Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

Jhanvi Parekh

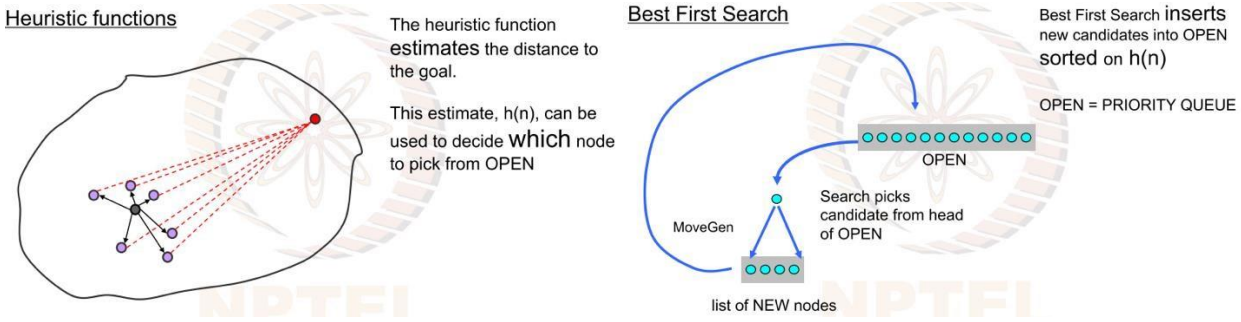
60009210033

D11

Experiment 3 (Heuristic Search)

Aim: Comparative analysis of Heuristic based methods.

Theory:



Algorithm for Best First Search

```
Best-First-Search(S) 1
OPEN ← (S, null, h(S)) []
2 CLOSED ← empty list
3 while OPEN is not empty
4 nodePair ← head OPEN 5 (N, , ) ← nodePair
6   if GoalTest(N) = true
7   return ReconstructPath(nodePair, CLOSED)
8   else CLOSED ← nodePair CLOSED
9   neighbours ← MoveGen(N)
10 newNodes ← RemoveSeen(neighbours, OPEN, CLOSED)
11 newPairs ← MakePairs(newNodes, N) 12 OPEN ← sorth( newPairs ++ tail OPEN )
13 return empty list
```

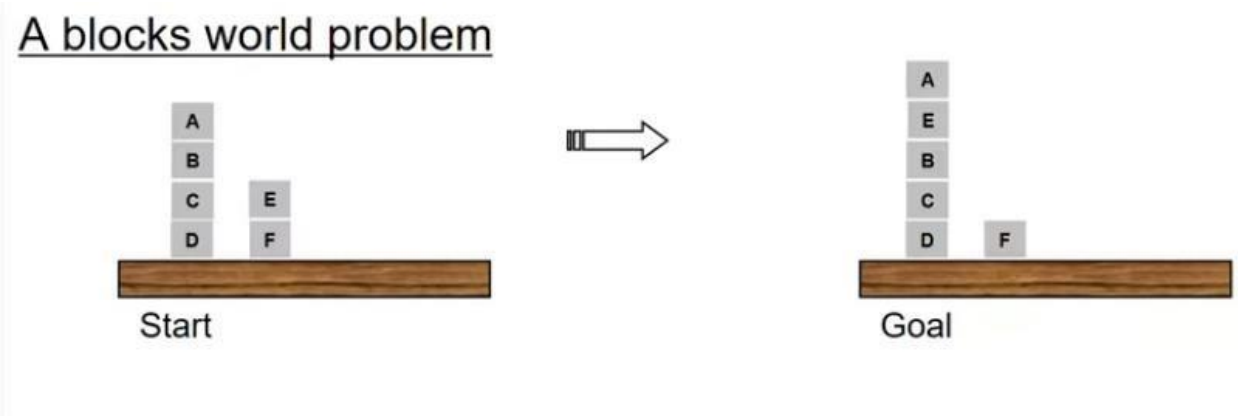
Algorithm Hill climbing

Hill-Climbing(S)
1 N ← S
2 do bestEver ← N
3 N ← head sorth MoveGen(bestEver)
4 while h(N) is better than h(bestEver)

5 return bestEver

Lab Assignment to do:

1. Design any two different heuristics for a given blocks world problem and show that one is better than another using Hill Climbing and Best First Search.



Link: https://colab.research.google.com/drive/1UZOWppz-NUNIKQ5zVoIbwhMLYGv5AKO0?usp=sharing

```
def generate_blocks_world_moves(state):
    def move(state, from_stack, to_stack):
        if not state[from_stack]:
            return None
        block_to_move = state[from_stack][-1]
        if not state[to_stack] or block_to_move < state[to_stack][-1] or block_to_move > state[to_stack][-1]:
            new_state = [stack[:] for stack in state]
            new_state[to_stack].append(new_state[from_stack].pop())
            return new_state
        else:
            return None
    moves = []
    num_stacks = len(state)
    for from_stack in range(num_stacks):
        for to_stack in range(num_stacks):
            if from_stack != to_stack:
                new_state = move(state, from_stack, to_stack)
                if new_state:
                    moves.append((from_stack, to_stack, new_state))
    return moves


initial_state = [['D', 'C','B','A'], ['F','E'], []]
moves = generate_blocks_world_moves(initial_state)


for move in moves:
    from_stack, to_stack, new_state = move
    print(f"Move {initial_state[from_stack][-1]} from stack {from_stack} to stack {to_stack}:")
    print(new_state)
```

```
Move A from stack 0 to stack 1:
[['D', 'C', 'B'], ['F', 'E', 'A'], []]
Move A from stack 0 to stack 2:
[['D', 'C', 'B'], ['F', 'E'], ['A']]
Move E from stack 1 to stack 0:
[['D', 'C', 'B', 'A', 'E'], ['F'], []]
Move E from stack 1 to stack 2:
[['D', 'C', 'B', 'A'], ['F'], ['E']]
```

```
import copy


def movegen(curr_state):
    global closed, open_list
    state = copy.deepcopy(curr_state)
    neighbors = []
    for i in range(len(state)):
        temp = copy.deepcopy(state)
        if len(temp[i]) > 0:
            elem = temp[i].pop()
            for j in range(len(temp)):
                temp1 = copy.deepcopy(temp)
                if j != i:
                    temp1[j] = temp1[j] + [elem]
                    if (temp1 not in closed and temp1 not in open_list):
                        neighbors.append(temp1)
    return neighbors



# Example state: [[3, 2, 1], [], []]
curr_state = [["d","c","b","a"], ["f","e"],[]]
closed = []
open_list = []
neighbors = movegen(curr_state)
print(neighbors)
```

```
[[['d', 'c', 'b'], ['f', 'e', 'a'], []], [['d', 'c', 'b'], ['f', 'e'], ['a']], [['d', 'c', 'b', 'a', 'e'], ['f'], []], [['d', 'c', 'b',
```

```
#Heuristic 1 : considering position of blocks
def heuristic1(curr_state):
    global goal_state, n_goal
    h_val = 0
```

```
    d_cur = dict((j,(x, y)) for x, i in enumerate(cur) for y, j in enumerate(i))
    d_goal = {
    "a": (0,4),  #Block 1 is in stack 0, position 0
    "b": (0,2),
    "c": (0,1),
    "d": (0,0),
    "e": (0,3),
    "f": (1,0)
}

    for i in range(3):
        for j in range(len(cur[i])):
            curx, cury = d_cur[cur[i][j]]
            goalx, goaly = d_goal[cur[i][j]]
            if( goaly == cury and goalx == curx):
                h_val += 1
            else:
                h_val -= 1
    return h_val


#Heuristic 2 : Considering heights of block
def heuristic2(curr_state):
    global goal_state, d_goal
    h_val = 0
    cur = copy.deepcopy(curr_state)
    d_cur = dict((j,(x, y)) for x, i in enumerate(cur) for y, j in enumerate(i))
    for i in range(3):
        for j in range(len(cur[i])):
            curx, cury = d_cur[cur[i][j]]
            goalx, goaly = d_goal[cur[i][j]]
            if( goaly == cury):
                h_val += (cury+1)
            else:
                h_val -=(cury+1)

    return h_val



#Heuristic 3 : Manhatten distance between start and goal state
def heuristic3(curr_state):
    global goal_state, d_goal
    h_val = 0
    cur = copy.deepcopy(curr_state)
    d_cur = dict((j,(x, y)) for x, i in enumerate(cur) for y, j in enumerate(i))
    for i in range(3):
        for j in range(len(cur[i])):
            curx, cury = d_cur[cur[i][j]]
            goalx, goaly = d_goal[cur[i][j]]
            h_val += (abs(curx-goalx) + abs(cury-goaly))
    return h_val


#Function to find if goal state is reached or not
def goaltest(cur_state):
    global goal_state
    for i in range(3):
        if(len(goal_state[i])!=len(cur_state[i])):
            return False
        for j in range(len(goal_state[i])):
            if(goal_state[i][j]!=cur_state[i][j]):
                return False
    return True


# Assuming goal_state and d_goal are defined elsewhere
curr_state = [["d","c","b","a"], ["f","e"], []]

# Example usage
h1_value = heuristic1(curr_state)
h2_value = heuristic2(curr_state)
h3_value = heuristic3(curr_state)

print(f"Heuristic 1 Value: {h1_value}")
print(f"Heuristic 2 Value: {h2_value}")
print(f"Heuristic 3 Value: {h3_value}")
```

```
Heuristic 1 Value: 2
Heuristic 2 Value: 1
Heuristic 3 Value: 4
```

BEST FIRST SEARCH

```
from operator import itemgetter
import sys


# Define your start_state and goal_state here
start_state = [["d", "c", "b", "a"], ["f", "e"], []]
goal_state = [[], [], ["a", "b", "c", "d", "e", "f"]]


# Define movegen function
def movegen(curr_state):
    global closed, open_list
    state = copy.deepcopy(curr_state)
    neighbors = []
    for i in range(len(state)):
        temp = copy.deepcopy(state)
        if len(temp[i]) > 0:
            elem = temp[i].pop()
            for j in range(len(temp)):
                temp1 = copy.deepcopy(temp)
                if j != i:
                    temp1[j] = temp1[j] + [elem]
                    if (temp1 not in closed and temp1 not in open_list):
                        neighbors.append(temp1)
    return neighbors


# Define goaltest function
def goaltest(cur_state):
    global goal_state
    for i in range(3):
        if(len(goal_state[i]) != len(cur_state[i])):
            return False
        for j in range(len(goal_state[i])):
            if(goal_state[i][j] != cur_state[i][j]):
                return False
    return True
```

```
# Define heuristic1 function
def heuristic1(curr_state):
    global goal_state
    h_val = 0
    cur = copy.deepcopy(curr_state)
    d_cur = dict((j, (x, y)) for x, i in enumerate(cur) for y, j in enumerate(i))
    d_goal = {
        "a": (0,4),  #Block 1 is in stack 0, position 0
        "b": (0,2),
        "c": (0,1),
        "d": (0,0),
        "e": (0,3),
        "f": (1,0)
    }
    for i in range(3):
        for j in range(len(cur[i])):
            curx, cury = d_cur[cur[i][j]]
            goalx, goaly = d_goal[cur[i][j]]
            if goaly == cury and goalx == curx:
                h_val += 1
            else:
                h_val -= 1
    return h_val


# Define heuristic2 function
def heuristic2(curr_state):
    h_val = 0
    cur = copy.deepcopy(curr_state)
    d_cur = dict((j, (x, y)) for x, i in enumerate(cur) for y, j in enumerate(i))
    for i in range(3):
```

```
        for j in range(len(cur[i])):
            curx, cury = d_cur[cur[i][j]]
            h_val += cury + 1
    return h_val


# Define heuristic3 function
def heuristic3(curr_state):
    h_val = 0
    cur = copy.deepcopy(curr_state)
    d_cur = dict((j, (x, y)) for x, i in enumerate(cur) for y, j in enumerate(i))
    d_goal = {
        "a": (0,4),  #Block 1 is in stack 0, position 0
        "b": (0,2),
        "c": (0,1),
        "d": (0,0),
        "e": (0,3),
        "f": (1,0)
    }
    for i in range(3):
        for j in range(len(cur[i])):
            curx, cury = d_cur[cur[i][j]]
            goalx, goaly = d_goal[cur[i][j]]
            h_val += abs(curx - goalx) + abs(cury - goaly)
    return h_val


# Define the BFS functions
def bfs1():
    global closed, open_list, heap, start_state, goal_state
    open_list = []  # Define open_list within this function
    heap = []      # Define heap within this function
    closed = []

    current_state = copy.deepcopy(start_state)
    open_list.append(copy.deepcopy(start_state))

    while True:
        closed.append(copy.deepcopy(current_state))
        if goaltest(current_state):
            return "Goal state reached"
        open_list.remove(current_state)
        prev_heu = heuristic1(current_state)
        neighbors = movegen(current_state)
        for i in neighbors:
            open_list.append(i)
            heap.append([i, heuristic1(i)])
        list = [current_state, prev_heu]
        if list in heap:
            heap.remove(list)
        if len(open_list) == 0:
            return "Goal state can't be reached"
        current_heap = copy.deepcopy(max(heap, key=itemgetter(1)))
        current_state = current_heap[0]


def bfs2():
    global closed, open_list, heap, start_state, goal_state
    open_list = []  # Define open_list within this function
    heap = []      # Define heap within this function
    closed = []

    current_state = copy.deepcopy(start_state)
    open_list.append(copy.deepcopy(start_state))

    while True:
        closed.append(copy.deepcopy(current_state))
        if goaltest(current_state):
            return "Goal state reached"
        open_list.remove(current_state)
        prev_heu = heuristic2(current_state)
        neighbors = movegen(current_state)
        for i in neighbors:
            open_list.append(i)
            heap.append([i, heuristic2(i)])
        list = [current_state, prev_heu]
```

```
            heap.remove(list)
        if len(open_list) == 0:
            return "Goal state can't be reached"
        current_heap = copy.deepcopy(max(heap, key=itemgetter(1)))
        current_state = current_heap[0]


def bfs3():
    global closed, open_list, heap, start_state, goal_state
    open_list = []  # Define open_list within this function
    heap = []      # Define heap within this function
    closed = []

    current_state = copy.deepcopy(start_state)
    open_list.append(copy.deepcopy(start_state))

    while True:
        closed.append(copy.deepcopy(current_state))
        if goaltest(current_state):
            return "Goal state reached"
        open_list.remove(current_state)
        prev_heu = heuristic3(current_state)
        neighbors = movegen(current_state)
        for i in neighbors:
            open_list.append(i)
            heap.append([i, heuristic3(i)])
        list = [current_state, prev_heu]
        if list in heap:
            heap.remove(list)
        if len(open_list) == 0:
            return "Goal state can't be reached"
        current_heap = copy.deepcopy(min(heap, key=itemgetter(1)))
        current_state = current_heap[0]


result1 = bfs1()
result2 = bfs2()
result3 = bfs3()

print("Result of BFS1:", result1)
print("Result of BFS2:", result2)
print("Result of BFS3:", result3)
```

```
Result of BFS1: Goal state reached
Result of BFS2: Goal state reached
Result of BFS3: Goal state reached
```

HILL CLIMBING

```
def hillClimbing1():
    global closed, open_list, heap, start_state, goal_state
    current_state = copy.deepcopy(start_state)
    open_list.append(copy.deepcopy(start_state))
    while(True):
        closed.append(copy.deepcopy(current_state))
        if(goaltest(current_state)):
            f_out.write("Goal state reached\n\n")
            return current_state
        prev_heu = heuristic1(current_state)
        neighbors = movegen(current_state)
        for i in neighbors:
            h = heuristic1(i)
            heap.append([i,h])

        current_heap = copy.deepcopy(max(heap,key=itemgetter(1)))
        if(current_heap[1] <= prev_heu):
            f_out.write("Goal state can't be reached\n\n")
            return current_state

        current_state = current_heap[0]
        heap = []

def hillClimbing2():
    global closed, open_list, heap, start_state, goal_state
    current_state = copy.deepcopy(start_state)
```

```
    while(True):
        closed.append(copy.deepcopy(current_state))
        if(goaltest(current_state)):
            f_out.write("Goal state reached\n\n")
            return current_state
        prev_heu = heuristic2(current_state)
        neighbors = movegen(current_state)
        for i in neighbors:
            h = heuristic2(i)
            heap.append([i,h])

        current_heap = copy.deepcopy(max(heap,key=itemgetter(1)))
        if(current_heap[1] <= prev_heu):
            f_out.write("Goal state can't be reached\n\n")
            return current_state

        current_state = current_heap[0]
        heap = []


def hillClimbing3():
    global closed, open_list, heap, start_state, goal_state
    current_state = copy.deepcopy(start_state)
    open_list.append(copy.deepcopy(start_state))
    while(True):
        closed.append(copy.deepcopy(current_state))
        if(goaltest(current_state)):
            f_out.write("Goal state reached\n\n")
            return current_state
        prev_heu = heuristic3(current_state)
        neighbors = movegen(current_state)
        for i in neighbors:
            h = heuristic3(i)
            heap.append([i,h])

        current_heap = copy.deepcopy(min(heap,key=itemgetter(1)))
        if(current_heap[1] >= prev_heu):
            f_out.write("Goal state can't be reached\n\n")
            return current_state

        current_state = current_heap[0]
        heap = []
```

```
import copy
from operator import itemgetter


# Define your movegen, goaltest, heuristic1, heuristic2, and heuristic3 functions here.
def movegen(curr_state):
    state = copy.deepcopy(curr_state)
    neighbors = []
    for i in range(len(state)):
        temp = copy.deepcopy(state)
        if len(temp[i]) > 0:
            elem = temp[i].pop()
            for j in range(len(temp)):
                temp1 = copy.deepcopy(temp)
                if j != i:
                    temp1[j] = temp1[j] + [elem]
                    neighbors.append(temp1)
    return neighbors


def goaltest(cur_state, goal_state):
    for i in range(3):
        if len(goal_state[i]) != len(cur_state[i]):
            return False
        for j in range(len(goal_state[i])):
            if goal_state[i][j] != cur_state[i][j]:
                return False
    return True


def heuristic1(curr_state):
    h_val = 0
    cur = copy.deepcopy(curr_state)
```

```
    for i in range(3):
        for j in range(len(cur[i])):
            if j < len(goal_state[i]) and goal_state[i][j] == cur[i][j]:
                h_val += 1
    return h_val


def heuristic2(curr_state):
    h_val = 0
    cur = copy.deepcopy(curr_state)
    for i in range(3):
        for j in range(len(cur[i])):
            if j < len(goal_state[i]) and goal_state[i][j] == cur[i][j]:
                h_val += 1
    return h_val



def heuristic3(curr_state):
    h_val = 0
    cur = copy.deepcopy(curr_state)
    d_cur = dict((j, (x, y)) for x, i in enumerate(cur) for y, j in enumerate(i))
    for i in range(3):
        for j in range(len(cur[i])):
            curx, cury = d_cur[cur[i][j]]
            goalx, goaly = d_goal[cur[i][j]]
            h_val += (abs(curx - goalx) + abs(cury - goaly))
    return h_val


# Define your start_state and goal_state here
start_state = [["d", "c", "b", "a"], ["f", "e"], []]
goal_state = [[], [], ["a", "b", "c", "d", "e", "f"]]


# Implement the Hill Climbing algorithm
def hillClimbing(heuristic_func, start_state, goal_state):
    current_state = copy.deepcopy(start_state)
    while True:
        if goaltest(current_state, goal_state):
            print("Goal state reached")
            return current_state
        neighbors = movegen(current_state)
        heap = []
        for i in neighbors:
            h = heuristic_func(i)
            heap.append([i, h])
        current_heap = max(heap, key=itemgetter(1))
        current_state = current_heap[0]


# Example usage
print("Result of Hill Climbing 1:")
result_hill_climbing1 = hillClimbing(heuristic1, start_state, goal_state)

print("\nResult of Hill Climbing 2:")
result_hill_climbing2 = hillClimbing(heuristic2, start_state, goal_state)

print("\nResult of Hill Climbing 3:")
result_hill_climbing3 = hillClimbing(heuristic3, start_state, goal_state)
```

```
Result of Hill Climbing 1:
Goal state reached

Result of Hill Climbing 2:
Goal state reached

Result of Hill Climbing 3:
```