CS 312 Assignment 2 Report; Team - 21

M V Karthik (200010030), Josyula V N Taraka Abhishek (200010021)

January 2, 2022

1 Introduction

In this task, Block World Domain had to be implemented. Blocks World Domain Game starts with an initial state consisting of a fixed number of blocks arranged in 3 stacks and we can move only top blocks of the stacks. Blocks World is a planning problem where we know goal state beforehand and path to Goal state is more important. Essentially we have to achieve a goal state that is a particular arrangement of blocks by moving these blocks.

We compare the Best-First-Search and HillClimbing alogorithms on:

1. Path Length
2. Number of states explored
3. Time Taken

2 Pseudo Codes

We used heapq module in python to access maximum heuristic state in open list in optimal time.

Pseudo code for alogorithms are given below.

2.1 Function MoveGen(state)

Algorithm 1: MoveGen(state)

Function $MoveGen(state)$;
$NextStates \leftarrow [\ ]$;
for $NeighbourNofstateinorder(Heuristicvalue)$ **do**
| $N = PriorityQue.pop()$ $NextStates.append(N)$
end
return $NextStates$

2.2 GoalTest(state)

Returns True if input state is goal state

Algorithm 2: GoalTest(state)

Function *GoalTest*(*state*) ;
if *state.value* == *goalState.value* **then**
  returnTrue
end
returnFalse

2.3 Ord Heuristic:

Add the order of the state if block is not in correct stack and absolut value of the difference if block is in correct stack and not at correct position.

```
def OrdHeuristic(s: State):
g = goal
ret = 0
for i in range(len(s.grid)):
    for j in s.grid[i]:
        if j in g.grid[i]:
            ret += abs(ord(j)-ord(g.grid[i][g.grid[i].index(j)]))
        else:
            ret += abs(ord(j))
return ret
```

2.4 Manhattan Heuristic:

Add the manhattan distance of the block if block is not in correct stack and absolut value of the difference if block is in correct stack and not at correct position.

```
def ManhattanHeuristic(s: State):
g = goal
ret = 0
for i in range(3):
    for j in g.grid[i]:
        if j in s.grid[i]:
            ret +=  abs((g.grid[i].index(j) - s.grid[i].index(j)))
        else:
            for k in range(3):
                if j in  s.grid[k]:
                    ret += abs(i -k)
                    + abs(g.grid[i].index(j)-s.grid[k].index(j))
return ret
```

2.5 PositionBased Heuristic

Subtract the heigth of the block if block is on the correct block with respect to the goal state otherwise add the height of the block.

```
def PositionBased(s: State):
ret = 0
for i in range(len(s.grid)):
    for j in range(len(s.grid[i])):
        done = False
        if j != 0:
            for k in range(len(goal.grid)):
                if s.grid[i][j] in goal.grid[k] and s.grid[i][j-1]
                    == goal.grid[k][goal.grid[k].index(s.grid[i][j])-1]:
                    ret += j+1
                    done = True
                    break
            if done == False:
                ret -= j+1
        else:
            for k in range(len(goal.grid)):
                if goal.grid[k] != [] and s.grid[i][j] == goal.grid[k][0]:
                    ret += j+1
                    done = True
                    break
            if done == False:
                ret -= j+1
return -1*ret
```

2.6 L2Norm Heuristic

Instead of using manhattan distance, we use L2Norm.

```
def L2Norm(s: State):
g = goal
ret = 0
for i in range(3):
    for j in g.grid[i]:
        if j in s.grid[i]:
            ret +=  abs((g.grid[i].index(j) - s.grid[i].index(j)))**2
        else:
            for k in range(3):
                if j in  s.grid[k]:
                    ret += (abs(i -k)**2
                    + abs(g.grid[i].index(j)-s.grid[k].index(j))**2)
return ret
```

3 How to run?

The code should be run as:

$*python3* 21.*py* *input.txt*

output will be printed in terminal.

4 Data and Inference

All statistics related to output are listed here. These are the statistics for the input example given in problem statement. Where 1 for OrdHeuristic, 2 for ManhattanHeuristic, 3 for L2Norm or 4 for PositionBased heuristic.

Algorithm	Heuristic	Path length	States Explored	Time Taken(s)	Goal Reached
BFS	1	77	3917	3.72	Y
BFS	2	60	1810	0.57	Y
BFS	3	66	2666	1.13	Y
BFS	4	23	319	0.02	Y
HC	1	3	13	0.00	N
HC	2	3	13	0.00	N
HC	3	3	13	0.00	N
HC	4	2	9	0.00	N

As expected, the Hill Climbing algorithm runs faster than Best First Search due to its greedy nature and lesser number of states explored. We couldn't reach Optimal State in HillClimbing because of a drawback for the HillClimbing algorithm found a local maximum and is stuck in that. This is very general that optimal state is not reached in HillClimbing Algorithm.

Out of all heuristic functions Position based heuristic was effective because of the fact it is based on relative postion of blocks.

5 Conclusion

From the results above, it is seen that Best First Search (BFS) always finds an optimal solution with the trade off of time, as it explores all possible N! states in the solution space. Conversely, on the other hand, the Hill Climbing Algorithm, has lesser execution time due recursive greedy selection in iterations but it cannot guarantee an optimal solution in all cases