

# Artificial Intelligence Lab Assignment 2

Group Number: 27

Pavan Kumar V Patil 200030041

Karthik J Ponarkar 200010022

January 2022

## Contents

<b>1</b>	<b>Brief description about the domain</b>	<b>3</b>
1.1	State space . . . . .	3
1.2	Start node and goal node . . . . .	3
<b>2</b>	<b>Pseudocode</b>	<b>4</b>
2.1	Goal test function . . . . .	4
2.2	MoveGen for best first search . . . . .	4
2.3	MoveGen for Hill climbing algorithm . . . . .	4
<b>3</b>	<b>Heuristic functions</b>	<b>5</b>
3.1	Heuristic 1 . . . . .	5
3.2	Heuristic 2 . . . . .	5
3.3	Heuristic 3 . . . . .	5
<b>4</b>	<b>BFS analysis and observation</b>	<b>6</b>
4.1	Analysis . . . . .	6
4.2	Observation . . . . .	6
<b>5</b>	<b>Comparison between BFS and Hill Climbing</b>	<b>7</b>
5.1	States explored . . . . .	7
5.2	Time Taken . . . . .	7
5.3	Reaching the optimal solution . . . . .	7

# 1 Brief description about the domain

## 1.1 State space

- According to the given question, each state consists of maximum 6 neighbours.
- State space  $S$  is a set which consists of all possible configuration of the boxes in the set of 3 stacks.

## 1.2 Start node and goal node

- Start node and goal node is fetched from the given input file named "input.txt".
- Start node is the initial state where, BFS or Hill Climbing algorithm is applied. Goal node is the final state which should be obtained using these algorithms from the start state.
- Goal node is useful to find the heuristic value of every state we visit.
- Goal node is also useful in the goal test function which checks whether the current is a final solution state or not.

## 2 Psuedocode

---

### 2.1 Goal test function

```
function GOALTEST(node)
  if node.stacks == goal.stacks then
    return true
  end if
  return false
end function
```

---

### 2.2 MoveGen for best first search

```
function MOVEGEN(node)
  global openlist
  global closedlist
  for eachState in validNextStates(node) do
    if eachState.stacks  $\notin$  closedlist then
      eachState.hValue = heuristic(eachState)
      openlist.append(eachState)
    end if
  end for
end function
```

---

### 2.3 MoveGen for Hill climbing algorithm

```
function MOVEGEN(node)
  neighbours = []
  for eachState in validNextStates(node) do
    eachState.hValue = heuristic(eachState)
    neighbours.append(eachState)
  end for
  return neighbours
end function
```

---

## 3 Heuristic functions

### 3.1 Heuristic 1

- This function assigns a value of +1 to the boxes of the current stack which are present in their right position with respect to the goal stack's boxes and assigns -1 to the boxes which are not present in their right position with respect to the goal stacks.
- For a block to be at its right position, it should be present in the correct stack and height.
- The sum of assigned values to all the boxes is the heuristic value returned by this function for a state.
- Higher the heuristic value returned by this function for a state, more is priority to choose that node during the best first search.
- Using this heuristic function for the **bfs** leads to a maximization problem.

### 3.2 Heuristic 2

- This function assigns a value of +1 to the boxes of the current stack which have the same height with respect to the goal stack's boxes and assigns -1 to the boxes which do not have the same height with respect to the goal stack's boxes.
- The sum of assigned values to all the boxes is the heuristic value returned by this function for a state.
- Higher the heuristic value returned by this function for a state, more is priority to choose that node during the best first search.
- Using this heuristic function for the **bfs** leads to a maximization problem.

### 3.3 Heuristic 3

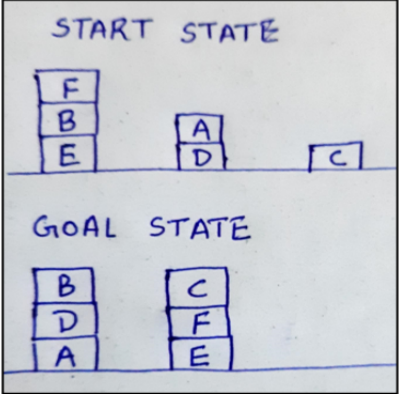
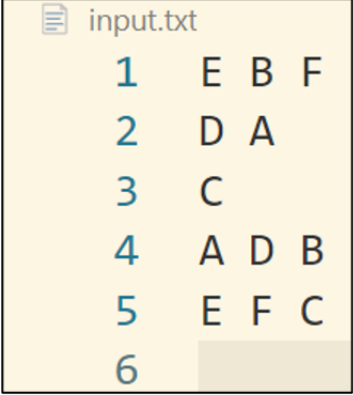
- This function assigns a value of +1 to the boxes of the current stack which are present in the same stack with respect to the goal stack's boxes and assigns -1 to the boxes which are not present in the same stack with respect to the goal stack's boxes.
- The sum of assigned values to all the boxes is the heuristic value returned by this function for a state.
- Higher the heuristic value returned by this function for a state, more is priority to choose that node during the best first search.
- Using this heuristic function for the **bfs** leads to a maximization problem.

## 4 BFS analysis and observation

### 4.1 Analysis

- In this algorithm, we begin with the start state. In the beginning, start state is the current state where each current state consist of maximum 6 neighbours.
- We declare two global lists which are open and close, where open list is a priority queue based on heuristic value of the states which are not visited and closed list consists of set of states(stacks) which are already explored.
- In these neighbours, we add neighbours which are not visited to the open list and find the maximum or minimum heuristic value of a state according to different heuristic functions.
- We deque the maximum or minimum heuristic valued state from the open list, then explore its neighbouring states using moveGen() function and if it is not a goal state we append it to closed it. If it is our goal state, we print the result and return from the algorithm.

### 4.2 Observation

		
BFS algorithm	Number of States explored	Goal reached
Heuristic1()	703	YES
Heuristic2()	340	YES
Heuristic3()	1303	YES

## 5 Comparison between BFS and Hill Climbing

### 5.1 States explored

- States explored in general would be more for the BFS approach than the hill climbing approach.
- In worst case, the BFS algorithm will explore all the search space and Hill Climbing algorithm will at most explore  $O(bd)$ . Where  $b$  is the maximum beam width and  $d$  is the depth of search graph from the start state.

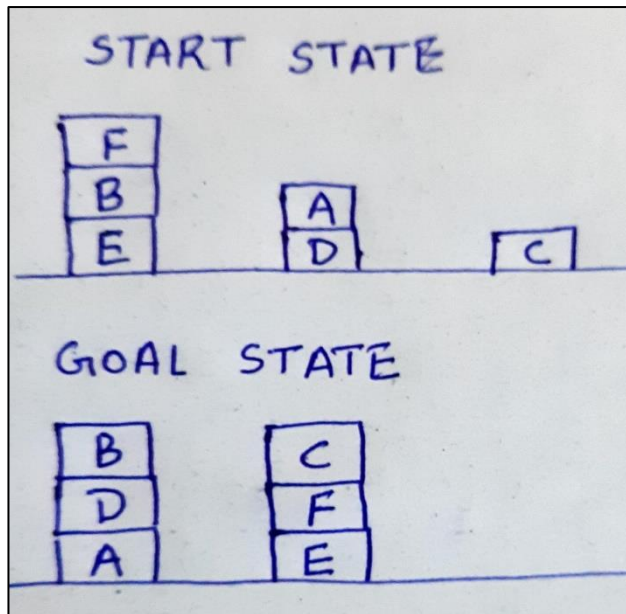
### 5.2 Time Taken

- Suppose  $b$  is the maximum beam width and  $d$  is depth of the search graph from the start state.
- In worst case, BFS algorithm will visit all the possible states which would result in a worst case time complexity of  $O(b^d)$
- In worst case, Hill Climbing algorithm will visit  $b$  new nodes at each level for maximum  $d$  times. Therefore, worst case time complexity of this algorithm would be  $O(bd)$
- Hence, we can conclude that Hill Climbing algorithm performs better in terms of time complexity as compared to BFS algorithm.

### 5.3 Reaching the optimal solution

- BFS is more optimal than Hill Climbing because BFS ensures the enclosure of the entire search space. This implies that if a solution state is present in the search space BFS will definitely find it.
- But in case of Hill Climbing, we approach the neighbouring states only when they seem to be more optimal than the current state. Mathematically, at least one of the neighbouring states should have higher heuristic value than the current state. If it is not the case, this function just returns current state because it is locally most optimal and has higher heuristic value than its neighbouring states.
- Hence, we conclude that Hill Climbing does not enclose the entire search space and BFS is more optimal in reaching the solution state.

# Result



input.txt

1	E	B	F
2	D	A	
3	C		
4	A	D	B
5	E	F	C
6			

## Best First Search

Heuristic1

```
=>> Goal reached!  
=>> Number of states explored: 703  
=>> The final state is goal state:  
[['A', 'D', 'B'], ['E', 'F', 'C'], []]
```

Heuristic2

```
=>> Goal reached!  
=>> Number of states explored: 340  
=>> The final state is goal state:  
[['A', 'D', 'B'], ['E', 'F', 'C'], []]
```

Heuristic3

```
=>> Goal reached!  
=>> Number of states explored: 1303  
=>> The final state is goal state:  
[['A', 'D', 'B'], ['E', 'F', 'C'], []]
```

## Hill Climbing

Heuristic1

```
=>> Goal not reached!  
=>> Number of states explored: 2  
=>> The final state reached is:  
[['E', 'B', 'F'], ['D', 'A', 'C'], []]
```

Heuristic2

```
=>> Goal not reached!  
=>> Number of states explored: 2  
=>> The final state reached is:  
[['E', 'B', 'F'], ['D', 'A', 'C'], []]
```

Heuristic3

```
=>> Goal not reached!  
=>> Number of states explored: 3  
=>> The final state reached is:  
[['E', 'B'], ['D', 'A', 'C', 'F'], []]
```