22.31	30	9	3.35
	Euclidean	Yes		22	22.2	29	9	3.28
	Hamming	Yes		22	22.29	29	9	3.31
	Linear Conflict	Yes		22	22.29	29	9	3.31
A*	Manhattan	Yes	Yes	22	22.18	29	9	3.28
	Euclidean	Yes		22	22.18	29	9	3.28
	Hamming	Yes		22	22.18	29	9	3.28
	Linear Conflict	Yes		22	22.18	29	9	3.28
IDA*	Manhattan	Yes	Yes	22	22.18	29	9	3.28
	Hamming	Yes		22	22.18	29	9	3.28
	Linear Conflict	Yes		22	22.18	29	9	3.28
IDDFS		Yes	Yes	22	22.18	29	9	3.28
Hill Climbing	Manhattan	Yes	No	22	22.36	32	9	6.57
	Euclidean	Yes		23	23.39	40	9	5.93
	Hamming	Yes		25	24.3	38	9	6.02
	Linear Conflict	Yes		25	24.3	38	9	6.02
15 Puzzle Cost								
IDA*	Hamming	Yes	Yes	42	42.12	49	34	3.96
	Linear Conflict	Yes		42	42.12	49	34	3.96
Hill Climbing	Manhattan	Yes	No	None				
	Euclidean	Yes		None				
	Hamming	Yes		None				
	Linear Conflict	Yes		None				

Table 1. Cost of 8-Puzzle and 15-Puzzle solution

Algorithm	Heuristic	Complete	Optimal	Median	Mean	Worst	Best	Standard Deviation
8 Puzzle Time								
BFS		Yes	Yes	6.59	6.64	15.35	0.03	3.84
DFS		Yes	No	None				
Dijkstra		Yes	Yes	6.27	6.39	14.39	0.69	3.29
GBFS	Manhattan	Yes	No	0.06	0.09	0.53	0.0	0.09
	Euclidean	Yes		0.11	0.22	1.58	0.0	0.26
	Hamming	Yes		0.04	0.06	0.44	0.0	0.07
	Linear Conflict	Yes		0.04	0.06	0.45	0.0	0.07
A*	Manhattan	Yes	Yes	0.05	0.08	0.59	0.0	0.09
	Euclidean	Yes		0.11	0.22	1.92	0.0	0.27
	Hamming	Yes		0.04	0.06	0.31	0.0	0.06
	Linear Conflict	Yes		0.04	0.06	0.37	0.0	0.06
IDA*	Manhattan	Yes	Yes	0.05	0.1	0.6	0.0	0.11
	Hamming	Yes		0.04	0.07	0.43	0.0	0.08
	Linear Conflict	Yes		0.04	0.07	0.54	0.0	0.08
IDDFS		Yes	Yes	24.47	82.92	1242.99	0.02	144.29
Hill Climbing	Manhattan	Yes	No	0.02	0.02	0.03	0.01	0.01
	Euclidean	Yes		0.03	0.03	0.04	0.01	0.01
	Hamming	Yes		0.03	0.03	0.06	0.01	0.01
	Linear Conflict	Yes		0.03	0.03	0.06	0.01	0.01
15 Puzzle Time								
IDA*	Hamming	Yes	Yes	45.95	59.5	195.32	5.52	55.45
	Linear Conflict	Yes		46.5	60.32	196.08	5.47	56.15
Hill Climbing	Manhattan	Yes	No	None				
	Euclidean	Yes		None				
	Hamming	Yes		None				
	Linear Conflict	Yes		None				

Table 2. Time Complexity of 8-Puzzle and 15-Puzzle solution

Results and Analysis:

We have solved the 8-Puzzle problem and the 15-Puzzle problem. For the dataset, 1000 mixed board configuration of the 8-Puzzle problem has been used. Here, path cost is also included. To solve the 15-Puzzle problem, 100 mixed board configuration has been used. We have included the cost and timetable for all the algorithms implemented. We have included the parameters median, mean, best, worst and standard deviation. The tables provide a detailed analysis of the implemented algorithms. Apart from cost and time, we have also found out the parameters for the number of nodes expanded for both 8 and 15 puzzle problems.

Limitations

In 15-Puzzle, with more than 50 steps, is too long to solve even with pruning and heuristics. It will have $6.3 * 10^{58}$ states w/o any pruning. We were not able to get the cost of 15 puzzle problem for hill climbing, even after considering 4 heuristics in addition to DFS for 8 puzzle problem.

Conclusion

We have implemented 8 and 2 algorithms for 8 and 15 puzzle problems, respectively. According to the cost tables, out of all the algorithms implemented BFS, Dijkstra, A*, IDA* and IDDFS have minimal cost and low standard deviation in comparison to the other algorithms that have been implemented for the 8 puzzle problem. And because these algorithms are also optimal, we conclude that they are a promising approach for the 8 puzzle problem.

References

[1] A* search algorithm. https://en.wikipedia.org/wiki/A*_search_algorithm.

[2] Best-first search. https://en.wikipedia.org/wiki/Best-first_search.

[3] Breadth-first search. https://en.wikipedia.org/wiki/Breadth-first_search.

[4] Depth-first search. https://en.wikipedia.org/wiki/Depth-first_search.

[5] Dijkstra's algorithm. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

[6] Hill climbing. https://en.wikipedia.org/wiki/Hill_climbing.

[7] Iterative deepening a*. https://en.wikipedia.org/wiki/Iterative_deepening_A*.

[8] Iterative deepening depth-first search. https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search.