AI - LAB
Assignment-2
Report

Arvind Kumar M - 200020008

Tarun Saini - 200010051

January 2, 2022

Contents

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.

2 State space

The state space is the Euclidean space in which the variables on the axes are the state variables. The state of the system can be represented as a vector within that space. To abstract from the number of inputs, outputs and states, these variables are expressed as vectors. In this problem statement we have used that state space as a stack of the Block World. Since, a state/block has to moved such that it reaches the goal state, the solution space can consist of N! possible exchanges.

3 Start state and Goal state

The start states and goal states will be given in the input file. However, such an example is given in the problem statement and sample I/O.
An example of Start State: is as below: Initial State is:
E B F
D A
C
Correspondingly, the Goal State: is as below:
Goal State is:
A D B
E F C

4 Pseudo code for Movegen()

```
def movegen(current state):
    for each stack in current state :
        elt = stack.pop()
            for remaining stack:
                state = stack.push(elt)
            if state not in CLOSED:
                Neighbors.append(state)
        return Neighbors
```

5 Pseudo Code for Goaltest()

```
def goaltest(current state):
    for each stack in current state:
        if(current state[elt] != goal state[elt]):
            return False
    return True
```

6 Heuristic Functions

6.0.1 Heuristic 1

In this heuristic, we consider +1 for correctly placed block and -1 for wrongly placed.

```
def heuristic1(current state):
  h_val = 0
  for each element in current state:
    if(position in current state == position in goal state):
      h_val += 1
    else:
      h_val -= 1
  return h_val
```

6.0.2 Heuristic 2

In the second one, we take positive height for correctly placed blocks and -height for misplaced blocks.

```
def heuristic2(current state):
  h_val = 0
  for each element in current state:
    if(height in current state == height in goal state):
      h_val += height
    else:
      h_val -= height
  return h_val
```

6.0.3 Heuristic 3

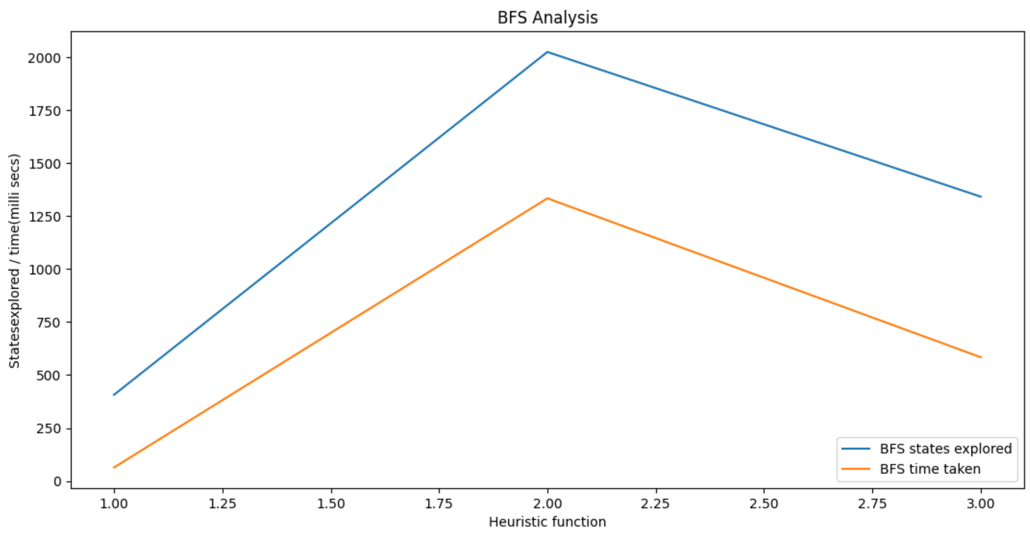
In the third one, we take manhatten distance for each block between current state and goal state.

```
def heuristic3(current state):
  h_val = 0
  for each element in current state:
    manhatten_dis = |goal_X - cur_X|+ |goal_Y - cur_Y|
      h_val += manhatten_dis
  return h_val
```

7 Best First Search analysis

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.

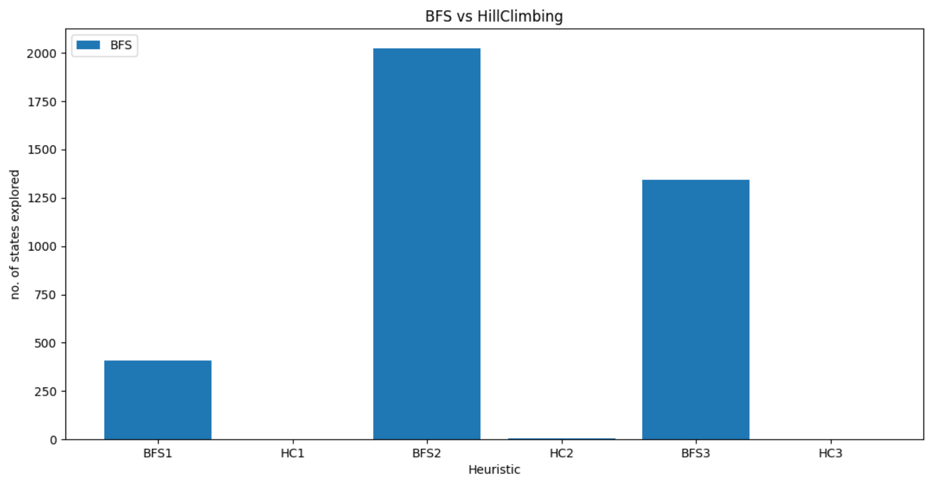
Below graph shows the Best First Search's States explored and time taken in milli seconds for each heuristic



8 Comparision Between BFS and Hill Climbing

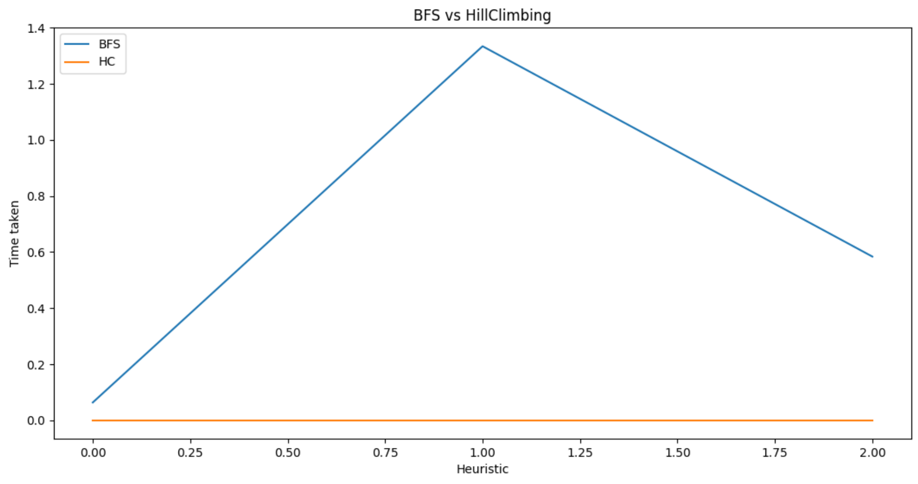
States explored

States explored in HillClimbing is very less since it attains local extremum after some time.



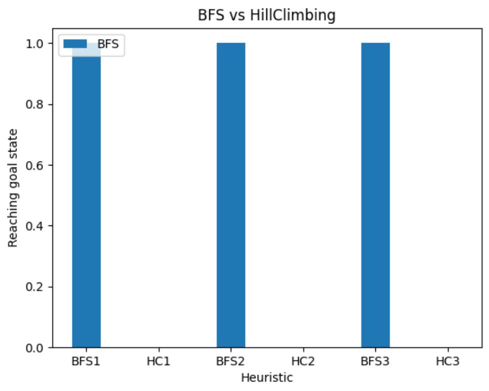
Time taken

Similar to states, time taken is also less in HillClimbing.Both time and states explored in BFS is dependend on Heuristic function.



Reaching the optimal state

Since BFS explores all N! states, goal state is guaranteed there. On the other-hand, goal state may or may not be reached in HillClimbing as it may stuck in local extremum.



In the above graph, reahing goal state means 1 and not reaching means 0.

9 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