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

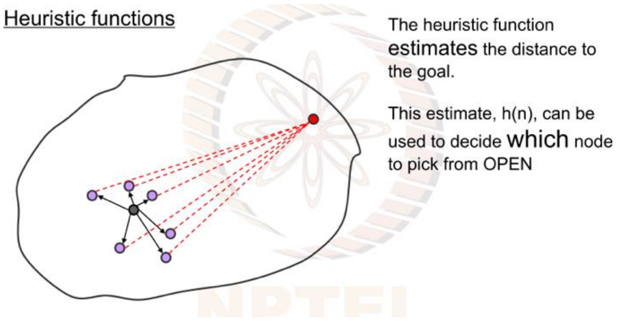
Experiment 3

SHIVAM NAGORI 60009210083 D12

(Heuristic Search)

Aim: Comparative analysis of Heuristic based methods.

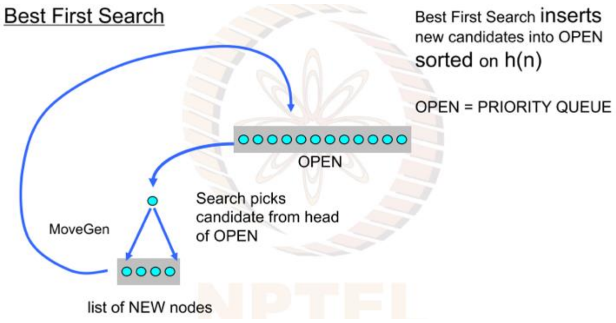
Theory:



Heuristic functions

The heuristic function estimates the distance to the goal.

This estimate, h(n), can be used to decide which node to pick from OPEN



Best First Search

Best First Search inserts new candidates into OPEN sorted on h(n)

OPEN = PRIORITY QUEUE

Algorithm for Best First Search

```
Best-First-Search(S)
OPEN ← (S, null, h(S)) []
CLOSED ← empty list
while OPEN is not empty
nodePair ← head OPEN
(N, , ) ← nodePair
if GoalTest(N) = true
return ReconstructPath(nodePair, CLOSED)
else CLOSED ← nodePair CLOSED
neighbours ← MoveGen(N)
newNodes ← RemoveSeen(neighbours, OPEN, CLOSED)
newPairs ← MakePairs(newNodes, N)
OPEN ← sorth( newPairs ++ tail OPEN )
return empty list
```

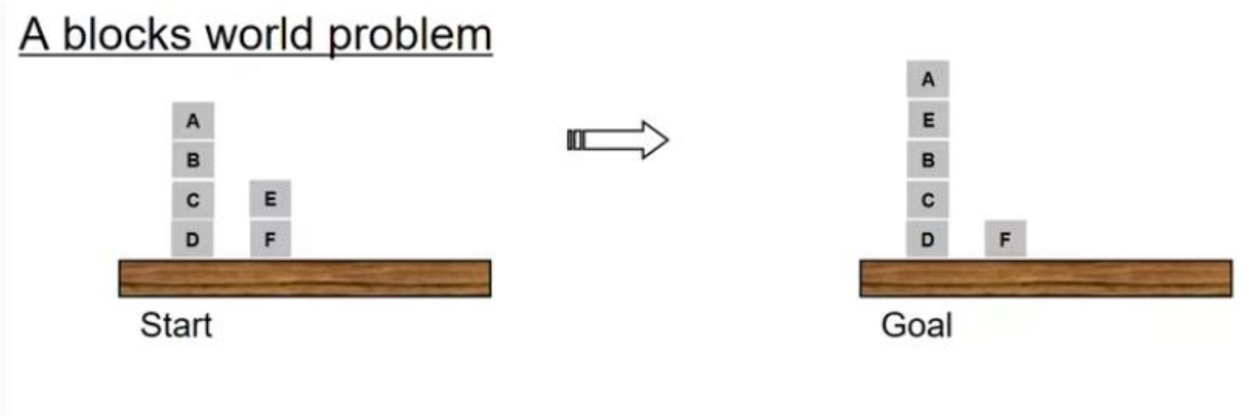
Algorithm Hill climbing

```
Hill-Climbing(S)
N ← S
```

```
do bestEver ← N
N ← head sorth MoveGen(bestEver)
while h(N) is better than h(bestEver)
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.



CODE:

```
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
```

```
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
```

```
#Heuristic 1 : considering position of blocks
def heuristic1(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))
    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
```

```
#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}")
```

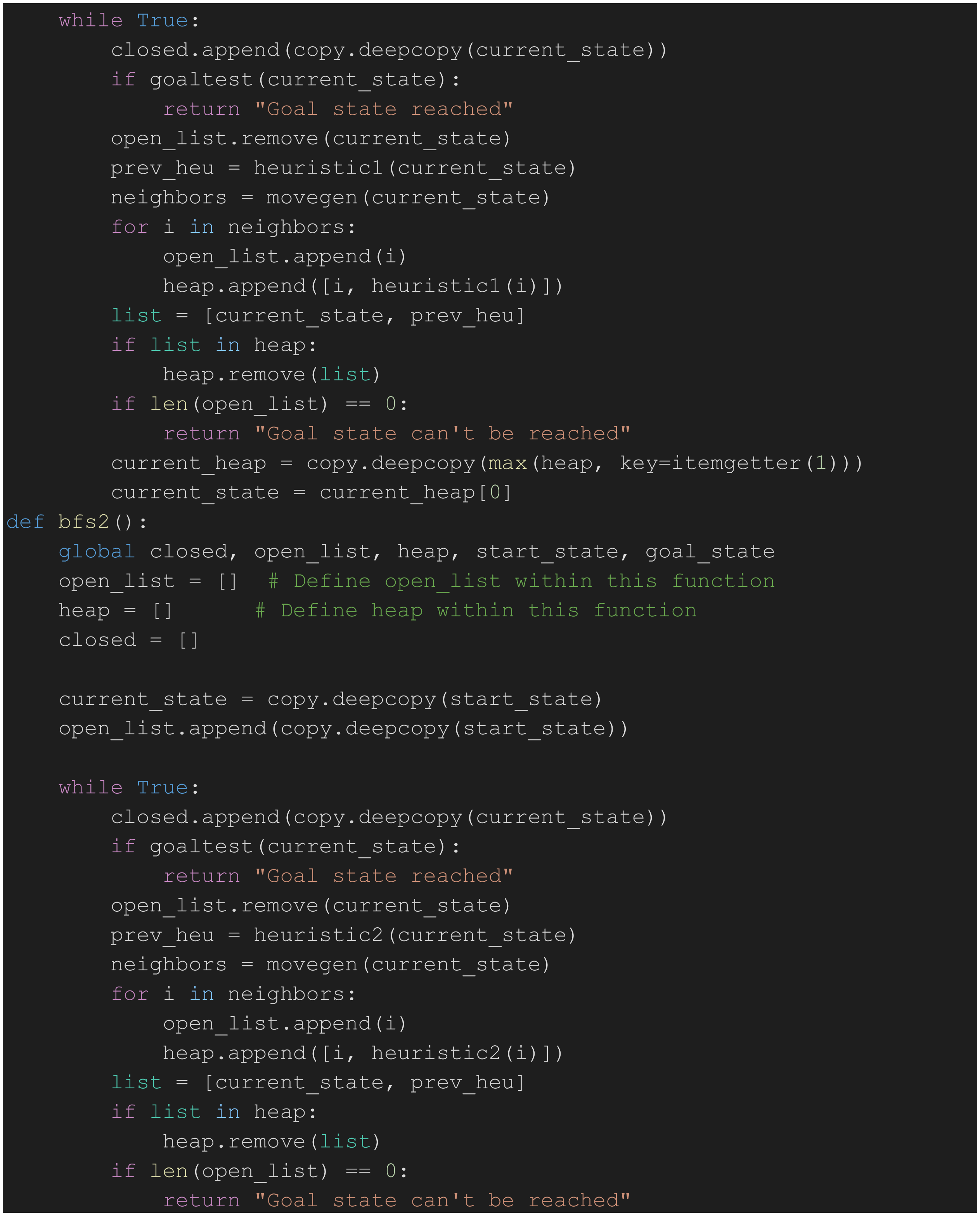
BFS:

```
# 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 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]
        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]
result1 = bfs1()
result2 = bfs2()

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

SOLN:

```
Result of BFS1: Goal state reached
Result of BFS2: 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)
    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 = 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 = []
```

```
# 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
```

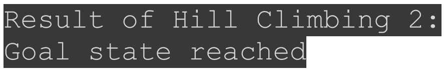
```
def heuristic1(curr_state):
    h_val = 0
    cur = copy.deepcopy(curr_state)
    for i in range(3):
        for j in range(len(cur[i])):
            if j < len(goal_state[i]) and goal_state[i][j] == cur[i][j]:
                h_val += 1
    return h_val
def heuristic2(curr_state):
    h_val = 0
    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

# 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]
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)
```

```
SOLUTION:
```

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



```
Result of Hill Climbing 2:
Goal state reached
```