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 (DJS22DSC502)

AY: 2024-25

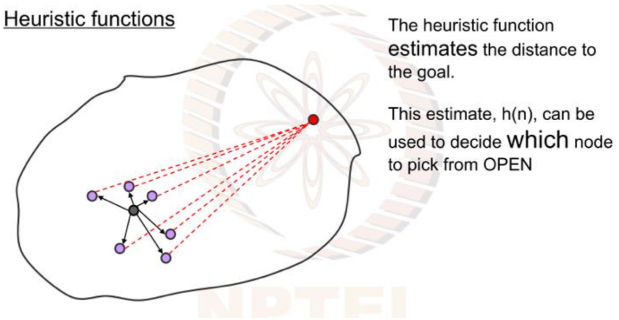
Experiment 3

(Heuristic Search)

DHRUV SHAH 60009220132 D11

Aim: Comparative analysis of Heuristic based methods.

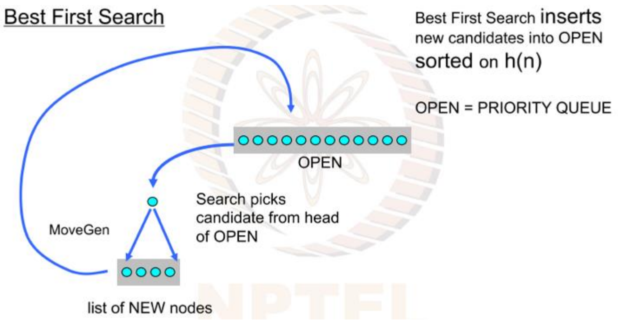
Theory:



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
```

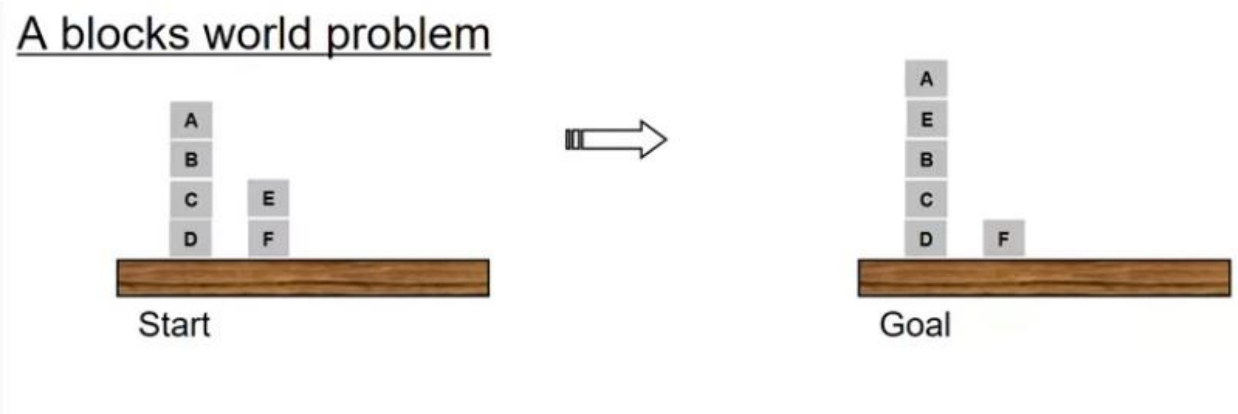
Algorithm Hill climbing

```
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.



```
from heapq import heapify,heappush,heappop


class BlockGame:
  def __init__(self,startState,goalState):
    self.startState = startState
    self.goalState = goalState

    self.goalPrevs = self.getGoalPrevs()

  def getGoalPrevs(self):
    goalPrevs = {}
    for i in range(len(self.goalState)):
      for j in range(len(self.goalState[i])):
        if j == 0:
          goalPrevs[self.goalState[i][j]] = '0'
        else:
          goalPrevs[self.goalState[i][j]] = self.goalState[i][j-1]
    return goalPrevs

  def moveGen(self,state):
    states = []
    for i in range(len(state)):
      for j in range(len(state)):
        if i != j:
          newState = []
          for k in range(len(state)):
            newState.append(state[k].copy())

          if len(newState[i]) > 0:
            newState[j].append(newState[i].pop())
            states.append(newState)
    return states

  def checkGoal(self,state):
    return state == self.goalState

  def h1(self,state):
    score = 0
    for i in range(len(state)):
      for j in range(min(len(state[i]),len(self.goalState[i]))):
        if state[i][j] != self.goalState[i][j]:
          score += 1
        else:
          score -=1
    return score

  def h2(self,state):
    cost = 0

    goal_positions = {}
    for i, tower in enumerate(self.goalState):
      for j, block in enumerate(tower):
          goal_positions[block] = (i, j)
    for i, tower in enumerate(state):
     for j, block in enumerate(tower):
      if block in goal_positions:
        goal_tower_index, goal_block_index = goal_positions[block]
        if i == goal_tower_index and j == goal_block_index:
          cost += j + 1
        else:
          cost -= j + 1

    return cost

  def hill_Climbing(self,state,currScore,heuristic='h2'):
    if self.checkGoal(state):
      return [state]
    ans = None
    for newState in self.moveGen(state):
      if heuristic == 'h1':
        score = self.h1(newState)
      else:
        score = self.h2(newState)
      if score > currScore:
        ans = self.hill_Climbing(newState,score,heuristic)
        if ans != None:
          return [state] + ans
    return ans

  def bestFirst(self,state,visited={},heuristic='h2'):
    if tuple(map(tuple,state)) in visited:
      return None
    if self.checkGoal(state):
      return [state]

    ans = None
    pq = []
    visited[tuple(map(tuple,state))] = True

    newStates = [i for i in self.moveGen(state) if tuple(map(tuple,i)) not in visi
    for i in newStates:
      if heuristic == 'h1':
        score = self.h1(i)
      else:
        score = self.h2(i)
      heappush(pq,(-score,i))

    for score,state in pq:
      ans = self.bestFirst(state,visited,heuristic)
      if ans != None:
        return [state] + ans
    return ans
```

```
startState = []
goalState = []

print("Start State : ")
for i in range(3):
  startState.append(input(f"Enter elements in stack {i+1} : ").split())

print("Goal State : ")
for i in range(3):
  goalState.append(input(f"Enter elements in stack {i+1} : ").split())

game = BlockGame(startState,goalState)
```

```
Start State :
Enter elements in stack 1 : D C B A
Enter elements in stack 2 : F E
Enter elements in stack 3 :
Goal State :
Enter elements in stack 1 : D C B E A
Enter elements in stack 2 : F
Enter elements in stack 3 :
```

```
ans = game.hill_Climbing(startState, game.h2(startState), "h1")
print(ans)
```

```
None
```

```
ans = game.hill_Climbing(startState,game.h2(startState),"h2")
for i in ans:
  print(i)
```

```
[['D', 'C', 'B', 'A'], ['F', 'E'], []]
[['D', 'C', 'B'], ['F', 'E', 'A'], []]
[['D', 'C', 'B'], ['F', 'E'], ['A']]
[['D', 'C', 'B', 'E'], ['F'], ['A']]
[['D', 'C', 'B', 'E', 'A'], ['F'], []]
```

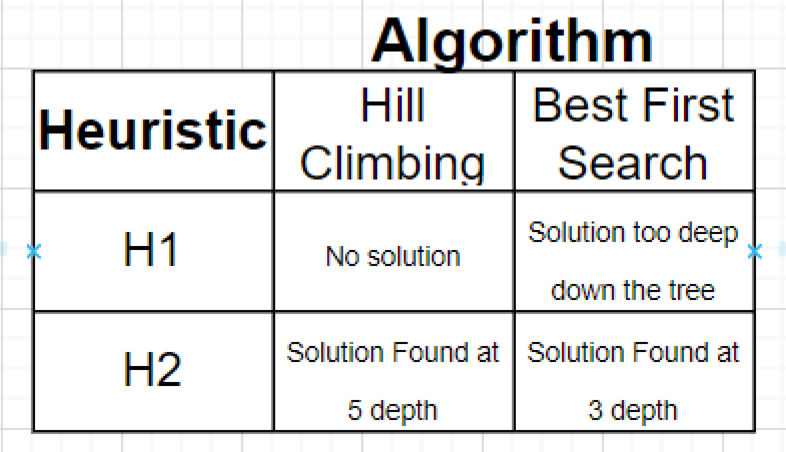
```
ans = game.bestFirst(startState,{},'h1') # Maximum Recursion Depth Reached
```

```
ans = game.bestFirst(startState,{},'h2')
for i in ans:
  print(i)
```

```
[['D', 'C', 'B'], ['F', 'E'], ['A']]
[['D', 'C', 'B', 'E'], ['F'], ['A']]
[['D', 'C', 'B', 'E', 'A'], ['F'], []]
[['D', 'C', 'B', 'E', 'A'], ['F'], []]
```

Conclusion:

We have successfully implemented Hill Climbing algorithm and Best First search alorithm to solve the Block Game using 2 heuristics We have seen that using the correct heuristic and correct algorithm matters a lot when solving such problems.



	Algorithm	
Heuristic	Hill Climbing	Best First Search
H1	No solution	Solution too deep down the tree
H2	Solution Found at 5 depth	Solution Found at 3 depth