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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 1

Aim: Identify PEAS for different Application.

PROBLEMS:

1. Satellite image analysis system:

Description: It consists of the images of earth and satellites taken by the means of artificial satellites.

Performance Measure: Correct image categorization

Environment: Downlink from orbiting satellite

Actuators: Display categorization of scene

Sensors: Colour pixel arrays

Environment type:

- Partially Observable
- Collaborative
- Multi-agent
- Dynamic
- Continuous
- Stochastic

2. Chatbot

Description: A chatbot is software that simulates human-like conversations with users via text messages on chat.

Performance Measure: Learning ability, Accuracy, Contextual reply

Environment: Google assistant, AppleSiri etc

Actuators: Databases, query searches

Sensors: Hardware like keyboard, user inputs

Environment type:

- Partially Observable
- Single-agent
- Dynamic
- Continuous
- Stochastic

3. An Essay Evaluator:

Description: It grades automatically grades the essay given to the software, and checks plagiarism and evaluates according to language and proper grammar.

Performance Measures: awards scores for quality, penalizes crap, detection of plagiarism, impartiality, usefulness of explanation of grading

Environment: Pdf, word file

Actuator: None, this can be a pure softbot

Sensors: File reading software, (perhaps even OCR)

Environment type:

- Partially Observable
- Single-agent
- Static
- Discrete
- Deterministic

4. Shopping for used books on the internet:

Description: This system gives the information of the second hand books available on the internet according to the preferences given by us.

Performance measure: Price, quality

Environment: Web, vendors, shippers

Actuators: Fill-in the form, follow URL, display to the user

Sensors: HTML

Environment type:

- Partially Observable
- Multi-agent
- Dynamic
- Continuous
- Stochastic

5. Refinery controller:

Description: It makes the labour work less by calculating the purity and improve it.

Performance measure: Maximize purity, yield, safety

Environment: Refinery, operators

Actuators: Valves, pumps, heaters, displays

Sensors: Temperature, pressure, chemical sensors.

Environment type:

- Fully Observable
- Single-agent
- Static
- Continuous
- Deterministic

6. Spam/ham classifier:

Description: It classifies all the spam and important mails properly.

Performance measure: Correct classification

Environment: Text, numbers, Alphanumeric-characters

Actuators: Screen display (Form)

Sensors: Keyboard.

Environment type:

- Fully Observable
- Multi-agent
- Dynamic
- Continuous
- Deterministic

7. Bidding on an item in the auction:

Description: It helps the bidders by automatically bidding for the required items given to the software.

Performance measure: Cost of the item, quality of the item, value of the item, necessity of the item

Environment: Auctioneer, bidders, items which are to be bid

Actuators: Speakers, microphones, display items, budget

Sensors: Camera, price monitor, eyes, ears of the attendees.

Environment type:

- Fully Observable
- Single-agent
- Static
- Discrete
- Deterministic

8. Chess player:

Description: It helps improve the other persons skills, and kill their boredom of being alone.

Performance measure: Once reached a terminal state (draw, win): -1 for losing, 0 for draw, 1 for winning

Environment: Chess board, and opponent

Actuators: Pairs of coordinates: (x1,y1) -> (x2, y2). Specify the source piece and the target position (for castling, the king's move is specified)

Sensors: Board perceived as a 8x8 matrix. Each element in the matrix can take one of the following values: E, BP, WP, BB, WB, BK, WN, BR, WN, BQ, WK, BK, WK

Environment type:

- Fully Observable
- Single-agent
- Dynamic
- Discrete
- Deterministic

9. Exploring the subsurface oceans of Titan:

Description: It helps to discover many unknown facts about the oceans by clicking quality pictures.

Performance measure: Safety, image quality, video quality

Environment: Ocean, water

Actuators: Mobile diver, steering, brake, accelerator

Sensors: Video, accelerometers, depth sensor, GPS

Environment type:

- Fully Observable
- Single-agent
- Dynamic
- Discrete
- Deterministic

10. Interactive English tutor:

Description: It teaches the student grammar, takes regular test, helps increase the fluency of speaking English by showing relevant videos.

Performance measure: Maximize students score in test

Environment: Set of students, testing agency

Actuators: Display exercises, suggestions, corrections

Sensors: Keyboard entry.

Environment type:

- Fully Observable
- Single-agent
- Static
- Discrete
- Deterministic

Conclusion: Thus, we have successfully studied the various case studies along with their PEAS description.

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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 2

AIM: Identify and analyze uninformed search Algorithm to solve the problem. Implement BFS/DFS/DFID search algorithms to reach goal state.

THEORY:

1. **BFS:**
   BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes. As the name BFS suggests, you are required to traverse the graph breadthwise as follows: First move horizontally and visit all the nodes of the current layer, Move to the next layer.

2. **DFS:**
   Depth-first search (DFS) is an algorithm for searching a graph or tree data structure. The algorithm starts at the root (top) node of a tree and goes as far as it can down a given branch (path), then backtracks until it finds an unexplored path, and then explores it. The algorithm does this until the entire graph has been explored. Many problems in computer science can be thought of in terms of graphs. For example, analysing networks, mapping routes, scheduling, and finding spanning trees are graph problems. To analyse these problems, graph-search algorithms like depth-first search are useful.

3. **DFID:**
   Depth First Iterative Deepening is an iterative searching technique that combines the advantages of both Depth-First search (DFS) and Breadth-First Search (BFS). DFID expands all nodes at a given depth before expanding any nodes at greater depth. So, it is guaranteed to find the shortest path or optimal solution from start to goal state.

CODE:

1. **BFS:**

```
def bfs(graph, current_node, goal_node):
    visit_complete = []
    visit_complete.append(current_node)
    queue = []
    queue.append(current_node)
    while queue:
        s = queue.pop(0)
        if goal_node == s:
            print(s)
            break
        print(s, end='->')
        for neighbour in graph[s]:
            if neighbour not in visit_complete:
                visit_complete.append(neighbour)
                queue.append(neighbour)
nodes = int(input('Enter the number of nodes: '))
graph = {}
for i in range(0, nodes):
    node_name = input(f'Enter {i + 1} node name: ')
    connected = int(
        input(f'Enter number of nodes connected to node {node_name}: '))
    connected_nodes = []
    if connected != 0:
        print('Enter the nodes: ')
    for j in range(0, connected):
        connected_nodes.append(input())
    graph[node_name] = connected_nodes
source_node = input('Enter the source node: ')
goal_node = input('Enter the goal node: ')
print('The path is: ', end='')
bfs(graph, source_node, goal_node)
```

```
Enter the number of nodes: 4
Enter 1 node name: 0
Enter number of nodes connected to node 0: 2
Enter the nodes:
1
2
Enter 2 node name: 1
Enter number of nodes connected to node 1: 1
Enter the nodes:
2
Enter 3 node name: 2
Enter number of nodes connected to node 2: 2
Enter the nodes:
0
3
Enter 4 node name: 3
Enter number of nodes connected to node 3: 1
Enter the nodes:
3
Enter the source node: 2
Enter the goal node: 1
The path is: 2->0->3->1
PS C:\.vscode\college>
```

2. DFS:

```python
import sys
visited = set()
def dfs(visited, graph, node, goal):
    if node not in visited:
        if goal == node:
            print(node)
            sys.exit(0)
        print(node, end = '->')
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour, goal)
nodes = int(input('Enter the number of nodes: '))
graph = {}
for i in range(0, nodes):
    node_name = input(f'Enter {i + 1} node name: ')
    connected = int(
        input(f'Enter number of nodes connected to node {node_name}: '))
    connected_nodes = []
    if connected != 0:
        print('Enter the nodes: ')
    for j in range(0, connected):
        connected_nodes.append(input())
```

```
    graph[node_name] = connected_nodes
source_node = input('Enter the source node: ')
goal_node = input('Enter the goal node: ')
print('The path is: ', end='')
dfs(visited, graph, source_node, goal_node)
```

```
Enter the number of nodes: 5
Enter 1 node name: 0
Enter number of nodes connected to node 0: 3
Enter the nodes:
1
2
3
Enter 2 node name: 1
Enter number of nodes connected to node 1: 1
Enter the nodes:
2
Enter 3 node name: 2
Enter number of nodes connected to node 2: 1
Enter the nodes:
4
Enter 4 node name: 3
Enter number of nodes connected to node 3: 0
Enter 5 node name: 4
Enter number of nodes connected to node 4: 0
Enter the source node: 0
Enter the goal node: 4
The path is: 0->1->2->4
PS C:\.vscode\college>
```

3. DFID:

```
path_list = []
def DLS(graph, src, target, max_depth):
    if src == target:
        return True
    if max_depth <= 0:
        return False
    for i in graph[src]:
        path_list.append(i)
        if (DLS(graph, i, target, max_depth - 1)):
            return True
    return False
def IDDFS(graph, src, target, max_depth):
    for i in range(max_depth):
        path_list.clear()
        if (DLS(graph, src, target, i)):
            return True
    return False
nodes = int(input('Enter the number of nodes: '))
graph = {}
for i in range(0, nodes):
    node_name = input(f'Enter {i + 1} node name: ')
    connected = int(
        input(f'Enter number of nodes connected to node {node_name}: '))
    connected_nodes = []
    if connected != 0:
        print('Enter the nodes: ')
    for j in range(0, connected):
        connected_nodes.append(input())
    graph[node_name] = connected_nodes
source_node = input('Enter the source node: ')
goal_node = input('Enter the goal node: ')
max_depth = int(input('Enter the max depth: '))
if IDDFS(graph, source_node, goal_node, max_depth) == True:
    print ("Target is reachable from source within max depth")
else :
    print ("Target is NOT reachable from source within max depth")
print('The path is: ', end = '')
for i in range(len(path_list)):
    print(path_list[i], end = '')
    if i != len(path_list) - 1:
       print('->', end = '')
```

```
Enter the number of nodes: 7
Enter 1 node name: 0
Enter number of nodes connected to node 0: 3
Enter the nodes:
1
2
4
Enter 2 node name: 1
Enter number of nodes connected to node 1: 2
Enter the nodes:
3
5
Enter 3 node name: 2
Enter number of nodes connected to node 2: 1
Enter the nodes:
6
Enter 4 node name: 3
Enter number of nodes connected to node 3: 0
Enter 5 node name: 4
Enter number of nodes connected to node 4: 0
Enter 6 node name: 5
Enter number of nodes connected to node 5: 0
Enter 7 node name: 6
Enter number of nodes connected to node 6: 0
Enter the source node: 0
Enter the goal node: 6
Enter the max depth: 2
Target is NOT reachable from source within max depth
The path is: 1->2->4
PS C:\.vscode\college>
```

CONCLUSION: Thus, we have successfully analysed the three uninformed search techniques: BFS, DFS, and DFID.

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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 3

Aim: Implementation of A* Search Algorithm.

Theory:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solves the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands fewer search trees and provides optimal results faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In the A* search algorithm, we use the search heuristic as well as the cost to reach the node. Hence, we can combine both costs as follows, and this sum is called a fitness number.

Algorithm of A* search:

1. Place the starting node in the OPEN list.
2. Check if the OPEN list is empty or not, if the list is empty then return failure and stop.
3. Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is the goal node, then return success and stop, otherwise.
4. Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute the evaluation function for n' and place it into the Open list.
5. Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
6. Return to Step 2.

Code:

```
def a_star_algorithm(graph, h, start, stop):
    open_list = set([start])
    closed_list = set([])
    poo = {}
    poo[start] = 0
    par = {}
    par[start] = start
    while len(open_list) > 0:
        n = None
        for v in open_list:
            if n == None or poo[v] + h[v] < poo[n] + h[n]:
                n = v
        if n == None:
            print('Path does not exist!')
            return None
        if n == stop:
            reconst_path = []
            while par[n] != n:
                reconst_path.append(n)
                n = par[n]
            reconst_path.append(start)
            reconst_path.reverse()
            return reconst_path, poo[stop]
        for (m, weight) in graph[n]:
            if m not in open_list and m not in closed_list:
                open_list.add(m)
                par[m] = n
                poo[m] = poo[n] + weight
            else:
                if poo[m] > poo[n] + weight:
                    poo[m] = poo[n] + weight
                    par[m] = n
            if m in closed_list:
                closed_list.remove(m)
                open_list.add(m)
        open_list.remove(n)
        closed_list.add(n)
    print('Path does not exist!')
    return None

nodes = int(input('Enter the number of nodes: '))
graph = {}
h = []
```

```
for i in range(0, nodes):
    node_name = input(f'Enter node {i + 1} name: ')
    h[node_name] = int(input('Enter its huristic value: '))
    connected = int(input(f'Enter number of nodes connected to node {node_name}: '))
    connected_nodes = []
    for j in range(0, connected):
        sub_node = input('Enter the node: ')
        distance = int(input('Enter it\'s distance: '))
        connected_nodes.append((sub_node, distance))
        graph[node_name] = connected_nodes

source_node = input('Enter the source node: ')
goal_node = input('Enter the goal node: ')
path, cost = a_star_algorithm(graph, h, source_node, goal_node)
print('The path is: ', end='')
for i in path:
    print(i, end='')
    if i != path[-1]:
        print('->', end='')
    else:
        print()

print(f'The final path cost is: {cost}')
```

Output:

```
PS C:\Users\HP\VSC> python -u "c:\Users\HP\VSC\Artificial Intelligence\A_Star.py"
Enter the number of nodes: 10
Enter node 1 name: A
Enter its huristic value: 10
Enter number of nodes connected to node A: 2
Enter the node: B
Enter it's distance: 6
Enter the node: F
Enter it's distance: 3
Enter node 2 name: B
Enter its huristic value: 8
Enter number of nodes connected to node B: 2
Enter the node: C
Enter it's distance: 3
Enter the node: D
Enter it's distance: 2
Enter node 3 name: C
Enter its huristic value: 5
Enter number of nodes connected to node C: 2
Enter the node: D
Enter it's distance: 1
Enter the node: E
Enter it's distance: 5
Enter node 4 name: D
Enter its huristic value: 7
Enter number of nodes connected to node D: 1
Enter the node: E
Enter it's distance: 8
Enter node 5 name: E
Enter its huristic value: 3
Enter number of nodes connected to node E: 2
Enter the node: I
Enter it's distance: 5
Enter the node: J
Enter it's distance: 5
Enter node 6 name: F
Enter its huristic value: 6
Enter number of nodes connected to node F: 2
Enter the node: G
Enter it's distance: 1
Enter the node: H
Enter it's distance: 7
Enter node 7 name: G
Enter its huristic value: 5
Enter number of nodes connected to node G: 1
Enter the node: I
Enter it's distance: 3
Enter node 8 name: H
Enter its huristic value: 3
Enter number of nodes connected to node H: 1
Enter the node: I
Enter it's distance: 2
Enter node 9 name: I
Enter its huristic value: 1
Enter number of nodes connected to node I: 1
Enter the node: J
Enter it's distance: 3
Enter node 10 name: J
Enter its huristic value: 6
Enter number of nodes connected to node J: 0
Enter the source node: A
Enter the goal node: H
The path is: A->F->H
The final path cost is: 10
```

Conclusion:

Learnt about A* Search Algorithm along with its steps and implemented it in code.

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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 4

Aim: To study and implement Hill-Climbing Search

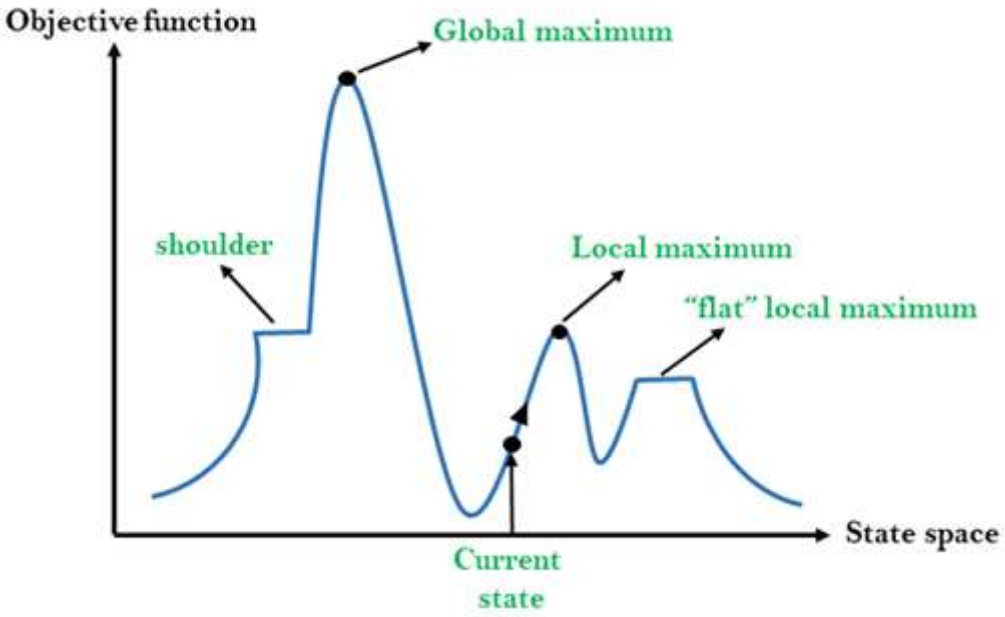
Theory:

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbour has a higher value. Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance travelled by the salesman.

It is also called greedy local search as it only looks to its good immediate neighbour state and not beyond that. A node of hill climbing algorithm has two components which are state and value. Hill Climbing is mostly used when a good heuristic is available. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features:

1. Generate and Test variant: Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
2. Greedy approach: Hill-climbing algorithm search moves in the direction which optimizes the cost.
3. No backtracking: It does not backtrack the search space, as it does not remember the previous state



Problems:

1. **Local Maximum:** A local maximum is a peak state in the landscape which is better than each of its neighbouring states, but there is another state also present which is higher than the local maximum
2. **Plateau:** A plateau is the flat area of the search space in which all the neighbour states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area
3. **Ridges:** A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Code:

```
import copy
def s(init_state,goal_state):
    global visited_states



    current_state = copy.deepcopy(init_state)
    index=1

    while(True):


        visited_states.append(copy.deepcopy(current_state))

        print("State:",index,current_state,h(current_state,goal_state))
        index=index+1
        prev_heu=h(current_state,goal_state)


        child = generate(current_state,prev_heu,goal_state)

        if child==0:

            return


        current_state = copy.deepcopy(child)


visited_states = []

def h(curr_state,goal_state):
    goal_=goal_state[4]
    val=0
    for i in range(len(curr_state)):
        check_val=curr_state[i]
        if len(check_val)>0:
            for j in range(len(check_val)):
                if check_val[j]!=goal_[j]:
                    val-=j
                else:
                    val+=j
    return val
```

```
def generate(curr_state,prev_heu,goal_state):
    global visited_states
    state = copy.deepcopy(curr_state)
    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 visited_states):
                        curr_heu=h(temp1,goal_state)

                        if curr_heu>prev_heu:
                            child = copy.deepcopy(temp1)
                            return child

    return 0

def hillclimb():

    global visited_states

    init_state = [[],[],[],[],['D','C','E','B','A']]
    goal_state = [[],[],[],[],['A','B','C','D','E']]

    s(init_state,goal_state)

hillclimb()
```

Output:

```
State: 1 [[], [], [], [], ['D', 'C', 'E', 'B', 'A']] -10
State: 2 [['A'], [], [], [], ['D', 'C', 'E', 'B']] -6
State: 3 [['A', 'B'], [], [], [], ['D', 'C', 'E']] -2
State: 4 [['A', 'B'], ['E'], [], [], ['D', 'C']] 0
State: 5 [['A', 'B', 'C'], ['E'], [], [], ['D']] 3
State: 6 [['A', 'B', 'C', 'D'], ['E'], [], [], []] 6
State: 7 [['A', 'B', 'C', 'D', 'E'], [], [], [], []] 10


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion: Thus, we successfully studied and implemented Hill-Climbing Search

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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

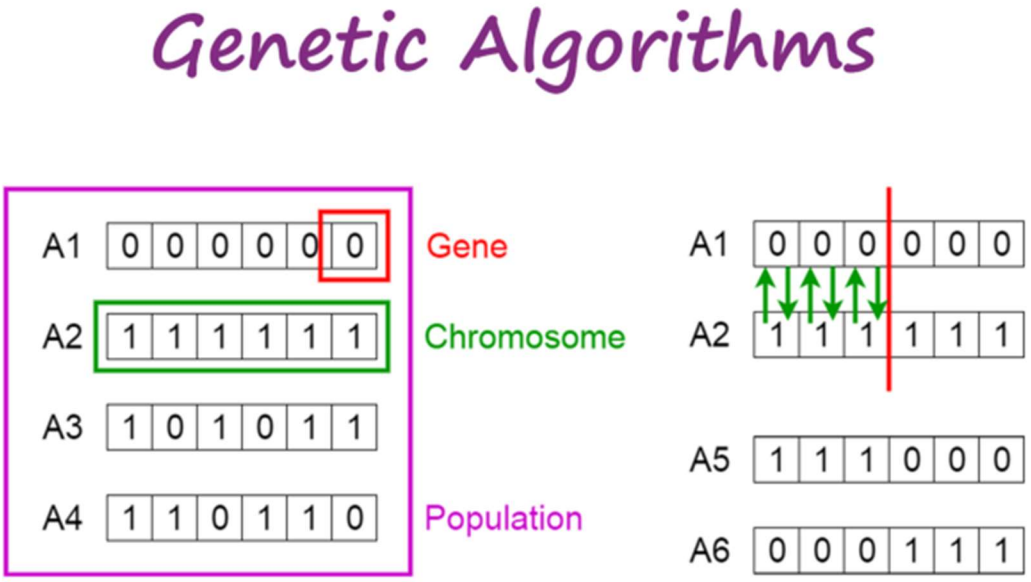
AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 5

Aim: To study and implement Genetic Algorithm

Theory:
A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.



Five phases are considered in a genetic algorithm.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

Initial Population
The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem you want to solve.An individual is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution).

Fitness Function
The fitness function determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

Selection
The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation. Two-pairs of individuals (parents) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

Crossover
Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes.

Mutation
In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped.

Code:

```
import math
from numpy.random import rand, randint

def crossover(parent1, parent2, r_cross):
    child1, child2 = parent1.copy(), parent2.copy()
    r = rand()
    point = 0
    if r > r_cross:
        point = randint(1, len(parent1) - 2)
        child1 = parent1[:point] + parent2[point:]
        child2 = parent2[:point] + parent1[point:]
    return child1, child2, point

def mutate(chromosome, r_mut):
    for i in range(len(chromosome)):
        if rand() < r_mut:
            chromosome[i] = 1 - chromosome[i]
    return chromosome

def bin_to_dec(bin):
    decimal = 0
    for i in range(len(bin)):
        decimal += bin[i] * pow(2, 4 - i)
    return decimal

def dec_to_bin(dec):
    binaryVal = []
    while dec > 0:
        binaryVal.append(dec % 2)
        dec = math.floor(dec / 2)
    for _ in range(5 - len(binaryVal)):
        binaryVal.append(0)
    binaryVal = binaryVal[::-1]
    return binaryVal

def fitness_function(x):
    return pow(x, 2)

def genetic_algorithm(iterations, population_size, r_cross, r_mut):
    input = [randint(0, 32) for _ in range(population_size)]
    pop = [dec_to_bin(i) for i in input]
    for generation in range(iterations):
        print(f"\nGeneration : {generation+1}", end="\n\n")
        decimal = [bin_to_dec(i) for i in pop]
        fitness_score = [fitness_function(i) for i in decimal]
        f_by_sum = [
            fitness_score[i] / sum(fitness_score) for i in range(population_size)
        ]
```

```
        exp_cnt = [
            fitness_score[i] / (sum(fitness_score) / population_size)
            for i in range(population_size)
        ]
        act_cnt = [round(exp_cnt[i]) for i in range(population_size)]
        print(
            "SELECTION\n\nInitial Population\tDecimal Value\tFitness Score\tFi/Sum\tExpected
count\tActual Count"
        )
        for i in range(population_size):
            print(
                pop[i],
                "\t",
                decimal[i],
                "\t\t",
                fitness_score[i],
                "\t\t",
                round(f_by_sum[i], 2),
                "\t\t",
                round(exp_cnt[i], 2),
                "\t\t",
                act_cnt[i],
            )
        print("Sum : ", sum(fitness_score))
        print("Average : ", sum(fitness_score) / population_size)
        print("Maximum : ", max(fitness_score), end="\n")
        max_count = max(act_cnt)
        min_count = min(act_cnt)
        max_count_index = 0
        for i in range(population_size):
            if max_count == act_cnt[i]:
                max_count_index = i
                break
        for i in range(population_size):
            if min_count == act_cnt[i]:
                pop[i] = pop[max_count_index]
        crossover_children = list()
        crossover_point = list()
        for i in range(0, population_size, 2):
            child1, child2, point_of_crossover = crossover(
                pop[i], pop[i + 1], r_cross)
            crossover_children.append(child1)
            crossover_children.append(child2)
            crossover_point.append(point_of_crossover)
            crossover_point.append(point_of_crossover)
        print(
            "\nCROSS OVER\n\nPopulation\t\tMate\t Crossover Point\t Crossover Population"
        )
        for i in range(population_size):
```

```
            if (i + 1) % 2 == 1:
                mate = i + 2
            else:
                mate = i
            print(
                pop[i],
                "\t",
                mate,
                "\t",
                crossover_point[i],
                "\t\t\t",
                crossover_children[i],
            )
        mutation_children = list()
        for i in range(population_size):
            child = crossover_children[i]
            mutation_children.append(mutate(child, r_mut))
        new_population = list()
        new_fitness_score = list()
        for i in mutation_children:
            new_population.append(bin_to_dec(i))
        for i in new_population:
            new_fitness_score.append(fitness_function(i))
        print("\nMUTATION\n\nMutation population\t New Population\t Fitness Score")
        for i in range(population_size):
            print(
                mutation_children[i],
                "\t",
                new_population[i],
                "\t\t",
                new_fitness_score[i],
            )
        print("Sum : ", sum(new_fitness_score))
        print("Maximum : ", max(new_fitness_score))
        pop = mutation_children

genetic_algorithm(iterations=2, population_size=4, r_cross=0.5, r_mut=0.05)
```

Output:

```
Maximum :  729

CROSS OVER

Population              Mate     Crossover Point         Crossover Population
[1, 0, 1, 0, 0]          2       1                       [1, 1, 0, 1, 1]
[1, 1, 0, 1, 1]          1       1                       [1, 0, 1, 0, 0]
[1, 1, 0, 1, 1]          4       0                       [1, 1, 0, 1, 1]
[1, 1, 0, 1, 1]          3       0                       [1, 1, 0, 1, 1]

MUTATION

Mutation population      New Population  Fitness Score
[1, 1, 0, 1, 0]          26              676
[1, 0, 1, 0, 0]          20              400
[1, 1, 0, 1, 1]          27              729
[1, 1, 0, 1, 1]          27              729
Sum :  2534
Maximum :  729

Generation : 2

SELECTION

Initial Population      Decimal Value   Fitness Score   Fi/Sum  Expected count  Actual Count
[1, 1, 0, 1, 0]          26              676             0.27            1.07            1
[1, 0, 1, 0, 0]          20              400             0.16            0.63            1
[1, 1, 0, 1, 1]          27              729             0.29            1.15            1
[1, 1, 0, 1, 1]          27              729             0.29            1.15            1
Sum :  2534
Average :  633.5
Maximum :  729

CROSS OVER

Population              Mate     Crossover Point         Crossover Population
[1, 1, 0, 1, 0]          2       1                       [1, 1, 0, 1, 0]
[1, 1, 0, 1, 0]          1       1                       [1, 1, 0, 1, 0]
[1, 1, 0, 1, 0]          4       1                       [1, 1, 0, 1, 0]
[1, 1, 0, 1, 0]          3       1                       [1, 1, 0, 1, 0]

MUTATION

Mutation population      New Population  Fitness Score
[1, 0, 0, 0, 0]          16              256
[1, 1, 0, 1, 0]          26              676
[1, 1, 0, 1, 0]          26              676
[1, 1, 0, 1, 0]          26              676
Sum :  2284
Maximum :  676
```

Conclusion: Thus, we successfully studied and applied Genetic Algorithm

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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 6

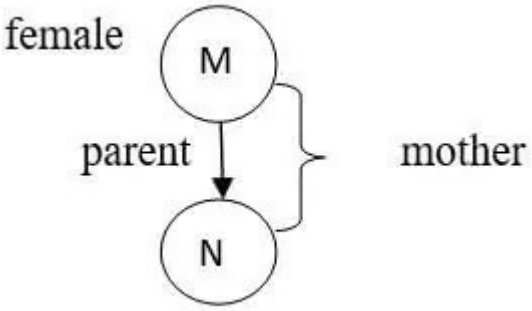
Aim: Implement Family tree using Prolog.

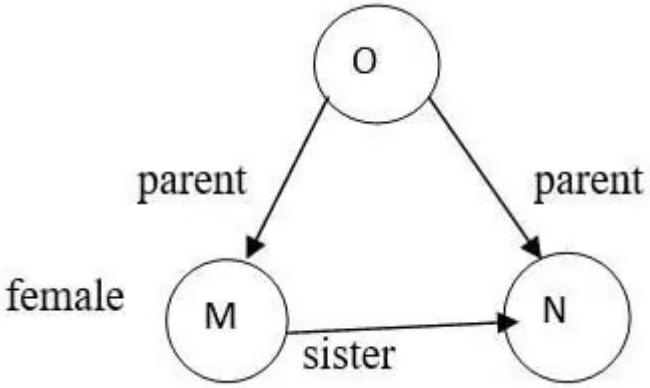
Theory:

Prolog is a language built around the Logical Paradigm: a declarative approach to problem solving.
There are only three basic constructs in Prolog: facts, rules, and queries.
A collection of facts and rules is called a knowledge base (or a database) and Prolog programming is all about writing knowledge bases. That is, Prolog programs simply are knowledge bases, collections of facts and rules which describe some collection of relationships that we find interesting.

Family Tree-

The program in prolog specifies the relationship between objects and the properties of objects; the family trees tell us how to construct a database of families. The database also contains facts and rules; let us consider the example “Sumit has a car.” We can declare the original relationship between two objects where one object is Sumit. Another object is a car; if we say, “does Sumit own a car?'' There are many types of relationships; some of them are ruled by using rules in the program, we can find the relationship between objects used, and it is not defined as a fact. Tree diagrams are very good in representations. The information is clearly mentioned, and due to that, users can understand easily, our programs in prolog are the sets of clauses.





Program:

```
female(pammi). female(lizza). female(patty).
female(anny). male(jimmy). male(bobby).
male(tomy). male(pitter). parent(pammi,bobby).
parent(tomy,bobby). parent(tomy,lizza).
parent(bobby,anny). parent(bobby,patty).
parent(patty,jimmy). parent(bobby,pitter).
parent(pitter,jimmy). mother(X,Y):-
parent(X,Y),female(X). father(X,Y):-
parent(X,Y),male(X). grandparent(X,Y):-
parent(X,Z),parent(Z,Y). grandmother(X,Z):-
mother(X,Y),parent(Y,Z). grandfather(X,Z):-
father(X,Y),parent(Y,Z). haschild(X):-
parent(X,_). sister(X,Y):-
parent(Z,X),parent(Z,Y),female(X),X\==Y.
brother(X,Y):-
parent(Z,X),parent(Z,Y),male(X),X\==Y.
```

Output:

Conclusion: Hence, we learned about Prolog language and implement Family tree in same

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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 7

Aim: Implement Wumpus World program in Prolog.

Theory:

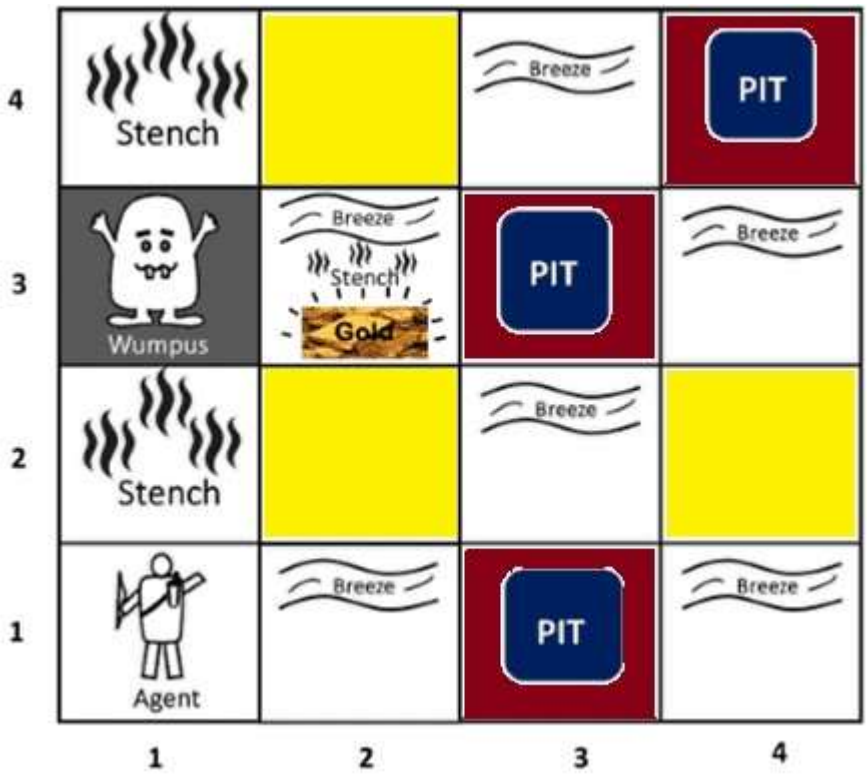
1. The Wumpus World in Artificial intelligence

The Wumpus world is a basic world example that demonstrates the value of a knowledge-based agent and how knowledge representation is represented. It was inspired by Gregory Yob's 1973 video game Hunt the Wumpus.

The Wumpus world is a cave with 4/4 rooms and pathways connecting them. As a result, there are a total of 16 rooms that are interconnected. We now have a knowledge-based AI capable of progressing in this world. There is an area in the cave with a beast named Wumpus who eats everybody who enters. The agent can shoot the Wumpus, but he only has a single arrow. There are some Pits chambers in the Wumpus world that are bottomless, and if an agent falls into one, he will be stuck there indefinitely. The intriguing thing about this cave is that there is a chance of finding a gold heap in one of the rooms. So the agent's mission is to find the gold and get out of the cave without getting eaten by Wumpus or falling into Pits. the agent returns with gold, he will be rewarded, but if he is devoured by Wumpus or falls into the pit, he will be penalized.

Note: Wumpus is immobile in this scene.

A sample diagram for portraying the Wumpus world is shown below. It depicts some rooms with Pits, one room with Wumpus, and one agent in the world's (1, 1) square position.



There are also some components which can help the agent to navigate the cave. These components are given as follows:

- The rooms adjacent to the Wumpus room are stinky, thus there is a stench there.
- The room next to PITs has a breeze, so if the agent gets close enough to PIT, he will feel it.
- If and only if the room contains gold, there will be glitter.
- If the agent is facing the Wumpus, the agent can kill it, and Wumpus will cry horribly, which can be heard anywhere.

2. PEAS description of Wumpus world:

We have given PEAS description as below to explain the Wumpus world:

Following are some basic facts about propositional logic:

Performance measure:

- If the agent emerges from the cave with the gold, he will receive 1000 bonus points.
- If you are devoured by the Wumpus or fall into the pit, you will lose 1000 points.
- For each action, you get a -1, and for using an arrow, you get a -10.
- If either agent dies or emerges from the cave, the game is over.

Environment:

- A 4*4 grid of rooms.
- Initially, the agent is in room square [1, 1], facing right.
- Except for the first square [1,1], the locations of Wumpus and gold are picked at random.
- Except for the initial square, every square of the cave has a 0.2 chance of being a pit.

Actuators:

- Left turn
- Right turn
- Move forward
- Grab
- Release
- Shoot **Sensors:**

- If the agent is in the same room as the Wumpus, he will smell the stench. (Not on a diagonal.)
- If the agent is in the room directly adjacent to the Pit, he will feel a breeze.
- The agent will notice the gleam in the room where the gold is located.
- If the agent walks into a wall, he will feel the bump.
- RWhen the Wumpus is shot, it lets out a horrifying scream that can be heard from anywhere in the cave.
- These perceptions can be expressed as a five-element list in which each sensor will have its own set of indicators.
- For instance, if an agent detects smell and breeze but not glitter, bump, or shout, it might be represented as [Stench, Breeze, None, None, None].

3. The Wumpus world Properties:

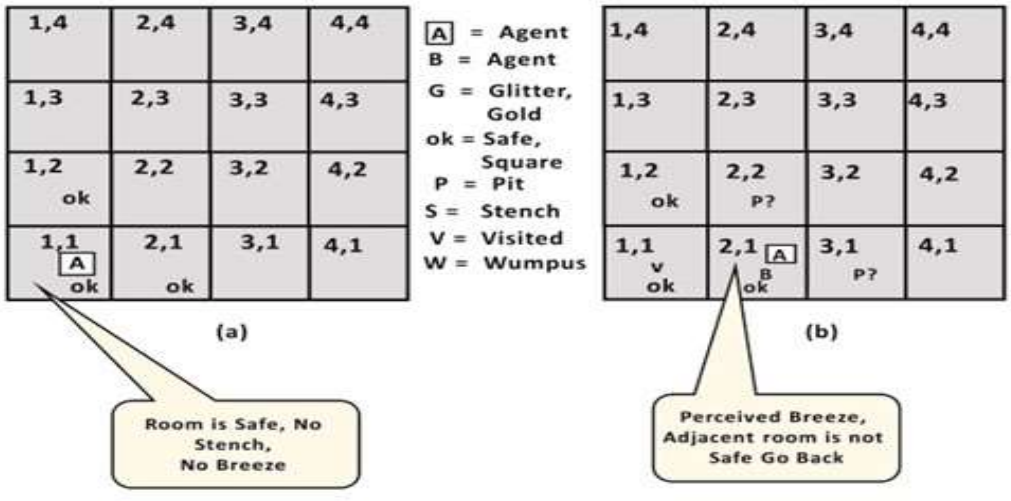
- Partially observable: The Wumpus universe is only partially viewable because the agent can only observe the immediate environment, such as a nearby room.
- Deterministic: It's deterministic because the world's result and outcome are already known.
- Sequential: It is sequential because the order is critical.
- Static: Wumpus and Pits are not moving, thus it is static.
- Discrete: There are no discrete elements in the environment.
- One agent: We only have one agent, and Wumpus is not regarded an agent, hence the environment is single agent.

4. Exploring the Wumpus world:

Now we will explore Wumpus world a bit and will explain how the agent will find its goal applying logical reasoning.

Agent's First step: At first, the agent is in the first room, or square [1,1], and we all know that this room is safe for the agent, thus we will add the sign OK to the below diagram (a) to represent that room is safe. The agent is represented by the letter A, the breeze by the letter B, the glitter or gold by the letter G, the visited room by the letter V, the pits by the letter P, and the Wumpus by the letter W.

Agent does not detect any wind or Stench in Room [1,1], indicating that the nearby squares are similarly in good condition.



Agent's first step

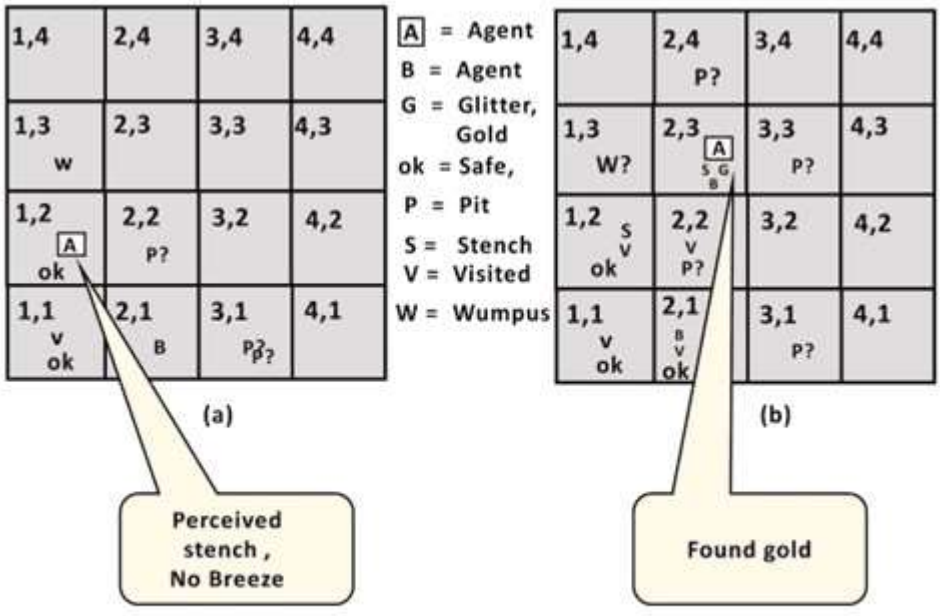
Agent's second Step:

Now that the agent must go forward, it will either go to [1, 2] or [2, 1]. Let's say agent enters room [2, 1], where he detects a breeze, indicating Pit is present. Because the pit might be in [3, 1] or [2, 2], we'll add the sign P? to indicate that this is a Pit chamber.

Now the agent will pause and consider his options before doing any potentially destructive actions. The agent will return to room [1, 1]. The agent visits the rooms [1,1] and [2,1], thus we'll use the symbol V to symbolize the squares he's been to.

Agent's third step:

The agent will now proceed to the room [1,2], which is fine. Agent detects a stink in the room [1,2], indicating the presence of a Wumpus nearby. However, according to the rules of the game, Wumpus cannot be in the room [1,1], and he also cannot be in [2,2]. (Agent had not detected any stench when he was at [2,1]). As a result, the agent infers that Wumpus is in the room [1,3], and there is no breeze at the moment, implying that there is no Pit and no Wumpus in [2,2]. So that's safe, and we'll designate it as OK, and the agent will advance [2,2] .



Agent's third step

Agent's fourth step:

Because there is no odor and no breeze in room [2,2], let's assume the agent decides to move to room [2,3]. Agent detects glitter in room [2,3], thus it should collect the gold and ascend out of the cave.

Code:

```
% Declaring dynamic methods

:- dynamic ([
agent_location/1,
gold_location/1,
pit_location/1,
time_taken/1,        score/1,
visited/1,
visited_cells/1,
world_size/1,
wumpus_location/1,
          isPit/2,
isWumpus/2,
          isGold/2
            ]).


%------------------------------------------------------------------------------ % To
start the game
 start
:-
   format('Initializing started...~n', []),     init,
   format('Let the game begin!~n', []),     take_steps([[1,1]]).

%------------------------------------------------------------------------------ %
Scheduling simulation:

step_pre(VisitedList) :-     agent_location(AL),
gold_location(GL),
   wumpus_location(WL),     score(S),
   time_taken(T),

   ( AL=GL -> writeln('WON!'), format('Score: ~p,~n Time: ~p', [S,T])
```

```
    ; AL=WL -> format('Lost: Wumpus eats you!~n', []),
format('Score: ~p,~n Time: ~p', [S,T]) ;
take_steps(VisitedList)
    ).

take_steps(VisitedList) :-
make_percept_sentence(Perception),      agent_location(AL),
    format('I\'m in ~p, seeing: ~p~n', [AL,Perception]),

    update_KB(Perception),      ask_KB(VisitedList,
Action),
    format('I\'m going to: ~p~n', [Action]),

    update_time,
    update_score,

    agent_location(Aloc),      VL
= [Aloc|VisitedList],
    standing,
step_pre(VL).

%------------------------------------------------------------------------------ %
Updating states

update_time :-      time_taken(T),
NewTime is T+1,      retractall(
time_taken(_) ),      assert(
time_taken(NewTime) ).

update_score :-      agent_location(AL),
gold_location(GL),
wumpus_location(WL),
update_score(AL, GL, WL).

update_score(P) :-      score(S),
    NewScore is S+P,
retractall( score(_) ),      assert(
score(NewScore) ).

update_score(AL, AL, _) :-      update_score(1000).

update_score(_,_,_) :-
    update_score(-1).
```

```
update_agent_location(NewAL) :-
retractall( agent_location(_) ),     assert(
agent_location(NewAL) ).

is_pit(no,  X) :-     \+
pit_location(X).
is_pit(yes, X) :-     pit_location(X).

%------------------------------------------------------------------------------ %
Display standings

standing :-     wumpus_location(WL),
gold_location(GL),
   agent_location(AL),

   ( is_pit(yes, AL) -> format('Agent was fallen into a pit!~n', []),       fail
   ; stnd(AL, GL, WL)       %\+
pit_location(yes, Al),
   ).

stnd(_, _, _) :-     format('There\'s still something to
do...~n', []).

stnd(AL, _, AL) :-     format('YIKES! You\'re eaten by the
wumpus!', []),     fail.

stnd(AL, AL, _) :-     format('AGENT FOUND THE
GOLD!!', []),     true.
%------------------------------------------------------------------------------ %
Perceptotion

make_perception([_Stench,_Bleeze,_Glitter]) :-     agent_location(AL),
   isStinky(AL), isBleezie(AL),
   isGlittering(AL).

test_perception :-
    make_percept_sentence(Percept),          format('I
feel ~p, ',[Percept]).

make_percept_sentence([Stench,Bleeze,Glitter]) :-
    smelly(Stench),       bleezy(Bleeze),
    glittering(Glitter).
```

```
%------------------------------------------------------------------------ %
Initializing

init :-     init_game,
init_land_fig72,
init_agent,
init_wumpus.

init_game :-     retractall(
time_taken(_) ),     assert(
time_taken(0) ),

   retractall( score(_) ),
   assert( score(0) ),

   retractall( visited(_) ),     assert(
visited(1) ),

   retractall( isWumpus(_,_) ),
   retractall( isGold(_,_) ),

   retractall( visited_cells(_) ),     assert(
visited_cells([]) ).

% To set the situation described in Russel-Norvig's book (2nd Ed.),
% according to Figure 7.2
init_land_fig72 :-     retractall(
world_size(_) ),
   assert( world_size(4) ),

   retractall( gold_location(_) ),
   assert( gold_location([3,2]) ),

   retractall( pit_location(_) ),     assert(
pit_location([4,4]) ),     assert(
pit_location([3,3]) ),     assert(
pit_location([1,3]) ).

init_agent :-     retractall(
agent_location(_) ),     assert(
agent_location([1,1]) ),

   visit([1,1]).
```

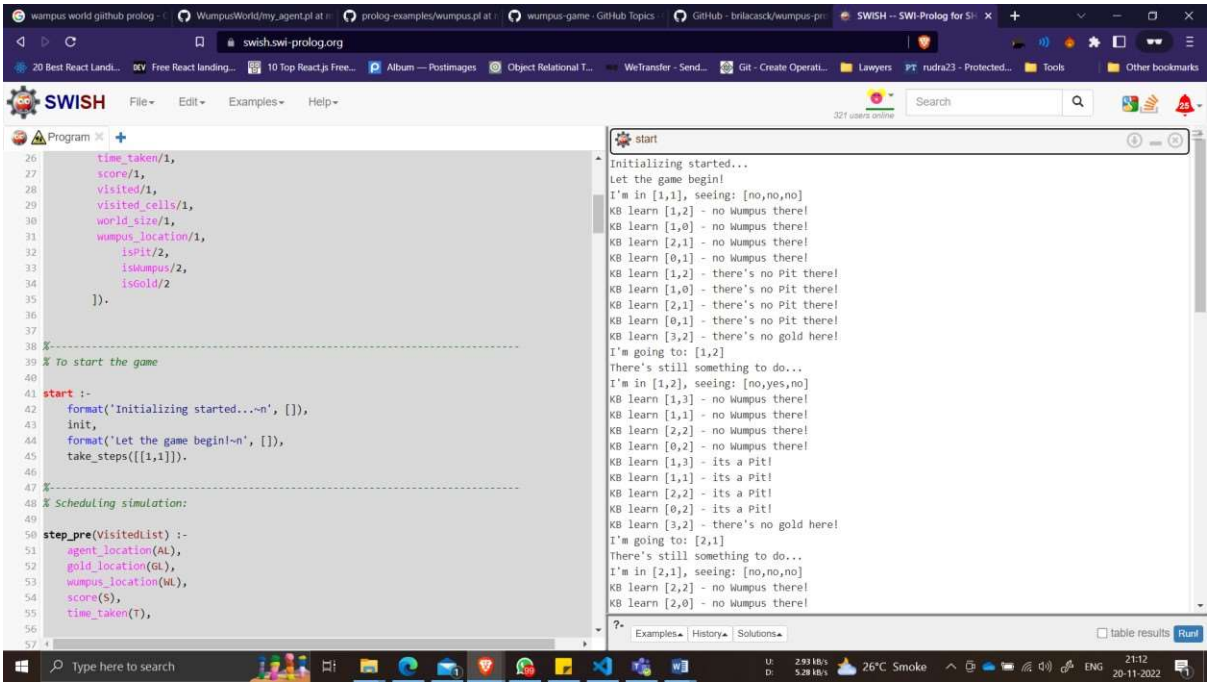
```
init_wumpus :-     retractall(
wumpus_location(_) ),     assert(
wumpus_location([4,1]) ).

visit(Xs) :-     visited_cells(Ys),
retractall( visited_cells(_) ),
assert( visited_cells([Ys|Xs]) ).

%------------------------------------------------------------------------------ %
Perceptors

%%% Institiation error!!!

%adj(X,Y) :- %
world_size(WS),
%    ( X is Y+1, Y   < WS %
; X is Y-1, Y-1 > 0 %    ).

adj(1,2). adj(2,1).
adj(2,3). adj(3,2).
adj(3,4). adj(4,3).

adjacent( [X1, Y1], [X2, Y2] ) :-
    ( X1 = X2, adj( Y1, Y2 )
    ; Y1 = Y2, adj( X1, X2 )
    ).

%adjacent([X1,Y],[X2,Y]) :- %
adj(X1,X2).

%adjacent([X,Y1],[X,Y2]) :- %
adj(Y1,Y2).

isSmelly(Ls1) :-     wumpus_location(
Ls2 ),     adjacent( Ls1, Ls2 ).

isBleezy(Ls1) :-     pit_location(
Ls2 ),     adjacent( Ls1, Ls2 ).

isGlittering( [X1, Y1] ) :-     gold_location(
[X2, Y2] ),
```

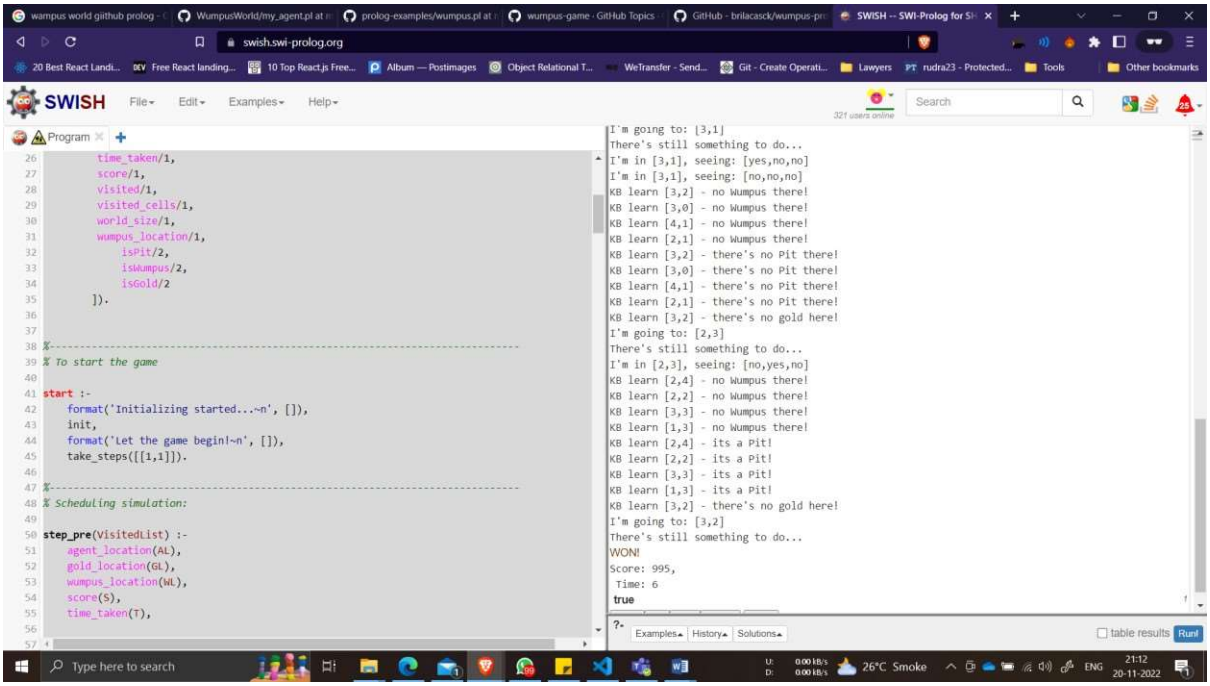
```
   X1 = X2,
Y1 = Y2.

bleezy(yes) :-
agent_location(AL),
isBleezy(AL). bleezy(no).

smelly(yes) :-     agent_location(AL),
isSmelly(AL).
smelly(no).

glittering(yes) :-     agent_location(AL),
isGlittering(AL).
glittering(no).

%-----------------------------------------------------------------------------
% Knowledge Base:

update_KB( [Stench,Bleeze,Glitter] ) :-
add_wumpus_KB(Stench),     add_pit_KB(Bleeze),
add_gold_KB(Glitter).

% if it would be 'yes' -> it would mean the player is eaten ;]
add_wumpus_KB(no) :-
%agent_location(L1),     %adjacent(L1,
L2),     %assume_wumpus(no, L2).
agent_location([X,Y]),
   world_size(_),

   % Checking needed!!
   % adj will freeze for (4,_) !!

   Z1 is Y+1, assume_wumpus(no,[X,Z1]),
   Z2 is Y-1, assume_wumpus(no,[X,Z2]),
   Z3 is X+1, assume_wumpus(no,[Z3,Y]),     Z4
is X-1, assume_wumpus(no,[Z4,Y]).

add_pit_KB(no)                              :-
agent_location([X,Y]),          Z1  is  Y+1,
assume_pit(no,[X,Z1]),
   Z2 is Y-1, assume_pit(no,[X,Z2]),
   Z3 is X+1, assume_pit(no,[Z3,Y]),     Z4
is X-1, assume_pit(no,[Z4,Y]).
```

```
% Checking needed!! If its not already in the KB !!!
add_pit_KB(yes) :-     agent_location([X,Y]),     Z1 is
Y+1, assume_pit(yes,[X,Z1]),
   Z2 is Y-1, assume_pit(yes,[X,Z2]),
   Z3 is X+1, assume_pit(yes,[Z3,Y]),     Z4
is X-1, assume_pit(yes,[Z4,Y]).

add_gold_KB(no) :-     gold_location(GL),
assume_gold(no, GL).

add_gold_KB(yes) :-
gold_location([X1,Y1]),
agent_location([X2,Y2]),     X1 =
X2, Y1 = Y2,     assume_gold(yes,
[X1,Y1]).

assume_wumpus(no, L) :-
   retractall( isWumpus(_, L) ),     assert(
isWumpus(no, L) ),     format('KB learn ~p - no
Wumpus there!~n', [L]).

assume_wumpus(yes, L) :-
   %wumpus_healthy, % Will be included ...
   retractall( isWumpus(_, L) ),     assert( isWumpus(yes, L) ),
format('KB learn ~p - possibly the Wumpus is there!~n', [L]).

assume_pit(no, L) :-     retractall( isPit(_, L) ),     assert(
isPit(no, L) ),     format('KB learn ~p - there\'s no Pit
there!~n', [L]).

assume_pit(yes, L) :-     retractall( isPit(_, L)
),     assert( isPit(yes, L) ),     format('KB
learn ~p - its a Pit!~n', [L]).

assume_gold(no, L) :-     retractall( isGold(_, L) ),
assert( isGold(no, L) ),     format('KB learn ~p - there\'s
no gold here!~n', [L]).

assume_gold(yes, L) :-     retractall( isGold(_, L) ),
assert( isGold(yes, L) ),     format('KB learn ~p - GOT THE
GOLD!!!~n', [L]).
```

```
permitted([X,Y]) :-     world_size(WS),
    0 < X, X < WS+1,     0
< Y, Y < WS+1.

ask_KB(VisitedList, Action) :-
isWumpus(no, L),     isPit(no, L),
permitted(L),     not_member(L,
VisitedList),
update_agent_location(L),
Action = L.

not_member(_, []).
not_member([X,Y], [[U,V]|Ys]) :-     (
X=U,Y=V -> fail     ;
not_member([X,Y], Ys)
    ).
```

Output:





Conclusion: We successfully implemented Wumpus World program in Prolog.

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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 8

Aim: Study experiment on Planning Problem.

Theory:

Artificial Intelligence is a critical technology in the future. Whether it is intelligent robots or self-driving cars or smart cities, they will all use different aspects of Artificial Intelligence!!! But to create any such AI project, **Planning** is very important. So much so that Planning is a critical part of Artificial Intelligence which deals with the actions and domains of a particular problem. Planning is considered as the reasoning side of acting. Everything we humans do is with a certain goal in mind and all our actions are oriented towards achieving our goal. In a similar fashion, planning is also done for Artificial Intelligence. For example, reaching a particular destination requires planning. Finding the best route is not the only requirement in planning, but the actions to be done at a particular time and why they are done is also very important. That is why planning is considered as the reasoning side of acting. In other words, planning is all about deciding the actions to be performed by the Artificial Intelligence system and the functioning of the system on its own in domain independent situations.

What is a Plan?

For any planning system, we need the **domain description, action specification**, and **goal description**. A plan is assumed to be a sequence of actions and each action has its own set of preconditions to be satisfied before performing the action and also some effects which can be positive or negative.
So, we have Forward State Space Planning (FSSP) and Backward State Space Planning (BSSP) at the basic level.

1. Forward State Space Planning (FSSP)

FSSP behaves in a similar fashion like forward state space search. It says that given a start state S in any domain, we perform certain actions required and acquire a new state S' (which includes some new conditions as well) which is called progress and this proceeds until we reach the goal state. The actions have to be applicable in this case.

- **Disadvantage:** Large branching factor
- **Advantage:** Algorithm is Sound

2. Backward State Space Planning (BSSP)

BSSP behaves in a similar fashion like backward state space search. In this, we move from the goal state g towards sub-goal g' that is finding the previous action to be done to achieve that respective goal. This process is called regression (moving back to the previous goal or sub-goal). These sub-goals have to be checked for consistency as well. The actions have to be relevant in this case.

- **Disadvantage:** Not a sound algorithm (sometimes inconsistency can be found)
- **Advantage:** Small branching factor (very small compared to FSSP)

Example:

Planning in artificial intelligence is about decision-making actions performed by robots or computer programs to achieve a specific goal.

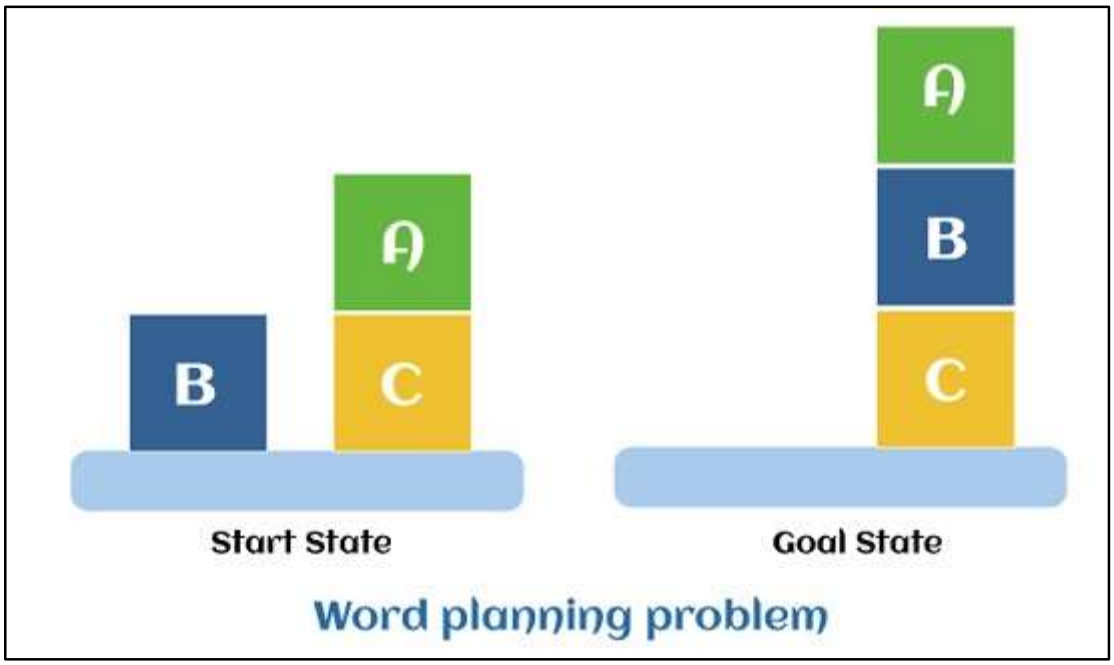
Execution of the plan is about choosing a sequence of tasks with a high probability of accomplishing a specific task.

Block-world planning problem

- The block-world problem is known as the Sussmann anomaly.
- The non-interlaced planners of the early 1970s were unable to solve this problem. Therefore it is considered odd.

- When two sub-goals, G1 and G2, are given, a non-interleaved planner either produces a plan for G1 that is combined with a plan for **G2** or vice versa.
- In the block-world problem, three blocks labeled 'A', 'B', and 'C' are allowed to rest on a flat surface. The given condition is that only one block can be moved at a time to achieve the target.

The start position and target position are shown in the following diagram.



Components of the planning system

The plan includes the following important steps:

- Choose the best rule to apply the next rule based on the best available guess. o Apply the chosen rule to calculate the new problem condition.
- Find out when a solution has been found. o Detect dead ends so they can be discarded and direct system effort in more useful directions.
- Find out when a near-perfect solution is found.

Target stack plan o It is one of the most important planning algorithms used by STRIPS. o Stacks are used in algorithms to capture the action and complete the target. A knowledge base is used to hold the current situation and actions.

- A target stack is similar to a node in a search tree, where branches are created with a choice of action.

The important steps of the algorithm are mentioned below:

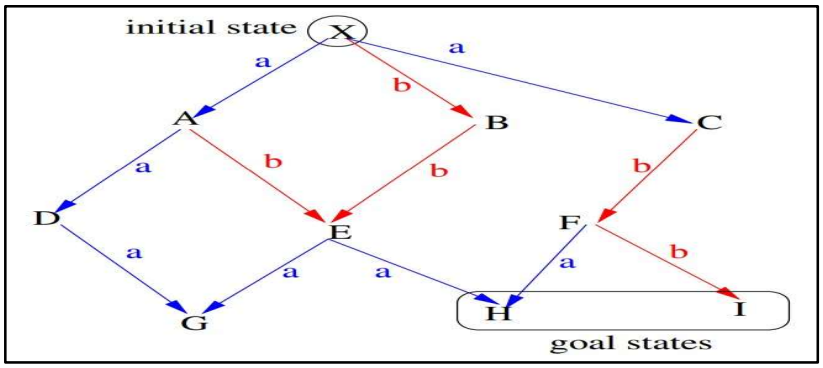
1. Start by pushing the original target onto the stack. Repeat this until the pile is empty. If the stack top is a mixed target, push its unsatisfied sub-targets onto the stack.
2. If the stack top is a single unsatisfied target, replace it with action and push the action precondition to the stack to satisfy the condition.

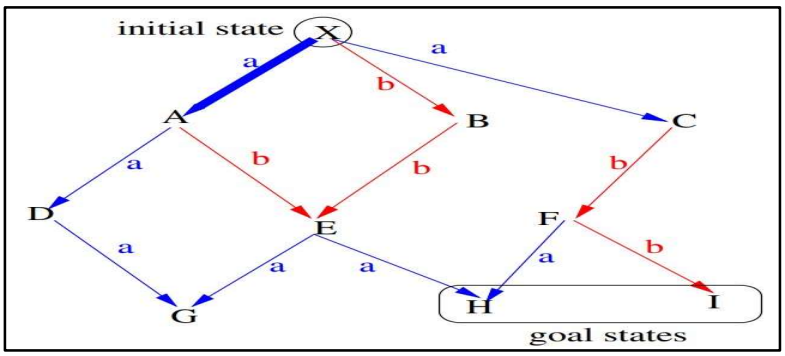
iii. If the stack top is an action, pop it off the stack, execute it and replace the knowledge base with the action's effect.

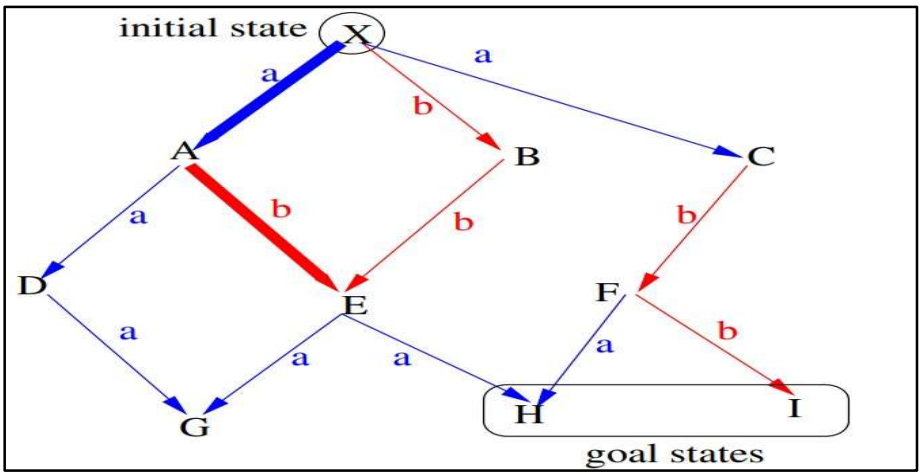
If the stack top is a satisfactory target, pop it off the stack.

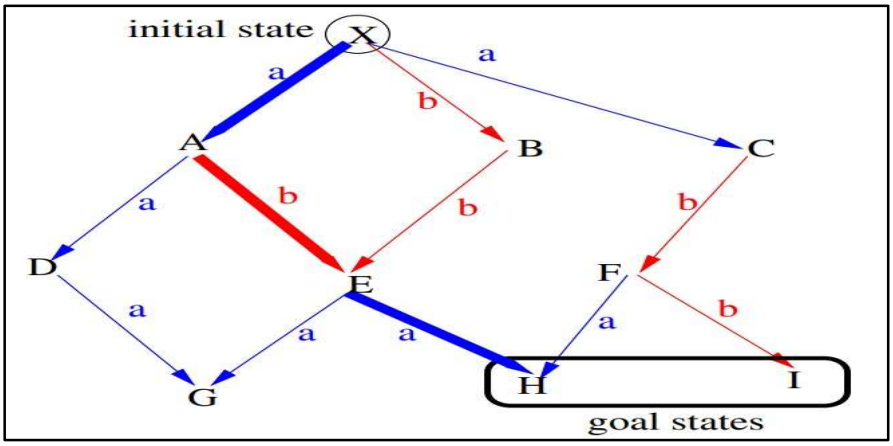
➢ Planning Through State Space Search

- We can view planning problems as searching for goal nodes in a large labeled graph (transition system)
- Nodes are defined by the value assignment to the fluents = states
- Labeled edges are defined by actions that change the appropriate fluents
- Use graph search techniques to find a (shortest) path in this graph! Note: The graph can become huge: 50 Boolean variables lead to $2^{50} = 10^{15}$ states
- Create the transition system on the fly and visit only the parts that are necessary






initial state
X
a
a
b
A
B
C
a
b
b
b
D
E
F
a
a
a
a
b
G
H
I
goal states


initial state
X
a
a
b
A
B
C
a
b
b
b
D
E
F
a
a
a
a
b
G
H
I
goal states

Let's Consider a planning problem and try to solve it with Forward state space search planning and Backward state space search planning.

- **Forward state space search planning**

Search through transition system starting at initial state

1. Initialize partial plan $\Delta := \langle\,\rangle$ and start at the unique initial state $\mathbf{I}$ and make it the current state S
2. Test whether we have reached a goal state already: $\mathbf{G} \subseteq S$? If so, return plan Δ.
3. Select one applicable action o_i non-deterministically and
   - compute successor state $S := App(S, o_i)$,
   - extend plan $\Delta := \langle \Delta, o_i \rangle$, and continue with step 2.

Instead of non-deterministic choice use some search strategy.
Progression planning can be easily extended to more expressive planning languages

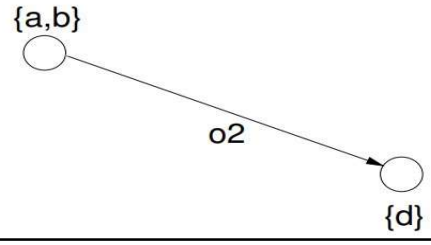
Problem:

$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
&\quad\ o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
&\quad\ o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$

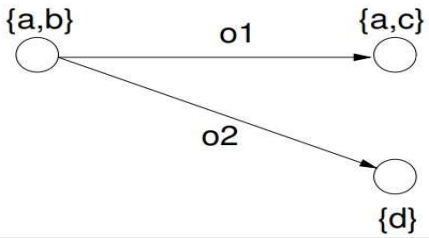
{a,b}

Solution:

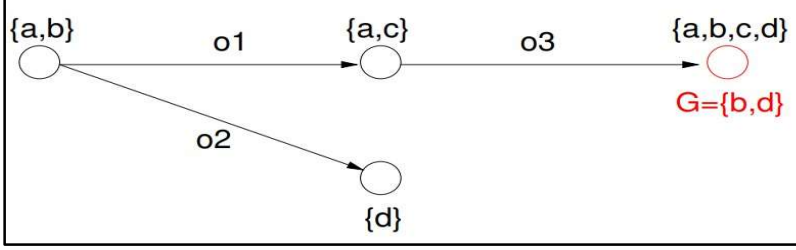
$$\begin{aligned}\mathcal{S} &= \{a, b, c, d\},\\ \mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\}\rangle,\\ &\quad\ o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\}\rangle,\\ &\quad\ o_3 = \langle \emptyset, \{c\}, \{b, d\}\rangle,\\ \mathbf{I} &= \{a, b\}\\ \mathbf{G} &= \{b, d\}\end{aligned}$$



$$\begin{aligned}\mathcal{S} &= \{a, b, c, d\},\\ \mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\}\rangle,\\ &\quad\ o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\}\rangle,\\ &\quad\ o_3 = \langle \emptyset, \{c\}, \{b, d\}\rangle,\\ \mathbf{I} &= \{a, b\}\\ \mathbf{G} &= \{b, d\}\end{aligned}$$



$$\begin{aligned}\mathcal{S} &= \{a, b, c, d\},\\ \mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\}\rangle,\\ &\quad\ o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\}\rangle,\\ &\quad\ o_3 = \langle \emptyset, \{c\}, \{b, d\}\rangle,\\ \mathbf{I} &= \{a, b\}\\ \mathbf{G} &= \{b, d\}\end{aligned}$$



- **Backward state space search planning Problem:**

Search through transition system starting at goal states. Consider sets of states, which are described by the atoms that are necessarily true in them

1. Initialize partial plan $\Delta := \langle\,\rangle$ and set $\mathbf{S} := \mathbf{G}$
2. Test whether we have reached the unique initial state already: $\mathbf{I} \supseteq \mathbf{S}$? If so, return plan Δ.
3. Select one action o_i non-deterministically which does not make (sub-)goals false ($\mathbf{S} \cap \neg \mathit{eff}^-(o_i) = \emptyset$) and
   - compute the regression of the description $\mathbf{S}$ through o_i:

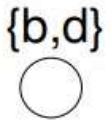
$$\mathbf{S} := \mathbf{S} - \mathit{eff}^+(o_i) \cup \mathit{pre}(o_i)$$

   - extend plan $\Delta := \langle o_i, \Delta \rangle$, and continue with step 2.

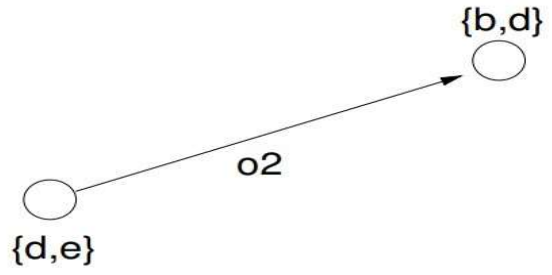
Instead of non-deterministic choice use some search strategy
Regression becomes much more complicated, if e.g. conditional effects are allowed. Then the result of a regression can be a general Boolean formula

Solution:

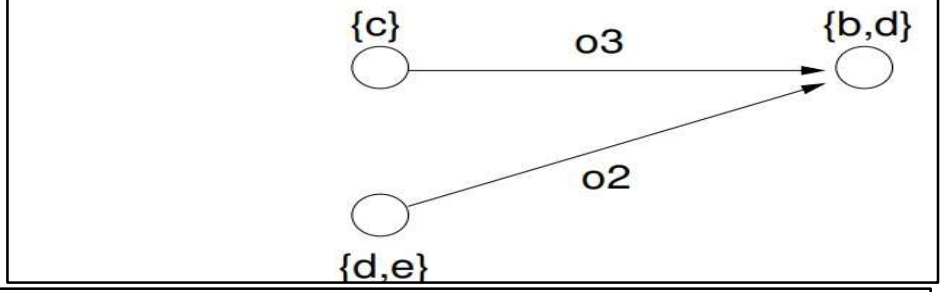
$$\mathcal{S} = \{a, b, c, d, e\},$$
$$\mathbf{O} = \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle,$$
$$o_2 = \langle \emptyset, \{e\}, \{b\} \rangle,$$
$$o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle,$$
$$\mathbf{I} = \{a, b\}$$
$$\mathbf{G} = \{b, d\}$$



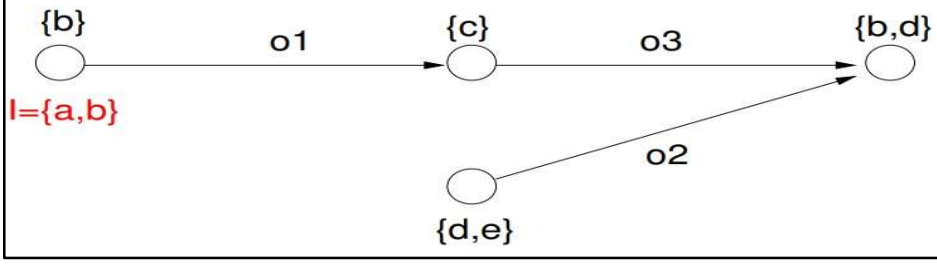
$$\mathcal{S} = \{a, b, c, d, e\},$$
$$\mathbf{O} = \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle,$$
$$o_2 = \langle \emptyset, \{e\}, \{b\} \rangle,$$
$$o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle,$$
$$\mathbf{I} = \{a, b\}$$
$$\mathbf{G} = \{b, d\}$$



$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
&\quad\ o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
&\quad\ o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
&\quad\ o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
&\quad\ o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &= \{b, d\}
\end{aligned}
$$



Conclusion: -Thus we have successfully performed study on planning problem.

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)

A.Y. 2022-2023

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 9

Aim: Neural Network Perceptron Learning

Theory:

Perceptrons are a type of artificial neuron that predates the sigmoid neuron. It appears that they were invented in 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory.

A perceptron can have any number of inputs, and produces a binary output, which is called its activation.

First, we assign each input a weight, loosely meaning the amount of influence the input has over the output.
To determine the perceptron's activation, we take the weighted sum of each of the inputs and then determine if it is above or below a certain threshold, or *bias, *represented by b.

The formula for perceptron neurons can can be expressed like this:

Algorithm:

```
 def perceptron(inputs, bias)
weighted_sum = sum {
 for each input in inputs
  input.value * input.weight
 }

 if weighted_sum <= bias
  return 0
 if weighted_sum > bias
  return 1

end
```

Code:

```
def sgn(net_input):
    if net_input <= 0 :
        return -1
    return 1

def pattern_classifier(n_iterations, input, weight, desired_output, learning_rate):
    for iteration in range(n_iterations):
        print(f'Iteration {iteration+1}')
        output = []
        for i,X in enumerate(input):
            net_input = 0
            for j in range(len(X)):
                net_input+=weight[j]*X[j]
            generated_output = sgn(net_input)
            output.append(generated_output)
            if generated_output != desired_output[i]:
                difference = desired_output[i] - generated_output
                for position in range(len(weight)):
                    weight[position] = float("{:.2f}".format(weight[position] +
learning_rate*difference*X[position]))
        print(f'Generated Output vector for Iteration {iteration+1} : {output}')
        print(f'Weight vector after Iteration {iteration+1} : {weight}')
        print("------"*25)
        if output == desired_output:
            break
    return output,weight

def main():
    input = [
        [1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1], #L starts here
        [1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,1,1],
        [1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1],
        [0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,1,1],
        [1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,1,1,1],
        [0,1,0,0,0,0,1,0,0,0,1,0,0,0,0,1,1,1,1,1],
        [0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1],
        [1,0,0,0,0,1,0,0,0,0,1,0,1,0,0,1,1,0,1,1],
        [0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,1,1,0,1,1],
        [1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,1,1],
```

```
    [0,1,0,1,0,1,1,0,1,1,1,0,1,0,1,1,0,0,0,1], #M starts here
    [1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,0,0,1],
    [1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,1,0,1],
    [1,1,0,1,1,1,0,1,0,1,1,0,1,0,1,1,0,0,0,1],
    [1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1],
    [1,0,0,0,1,1,1,0,1,1,1,0,0,0,1,1,0,0,0,1],
    [1,0,0,0,1,1,1,0,1,1,1,1,0,1,1,1,0,1,0,1],
    [1,1,0,1,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1],
    [1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,0,0,0,0,0],
    [1,0,0,0,1,1,1,1,1,1,1,0,1,0,1,1,0,0,0,1],
  ]
  desired_output = [1,1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
  initial_weight = [1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1]
  learning_rate = 0.05
  n_iterations = 3

  classification_output, weight_vector = pattern_classifier(n_iterations, input,
initial_weight, desired_output, learning_rate)

  count = 0
  for i, output in enumerate(classification_output):
    if output == desired_output[i]:
      count+=1

  accuracy = (count / len(input))*100

  print(f'Accuracy of Classifier : {accuracy} %')

  print('Classifying an Unknown Sample of L (Output = 1)')
  unknown_sample = [1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,0]
  print('Unknown Sample : ',unknown_sample)
  net_input=0
  for i in range(len(unknown_sample)):
    net_input+=weight_vector[i]*unknown_sample[i]
  predicted_output = sgn(net_input)
  print('Predicted Output : ', predicted_output)
  print("\n")

main()
```

Output:

```
Iteration 1
Generated Output vector for Iteration 1 : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1]
Weight vector after Iteration 1 : [0.2, 0.6, 0.0, 0.6, 0.2, -0.9, 0.4, 0.6, -0.6, 0.1, 0.1, -0.1, 0.4,
0.9, -0.9, 0.1, 1.0, -0.3, 1.0, 0.1]
---------------------------------------------------------------------------------------------------------------------------
Iteration 2
Generated Output vector for Iteration 2 : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1,
-1]
Weight vector after Iteration 2 : [0.1, 0.5, 0.0, 0.5, 0.1, -1.0, 0.4, 0.5, -0.6, 0.0, 0.0, -0.1, 0.3,
0.9, -1.0, 0.0, 1.0, -0.3, 1.0, 0.0]
---------------------------------------------------------------------------------------------------------------------------
Iteration 3
Generated Output vector for Iteration 3 : [1, 1, 1, 1, -1, 1, 1, 1, 1, 1, -1, -1, -1, 1, -1, -1, -1, -1, -1,
-1]
Weight vector after Iteration 3 : [0.1, 0.4, 0.0, 0.4, 0.0, -1.0, 0.4, 0.4, -0.6, -0.1, 0.0, -0.1, 0.2,
0.9, -1.0, 0.0, 1.1, -0.2, 1.1, 0.0]
---------------------------------------------------------------------------------------------------------------------------
Accuracy of Classifier : 90.0 %
Classifying an Unknown Sample of L (Output = 1)
Unknown Sample :  [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0]
Predicted Output :  1
```

Conclusion:

Neural Network Perceptron Learning implementation.

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)

ARTIFICIAL INTELLIGENCE

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 10

AIM: Case study of an AI Application.

Paper Link: https://ieeexplore.ieee.org/abstract/document/9325622

Introduction:

This paper is a comparative study for deep reinforcement learning with CNN, RNN, and LSTM in autonomous navigation. For the comparison, a PyGame simulator has been used with the final goal that the representative will learn to move without hitting four different fixed obstacles. Autonomous vehicle movements were simulated in the training environment and the conclusion drawn was that the LSTM model was better than the others.

Approach:

The research is wholly based on reinforcement learning which is a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones. This method assigns positive values to the desired actions to encourage the agent and negative values to undesired behaviors. This programs the agent to seek long-term and maximum overall reward to achieve an optimal solution.

These long-term goals help prevent the agent from stalling on lesser goals. With time, the agent learns to avoid the negative and seek the positive. This learning method has been adopted in artificial intelligence (AI) as a way of directing unsupervised machine learning through rewards and penalties.

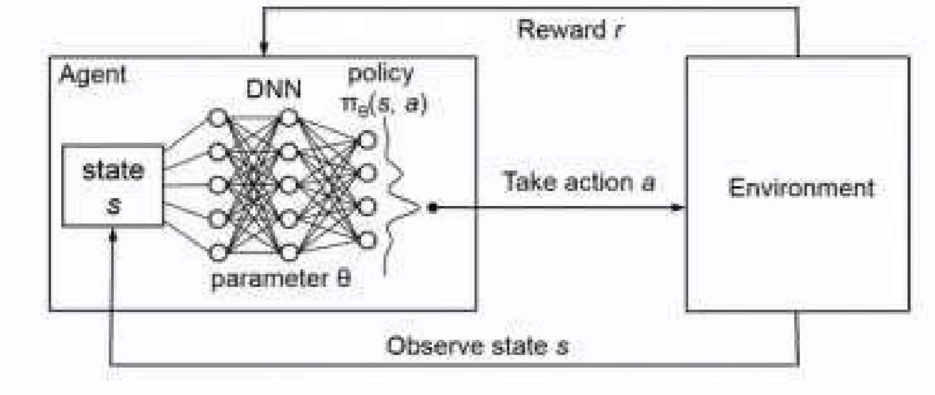
The main advantage of reinforcement learning in this scenario is that unlike deep learning (DL) algorithms, it does not require a data set during the training phase, increasing its popularity and making it

more suitable.

The PyGame simulator interface consists of an agent that learns to move without hitting 4 different randomly positioned obstacles and edges limiting the area. In addition, the paper presents a model-free, off policy approach in this study.

During the research, 4 algorithms were compared. They are:

1) **Deep Q-Network**:It trains on inputs that represent active players in areas or other experienced samples and learns to match those data with desired outputs. This is a powerful method in the development of artificial intelligence that can play games like chess at a high level, or carry out other high-level cognitive activities – the Atari or chess video game playing example is also a good example of how AI uses the types of interfaces that were traditionally used by human agents.

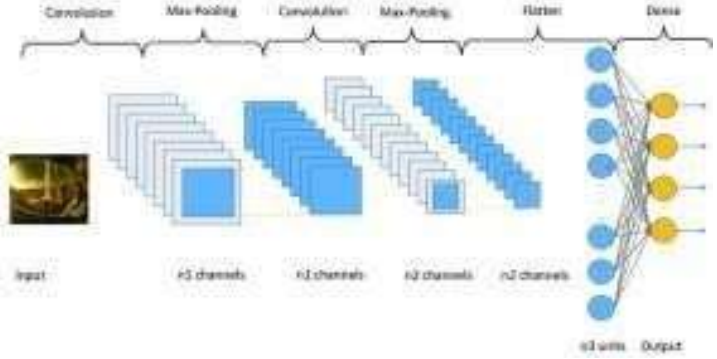


2) **CNN:** A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

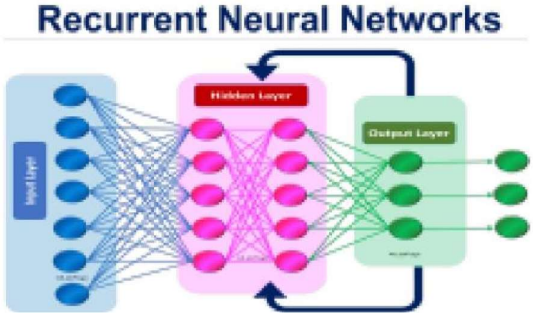
The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

The main purpose of the convolution process is to extract the feature map from the input data.



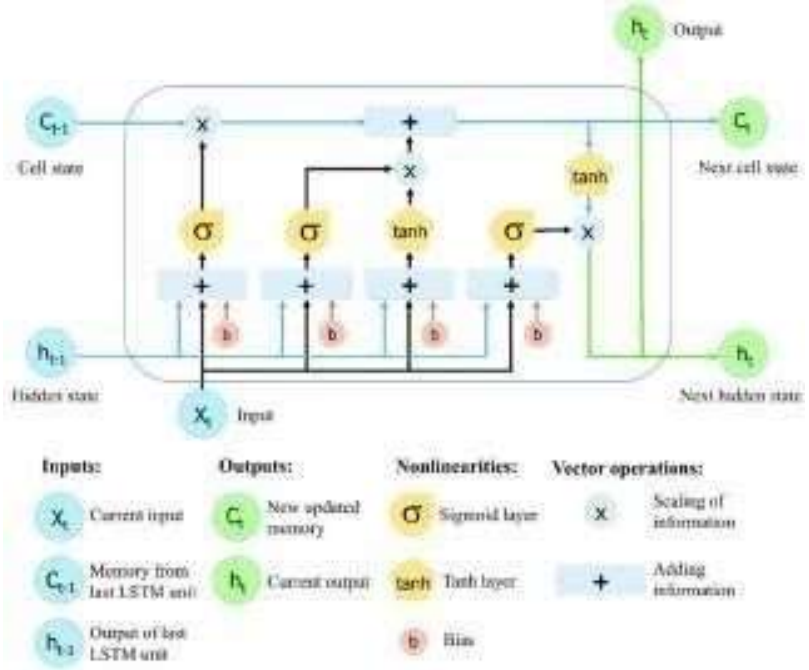
3) **RNN:** Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is the Hidden state, which stores some information about a sequence.



4) LSTM: Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber It tackled the

problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data. LSTM has a chain structure that contains four neural networks and different memory blocks called cells. Information is retained by the cells and the memory manipulations are done by the gates. There are three gates – Forget gate, Input gate and the output gate.

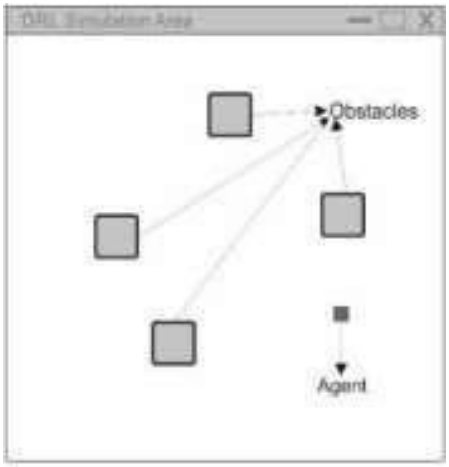
With LSTMs, there is no need to keep a finite number of states from beforehand as required in the hidden Markov model (HMM). LSTMs provide us with a large range of parameters such as learning rates, and input and output biases. Hence, no need for fine adjustments. The complexity to update each weight is reduced to O(1) with LSTMs, similar to that of Back Propagation Through Time (BPTT), which is an advantage.



SIMULATION:

In this work, PyGame library was used as a robot simulation

environment. 4 different obstacles were placed randomly in a 360 * 360 pixel area and the agent was allowed to float within the specified area without hitting these obstacles as shown below.



The agent loses -150 points when hitting obstacles during the learning phase and -50 points when it hits walls. It gets +2 points for every step where it does not hit walls and obstacles.

Four different actions in this simulation are shown in the table below.

Num.	Action
0	go to the left
1	go to the right
2	go to the up
3	go to the bottom

CONCLUSION:

Multiple deep learning algorithms were separately tested on the PyGame simulation interface and the conclusions were drawn. The first conclusion was that even though deep reinforcement learning (DRL) models provide fast and safe solutions for autonomous vehicles, their training time is very high. After training it was observed that RNN and LSTM,which are generally used to solve language processing problems, can also be successful in such autonomous navigation problems. The second conclusion was that while the LSTM model took the maximum time to train, it showed the highest success in the success-episode graphics. The paper then concludes with saying that this is a very rich field in terms of future research prospects.

Name : Ayush Jain
SAP ID : 60004200132
Div : TE B2
Computer Engineering

AI - Assignment 1

Q. 1) Write a short note :

b) Sensorless Planning :

→ 1) It is a kind of planning that is based on any perception. The algorithm ensures that plan should reach its goal at any cost.

2) Sensorless planning is also known as conformant planning. For the environment with no observations.

3) In these the problem of finding a sequence of actions for achieving a goal in presence of uncertainity in the initial state or action effects.

4) It works in no observations environment so it search the belief-state space to find the solution for sensorless problem rather than physical state.

5) This agent not reliable on sensor data and then it comes up with the plan that works in all possible case.

example : 1) You have a wall made from bricks.
(2) You have a can of white paints.

- Action : paint (brick), effect : colour (brick)
- Goal : Every brick should be painted white
- Suppose world isn't fully observable, we actually cannot observe the brick colour.

(c) Multiagent planning:

→ 1) In multiple agent environment each agent faces a multiagent planning problem in which it tries to achieve its goal.

2) It involves co-ordinating resources and activities of multiple agents.

3) It can involve agent for planning for a common goal or agent co-ordinating the plans for planning of others or on agent refining their own plans while negotiating over tasks or resources.

4) If new agent are introduced to a single agent environment, but single agent does not change its basic algorithm then it may perform poorly.

5) Agents are not in different to another agents intentions (like nature is). So agents can co-operate, complete or co-ordinate.

6) Sometimes distribution computations are easier to understand and develop.

7) Joint plans can be constructed, but must be aggregated with some form of co-ordination. if two agents are to agree on which joint plan to execute.

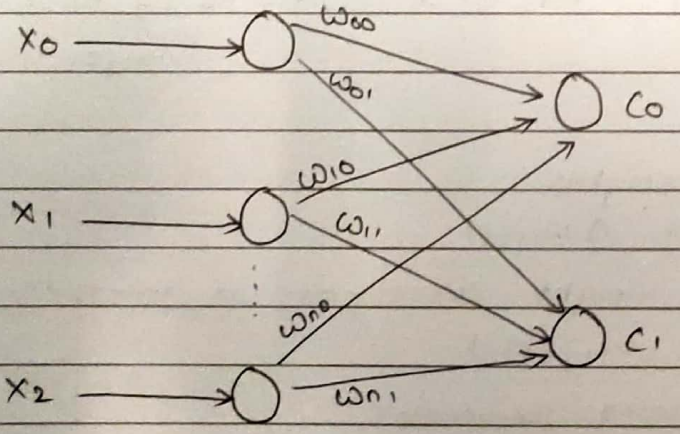
8) Example: In double tennis problem if each agent uses different plan then neither will return the ball.

9) Having a correct joint plan doesn't guarantee success the agent needs to arrive at some joint plan.

Q. 2) Write a short note on Self Organizing Map.

→

1) Self Organizing Map (SOM) is type of neural network inspired by bilogical neural networks in 1920.
2) It follows unsupervised learning approach and trained it network through a competitive learning algorithm.
3) SOM is used for clustering and mapping technique to map multidimensional data onto the lower-dimensional which allows people to reduce complex problem for easy interpretation.
4) SOM consists of two layers Input and output layers:



5) Each neuron in an SOM is assigned weight vector with the same dimensionality N as the input space.
6) Any given input pattern is compared to the weight vector of each neuron and closest to output weight, neuron is declared as winner.
7) The euclidean norm is commonly used to measure distance.

B) Algorithm:

Step 1: Initialization

Set initial synaptic weight to small random value between [0,1] and assigned small positive value to the learning parameters α.

Step 2: Activation and Similarity matching:

Activate network by applying input vector x. Find winner takes all (best neuron) neuron J_x at iteration p,

using euclidean distance.

$$J_x(p) = \min_{j=1}^{n} [x - w_p(x)]$$

nx no. of neuron in input layer.

Step 3: Learning

Update synaptic weights.

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

where $\Delta w_{ij}(p)$ = weight correction at interaction p

$$\Delta w_{ij}(p) = \eta [x_i - w_{ij}(p)]$$

↳ learning parameter.

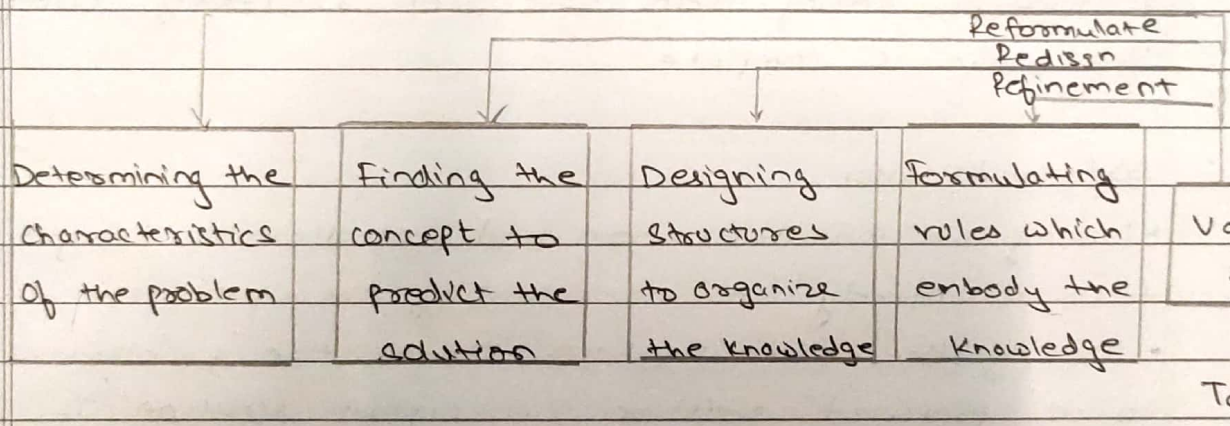
Step 4: Iteration:

Increase iteration p by 1, go back to step 2 and continue until the minimum-distance Euclidean criteria is satisfied, or no noticable changes occurs in the feature map.

Q. 3) Explain phases in building Expert Systems with ES Architecture in detail.

→ The following are stages for developing expert system:

1) Identification.
2) Conceptualisation.
3) Formalisation
4) Implementation
5) Testing (Validation, verification, maintenance)



Five stages of Expert System Development

(1) Identification:

- Before we can begin to develop on expert system, it is important to describe, with as much precision as possible, the problem which the system is intended to solve.
- It is also important to identify our resources domain experts and information such as reference books and manuals are usually located.

(2) Conceptualization:

- Once it has been identified for the problem an expert system is to solve the next stages involves analysing the problem further to ensure that it specifies as well as generalities are understood.
- This stage involves a circular procedure of iteration and re-iteration between the knowledge engineers and the domain expert.
- When both agree that the key concepts and the relationships among them have been adequetly conceptualised, this stage is complete.

(3) Formulaisation:

- During the indentification and formalisation stages, the focus is entirely on understanding the problem.
- During the formalisation stage, the problem is connected to its proposed solution, an expert solution is supplied by analyzing the relationships depicted in the conceptualization stage.

(4) Implementation

- In these stages the formalization concepts are programmed into the computers which has been chosen for system development, using the predetermined techniques and tools to implement 'first-pass' (prototype) of the system.
- If the prototype works at all, the knowledge engineer may be able to determine if the technique is chosen to implement the expert system were the appropriate.

- Once the prototype system has refined sufficiently to allow it to be executed, the expert system is ready to be tested thoroughly to ensure that it expertise's correctly.

5) Testing :

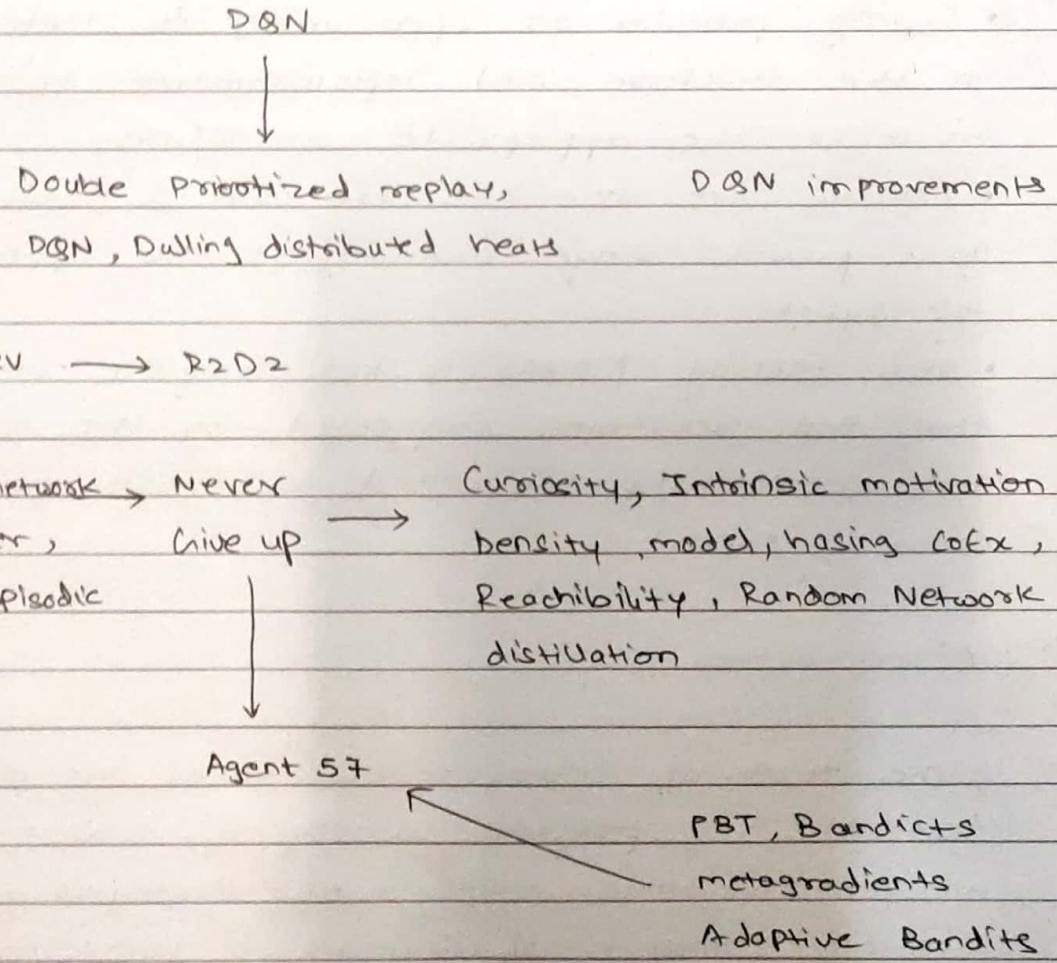
- Testing provides an opportunity to identify the weakness in the structure and implementation of the system and to make the appropriate correction.
- Results from the test are used to 'feedback' to return to a previous stage and adjust the performance of the system.
- The testing process is not complete until it indicates that the solutions suggested by the expert system are consistently as valid as those provided by human domain expert.

Q. 4) Atari Games :

→ 1) The truth of atari video games are a great way to test AI. They provide variety of challenges that forces an AI to dense number of strategies yet they have a clear measure of success, a score to test against.
2) The AI agent called agent 57 has learned to play all 57 atari video game in arcade learning environment- a collection of classic games, that researcher use to test the limit of their deep learning agent.
3) Developed by deep mind agent 57 uses the same deep

reinforcement learning algorithm to achieve superhuman game play levels even in the game that previous one's struggle with.

- Games such as Montezyma's revenge and pitFall require extensive exploration to obtain ~~guard~~ good performance.



- Training an AI to excel AI more than one taste of biggest open challenge in deep learning. The ability to learn 57 different tasks make agent 57 more versatile than previous AI game but it is still can't learn to play more than one game at a time.

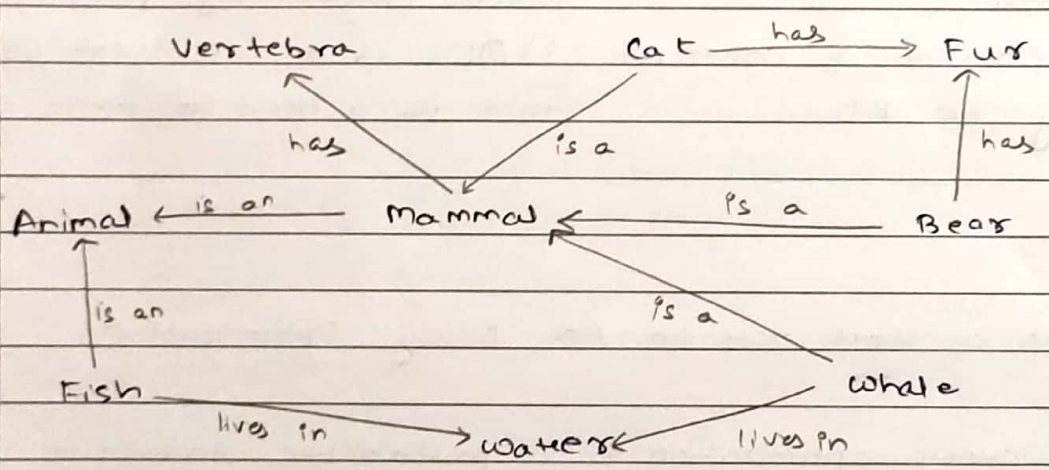
Name: Ayush Jain
SAP ID: 60004200132
Div: B2
Computer Engineering

AI- Assignment 2

Q. 1) Write a short note on:

(i) Semantic Networks

→
- AI agents have to store and organize information in their memory.
- One of the ways they do is by using semantic Networks. They are a way of representing relationships between objects and ideas.
- For example, a network might tell a computer the relationship between different animals. (a cat is a mammal, a cat is a whiskers)
- example diagram:



ii) Compare RDF and OWL

→

RDF	OWL
1) It stands for Resource description framework.	1) It stands for Web Ontology language.
2) It is special framework found online that is tasked with the representation of online exchange of data.	2) OWL is a special language used in the description of ontologies online.
3) RDF refers to only the structure of data as it is available.	3) OWL refers to different semantic relationships of which bring in various programming practices.
4) Exploration of content easly on RD	4) OWL is an excellent solution when there is a need to make implicit ref.

Q.2) Write a short note on Ant Colony Optimization.

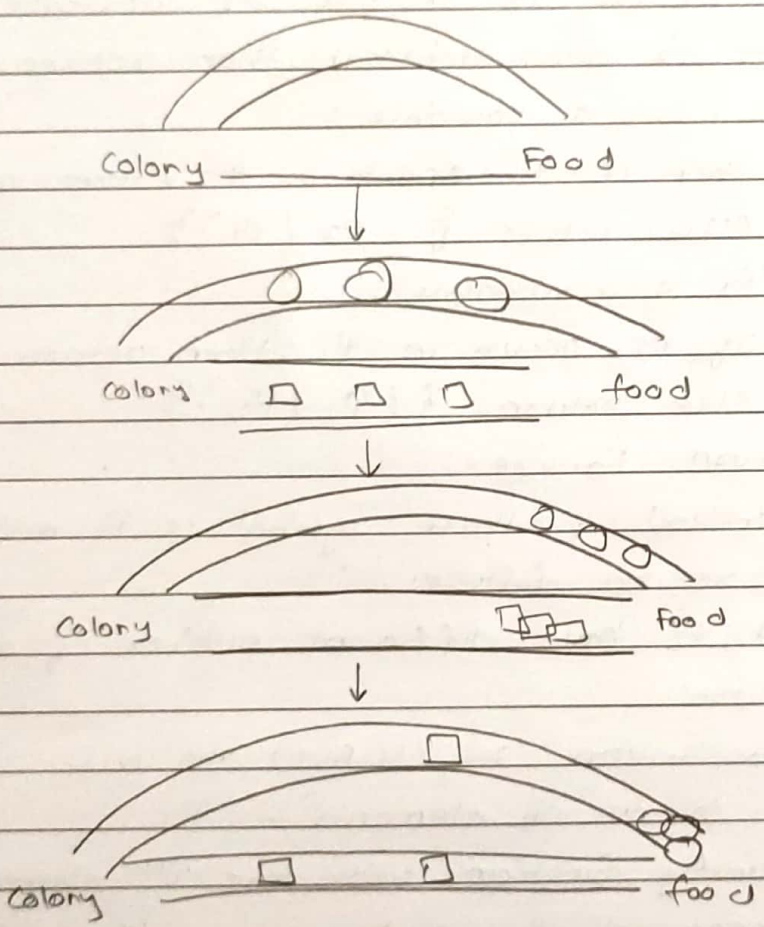
→

1) Ant colony optimization is a probablistic technique for finding optimal paths. In CS, the ant colony optimization algorithm is used for solving different computational problems.
2) This algorithm is introduced based on the behaviour of an ant seeking a path between their colony and a source food.
3) Initially, it was used to solve problems like TSP. It is also used to solve optimization problems.
4) While searching ants roaming around their colonies. An ant

repeatedly hops from one place to another to find the food.

5) While searching, it deposits an organic compound called pheromone on the ground.

6) When returning its deposits pheromone on the paths based on the quantity and quality of the food.



Q. 3) Explain Unification algorithm

→ • Unification is the process of finding a substitute that makes two seperate logical atomic expression identical.

• Unification Algorithm:

(1) If ψ_1 or ψ_2 is a var or constant then:

a) If ψ_1 or ψ_2 are identical, then return NIL

b) Else if ψ is a variable:

(a) then if ψ_1 occurs in ψ_2, then return Failure

(b) Else return $\{(\psi_2 / \psi_1)\}$

c) Else if ψ_2 is a variable:

(a) If ψ_2 occurs in ψ_1, then return Failure.

(b) Else return $\{(\psi_1 / \psi_2)\}$

d) Else return Failure.

(2) If the initial predicate symbol is ψ_1 and ψ_2 are not same, then return failure.

(3) If ψ_1 and ψ_2 have different number of arguments, then return failure.

(4) Set Substitution set (SUBST) to NIL

(5) For i=1 to no. of elements in ψ_1

(a) Call unify function with the i^{th} element of ψ_1 and i^{th} element of ψ_2, and put result in S.

(b) If S = Failure then return failure.

(c) If S ≠ NILL then

(a) Apply S to the remainder of both L_1 and L_2

(b) SUBST = APPEND(S, SUBST)

(6) Return Subset

Example:

UNIFY (knows (Richard, x) ; knows (Richard , John))

Here ψ_1 = knows (Richard, x)

ψ_2 = knows (Richard, John)

So $\Rightarrow$: { knows (Richard, x), knows (Richard, John) }

$S_1 \Rightarrow$ { knows (Richard, John) ; knows (Richard, John) }

Successfully unified.

Unifier : { John / x }

Q.4) Explain 'Bayesian Belief Network'

→ 1) Bayesian belief Network is a key computer technology for dealing with probablistic events and to solve a problem which has uncertainity.

2) A Bayesian Network is a probabilistic graphical model which represents a set of vars and their conditional dependencies using a directed acyclic graph

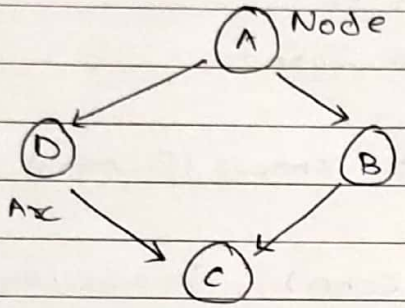
3) Bayesian networks are probabilistic because these networks are busy from probability distributions and also use probability theory for prediction.

4) It can also be used in various tasks including predictions, anomally detection, diagnostics, automated insights, reasoning, time series predictions and decision making under uncertainity.

5) It consist of two parts :

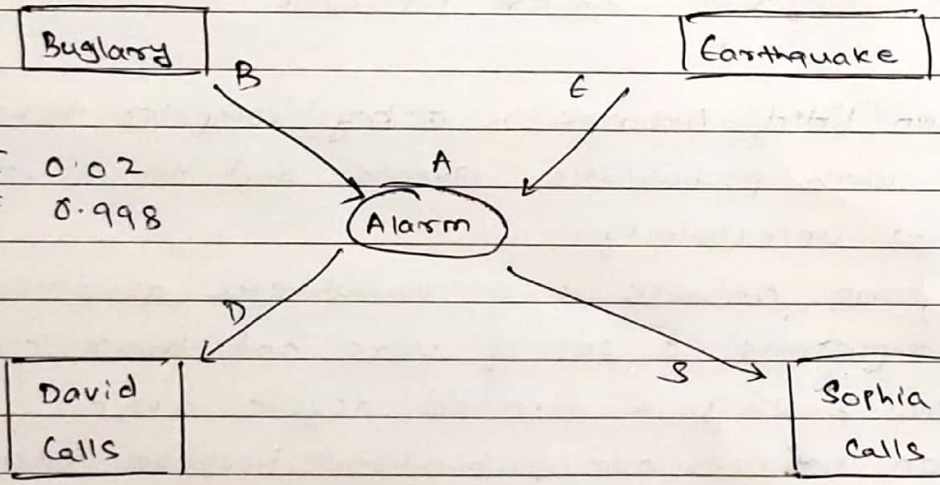
(1) Directed Acyclic graph.

(2) Table of conditional probabilities.



Example :

Calculate the probability that alarm has sounded, but there is neither a burglary nor an earthquake occurred, and David and Sophia both called the theory.



B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P(D=T)	P(D=F)
T	0.91	0.09
F	0.05	0.95

A	P(S=T)	P(S=F)
T	0.75	0.25
F	0.02	0.98

5) Define Fuzzy set. Explain various Fuzzy set operations with suitable example.

→ i) Fuzzy set is a set having degrees of membership between 1 and 0. Fuzzy set are represented with ~ character. For example, Number of cars following traffic signals at a particular out of all cars present will have membership value between [0,1].

2) Partial membership exists when members of one fuzzy set can also be part of other fuzzy set in the same universe.

3) A fuzzy set $\tilde{A}$ in the universe of disclosure, U can be defined as a set of ordered pairs and it is given by

$$\tilde{A} = \{ (x, \mu_{\tilde{A}}(x) \mid x \in X \}$$

When the universe of disclosure, μ is discrete of finite, fuzzy set $A\sim$ is given by:

$$\tilde{A} = \sum_i \frac{\mu_{\tilde{A}}(x_i)}{x_i} \quad , \quad \tilde{A} = \int \frac{\mu_{\tilde{A}}(x)}{x}$$

4) Fuzzy set operations:

a) Union:

This operations combines two fuzzy sets into once, taking the max value of each element from the two sets.

Example, consider 2 fuzzy set.

$A = \{0.3, 0.7, 0.9\}$

$B = \{0.4, 0.6, 0.8\}$

$A \cup B = \{0.4, 0.7, 0.9\}$

(b) Intersection:

This operation takes the minimum value of each element from 2 fuzzy set. Using same sets from above,

$A \cap B = \{0.3, 0.6, 0.8\}$

(c) Complement:

This operation inverts the membership values of a fuzzy set, so that elements that were previously members have zero membership and elements that were not membership have a membership value of 1

Example: $C = \{0.2, 0.5, 0.8\}$

$C' = \{0.8, 0.5, 0.2\}$

(d) Algebraic Sum:

This operation adds the membership values of corresponding elements two fuzzy sets.

Using A and B from before:

$A \oplus B = \{0.7, 1.3, 1.7\}$

(e) Algebraic Product:

This operation multiplies the membership values of corresponding elements in two fuzzy sets.

Using A & B from before:

$A \otimes B = \{0.12, 0.42, 0.72\}$