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

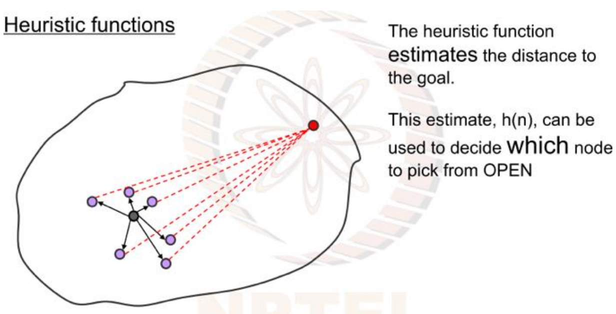
(Heuristic Search)

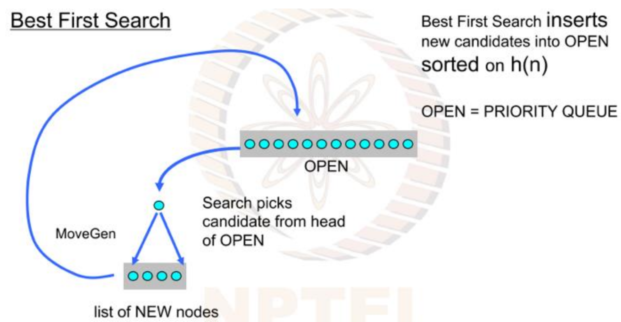
60009210105 Amitesh Sawarkar

D 12

Aim: Comparative analysis of Heuristic based methods.

Theory:





Algorithm for Best First Search

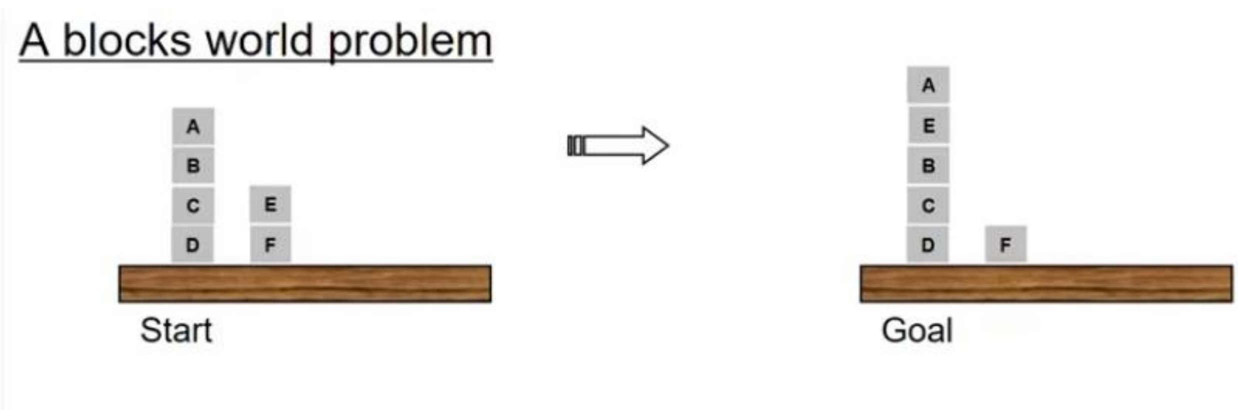
```
Best-First-Search(S)
OPEN ← (S, null, h(S)) []
CLOSED ← empty list
while OPEN is not empty
nodePair ← head OPEN
(N, , ) ← nodePair
if GoalTest(N) = true
return ReconstructPath(nodePair, CLOSED)
else CLOSED ← nodePair CLOSED
neighbours ← MoveGen(N)
newNodes ← RemoveSeen(neighbours, OPEN, CLOSED)
newPairs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

Algorithm Hill climbing

Hill-Climbing(S)
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.



```python
print("Enter the inital test's blocks from bottom to top")

i1 = []
i2 = []

n1 = int(input("Enter the number of blocks in i1: "))
for i in range(n1):
    i1.append(input(f"Enter block {i + 1} for i1: "))

n2 = int(input("Enter the number of blocks in i2: "))
for i in range(n2):
    i2.append(input(f"Enter block {i + 1} for i2: "))

print(i1)
print(i2)
```

```
Enter the inital test's blocks from bottom to top
Enter the number of blocks in i1: 4
Enter block 1 for i1: D
Enter block 2 for i1: C
Enter block 3 for i1: B
Enter block 4 for i1: A
Enter the number of blocks in i2: 2
Enter block 1 for i2: F
Enter block 2 for i2: E
['D', 'C', 'B', 'A']
['F', 'E']
```

```python
initial_table = [i1, i2]
print(initial_table)
```

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

```
print("Enter the goal test's blocks from bottom to top")

i3 = []
i4 = []

n3 = int(input("Enter the number of blocks in i1: "))
for i in range(n3):
    i3.append(input(f"Enter block {i + 1} for i1: "))

n4 = int(input("Enter the number of blocks in i1: "))
for i in range(n4):
    i4.append(input(f"Enter block {i + 1} for i1: "))

print(i3)
print(i4)
```

```
Enter the goal test's blocks from bottom to top
Enter the number of blocks in i1: 5
Enter block 1 for i1: D
Enter block 2 for i1: C
Enter block 3 for i1: B
Enter block 4 for i1: E
Enter block 5 for i1: A
Enter the number of blocks in i1: 1
Enter block 1 for i1: F
['D', 'C', 'B', 'E', 'A']
['F']
```

```
goal_table = [i3, i4]
print(goal_table)
```

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

```python
#HILL CLIMBING

def heuristic_h1(state, goal_state):
  misplaced_blocks = 0
  for i in range(len(state)):
    for j in range(len(state[i])):
      if state[i][j] not in goal_state[i]:
        misplaced_blocks += 1
  return misplaced_blocks

def heuristic_h2(state, goal_state):
  total_moves = 0
  for i in range(len(state)):
    for block in state[i]:
      if block != ' ':
        if block in goal_state[i]:
          target_i = goal_state[i].index(block)
          total_moves += abs(target_i - i)
        else:
          total_moves += 1
  return total_moves
```

```
def hill_climbing_with_path(initial_state, heuristic, goal_state):
    current_state = initial_state
    best_state = current_state
    path = [current_state]

    while True:
        neighbors = generate_neighbors(current_state)
        for neighbor in neighbors:
            if heuristic(neighbor, goal_state) < heuristic(best_state, goal_state):
                best_state = neighbor

        if heuristic(best_state, goal_state) >= heuristic(current_state, goal_state):
            break

        current_state = best_state
        path.append(current_state)

    return path
```

```
print("Hill Climbing with H1:")
path_h1 = hill_climbing_with_path(initial_table, heuristic_h1, goal_table)
if path_h1:
    for idx, state in enumerate(path_h1):
        print(f"Iteration {idx}:")
        for stack in state:
            print(stack)
        print("------")
    print("H1 Cost:", heuristic_h1(path_h1[-1], goal_table))
else:
    print("No solution found for H1.")

print("\nHill Climbing with H2:")
path_h2 = hill_climbing_with_path(initial_table, heuristic_h2, goal_table)
if path_h2:
    for idx, state in enumerate(path_h2):
        print(f"Iteration {idx}:")
        for stack in state:
            print(stack)
        print("------")
    print("H2 Cost:", heuristic_h2(path_h2[-1], goal_table))
else:
    print("No solution found for H2.")
```

```
Hill Climbing with H1:
Iteration 0:
['D', 'C', 'B', 'A']
['F', 'E']
------
Iteration 1:
['D', 'C', 'B', 'A', 'E']
['F']
------
H1 Cost: 0

Hill Climbing with H2:
Iteration 0:
['D', 'C', 'B', 'A']
['F', 'E']
------
Iteration 1:
['D', 'C', 'B']
['F', 'E', 'A']
------
Iteration 2:
['D', 'C']
['F', 'E', 'A', 'B']
------
H2 Cost: 5
```

```
#BFS
import heapq

def best_first_search(initial_state, heuristic, goal_state):
  OPEN = [(initial_state, 0)]
  CLOSED = set()
  while OPEN:
    current_state, cost = heapq.heappop(OPEN)
    if current_state == goal_state:
      return current_state
    CLOSED.add(tuple(map(tuple, current_state)))
    neighbors = generate_neighbors(current_state)
    for neighbor in neighbors:
      if tuple(map(tuple, neighbor)) not in CLOSED:
        if neighbor not in (state for state, _ in OPEN):
          heapq.heappush(OPEN, (neighbor, heuristic(neighbor, goal_state)))
  return None
print("Best First Search with H1:")
solution_h1_bfs = best_first_search(initial_table, heuristic_h1, goal_table)
if solution_h1_bfs:
    print("Solution:", solution_h1_bfs)
    print("H1 Cost:", heuristic_h1(solution_h1_bfs, goal_table))
else:
    print("No solution found for Best First Search with H1.")

print("\nBest First Search with H2:")
solution_h2_bfs = best_first_search(initial_table, heuristic_h2, goal_table)
if solution_h2_bfs:
    print("Solution:", solution_h2_bfs)
    print("H2 Cost:", heuristic_h2(solution_h2_bfs, goal_table))
else:
    print("No solution found for Best First Search with H2.")
```

```
Best First Search with H1:
Solution: [['D', 'C', 'B', 'E', 'A'], ['F']]
H1 Cost: 0

Best First Search with H2:
Solution: [['D', 'C', 'B', 'E', 'A'], ['F']]
H2 Cost: 11
```