CS312 - Artificial Intelligence Lab
Assignment 2

Group 10

Rudraksh (190030038), Sumedhsingh (190030042)

Likhilesh (190030024)

Introduction

The aim of this assignment is to simulate the Blocks Word Domain Game using Best-First Search (BFS) and Hill Climbing (HC). This Game begins with a given number of blocks arranged in three stacks, and we can only move the top blocks of the stacks. We must move these blocks to reach a goal state, which is a specific arrangement of blocks. Blocks World is a planning problem in which the goal state is known ahead of time and the pathway to it is more significant.

1 Description of Domain

1.1 State Space

A state space is the set of all possible configurations of a system. In this Case we have stack of different boxes in which every new state represents different configuration of the boxes, we can move through the state space by changing this configuration and reaching to desired configuration (i.e., Goal state).

1.2 Start State & Goal State

We have chosen vector of strings to represent a state, where each string represents a stack.
For eg

Input (Start State):	**Output (Goal State):**
Stack of boxes 1: “fbe”	Stack of boxes 1: “bda”
Stack of boxes 2: “ad”	Stack of boxes 2: “cfe”
Stack of boxes 3: “c”	Stack of boxes 3: “”

1.3 Pseudo Codes

The section below has pseudo codes of a few important functions.

1.3.1 moveGen(state)

This function generates and returns the states that can be reached from a given state in one step. Note that at most 6 new neighbours are possible for a given state.

Algorithm 1 moveGen(state)

```
neighbours ← list() ▷ initialize neighbours to empty list
if state.stack1 not empty then
    neighbour1 ← state.copy()
    neighbour1.stack2.push(neighbour1.pop())
    neighbour1.setHeuristic()
    neighbours.append(neighbour1)

    neighbour2 ← state.copy()
    neighbour2.stack3.push(neighbour2.pop())
    neighbour2.setHeuristic()
    neighbours.append(neighbour2)
```

```
end if

if state.stack2 not empty then
    neighbour3 ← state.copy()
    neighbour3.stack1.push(neighbour3.pop())
    neighbour3.setHeuristic()
    neighbours.append(neighbour3)

    neighbour4 ← state.copy()
    neighbour4.stack3.push(neighbour4.pop())
    neighbour4.setHeuristic()
    neighbours.append(neighbour4)
end if

if state.stack3 not empty then
    neighbour5 ← state.copy()
    neighbour5.stack1.push(neighbour5.pop()
    neighbour5.setHeuristic()
    neighbours.append(neighbour5)

    neighbour6 ← state.copy()
    neighbour6.stack2.push(neighbour6.pop())
    neighbour6.setHeuristic()
    neighbours.append(neighbour6)
end if

return neighbours
```

1.3.2 goalTest(state, goalState)

This function returns true if the given state is goal state other wise returns false.

Algorithm 2 goalTest(state, goalState)

```
if state.stack1 != goalState.stack1 then
    return False
end if
if state.stack2 != goalState.stack2 then
    return False
end if
if state.stack3 != goalState.stack3 then
    return False
end if
return True
```

2 Heuristic Functions

2.1 Heuristic Function-1

In this heuristic function, we have implemented the following logic:

In an intermediate state, if the position of the particular block in a current state and goal state doesn’t matches, we increase the heuristic value considering it as a penalty.

Algorithm 3 Heuristic function 1

```
if(currentState(Block) != goalState(Block)){
    for all blocks:
        h(Block) += 1;
}
else
    continue; //Default value = 0
```

2.2 Heuristic Function-2

In this heuristic function, we implemented the following logic:

In an intermediate state,

Case 1: Increasing the heuristic value by 2 if there is no matching block in the same stack.

Case 2: Increasing the value by 1 if we find the block at same stack but not in same position.

Case 3: And if we find the block in same position then we don’t change the value.

Algorithm 3 Heuristic function 1

```
for all blocks in one stack:
    if(currentStack(Block) != goalStack(Block)){
        if(currentState(Block) == goalState(Block)){
            h(Block) += 1
        }
        else
            h(Block) += 2;
    }
    else
        continue; //Default value = 0
```

2.3 Heuristic Function-3

In this heuristic function, we implemented the following logic:

In an intermediate state,

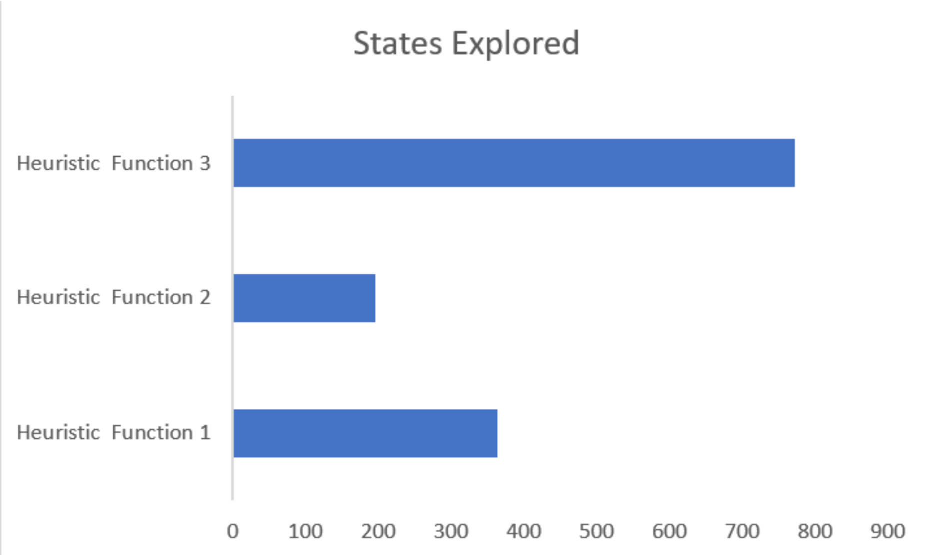
While traversing in the stack from bottom we make no changes in heuristic value until it matches the goal state stack, but if at some point it doesn’t match, we add the heuristic value by remaining height of the current state stack.

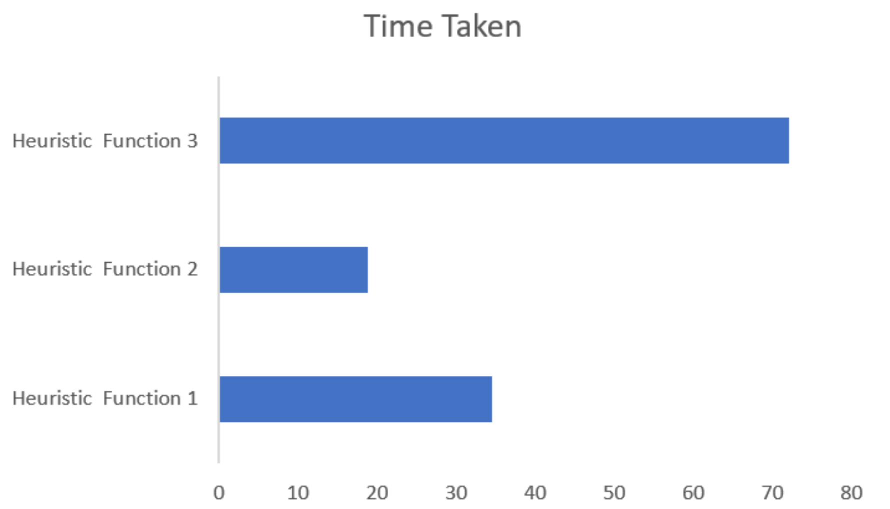
Algorithm 3 Heuristic function 1

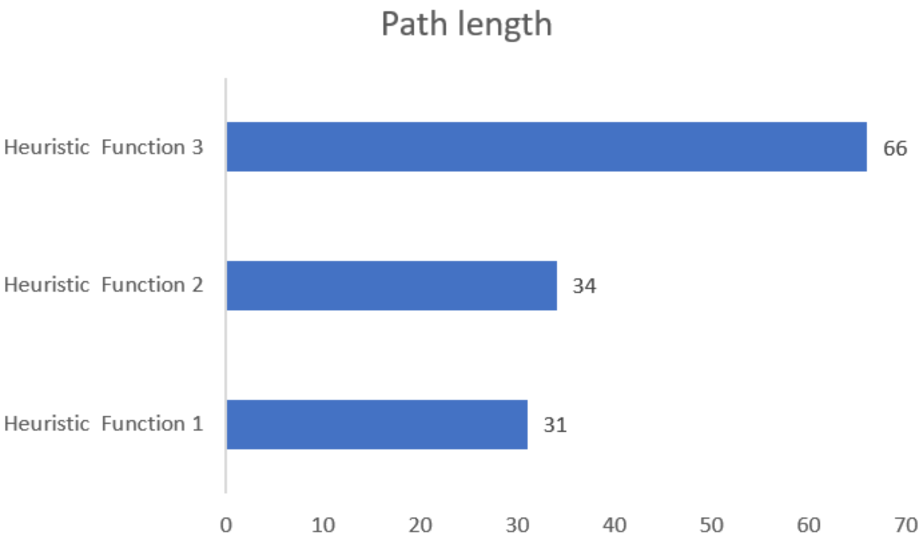
```
for all blocks in one stack{
    if(currentBaseStack(Block) != goalBaseStack(Block)){
        break at jth position of stack;
    }
    for(j to current.size())
        h(block)+=1
}
else
    continue; //Default value = 0
```

3 Best First Search Analysis

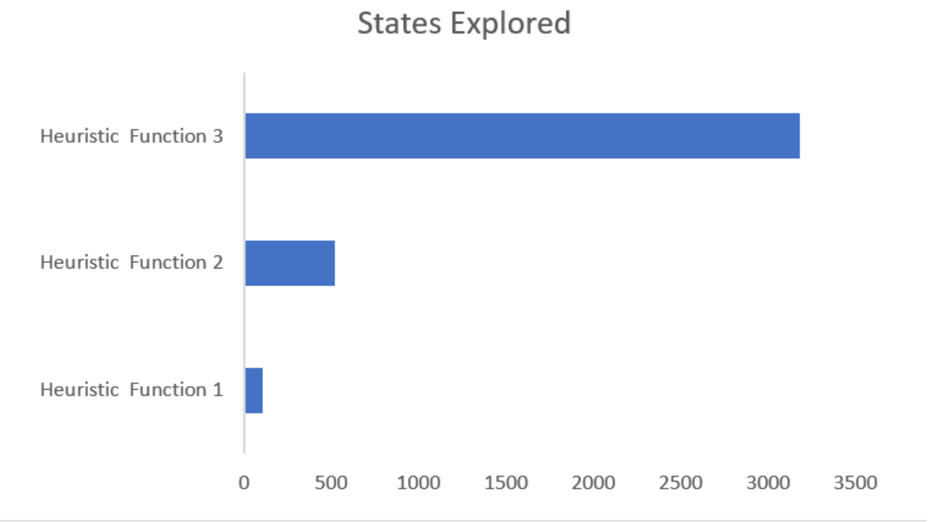
For Input 1:

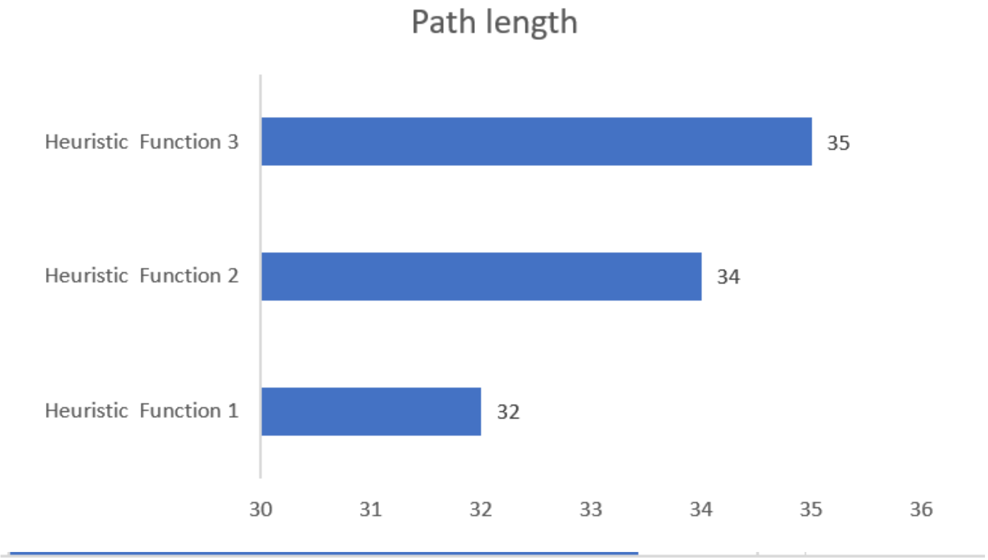


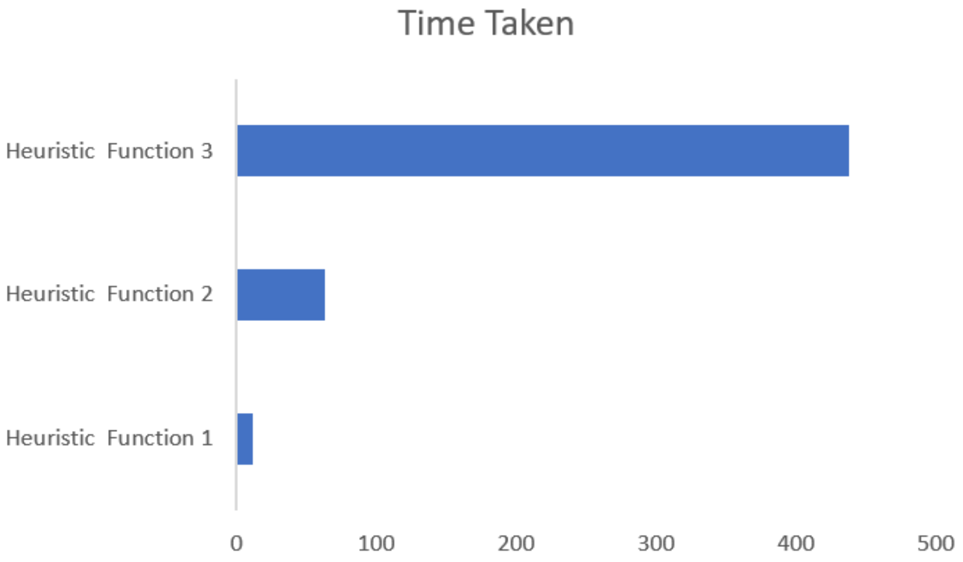

Time Taken
Heuristic Function 3
Heuristic Function 2
Heuristic Function 1
0
10
20
30
40
50
60
70
80


Path length
Heuristic Function 3
66
Heuristic Function 2
34
Heuristic Function 1
31
0
10
20
30
40
50
60
70

For Input 2:

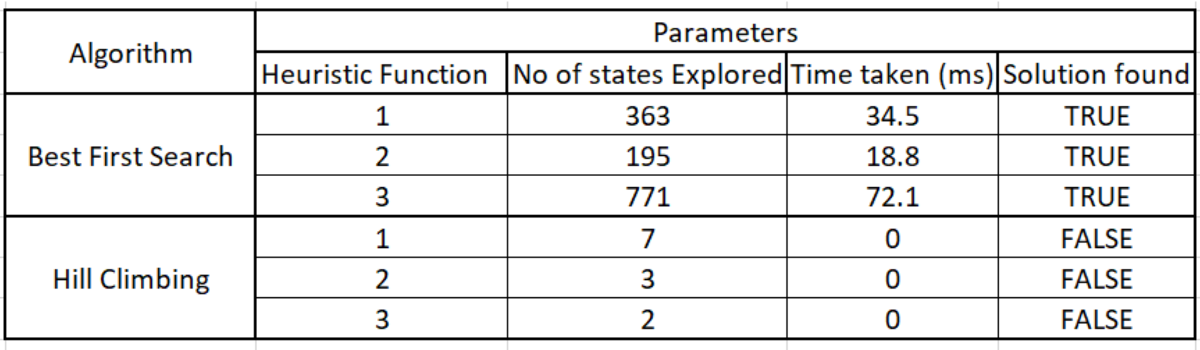

States Explored
Heuristic Function 3
Heuristic Function 2
Heuristic Function 1
0
500
1000
1500
2000
2500
3000
3500


Path length
Heuristic Function 3
Heuristic Function 2
Heuristic Function 1
35
34
32
30
31
32
33
34
35
36


Time Taken
Heuristic Function 3
Heuristic Function 2
Heuristic Function 1
0
100
200
300
400
500

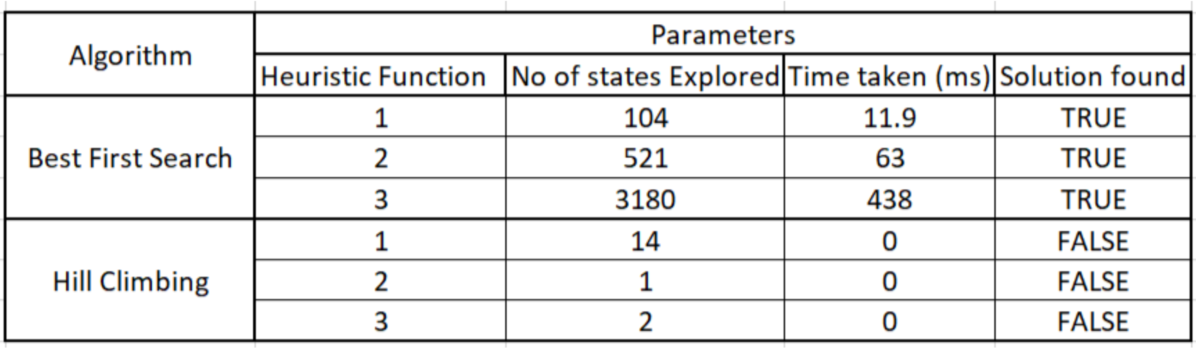
4 Best First Search and Hill Climb Comparison

For Input 1:



Algorithm	Parameters			
	Heuristic Function	No of states Explored	Time taken (ms)	Solution found
Best First Search	1	363	34.5	TRUE
	2	195	18.8	TRUE
	3	771	72.1	TRUE
Hill Climbing	1	7	0	FALSE
	2	3	0	FALSE
	3	2	0	FALSE

For Input 2:



Algorithm	Parameters			
	Heuristic Function	No of states Explored	Time taken (ms)	Solution found
Best First Search	1	104	11.9	TRUE
	2	521	63	TRUE
	3	3180	438	TRUE
Hill Climbing	1	14	0	FALSE
	2	1	0	FALSE
	3	2	0	FALSE

5 Inference and Conclusion

- Best first Search method explores larger number of state space depending upon priorities and heuristic functions, also reaches to the goal state if state space is finite and consists of goal state. Whereas Hill Climb is more of a greedy approach whose aim is to find local maxima and not global maxima (i.e., Goal State) in state space. Which implies it doesn’t always find us the goal state for us.

- Consequently, the time taken by Best First Search is also much higher than that by Hill Climbing because of the reason mentioned above.
- We can observe that Heuristic Function 1 is the best in terms of time taken and also path length followed by Heuristic Function 2 and then 3.
- As discussed, Hill Climb method takes less time but doesn’t guarantee a solution, so we can first use hill climb method to find solution but if we don’t get our solution, we can carry on to best first search method.
- Although in our case we didn’t find any solution from hill climb for both the inputs.