CS312:Artificial Intelligence Laboratory

Lab 2 Report Group 24

Utkarsh Prakash - 180030042 Manjeet Kapil - 180010021

1. Introduction

In this assignment we try to solve the block world problem by using Best First Search and Hill Climbing algorithm which is one of the most famous planning domains in artificial intelligence. Blocks World is a planning problem where we know goal state beforehand and path to Goal state is more important.

2. State Space

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 and we have to achieve a goal state that is a particular arrangement of blocks by moving these blocks. Only one block may be moved at a time. In other words, any blocks that are, at a given time, under another block cannot be moved.

3. Start Node and Goal Node

Start Node/Initial Node:

```
[3, 5, 4]

[2, 6, 1]

[]
```

Goal Node:

```
[4, 2]

[6, 1, 5, 3]

[]
```

The start node is the initial state/arrangement of the blocks and goal node is the target state/arrangement of the blocks. Both the start node and target node are given as inputs to the program. Here each list represents a stack/pile and the leftmost element represents the bottom most block in the stack.

4. Pseudo Code

MoveGen(state)

This function accepts a state and returns all the states that can be reached from the given state in one step i.e. by moving a block.

function MoveGen(state):

```
1. neighbors = []
2. for top_most_block in each stack:
3.      for each possible move as neighbor:
4.           neighbor.h = heuristic(neighbor, goal_node)
5.           neighbors.append(neighbor)
6. return neighbors
```

`heuristic()` is a function that calculates the value of the heuristic for the given node.

GoalTest(state, target_state)

This function checks whether the state is the target(goal) state or not. If it is a target state it returns true otherwise false.

function GoalTest(state, target_state):

```
1. if state.value == target_state.value then
2.      return true
3. return false
```

5. Heuristic Function

- **H1:** This function simply checks whether each block is on the correct block, with respect to the final configuration. We subtract one for every block that is on the block it is supposed to be on, and add one for every one that is on a wrong one. For example, the heuristic value for our start state is 6 since every block is not in its correct position and goal state is -6 since all the blocks are in the correct position. With such a function we are looking for smaller values of the heuristic function. In other words, the algorithm is performing steepest gradient descent.
- **H2:** This function looks at the entire pile that the block is resting on. If the configuration of the pile is correct, with respect to the goal, it subtracts one for every block in the pile,

or else it adds one for every block in that pile. For example, the heuristic value for the start node is 6 = 0 + 1 + 2 + 0 + 1 + 2. With such a function we are looking for smaller values of the heuristic function.

- **H3 (Manhattan Distance):** This function looks at the distance between each of the blocks in the current state and the goal state. The x-coordinate of the block is the number of the stack to which it belongs (counting the stack number from left) and the y-coordinate of the block is the position of the block from the bottom in the stack/pile. The Manhattan Distance is given by:

$$|current_x - goal_x| + |current_y - goal_y|$$

The value of the heuristic is given by the sum of the Manhattan distances for each block. For example, the heuristic value for the start node 12. With such a function we are looking for smaller values of the heuristic function.

6. Best First Search Analysis and Observation

Heuristics/ Parameters	**Number of state Explored**	**Path Length**	**Time Taken**	**Reaching optimal solution**
H1	332	148	0.029 sec	Yes
H2	5193	2139	4.69 sec	Yes
H3	436	149	0.039 sec	Yes

The Best-First Search algorithm is able to find the solution with all the three heuristics. This is because the graph Best-First Search algorithm is complete in finite domains. We also see that the change in heuristic can change the time taken by the algorithm. With the heuristic H2 the time taken increases significantly whereas with heuristic H1 and H3 the time taken is less. The amount of reduction in time complexity depends upon the quality of heuristic as H1 and H3 better capture the cost of reaching the goal state from the present state.

7. Comparison with Hill Climbing

These results are shown for following example:

Start Node/Initial Node:

```
[3, 5, 4]

[2, 6, 1]

[]
```

Goal Node:

```
[4, 2]

[6, 1, 5, 3]

[]
```

Best First Search

Heuristics/ Parameters	**Number of state Explored**	**Path Length**	**Time Taken**	**Reaching optimal solution**
H1	332	148	0.029 sec	Yes
H2	5193	2139	4.69 sec	Yes
H3	436	149	0.039 sec	Yes

Hill Climbing

Heuristics/ Parameters	**Number of State Explored**	**Path Length**	**Time taken**	**Reaching Optimal solution**
H1	63	14	0.0014 sec	Yes
H2	33	Stuck in local minima	0.0005 sec	N/A
H3	25	Stuck in local minima	0.0005 sec	N/A

As we can see from above results, the Hill Climbing algorithm is faster than the Best First Search algorithm due to its greedy approach. However, one disadvantage of the Hill Climbing algorithm is that it has chances of getting stuck in local minima i.e. the hill climbing algorithm is not complete.