CS5960 Artificial Intelligence Principles and Techniques

TOWER OF HANOI

Project by:

Rishab Balakrishnan (MSc. Data Science)

Prutha Edwankar (MSc. Artificial Intelligence)

Course Leader:

Professor Sara Bernardini

Royal Holloway University of London, Egham, Surrey

Index

1. Introduction

2. Search Algorithms Used
   2.1. A* Search Algorithm
   2.2. Breadth First Search Algorithm
   2.3. Rationale for implementations

3. Heuristic Functions

4. Results

5. Extension – Pattern Database

6. Conclusion

7. References

1. Introduction

Game Setting:

- Tower of Hanoi is a game with usually 3 pegs and a fixed number of disks.
- For this project, we have used 4 pegs and the number of discs (n) keep increasing up to a runtime of 10 minutes
- The initial state is defined by all the disks in the leftmost peg, i.e. Peg1 with the largest disk at the bottom and the smallest on the top.
- The goal state is where all disks are moved over to the rightmost peg, i.e. Peg4 with the largest disk at the bottom and the smallest on the top.

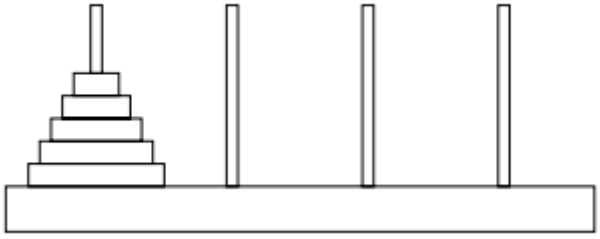

Fig 1.1 – Initial State [2]

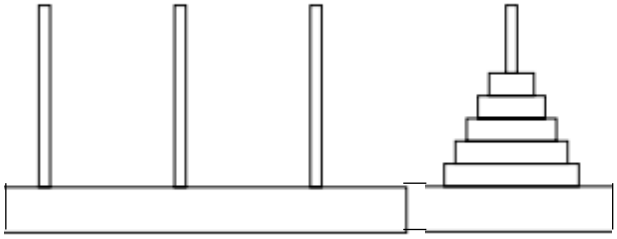

Fig 1.2 – Final State [2]

Game Rule:

- Possible to move only one disk per move to any other peg.
- Only the top most disk on a peg can be moved to other pegs, i.e. the larger sized disc has to always be below the smaller sized disk.

2. Search Algorithms Used

There were two kinds of search algorithms implemented for this puzzle game:

- A* Search algorithm
- Breadth first search

A* search algorithm is a smart and a flexible algorithm that can be used to solve large range of problems in the best and the shortest path (cost in our case). It gives a more realistic result as it considers the current state and considers how far the current state from the goal state is in order to draw the best possible path.

2.1. A Start Search Algorithm

In A* nodes are evaluated by combining:
g(n) - the cost to reach the node
h(n) - the cost to get from the node to the goal:

Thus, f(n)=g(n)+h(n).

Here, f(n)= estimated cost of the cheapest solution through n. Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated. [3]

In order to find the cheapest path, we need to have the lowest value of f(n), i.e. the lowest value of g(n) + h(n).

There are two conditions for the heuristics of A*:

The heuristic is admissible if it does not overestimate the cost to reach the final state.

i.e. h(n) <= h*(n) [2] h*(n) – true cost from node n

The heuristics is consistent when A* is applied to graph search. Heuristics is consistent when

h(n) <= c(n, a, n') + h(n') [3]

Properties of A*:

a. A* is complete unless there are infinite nodes with f <= C*
b. It is optimal as it does not expand f_{i+1} unless f_i is finished.
c. It increases exponentially in time.
d. A* keeps all the nodes in the memory thus making it smart.

The A* search algorithm that we implemented for the tower of Hanoi game puzzle is as follows:

- The g i.e. the cost increments by one for each state
- The h calculates the state from current position to the goal state
- Based on the h and g value the final state is reached in the least number of moves.[3]

2.2. Breadth First Search Algorithm

In Breadth-first search algorithm the root node is expanded first, then all the successors of the root node are expanded next. Further the successors, of these expanded nodes are expanded and so on. [2]

In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. Breadth-first search is an instance of the general graph-search algorithm in which the shallowest unexpanded node is chosen for expansion. Thus, new nodes go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first. There is one slight tweak on the general graph-search algorithm, which is that the goal test is applied to each node when it is generated rather than when it is selected for expansion. [2]

For the tower of Hanoi puzzle game the breadth first search algorithm was implemented by removing the heuristic function H and just keeping the cost value. By doing this each peg is explored and thus the number of states and the time taken is more as compared to A* as shown in the tables below.

2.3. Rationale for Implementation

The purpose of choosing A* and breadth first search algorithm was to establish the difference in the working efficiency of uninformed and informed search algorithm. The A star algorithm works on a more realistic basis by choosing the next move that is closest to the final state. In breadth first search the next move is decided on the basis of shortest path to the next move and not the shortest path to the final state.

3. Heuristic Function

The implementation of the search algorithms used for solving the Hanoi puzzle problem are extension of [1] where:

- The problem was converted from 3 pegs to 4 pegs
- G function adds 1 to the current cost path:
  G = cost + 1
- H that is the heuristics alternatively decrements and increments by 1 for each state on each peg to make the disks move in both the directions and for the last peg it is changed by 2 so that the states are changed in way that the largest weighing disc is placed at the bottom.
  For example if the current game is using 5 discs, and 3 are on the right position then F is decremented by 3.
- The final state is changed from peg 2 to peg 4
- A timer is added to calculate the runtime
- The number of discs to be moved are taken from the user.
- For breadth-first search H is kept 0. Thus the algorithm would be managed only always adds 1 to the current cost, and each peg is explored at each state.

An example result is presented below:

```
Enter number of disc 3
Starting State: ([3, 2, 1], [], [], [])
Final State: ([], [], [], [3, 2, 1])
Total Number of States: 6
Solution:
([3, 2, 1], [], [], [])
([3, 2], [1], [], [])
([3], [1], [2], [])
([], [1], [2], [3])
([], [1], [], [3, 2])
([], [], [], [3, 2, 1])
Time Taken: 0.015000 seconds
```

Results for a*

```
Enter number of disc 3
Starting State: ([3, 2, 1], [], [], [])
Final State: ([], [], [], [3, 2, 1])

Total Number of States: 17
Solution:
([3, 2, 1], [], [], [])
([3, 2], [1], [], [])
([3], [1], [2], [])
([], [1], [2], [3])
([1], [], [2], [3])
([], [], [2, 1], [3])
([], [], [2], [3, 1])
([2], [], [], [3, 1])
([], [2], [], [3, 1])
([1], [2], [], [3])
([], [2, 1], [], [3])
([], [2], [1], [3])
([2], [], [1], [3])
([], [], [1], [3, 2])
([1], [], [], [3, 2])
([], [1], [], [3, 2])
([], [], [], [3, 2, 1])
Time Taken: 0.015000 seconds
```

Result for BFS

From the above two results we can see that the number of states and the time taken for A* is less than that of BFS

4. Results

There were two results obtained by two different search algorithms:

Optimal Solution: Using A star Algorithm
$N_{max} = 10$
Runtime = 10.41 mins

Sub Optimal Solution: Breadth First Search Algorithm
$N_{max} = 6$
Runtime = 1.30 mins (above 6 discs the runtime exceeds 10 mins)

Table of comparison:

No. of Discs	Runtime for optimal Solution	Run time for sub optimal solution
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0.10
6	0	1.3
7	0.08	23.4
8	0.11	-
9	1.55	-
10	10.41	-

Table 1 – No. of discs vs Runtime

Note – Time is calculated in minutes

No. of Discs	Length of optimal solution	Length of sub optimal solution
1	2	4
2	4	16
3	6	17
4	11	165
5	28	740
6	74	2762
7	214	
8	709	
9	2127	
10	5342	

Table 2 – No. of disc vs Length of solution

Note – Length of solution is calculated in terms of number of states

5. Pattern Database

The PDB heuristic aims at achieving two tasks:

1. Collect all states for a subset of the problem (in our case, if the number of disks is 7: then we will collect all states for disks 5,6,7 to simplify the problem)
2. Implementing PDB speeds up results and also saves memory.
3. Link all possible states with their corresponding costs.
4. Cost definition = sum (number of rows difference for every element between existing state /next state and final state) + sum (number of column difference for every element between existing /next state and final state)

Sample output for all states in a List:

Index	Type	Size	Value
0	list	24	[(0, 5, 6, 7), (0, 5, 7, 6), (0, 6, 5, 7), (0, 6, 7, 5), (0, 7, 5, 6), ...

(Checks for the rules of the game not implemented as all possible states are covered)

Example for cost definition for PDB:

STATE 1 : Columns #0 ,# 1, # 2 are the pegs and FINAL STATE = Column #3

	0	1	2	3
0	4	1	3	4
1	0	0	2	3
2	0	0	0	2
3	0	0	0	1

Cost of state 1 = [(0) + (3 -0) + (2-1) + (1 -0)] + [(3 – 1) + (3-1) +(3-2) +(3-2)] = 11

The function written in the code will output the cost of the state when input is a NumPy array and the last column is the final state required.

6. Conclusion

The following conclusions are drawn from the test results

- A star search algorithm works more efficiently than the BFS search algorithm, as it can handle more number of discs for a runtime of 10 minutes and the length of the solution is also shorter.
- A* gives is a smart and flexible algorithm as it gives a more realistic result by calculating the cost per move and the cost from current node to the final node. Hence it chooses the lowest cost of F.
- BFS explores and expands every node so there is a possibility of repetition or getting stuck in an infinite loop. Thus it takes longer runtime.
- For the pattern database we were only able to implement the function to get the cost of each state.

References:

[1] https://github.com/n3k/HanoiTowers/tree/master/src

[2] Recent Progress in Heuristic Search: A Case Study of the Four-Peg Towers of Hanoi Problem.

[3] AI A Modern Approach. Stuart Russell, Peter Norvig