CS312: Lab-2

By: Mayank Mittal *(190030026)*
Ayush Gupta *(190030007)*

Block World Domain Game:

In our program, we are always maintaining three stacks. We ask for the number of blocks as input from the user. The Starting state and Goal State are then generated randomly. The output is then printed in the *output.txt* file. It contains the visited states for each Heuristic functions of both, Best First Search and Hill Climbing.

Move Generation Function:

```
def nextGenfunctionHelper(current_list, index, finalState, WhichHeurestic):
    list_of_neibors = []
    temp_list1 = copy.deepcopy(current_list)
    temp_list2 = copy.deepcopy(current_list)
    last_number = current_list[index][len(current_list[index]) - 1]
    temp_list1[index].pop()
    temp_list2[index].pop()
    temp_list1[(index+1) % 3].append(last_number)
    temp_list2[(index+2) % 3].append(last_number)
    val1 = CallWhichHeuristic(temp_list1, finalState, WhichHeurestic)
    val2 = CallWhichHeuristic(temp_list2, finalState, WhichHeurestic)
    list_of_neibors.append([val1, temp_list1])
    list_of_neibors.append([val2, temp_list2])
    return list_of_neibors


def nextGenfunction(current_list, finalState, WhichHeurestic):
    list_of_neibors = []
    if(len(current_list[0])):
        list_of_neibors += nextGenfunctionHelper(
            current_list, 0, finalState, WhichHeurestic)
    if(len(current_list[1])):
        list_of_neibors += nextGenfunctionHelper(
            current_list, 1, finalState, WhichHeurestic)
    if(len(current_list[2])):
        list_of_neibors += nextGenfunctionHelper(
            current_list, 2, finalState, WhichHeurestic)
    return list_of_neibors
```

As we can see from the above image, we take the last element from every one of the 3 stacks, and then add to one of the other two stacks to create neighbors. This acts as our Move generation function.

Goal Test:

Whenever the current state is the same as the final state, we finish the search. This condition is explicitly mentioned in both, the Best First Search and Hill Climb Search.

```
def BFS(startState, finalState, WhichHeurestic):

    queu = PriorityQueue()
    heuristic_value = CallWhichHeuristic(
        startState, finalState, WhichHeurestic)

    queu.put((heuristic_value, startState))
    visited = []
    visited.append(startState)
    length_path = 0

    while(not queu.empty()):
        value_at_top = queu.get()
        length_path += 1
        printState(value_at_top[1], length_path)

        if(value_at_top[1] == finalState):
            print("Number of states explored are " + str(length_path))
            return True

        next_Gen_neighbours = nextGenfunction(
            value_at_top[1], finalState, WhichHeurestic)

        for neighbours in next_Gen_neighbours:
            if neighbours[1] not in visited:
                queu.put((neighbours[0], neighbours[1]))
                visited.append(neighbours[1])
    return False
```

```
def HillClimbing(startState, finalState, WhichHeurestic):
    visited = []
    visited.append(startState)
    length_path = 0
    checkValue = CallWhichHeuristic(startState, finalState, WhichHeurestic)

    while(True):
        length_path += 1
        printState(startState, length_path)
        if(startState == finalState):
            print("Number of states explored are " + str(length_path))
            return True

        next_Gen_neighbours = nextGenfunction(
            startState, finalState, WhichHeurestic)

        Indicator = False
        for neighbours in next_Gen_neighbours:
            if neighbours[1] not in visited:
                visited.append(neighbours[1])
                if neighbours[0] <= checkValue:
                    Indicator = True
                    checkValue = neighbours[0]
                    startState = copy.deepcopy(neighbours[1])

        if(Indicator == False):
            print("Stuck at local maxima")
            print("There is no solution for Hill Climbing approach with this Heuristic and number of states explored are"
                  " " + str(length_path))
            return False
```

Heuristic Functions:

We have selected three Heuristic functions in our program. Since we are using min-heap Priority Queue in Python, we are returning the negative value for all the Heuristic functions:

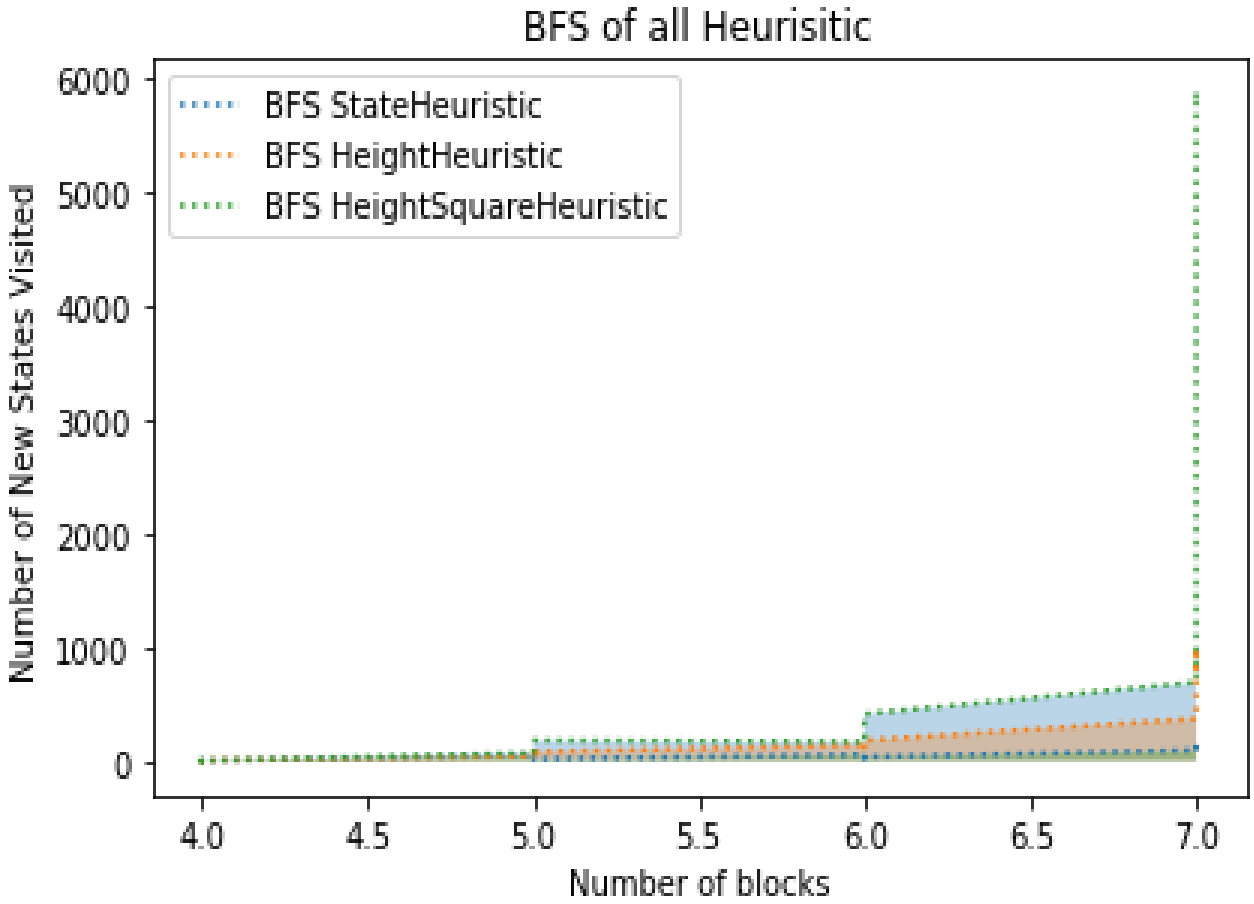
1) **StateHeuristic():** In this Heuristic function, we increment the cost by 1 if the relative position of the block with the goal state is the same, else decrement it by 1.
2) **HeightHeuristic():** In this Heuristic function, we increment the cost by the height if the relative position of the block with the goal state is the same, else decrement it by that same height.
3) **HeightSquareHeuristic():** This is the same as *HeightHeuristic()* except we increment/decrement by the square of the height.

Observation Table:

In the following table, *H1* denotes StateHeuristic, *H2* denotes HeightHeuristic, *H3* denoted HeightSquaredHeuristic, and *bfs* is for Best First Search and *hc* means Hill Climbing.

T denotes the search was successful, and *F* denotes the search was unsuccessful.

Blocks	***bfs H1***		***bfs H2***		***bfs H3***		***hc H1***		***hc H2***		***hc H3***	
4	*T*	15	*T*	24	*T*	34	*F*	4	*F*	3	*F*	3
4	*T*	2	*T*	2	*T*	2	*T*	2	*T*	2	*T*	2
4	*T*	8	*T*	10	*T*	9	*F*	2	*F*	4	*F*	4
5	*T*	57	*T*	51	*T*	78	*T*	9	*F*	6	*F*	6
5	*T*	25	*T*	122	*T*	148	*F*	3	*F*	6	*F*	6
5	*T*	23	*T*	87	*T*	188	*F*	3	*F*	3	*F*	3
6	*T*	61	*T*	147	*T*	176	*T*	13	*F*	3	*F*	3
6	*T*	34	*T*	108	*T*	173	*F*	9	*F*	1	*F*	1
6	*T*	42	*T*	187	*T*	418	*F*	6	*F*	2	*F*	2
7	*T*	101	*T*	376	*T*	698	*F*	4	*F*	4	*F*	4
7	*T*	178	*T*	1001	*T*	5871	*F*	7	*F*	2	*F*	2
7	*T*	106	*T*	765	*T*	5790	*F*	7	*F*	3	*F*	3



Analysis:

- Best First Search always finds the solution. It also visits all the neighboring states. Hence, it's **complete and optimal**.
- Hill Climbing does not always find the solution. Since it takes the most optimal neighbor and ignores the rest of the neighbors, it is also called Greedy Search. Hence, it's **incomplete and non-optimal.**
- If there are total N possible states (or b^d states) from the start state to the end state, then **time complexity** for:

1) Best First Search: *O(N*log(N)) log(N)* factor comes from the use of priority queue.
2) Hill Climbing: *O(∞)* As the search may not reach the result

- **Space Complexity** for
    1. Best First Search: *O(N)* where *N* is the number of all states
    2. Hill Climbing: Depth/Height of the tree formed from the starting state to the ending state.
- Best First Search takes a lot more time compared to Hill Climbing, as it visits all the neighboring states whereas Hill Climbing finishes early, with either the search failing, or it completes, time taken being the depth of the tree of all the states.
- Lastly, the performance of both the searches are highly dependent on the Heuristic Functions.
    - ➔ The *StateHeuristic()* function performed moderately well, it even completed the search in Hill Climbing for a few cases. And the number of states didn't increase rapidly as the number of blocks increased.
    - ➔ The *HeightHeuristic()* function performed worse than StateHeuristic but better than StateSquareHeuristic. It happened because the penalty we took for a block in the wrong stack was not optimal. We could have improved this Heuristic function
    - ➔ The *HeightSquareHeuristic()* function performed the worse, partially because the penalty in *HeightHeuristic()* was squared, thus increasing the possible non-optimal states to come.