CS 312: Artificial Intelligence Laboratory

Lab 2 report

Ashwin Waghmare(210010060)

Tejal Ladage(210010026)

• Introduction:

The objective of this task is to solve Block World Domain. Block World Domain has an initial state consisting 6 blocks arranged in 3 stacks and we can move only top blocks of the stacks. Blocks World is a planning problem where we know the goal state beforehand. We have to achieve a goal state after a particular arrangement of blocks by moving the block. Number of stacks allowed during any arrangement is always three.

• Description:

1. State space:

A state space is the set of all configurations that a given problem and its environment could achieve. Each configuration is called a state. Here the state space is all possible permutations of the given 'n' blocks on 3 stacks.

2. Start state:

Start state is the initial arrangement of blocks at the start of the game.

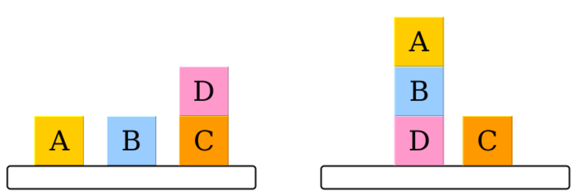
3. Goal state

Goal node is the arrangement which we want to achieve in order to complete the game. The start states and goal states will be given in the input file.

Example:

Start state = [['A', 'GROUND'], ['B', 'GROUND'], ['D', 'C', 'GROUND']]

Goal state = [['GROUND'], ['A', 'B', 'D', 'GROUND'], ['C', 'GROUND']] jb



Initial state Goal

• Pseudocode:

a) goalTest :

```
function goalTest (currentstate):
    if currentstate == goalstate:
        Return True
    else:
        Return False
```

b) MoveGen :

```
function MoveGen (current_state):
    initialise empty list for neighbours
    if stack1:
        tempcopy = copy(current_state)
        pop element from stack1 and append it to stack2
        append tempcopy to neighbours
        tempcopy = copy(current_state)
        pop element from stack1 and append it to stack3
        append tempcopy to neighbours


    if stack2:

        tempcopy = copy(current_state)
        pop element from stack2 and append it to stack1
        append tempcopy to neighbours
        tempcopy = copy(current_state)
        pop element from stack2 and append it to stack3
        append tempcopy to neighbours


    if stack3:

        tempcopy = copy(current_state)
        pop element from stack3 and append it to stack2
        append tempcopy to neighbours
        tempcopy = copy(current_state)
        pop element from stack3 and append it to stack1
        append tempcopy to neighbours
```

• Heuristic Functions:

A) Heuristic 1: In heuristic1, we implemented following logic:

1) +1 for block if it is in correct stack and on the top of correct block. -1 for block otherwise.

Example:

Start state: [‘F’, ‘B’, ‘A’, ‘GROUND], [‘D’, ‘GROUND’], [‘E’, ‘C’, ‘GROUND’]

Goal state: [‘F’, ‘B’, ‘GROUND’], [‘A’, ‘C’, ‘GROUND’], [‘E’, ‘D’, ‘GROUND’]

Heuristicvalue of start state = -1-1+1-1-1-1 = -4

Heuristic of goal state = +6

B) Heuristic 2: In this heuristic function, we implemented following logiC.

1)+1 x (level of block) if block is in correct stack and on the top of correct structure i.e., all the block below it must be in correct form

2)-1 x (level of block) otherwise

Example: Start state: [‘F’, ‘B’, ‘A’, ‘GROUND], [‘D’, ‘GROUND’], [‘E’, ‘C’, ‘GROUND’]

Goal state: [‘F’, ‘B’, ‘GROUND’], [‘A’, ‘C’, ‘GROUND’], [‘E’, ‘D’, ‘GROUND’]

Heuristic of start state = -1-2-3-1-1-2 = -10

Heuristic of goal state = +9

• Hillclimbing Table:

startState = [['A', 'B', 'C','GROUND'], ['D','E' ,'GROUND'], ['F','GROUND']]

goalState = [['C','GROUND'], ['A','E', 'GROUND'], ['B','D','F','GROUND']]

Heuristic function	States explored	Time Taken	Optimal Solution
Heuristic1	4	0.0049078 sec	Yes
Heuristic2	2	0.0024039 sec	No(Reaches local maxima)

startState = [['A', 'B', 'C','GROUND'], ['GROUND'], ['D','GROUND']]

goalState = [['C','GROUND'], ['A', 'GROUND'], ['B','D','GROUND']]

Heuristic function	States explored	Time Taken	Optimal Solution
Heuristic1	3	0.00428 sec	Yes
Heuristic2	3	0.00569 sec	Yes