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

Bhuvi Ghosh

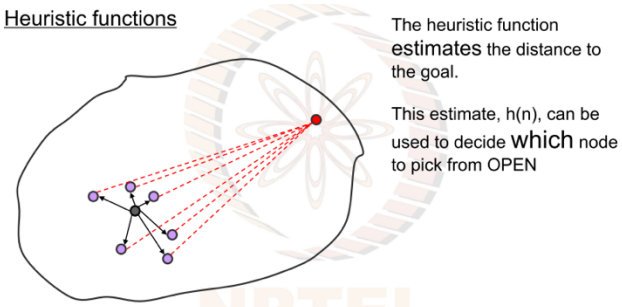
60009210191

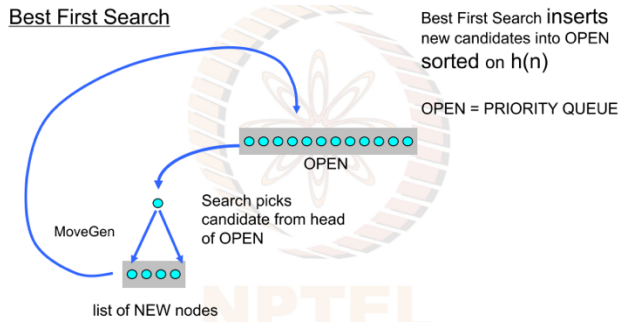
Experiment 3

(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)
```

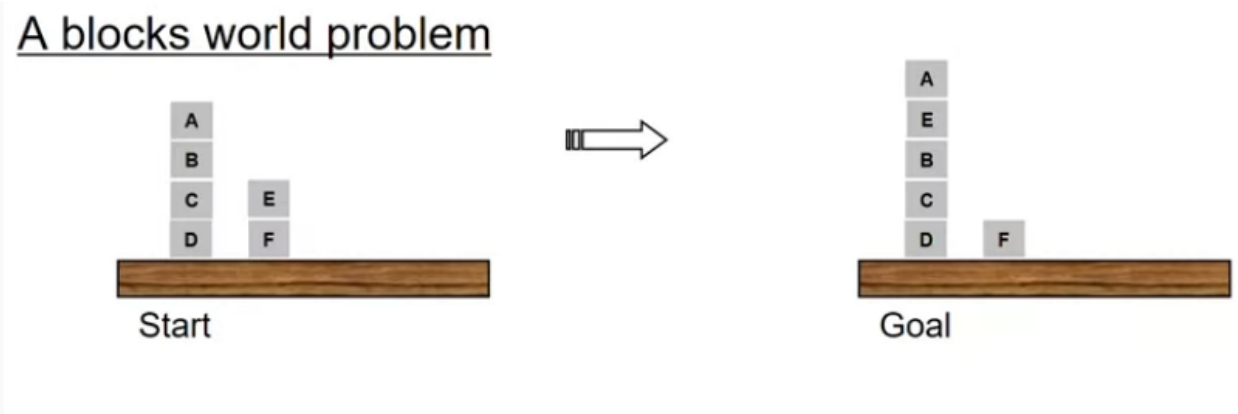
10 newNodes ← RemoveSeen(neighbours, OPEN, CLOSED)
11 newPairs ← MakePairs(newNodes, N)
12 OPEN ← sorth(newPairs ++ tail OPEN)
13 return empty list

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.



USING HEURISTIC 2:

```
def h2(state,goal):

    score = 0
    for i in range(len(goal)):
        for j in range(len(state[i])):
          if state[i][j] == goal[i][j]:
            score+=(j+1)
          else:
```

```
            score-=(j+1)

    return score




def generate_moves(start_state,goal):
    moves = []

    for i in range(3):
        if start_state[i]:
            for j in range(3):

                if i == j:

                    continue

                new_state = [stack.copy() for stack
in start_state]

                element = new_state[i].pop()
                new_state[j].append(element)

                h_score = h2(new_state,goal)

                moves.append((new_state,h_score))

    return moves
```

```
start_state = [['D', 'C', 'B', 'A'], ['F', 'E'],
[]]
goal_state = [['D', 'C', 'B', 'E', 'A'], ['F'], []]
from collections import Counter

flat_list = [item for stack in goal_state for item
in stack]

element_counts = Counter(flat_list)
sum_of_counts = sum(element_counts.values())

def reshape_goal(goal_state):
    max_len = sum_of_counts

    padded_goal_state = [stack + [''] * (max_len -
len(stack)) for stack in goal_state]

    return padded_goal_state

equalized_goal_state = reshape_goal(goal_state)

result =
generate_moves(start_state,equalized_goal_state)

result.sort(key=lambda x: x[1], reverse=True)

for state, score in result:
    print(f'State: {state}, H2 Score: {score}')
```

```
State: [['D', 'C', 'B'], ['F', 'E'], ['A']], H2 Score: 4
State: [['D', 'C', 'B'], ['F', 'E', 'A'], []], H2 Score: 2
State: [['D', 'C', 'B', 'A'], ['F'], ['E']], H2 Score: 2
State: [['D', 'C', 'B', 'A', 'E'], ['F'], []], H2 Score: -2
```

USING HEURISTIC 1:

```
def h1(state,goal):
  score = 0
  for i in range(len(goal)):
      for j in range(len(state[i])):
        if state[i][j] == goal[i][j]:
          score+=1
        else:
          score-=1

  return score

def generate_moves(start_state,goal):
    moves = []

    for i in range(3):
        if start_state[i]:
            for j in range(3):

                if i == j:
```

```
                    continue

                new_state = [stack.copy() for stack
in start_state]

                element = new_state[i].pop()
                new_state[j].append(element)

                h_score = h1(new_state,goal)

                moves.append((new_state,h_score))

    return moves

result =
generate_moves(start_state,equalized_goal_state)

result.sort(key=lambda x: x[1], reverse=True)

for state, score in result:
    print(f'State: {state}, H1 Score: {score}')
```

```
State: [['D', 'C', 'B'], ['F', 'E', 'A'], []], H1 Score: 2
State: [['D', 'C', 'B'], ['F', 'E'], ['A']], H1 Score: 2
State: [['D', 'C', 'B', 'A', 'E'], ['F'], []], H1 Score: 2
State: [['D', 'C', 'B', 'A'], ['F'], ['E']], H1 Score: 2
```