Tower of Hanoi Using A* Search Algorithm

Program Description:

In this project we have solve the tower of Hanoi puzzle using A* search and BFS using 4 pegs and disc ranging from 2 to 10 i.e 2,3….10. The heuristics used are minimum Euclidean distance from the last peg of the large disk in each pole and Manhattan distance of the topmost disc from the last peg, which has been discussed further. In this project I have teamed up with Vimal S. Patel student id : 101012783

State Space:

Every problem π specifies a state space Θ which is a 6 tuple i.e $\Theta = (S,A,c,T,I,S^G)$

In this the problem that we are solving are for n discs, for now let's consider the number of discs = 3, the state becomes as follows:

- S = This can be considered as the finite number of states for our problem i.e the total number of states required from [[2,1,0],[ ],[ ],[ ]] to [[ ], [ ],[ ] ,[2,1,0]]
- A = This are the finite set of Actions. In our problem action are, from each peg the topmost disc should be move to any other peg. This can be represented as follows: [[2, 1], [0], [ ], [ ],[ ]]
- c = It is represented as cost function. We are solving this problem with a unit cost i.e for each movement of disc the unit cost is 1.
- T = This is a transition relation which required to be deterministic. Consider if we move the topmost disc to the second peg, transition relation can be represented as follows: [[2,1,0],[ ],[ ],[ ]] -> [[2, 1], [0], [ ], [ ],[ ]]
- I = This is the representation of the initial state of the problem i.e at the start of the code all the discs are in the first peg. E.g : [[2,1,0],[ ],[ ],[ ]]
- S^G = This is the representation of the goal state. After the end of the implementation the goal state can be represented as : [[ ] , [ ],[ ] ,[2,1,0]]

Search Algorithms:

Breadth First Search

BFS expands the root node first and then the nodes which are generated by the root node is expanded next and their child nodes/successors and so on. In this the algorithm will process horizontally before proceeding vertically. It uses a data structure to keep track of the nodes, generally its first in first out (FIFO). It is a systematic strategy that expands all the paths at length 1 and those of length 2 and so on. BFS guarantees to find the solution and if there are multiple solutions it will find the shallowest goal state first.

A* Search:

It is an informed search algorithm formulated in terms of weighted graphs. It aims to find the path with the smallest cost to the goal state. At each iteration of the loop, the algorithm expands the state based on the estimated cost all the way to the goal. It does that with the help of the equation $f(n) = g(n) + h(n)$. Here $g(n)$ is the cost of the path from the starting node to node n and $h(n)$ is the cost of heuristic function which estimates the minimum cost from n to the goal state.

While trying to find the path with the cheapest cost we need to start with the lowest value of f and to choose an h that never overestimates the cost to reach the goal state.

Heuristics:

Manhattan distance of the top disk from the last peg:

Intuition:

In this we have considered the top most disk from the current state of the tower of corresponding pegs and calculated the Manhattan distance from the last peg. The formula for the used is as follows:

$$\text{Manhattan distance} = |\text{topmost disc}_i - \text{position of last peg}| + |\text{topmost disc}_j - \text{position of last peg}|$$

Here i and j represents the two corresponding pegs.

Consistency:

As in this heuristic we have returned the distance of the topmost disc from the last peg, the cost which will take to move the disk to the corresponding peg is 1 each time. This can be represented as follows:

Consider n =3, number of disks and the state of the tower is [[1], [], [2, 0], []]. In this case, in peg 3 there are two discs. So, disc 0 can be move to the 2nd or the 4th peg with unit cost = 1.

From this we can say that the next move will always have the unit cost which makes the heuristic **consistent.**

Admissibility:

For any state of the tower, if we are using this heuristic the cost of the disk to move to the last peg will be equals to 1 given that the disk is the top most disc.

Euclidean distance of the larger disk from the last peg:

Intuition:

In this we have considered the larger disc from the current state of the tower and calculated the Euclidean distance from the last peg. The formula for the used is as follows:

$$\text{Euclidean distance} = \sqrt{(larger\ disc\ frm\ peg - last\ peg)^2}$$

In the code we have returned the minimum distance of the disc which are closest to the last peg.

Consistency:

As in this heuristic we have returned the distance of the larger disc from the last peg, the cost which will take to move the disk to the corresponding peg is ≥ 2 if it is not the only disc in that peg.

Consider n =3, number of disks the state of the tower is [[1], [], [2, 0], []]. In this case in peg 3 there are two discs. So, disc 0 can be move to the 2nd or the 4th peg with unit cost = 1 and then the disc 2 will be moved to the corresponding peg with cost is equal to 2.

From this we can say that the next move will have the cost ≥ 2 which makes the heuristic **not consistent.**

Admissibility:

For any state of the tower, if we are using this heuristic the cost of the disk to move to the last peg will be ≥ 2 given that the disk is not the only disc in the tower.

Tables and Plots with the runtime of each algorithm vs. number of discs n:

For Heuristic 1:

No. of discs	Time taken by the heuristic	States Expanded	States visited
2	0.003	6	11
3	0.041	40	60
4	0.201	237	253
5	0.784	1005	1021
6	3.235	4025	4093
7	13.805	16368	16381
8	56.712	65514	65530
9	235.34	262117	262138
10	976.254 > 10 min	1048322	1048568

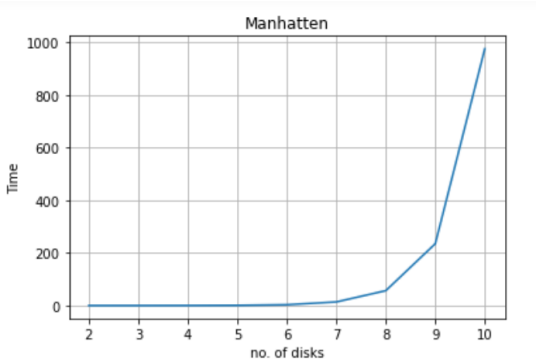

Figure 2: time taken v/s number of disc for first heuristic

For Heuristic 2:

No. of discs	Time taken by the heuristic	States Expanded	States visited
2	0.0099	11	15
3	0.048	59	63
4	0.208	254	255
5	0.837	1022	1023
6	3.539	4094	4095

7	14.23	16382	16383
8	58.947	65534	65535
9	243.82	262142	262143
10	1032.7724 > 10 min	1048567	1048575

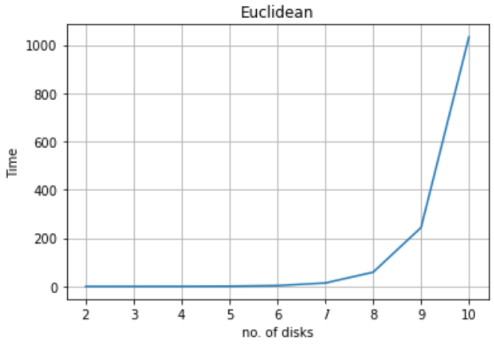

Figure 3 : time taken v/s number of disc for second heuristic

Optimal and Sub-optimal Solution:

We have used k as a maximum number of discs with runtime limit of ≈ 10 min. As Heuristic 1 is consistent we can consider him to provide an optimal solution for A* while BFS will provide the sub-optimal solution.

The comparison is shown in the below table.

No. of discs	Time taken by Optimal	Time taken by Sub-Optimal	Plaan length Optimal	Plan length Sub-Optimal
2	0.003	0.0019	3	3
3	0.041	0.042	5	5
4	0.201	0.191	9	9
5	0.784	0.828	13	13
6	3.235	3.313	17	17
7	13.805	14.901	25	25
8	56.712	57.583	33	33
9	235.34	238.473	41	41
10	976.254 > 10 min	979.824 > 10 min	49	49

From the above table we can say that the number of disc that can be handled optimally are $k = 9$ with the run time of 235.34 sec and 238.47 sec respectively. We can see that if the discs are increased the runtime for the code also increases.

Conclusion:

From the report we can say that the optimal solution is given by A* search using Manhattan distance from the last peg which is the heuristic that we have created. The code will work optimally with number of discs up to 9 and the runtime will increase with the increase in number of discs.

References:

- https://en.wikipedia.org/wiki/A*_search_algorithm
- https://en.wikipedia.org/wiki/Breadth-first_search
- Book: Artificial Intelligence a Modern Approach by Stuart Russel and Peter Norvig
- Lecture Slides.
- Lab 3 Solution.