User Manual, Results, analysis for various search methods

Language used for Coding : Python

Python version used : 3.6.0b1, So compatibility is required in order to run python .

I used Python IDLE platform to run all the use-cases. Code contains enough comments for understanding purpose.

Instructions running for the code:

1. BFS(Breadth first Search), DFS, IDS : 3 initial states(easy,medium,hard) are defined in the main function. One needs to be uncommented in order to run the use-cases. The same holds true for DFS and IDS as well.
2. Greedy Search: 3 initial states are defined in the main function. One state needs to be uncommented in order to run the use-cases. 2 heuristic functions definitions are defined. Needs to select the heuristic1 or heuristic2 while calling the greedy function.
3. A* search : Heuristic functions (heuristic1 and heuristic2) are present. Also g(n) is based on the depth of the node.
4. IDA* approach: two heuristic functions are present. Iteratively min fvalue of the child nodes is picked in each iteration.

Breadth First search:

Use-Case Type	Number of Nodes Visited	Maximum length of Nodelist	Number of nodes generated	Depth at which goal node is found	Time taken for the run(in secs)
Easy	42	35	77	5	0.001 secs
Medium	343	217	560	9	0.008 secs
Hard	181364	24982	181440	30	3.55964 secs

Moves taken for Easy Case : up right up left down

Moves taken for Medium Case: up right right down left left up right down

Moves taken for Worst Case: up left down down right right up up left left down down right right up up left left down down right right up up left left down down right up

Strength: It is complete if the solution exists. We see that it is able to find the solution even for hard use-case.

Weakness: Space complexity is very high in case of BFS search.

2. Also the time taken to search goal node is higher.

3. There is no informed search. It searches all the nodes in the breadth-first manner.

Depth First Search:

Use-Case Type	Number of Nodes Visited	Maximum length of Nodelist	number of nodes generated	Depth at which goal node is found	Time taken for the run(in secs)
Easy	59806	36330	96136	52267	14.73 secs
Medium	58391	35778	94169	51255	14.37 secs
Hard	4702	3650	8352	4578	0.1836 secs

Since number of moves are too high in case of DFS, so didn't print the moves for DFS.

Strength : It may find the solution quickly if it goes down the correct path. It does deep dive into the search space. As seen, it is able to find the goal node in case of hard use-case in smaller time compared to BFS.

Weakness: It becomes incomplete if it wanders down the wrong path. In case of easy and medium cases, the cost of finding the node is very high as shown in the table above. As seen in case of Hard case, its space complexity is lower, but in case of easier and medium use-cases, its space complexity is higher, as it may keep on generating the nodes again and again. Hence it is bad for deep or infinite depth state space.

Iterative Deepening Search :

Use-Case Type	Number of Nodes Visited (Counting in all the iterations)	Maximum length of Nodelist in the last iteration	number of nodes generated in the last iteration	Depth at which goal node is found	Time taken for the run(in secs)
Easy	89	6	28 (33 nodes in the last iteration)	5	0.031 secs
Medium	793	9	142 (255 in the last iteration)	9	0.15372 secs
Hard	401790	28	43024(57964 in the second last iteration)	32	4.17 secs

Moves taken for Easy Case : up right up left down

Moves taken for Medium Case: up right right down left left up right down

Moves taken for Worst Case: down right up up left left down down right right up up left left down right up left down down right right up up left left down down right right up left

Strength: 1. Searches iteratively in a depth first search manner. It reduces the high space complexity as in case of BFS.

2. It is good when the search space is huge and depth of solution is not known.

3. it is complete

Weakness: Will keep on searching the same nodes again and again in each iteration. Time taken is higher as it does the work iteratively.

Greedy-BestFirstSearch: (Output with heuristic based on the number of tiles out of place)

Use-Case Type	Number of Nodes Visited (Counting in all the iterations)	Maximum length of nodelist	Number of nodes generated	Depth at which goal node is found	Time taken for the run(in secs)
Easy	7	7	14	5	0.001secs
Medium	158	105	263	9	0.054 secs
Hard	1800	1196	2996	66	3.90 secs

Moves taken for Easy Case : up right up left down

Moves taken for Medium Case: up right right down left left up right down

Moves taken for Worst Case: up left down down right up left up right down right up left left down right right down left up left down right right up left up left down right right up left left down right right down left up right up left down left up right down down right up left up right down left left down right up right down left left up right

Strength : It is more informed search. It is fast just like single path toward the goal.

Weakness: It is incomplete just like DFS.

2. It is non-optimal just like DFS. It finds the goal node for hard use-case at 66 depth.

Greedy-BestFirstSearch: (Output with heuristic based on manhattan distance)

Use-Case Type	Number of Nodes Visited (Counting in all the iterations)	Maximum length of nodelist	Number of nodes generated	Depth at which goal node is found	Time taken for the run(in secs)
Easy	6	6	12	5	0.001 secs
Medium	242	176	418	19	0.45036 secs
Hard	279	218	497	54	0.65 secs

Moves taken for Easy Case : up right up left down

Moves taken for Medium Case: right up right down left up left down right right up left left down right up right down left

Moves taken for Worst Case: up left down down right up up left down down right up up left down right right up left left down right right down left up right up left left down right up right down left left down

right up left up right down down left up right up left down down right up

Better the heuristic function, more better the greedy algorithm.

Since manhattan distance heuristic is better than number of tiles out-of place heuristic, so 2nd approach searches the goal node for hard use-case at shorter depth.

A**StarSearch (output with heuristic based on the number of tiles out of place)**

Use-Case Type	Number of Nodes Visited	Maximum length of nodelist	Number of nodes generated	Depth at which goal node is found	Time taken for the run(in secs)
Easy	7	7	14	5	0.000007 secs
Medium	34	30	64	9	0.001secs
Hard				30	

System crashed while running hard case.

Moves taken for Easy Case : up right up left down

Moves taken for Medium Case: up right right down left left up right down

Moves taken for Worst Case: up left down down right right up up left left down down right right up up left left down down right right up up left left down down right up

Heuristic based on manhattan distance

Use-Case Type	Number of Nodes Visited (Counting in all the iterations)	Maximum length of nodelist	Number of nodes generated	Depth at which goal node is found	Time taken for the run(in secs)
Easy	6	6	12	5	0.001secs
Medium	18	16	34	9	0.003 secs
Hard				30	

System Crashed while running hard use-case.

Moves taken for Easy Case : up right up left down

Moves taken for Medium Case: up right right down left left up right down

Moves taken for Worst Case: up left down down right right up up left left down down right right up up left left down down right right up up left left down down right up

Strength : 1. It is complete and optimal. It finds the goal node optimally for easy, medium and hard use-cases.

Weakness: Heuristic function should be admissible.. It should never overestimate the actual cost of the best solution through node n.

I**daStarSearch (output with heuristic based on the number of tiles out of place)**

Use-Case Type	Number of Nodes Visited (Counting in all the iterations)	Maximum length of recursion	Time taken for the run(in secs)
Easy	19	5	0.000001 secs
Medium	87	9	0.003 secs
Hard	860	30	0.6 secs

Moves taken for Easy Case : up right up left down

Moves taken for Medium Case: up right right down left left up right down

Moves taken for Worst Case: up left down down right right up up left left down down right right up up left left down down right right up up left left down down right up

I**daStarSearch (output with heuristic based on manhattan distance)**

Use-Case Type	Number of Nodes Visited (Counting in all the iterations)	Maximum length of recursion	Time taken for the run(in secs)
Easy	19	5	0.000001 secs
Medium	45	9	0.0019999 secs
Hard	680	30	0.61 secs

Strength : It is complete and optimal. Space complexity is proportional to the longest path it traverses.

Dependent upon number of different values h(n) can take. Similar to recursive implementation of DFS.

It finds the goal node optimally than all the other search methods. Its space complexity is also lesser.