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)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2023-24

Name: Shashwat Shah SAP ID: 60004220126

Course: Artificial Intelligence Laboratory Course Code: DJ19CEL503

Year: T. Y. B. Tech. Sem: V Batch: C2-2

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	Σ Avg	Avg 1	Avg 2	Σ Avg
Course Outcome	1	2	2	2	2	4	3	3	3	5		3, 4	3, 5	
1. Knowledge (3) (Factual/Conceptual/Procedural/ Metacognitive)	3	3	3	3	2	2	2	3	3	3	3	3	3	3
2. Describe (3) (Factual/Conceptual/Procedural/ Metacognitive)	3	2	2	3	3	3	-	-	3	2	2	3	2	3
3. Demonstration (3) (Factual/Conceptual/Procedural/ Metacognitive)	2	3	2	3	2	3	3	3	2	2	3	2	2	2
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive) (3)	3	3	3	3	3	3	3	3	3	2	3	2	2	2
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive) (3)	-	-	-	-	-	-	3	2	-	-		-	-	
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value) (3)	3	3	3	2	3	3	3	3	3	3	3	3	3	3
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Total	14	14	13	14	13	14	14	14	14	12	14	13	12	13
Signature of the faculty member														

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks 14 Σ Avg. =	Assignment marks 13 Σ Avg. =	Total Term-work (25) = 24
Laboratory Scaled to (15) = 14	Assignment Scaled to (10) = 09	Sign of the Student: Shashwat

Signature of the Faculty member:
Name of the Faculty member: Dr Chetashri B

Signature of Head of the Department
Date: 5/12/23

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

AIM: Select a Problem Statement relative to AI

1) Identify the problem
2) PEAS description
3) Problem formulation

PART A)

Write the problem faced in each one of them

1) **AGENT**: Chess playing with a clock
    a. **Performance Measure:**
        i. Win:Lose ratio
        ii. Speed
        iii. Total game time
    b. **Environment:**
        i. Chessboard
        ii. Clock
    c. **Environment Type:**
        i. Fully observable
        ii. Discrete
    d. **Actuator:**
        i. Pausing of the clock
        ii. Movement of chess pieces
    e. **Sensors:**
        i. Movement arm
        ii. Servo Motors
        iii. Location of chess pieces using reed switches.
    f. **Problem**

2) **AGENT**: Driving a car
    a. **Performance Measure:**
        i. Speed
        ii. Time Taken

iii. Comfort
iv. Fuel Economy

b. **Environment:**
   i. Car
   ii. Road
   iii. Traffic
   iv. Signposts
   v. Potholes
c. **Environment Type:**
   i. Partially Observable
d. **Actuator:**
   i. Steering Wheel
   ii. Brake
   iii. Accelerator
   iv. Mirror
   v. Gearstick
e. **Sensors:**
   i. GPS
   ii. Odometer
   iii. Speedometer
   iv. Fuel tank capacity meter
f. **Problem**

3) **AGENT**: Interactive English tutor
   a. **Performance Measure:**
      i. Language Improvement
      ii. Increase in test score
      iii. Number of errors made per paragraph
   b. **Environment:**
      i. Classroom
      ii. Table
      iii. Chair
      iv. Students
      v. Whiteboard
      vi. Books
   c. **Environment Type:**
      i. Deterministic
   d. **Actuator:**
      i. Writing on whiteboard
      ii. Opening and reading the books

iii. Checking of test papers

e. **Sensors:**
   i. Eyes
   ii. Ears
   iii. Books
   iv. Test Papers
f. **Problem**

4) **AGENT**: Part picking robot
   a. **Performance Measure:**
      i. % Efficiency of the robot
   b. **Environment:**
      i. Parts
      ii. Conveyer belt
   c. **Environment Type:**
      i. Collaborative
   d. **Actuator:**
      i. Picking up the parts
      ii. Sorting the parts
   e. **Sensors:**
      i. Camera
      ii. Robot Arm
      iii. Distance Sensor
      iv. Servo motors
   f. **Problem**

5) **AGENT**: Satellite Image Analysis System
   a. **Performance Measure:**
      i. % Correct Analysis
      ii. Time taken
   b. **Environment:**
      i. Camera
   c. **Environment Type:**
      i. Dynamic
   d. **Actuator:**
      i. Capturing of Images
      ii. Movement of the satellite
   e. **Sensors:**
      i. Camera
      ii. Color Sensor

f. **Problem**

6) **AGENT**: Medical Diagnosis System
   a. **Performance Measure:**
      i. % of correct diagnosis
      ii. Time taken
      iii. Systems correctly found
      iv. Number of lawsuits
      v. Cost
   b. **Environment:**
      i. Patient
      ii. Hospital
   c. **Environment Type:**
      i. Deterministic
   d. **Actuator:**
      i. Asking questions
      ii. Recommending further tests
      iii. Printing reports
      iv. Dispensing medicines
   e. **Sensors:**
      i. Camera
      ii. Microphone
      iii. Speaker
      iv. Printer
   f. **Problem**

7) **AGENT**: Refinery Controller
   a. **Performance Measure:**
      i. % Efficiency
      ii. Speed
   b. **Environment:**
      i. Refinery workers
      ii. Machines
   c. **Environment Type:**
      i. Collaborative
   d. **Actuator:**
      i. Turn on/off systems
      ii. Adjust temperatures
      iii. Adjust pressures
   e. **Sensors:**

i. Temperature sensor
ii. Pressure sensor
iii. Proximity sensor

f. **Problem**

8) **AGENT**: Pokey playing
   a. **Performance Measure:**
      i. Rounds won
      ii. Number of correct moves
   b. **Environment:**
      i. Cards
      ii. Humans
   c. **Environment Type:**
      i. Discrete
   d. **Actuator:**
      i. Dealing the cards
      ii. Playing the cards
   e. **Sensors:**
      i. Camera
      ii. Color sensor
      iii. Servo motor
      iv. Movement Arm
   f. **Problem**

9) **AGENT**: Chatbot
   a. **Performance Measure:**
      i. Time taken
      ii. Grammatical accuracy
   b. **Environment:**
      i. Chatbot
      ii. Human
   c. **Environment Type:**
      i. Continuous
   d. **Actuator:**
      i. Displaying the questions
      ii. Taking responses
   e. **Sensors:**
      i. Keyboard
      ii. Screen
   f. **Problem**

10) **AGENT**: Soccer playing robot

a. **Performance Measure:**
    i. Number of goals scored
    ii. Number of goals saved
    iii. Number of penalties
    iv. Number of games won
b. **Environment:**
    i. Soccer field
    ii. Goal posts
    iii. Goat net
    iv. Humans (Other players)
c. **Environment Type:**
    i. Continuous
d. **Actuator:**
    i. Movement of the ball
e. **Sensors:**
    i. Servo motors
    ii. Proximity sensors
f. **Problem**

11) **AGENT**: Recommender system

a. **Performance Measure:**
    i. % Efficiency
b. **Environment:**
    i. Dataset
    ii. Input variables
c. **Environment Type:**
    i. **Continuous**
d. **Actuator:**
    i. Creating the algorithm
    ii. Using the algorithm
e. **Sensors:**
    i. Keyboard
    ii. Screen
f. **Problem**

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim: Perform uninformed searching techniques like BFS, DFS, etc.

Theory:

BFS:

Breadth-first search is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. BFS or Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighbouring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

Code:

```
# BFS
graph = {
    'S':['A','B','C'],
    'A':['B','D','S'],
    'B':['S','A','D'],
    'C':['S','G'],
    'D':['A','B','E'],
    'E':['D','G'],
    'G':['C','E']
}

opened = []
closed = []
goal = False
# ----------------------------------------------------------------

def expand(node):
    for x in graph[node]:
        if x not in closed and x not in opened:
            opened.append(x)

opened.append('S')
print(opened)
count = 0
```

```
while count < len(graph)-1 and not goal:
    node = opened.pop(0)

    if node not in closed:
        closed.append(node)

    expand(node)
    print(opened)
    if 'G' in closed:
        goal=True

    count+=1

print(goal)
```

output:

```
['S']
['A', 'B', 'C']
['B', 'C', 'D']
['C', 'D']
['D', 'G']
['G', 'E']
['E']
True
```

DFS:

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and explores as far as possible along each branch before backtracking

It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children. Because of the recursive nature, stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

Code:

```
#DFS

graph = {
    'S':['A','B','C'],
    'A':['B','D','S'],
    'B':['S','A','D'],
    'C':['S','G'],
    'D':['A','B','E'],
```

```
    'E':['D','G'],
    'G':['C','E']
}
opened=[]
close=[]
goal=False


def expand(node):
    for x in graph[node]:
        if x not in opened and x not in close:
            opened.insert(0,x)

opened.append('S')
print(opened,"\t",close)
count=0
while count<len(graph)-1 and not goal:
    node=opened.pop(0)
    if node not in close:
        close.append(node)
    expand(node)
    if 'G' in close:
        goal=True
    print(opened,"\t",close)
    count+=1

print(goal)
```

Output:

```
['S']    []
['C', 'B', 'A']         ['S']
['G', 'B', 'A']         ['S', 'C']
['E', 'B', 'A']         ['S', 'C', 'G']
True
```

Conclusion:

Thus, we successfully studied Uninformed Searching Algorithms like BFS, DFS.

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim: Perform uninformed searching techniques like DFID.

Theory:

DFID:

Depth-First Iterative Deepening (DFID) is a search algorithm used in computer science and artificial intelligence to efficiently explore a search space or a tree-like structure, particularly when searching for a target node or state. This algorithm combines the depth-first search strategy with an iterative approach to gradually increase the depth of exploration. The core idea behind DFID is to perform a depth-first search with an initial depth limit and, if the target node is not found within that limit, incrementally increase the depth limit until the target is discovered. This approach offers the best of both worlds: the memory efficiency of depth-first search and the guarantee of finding the shortest path, similar to breadth-first search. It is particularly useful in situations where the search space is unknown or large, and finding the shortest path to a goal is essential.

In practical applications, DFID is employed in scenarios such as puzzle-solving, game-playing, and route-finding problems, where the optimal solution is sought while minimizing memory usage. It provides a balance between memory and time complexity, ensuring that the solution found is the shortest one. By systematically deepening the search, DFID avoids some of the potential pitfalls of pure depth-first search and provides a systematic way to explore complex problem spaces, making it a valuable tool in the toolbox of search and optimization algorithms.

Code:

```
class Node:
    def __init__(self, val=None):
        self.val = val
        self.left = None
        self.right = None


def get_root():
    values = iter([3, 8, 6, 9, None, None, 11, 10, None, None,
                   12, None, None, 7, None, None, 4, 5, None, None, 13, None,
None])

    def tree_recur(itr):
        val = next(itr)
```

```
        if val is not None:
            node = Node(val)
            node.left = tree_recur(itr)
            node.right = tree_recur(itr)
            return node

    return tree_recur(values)

def dfids():
    root = get_root()
    res = float("inf")

    def dfids_search(node, depth, limit):
        if depth <= limit and node is not None:
            val = node.val
            if val == 12:
                nonlocal res
                res = min(res, depth)
            else:
                dfids_search(node.left, depth + 1, limit)
                dfids_search(node.right, depth + 1, limit)

    for limit in range(1,5):
        dfids_search(root, 0, limit)
        if res < float("inf"):
            return res
    return -1

if __name__ == "__main__":
   print("\nShortest Depth: ", dfids())
```

output:

```
Shortest Depth:  4
```

Conclusion:

Thus, we successfully studied Uninformed Searching Algorithms like DFID.

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim: Identify and analyze informed search Algorithm to solve the problem. Implement A* search algorithm to reach goal state.

Theory:

A*:

A* (pronounced "A-star") is a widely used and effective search algorithm in computer science and artificial intelligence. It is particularly employed in pathfinding and graph traversal problems, where you need to find the shortest path from a start node to a target node while considering the associated costs of moving through the graph or search space. A* is a heuristic search algorithm, which means it uses a combination of actual cost (the cost to reach a node from the start) and an estimated cost (a heuristic value) to make informed decisions during the search. This combination of real and estimated costs allows A* to efficiently explore the search space and find the optimal path while avoiding unnecessary exploration, making it highly efficient and effective.

The key to A*'s success is its ability to balance between completeness and efficiency. By using a heuristic, A* intelligently prioritizes the nodes to explore, favoring those that are likely to lead to the goal node and minimizing the number of nodes evaluated. This makes A* suitable for a wide range of applications, including GPS navigation, robotics, video game pathfinding, and more. The choice of heuristic can significantly impact A*'s performance, and when an admissible heuristic is used (one that never overestimates the cost to the goal), A* is guaranteed to find the optimal path, making it a powerful and versatile tool in solving complex search and optimization problems.

Code:

```
# A*
from collections import deque

class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        return self.adjac_lis[v]

    def heuristic(self, n):
        H = {
```

```
            'S': 15,
            '1': 14,
            '2': 10,
            '3':  8,
            '4': 12,
            '5': 10,
            '6': 10,
            '7': 0
        }

        return H[n]

    def a_star(self, start, stop):

        open_list = set([start])
        closed_list = set([])

        distance = {} # Distance from Start.
        distance[start] = 0

        adjacent_nodes = {} # Adjacent Mapping of all Nodes
        adjacent_nodes[start] = start

        while len(open_list) > 0:
            n = None

            for v in open_list:
                if n == None or distance[v] + self.heuristic(v) < distance[n]
+ self.heuristic(n):
                    n = v

            if n == None:
                print('Path does not exist!')
                return None

            if n == stop: # If current node is stop, we restart
                reconst_path = []

                while adjacent_nodes[n] != n:
                    reconst_path.append(n)
                    n = adjacent_nodes[n]

                reconst_path.append(start)

                reconst_path.reverse()

                print('\nPath found: {}\n'.format(reconst_path))
                return reconst_path
```

```
            for (m, weight) in self.get_neighbors(n): # Neighbours of current
node

                if m not in open_list and m not in closed_list:
                    open_list.add(m)
                    adjacent_nodes[m] = n
                    distance[m] = distance[n] + weight

                else: # Check if its quicker to visit n then m
                    if distance[m] > distance[n] + weight:
                        distance[m] = distance[n] + weight
                        adjacent_nodes[m] = n

                        if m in closed_list:
                            closed_list.remove(m)
                            open_list.add(m)


            open_list.remove(n) # Since all neighbours are inspected
            closed_list.add(n)
            print("OPEN LIST : ", end="")
            print(open_list)
            print("CLOSED LIST : ", end="")
            print(closed_list)
            print("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - -")

        print('Path does not exist!')
        return None


adjacent_list2 = {
    'S': [('1', 3), ('4', 4)],
    '1': [('S', 3), ('2', 4), ('4', 5)],
    '2': [('1', 4), ('3', 4), ('5', 5)],
    '3': [('2', 4)],
    '4': [('S', 4), ('1', 5), ('5', 2)],
    '5': [('4', 2), ('2', 5), ('6', 4)],
    '6': [('5', 4), ('7', 3)],
    '7': [('6', 3)],
}

g = Graph(adjacent_list2)
g.a_star('S', '7')
```

output:

```
OPEN LIST : {'4', '1'}
CLOSED LIST : {'S'}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OPEN LIST : {'5', '1'}
CLOSED LIST : {'4', 'S'}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OPEN LIST : {'1', '2', '6'}
CLOSED LIST : {'5', '4', 'S'}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OPEN LIST : {'2', '6'}
CLOSED LIST : {'5', '4', 'S', '1'}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OPEN LIST : {'3', '6'}
CLOSED LIST : {'1', '2', '4', '5', 'S'}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OPEN LIST : {'6'}
CLOSED LIST : {'3', '1', '2', '4', '5', 'S'}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OPEN LIST : {'7'}
CLOSED LIST : {'3', '1', '2', '4', '6', '5', 'S'}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Path found: ['S', '4', '5', '6', '7']

['S', '4', '5', '6', '7']
```

Conclusion:

Thus we implemented the A* algorithm.

Page No.
Date

Experiment 4

Shashwat Shah
60004220126
Div B C2-2

Aim : To implement local search algorithm: Hill climbing.

Theory : Hill climbing is a local search algorithm in AI to find the best possible solution to a problem. It is named after the idea that it climbs the hills of search space to find the most optimal solution but it can have limitations and may not be always find the global optimum. Current state begins with an initial state and explores the best neighbour iteratively to determine its next move.

Objective function - To evaluate the quality of a state, the goal is to maximize or minimize this function. It guides the searching towards better solutions

Advantages :-

Simplicity - It is straight forward to implement and understand.

Efficiency - It can quickly find local optimum due to its simple structure

Disadvantages.

i) Limited Exploration - It only considers the best immediate neighbours, hence entire graph cannot be explored.

ii) local optimum - It is prone to getting stuck in the local optimum and can miss the global optimum

iii) Dependency on initial state.

iv) It does not maintain a history of visited states.

Conclusion : Hence, we have implemented the local search algorithm, Hill climbing.

Experiment 4

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim:

Program to implement Local Search algorithm: 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. This 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. In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state. A node of hill climbing algorithm has two components which are state and value. It is mostly used when a good heuristic is available.

Following are some main features of Hill Climbing Algorithm:

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.

Greedy approach: Hill-climbing algorithm search moves in the direction which optimizes the cost.

No backtracking: It does not backtrack the search space, as it does not remember the previous states.

Types of Hill Climbing Algorithm are:

1. Simple hill Climbing
2. Steepest-Ascent hill-climbing
3. Stochastic hill Climbing

Algorithm

Step 1: Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.

Step 2: Loop until the solution state is found or there are no new operators present which can be applied to the current state.

a) Select a state that has not been yet applied to the current state and apply it to produce a new state.

b) Perform these to evaluate new state:

1. If the current state is a goal state, then stop and return success.
2. If it is better than the current state, then make it current state and proceed further.
3. If it is not better than the current state, then continue in the loop until a solution is found.

Step 3: Exit.

Code:

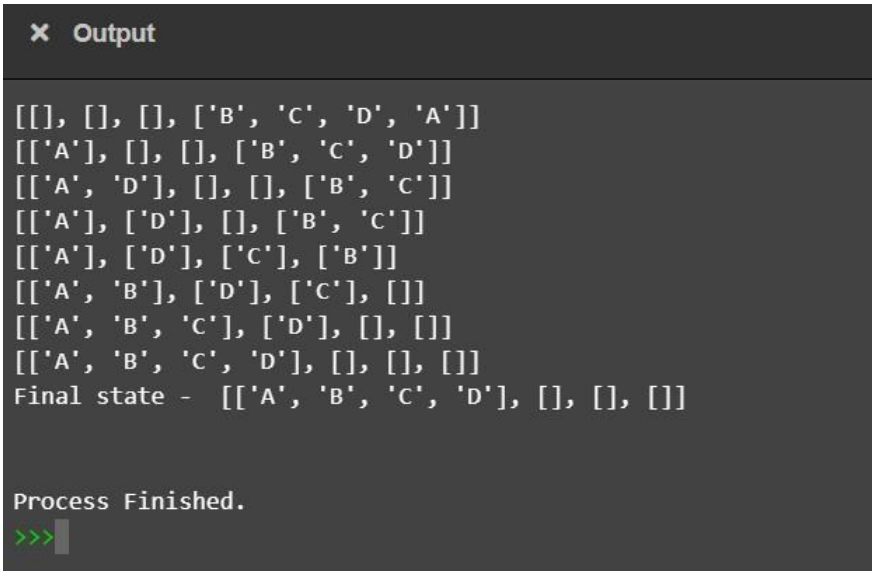
```
import copy visited_states
= [] def
gn(curr_state,prev_heu,go
al_state):     global
visited_states     state =
```

```
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=heuristic(temp1,goal_state)
if curr_heu>prev_heu:
                        child = copy.deepcopy(temp1)
return child


   return 0


def heuristic(curr_state,goal_state):
   goal_=goal_state[3]     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-=1
val-=j                else:
```

```
# val+=1
val+=j     return val

def sln(init_state,goal_state):
   global visited_states     if
(init_state == goal_state):
print (goal_state)
print("solution found!")
return

   current_state = copy.deepcopy(init_state)

   while(True):
      visited_states.append(copy.deepcopy(current_state))
print(current_state)
prev_heu=heuristic(current_state,goal_state)        child =
gn(current_state,prev_heu,goal_state)        if child==0:
print("Final state - ",current_state)            return

      current_state = copy.deepcopy(child)

def hc():
   global visited_states     initial
= [[],[],[],['B','C','D','A']]     goal
= [[],[],[],['A','B','C','D']]
sln(initial,goal)
```

hc()

Output:



Conclusion:

Hill Climbing algorithm is good in solving the optimization problem while using only limited computation power.

Page No.
Date

Experiment 5

Shashwat Shah
60004220126
C2-2 Div B

Aim: To implement genetic algorithms to solve optimization problems.

Theory: Inspired by Charles Darwin's theory of evolution and natural selection a genetic algorithm is a search heuristic reflecting the process of survival of the fittest for producing the next generations

There are 5 phases.

1) Initialization - A set of individuals called a population each being a solution to the given problem.
2) Fitness Assignment - To determine the ability of an individual to compete with the others and probability of selection for reproduction.
3) Selection - The selection of individuals for reproduction of the next generation
4) Reproduction - The creation of children in next generation is done in next steps. This variation operators can be applied for a crossover.
5) → Crossover - A crossover point is selected at random within the genes and the parts of both parents are swapped to create new individuals.
   → Mutation - Inserting Random genes in the offspring to increase their density vs the population. It is also done by flipping some bits in the genes.

Page No.

5) Steps 2, 3, 4 are repeated for a specific no. of times Once a condition is met, it ends.

Conclusion :- Genetic Algorithms can be used to find solution using local moves or renewing population with the best solutions

Experiment 5

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate “survival of the fittest” among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

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).

In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). We say that we encode the genes in a chromosome.

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 the 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 chances 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. For example, consider the crossover point to be 3 as shown below.

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.

Mutation: Before and After

Mutation occurs to maintain diversity within the population and prevent premature convergence.

Termination

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.

ALGORITHM:

```
Initialize a random population of individuals
Compute fitness of each individual
WHILE NOT finished DO
BEGIN /* produce new generation */
FOR population_size DO
BEGIN /* reproductive cycle */
Select two individuals from old generation, recombine the two
individuals to give two offspring
Make a mutation for selected individuals by altering a random bit
in a string
Create a new generation (new populations)
END
IF population has converged THEN finished :=
TRUE
END
```

Why use Genetic Algorithms

- They are Robust
- Provide optimisation over large space state.

- Unlike traditional AI, they do not break on slight change in input or presence of noise ● Application of Genetic Algorithms

Genetic algorithms have many applications, some of them are –

- Recurrent Neural Network
- Mutation testing
- Code breaking
- Filtering and signal processing
- Learning fuzzy rule base etc

CODE:

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

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

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

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 = [decimal_to_binary(i) for i in
            input] for generation in range(iterations): print(f'\nITERATION : {generation+1}',end='\n\n')
            decimal = [binary_to_decimal(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 \tDecimal Value\tFitness Score\t\tFi/Sum\t\tExpected
\tActual ') 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',a
ct_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]:
                                        maxIndex=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(mutation(child,r_mut))
                new_population = list() new_fitness_score = list() for i in mutation_children:
                new_population.append(binary_to_decimal(i)) for i in new_population:
                new_fitness_score.append(fitness_function(i)) print("\nMUTATION\n\nMutation
                population\t New Population\t Fitness") 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
            print("-----------------------------------------------------------------------------------------------")

def main(): iterations = 3 population_size = 4 r_cross = 0.5 r_mut = 0.05
        genetic_algorithm(iterations,population_size,r_cross,r_mut)

if __name__ == '__main__':
        main()
```

OUTPUT:

```
ITERATION : 1

SELECTION

Initial         Decimal Value   Fitness Score          Fi/Sum          Expected        Actual
[0, 1, 0, 0, 1]          9               81             0.11            0.43            0
[1, 0, 1, 0, 0]          20              400            0.53            2.12            2
[0, 0, 1, 1, 1]          7               49             0.06            0.26            0
[0, 1, 1, 1, 1]          15              225            0.3             1.19            1
Sum :  755
Average :  188.75
Maximum :  400

CROSS OVER

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

MUTATION

Mutation population      New Population  Fitness
[0, 1, 0, 0, 1]          9               81
[1, 0, 1, 0, 0]          20              400
[0, 1, 1, 1, 1]          15              225
[0, 1, 0, 0, 1]          9               81
Sum :  787
Maximum :  400
--------------------------------------------------------------------------------------------
ITERATION : 2

SELECTION

Initial         Decimal Value   Fitness Score          Fi/Sum          Expected        Actual
[0, 1, 0, 0, 1]          9               81             0.1             0.41            0
[1, 0, 1, 0, 0]          20              400            0.51            2.03            2
[0, 1, 1, 1, 1]          15              225            0.29            1.14            1
[0, 1, 0, 0, 1]          9               81             0.1             0.41            0
Sum :  787
Average :  196.75
Maximum :  400

CROSS OVER
Population              Mate     Crossover Point         Crossover Population
```

```
[0, 1, 0, 0, 1]          2       0                    [0, 1, 0, 0, 1]
[1, 0, 1, 0, 0]          1       0                    [1, 0, 1, 0, 0]
[0, 1, 1, 1, 1]          4       2                    [0, 1, 0, 0, 1]
[0, 1, 0, 0, 1]          3       2                    [0, 1, 1, 1, 1]

MUTATION

Mutation population      New Population  Fitness
[0, 1, 0, 0, 1]          9               81
[1, 0, 1, 0, 0]          20              400
[0, 1, 1, 0, 1]          13              169
[0, 1, 1, 1, 1]          15              225
Sum :  875
Maximum :  400
-------------------------------------------------------------------------------------------
ITERATION : 3

SELECTION

Initial          Decimal Value   Fitness Score          Fi/Sum          Expected        Actual
[0, 1, 0, 0, 1]          9               81              0.09            0.37            0
[1, 0, 1, 0, 0]          20              400             0.46            1.83            2
[0, 1, 1, 0, 1]          13              169             0.19            0.77            1
[0, 1, 1, 1, 1]          15              225             0.26            1.03            1
Sum :  875
Average :  218.75
Maximum :  400

CROSS OVER

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

MUTATION

Mutation population      New Population  Fitness
[0, 0, 1, 0, 0]          4               16
[1, 1, 0, 0, 1]          25              625
[0, 1, 1, 0, 1]          13              169
[0, 1, 1, 0, 1]          13              169
Sum :  979
Maximum :  625
-------------------------------------------------------------------------------------------
```

CONCLUSION:

We learnt about the Genetic Algorithm, its workings and its uses and also implemented it in a python program. We also learnt about other terms associated with genetic algorithm such as crossover, mutation, fitness score, etc.

Page No.
Date

Experiment 6

Shashwat Shah
60004220126
Div B C22

Aim: To study and implement perceptron learning algorithm.

Theory: Perception learning is a fundamental concept in machine learning and artificial neural networks. The perceptron was developed by Frank Rosenblt in 1957 and is considered one of the earliest neural network models.

Structure of a perception includes the input nodes associated with weights and an output node. The inputs and weights are multiplied and the weighted sum is passed through an activation function. The output is a binary decision based on whether the weighted sum exceeds a certain threshhold which is usually a step function

The process involves the following steps :-

i) Initialization - Initialize the inputs (x) and weights (w) and the learning constant (c)

ii) Calculation for each input compute the weighted sum and get the output theory

iii) Error calculation and weight update.

$$\Delta w = c \cdot (\text{desired} - \text{output}) x_i$$

$$w_{i+1} = \Delta w + w_i$$

iv) Repeat steps ii & iii for a specific number of iterations until the weight converge.

Page No.
Date

Advantage -

- Efficient method and very easy to understand and implement
- It laid the foundation of more complex neural networks.
- These are yet used in certain applications in context of neural network history & development.

Limitation -

Perception learning algorithm may not converge if the data is not linearly seperable.

It is sensitive to the choice of initial weights and learning rate.

Conclusion: Perception learning is a straight forward method for training a basic neural network model while it has limitations it played a crucial role in history of neural network and set the foundation for sophisticated models.

Experiment – 7

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim: Perceptron training algorithm for L and M classification

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:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

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: Hence we have performed **Perceptron training algorithm for L and M classification**

Page No.
Date

Experiment 7

Shashwat Shan
60004220126
DN B C2-2

Aim: Implementation of family tree in Prolog.

Theory: Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog logic is expressed as relations (called facts and rules), the core heart of the prolog lies in the logic being applied

Formulation or computation is carried out by running a query over these relations. Prolog features are 'logic variable' which means that they behave like uniform data structures. a backtracking strategy to search for proofs, a pattern matching facility, mathematical variables, and input and out are interchangeable. Prolog is a declarative language that means we can specify what problem we want to solve rather than how to solve it. Prolog is used in some areas like database, natural language processing and artificial intelligence, but it is pretty useless in some areas like numerical algorithms or instance graphics.

Key features

1) Unification - The basic idea is, can the given terms be made to represent the same structure.
2) Backtracking - When a task fails, prolog traces backward and tries to satisfy the previous tasks
3) Recursion - It is the basis for any search in the program.

Conclusion: Thus, we successfully implemented family tree in Prolog.

Experiment 7

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim: Implementation of Wumpus World Program in Prolog.

Theory:

Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called Facts and Rules). The core heart of the prolog lies in the logic being applied.

Formulation or Computation is carried out by running a query over these relations. Prolog features are 'Logical variable', which means that they behave like uniform data structures, a backtracking strategy to search for proofs, a pattern-matching facility, mathematical variables, and input and out are interchangeable. Prolog is a declarative language that means we can specify what problem we want to solve rather than how to solve it. Prolog is used in some areas like databases, natural language processing, and artificial intelligence, but it is pretty useless in some areas like numerical algorithms or instance graphics.

Key Features of Prolog:

1. Unification: The basic idea is, can the given terms be made to represent the same structure.
2. Backtracking: When a task fails, prolog traces backward and tries to satisfy the previous task.
3. Recursion: Recursion is the basis for any search in the program.

Prolog provides an easy to build a database. Doesn't need a lot of programming effort. Pattern matching is easy. Search is recursion based. It has built-in list handling. Makes it easier to play with any algorithm involving lists. LISP (another logic programming language) dominates over prolog with respect to I/O features. Sometimes input and output is not easy.

Applications:

- Specification Language
- Robot Planning
- Natural language understanding
- Machine Learning
- Problem Solving
- Intelligent Database retrieval
- Expert System
- Automated Reasoning

Code:

```
/* Facts */ male(jack).

male(oliver).

male(ali).

male(james).

male(simon).

male(harry).

male(bhupesh).

male(ram).

female(helen).

female(sophie).
```

female(jess).

female(lily).

female(sita).

parent_of(jack,jess).

parent_of(jack,lily).

parent_of(helen, jess).

parent_of(helen, lily).

parent_of(oliver,james).

parent_of(sophie, james).

parent_of(jess, simon).

parent_of(ali, simon). parent_of(lily,

harry). parent_of(james, harry).

parent_of(jack,bhupesh).

parent_of(helen,bhupesh).

parent_of(ram,helen).

parent_of(sita,helen).

parent_of(ram,sophie).

parent_of(sita,sophie).

/* Rules */ father_of(X,Y):-

male(X), parent_of(X,Y).

mother_of(X,Y):- female(X),

parent_of(X,Y).

grandfather_of(X,Y):- male(X), parent_of(X,Z), parent_of(Z,Y). grandmother_of(X,Y):-

female(X), parent_of(X,Z), parent_of(Z,Y). sister_of(X,Y):- %(X,Y or Y,X)%

female(X),

```
father_of(F,     Y), father_of(F,X),X
\=       Y.
sister_of(X,Y):-
          female(X),
mother_of(M, Y),
mother_of(M,X),X     \=       Y.
aunt_of(X,Y):-
          female(X),
parent_of(Z,Y), sister_of(Z,X),!.
brother_of(X,Y):- %(X,Y or
Y,X)%                male(X),
father_of(F,                Y),
father_of(F,X),X      \=      Y.
brother_of(X,Y):-     male(X),
mother_of(M,                Y),
mother_of(M,X),X   \=      Y.
uncle_of(X,Y):-
parent_of(Z,Y), brother_of(Z,X).
ancestor_of(X,Y):-
parent_of(X,Y).
ancestor_of(X,Y):-
parent_of(X,Z),
ancestor_of(Z,Y).
```

Output:

mother_of(X,jess)

X = helen

Next | 10 | 100 | 1,000 | Stop

```
?- mother_of(X,jess)
```

grandfather_of(X,simon)

X = jack

Next | 10 | 100 | 1,000 | Stop

```
?- grandfather_of(X,simon)
```

sister_of(X,jess)

X = lily

Next | 10 | 100 | 1,000 | Stop

```
?- sister_of(X,jess)
```

brother_of(X,jess)

X = bhupesh

Next | 10 | 100 | 1,000 | Stop

```
?- brother_of(X,jess)
```

aunt_of(X,jess)

X = sophie

```
?- aunt_of(X,jess)
```

ancestor_of(X,jess)

X = jack

Next | 10 | 100 | 1,000 | Stop

```
?- ancestor_of(X,jess)
```

Conclusion: Thus, we successfully implemented family tree in prolog.

Page No.
Date

Experiment 8

Shashwat Shah
60004220126
Comps B C22

Aim : Implementation on any AI problem : Wumpus world, tic. tac -toe theory.

Theory : Wumpus world.

The wumpus world is a cave of 16 rooms (4x4) each room is connected to others through walkway, (no rooms are connected diagnally. The knowledge based agent starts from room [1, 1]. The cave has some pits, treasure and a beast named wumpus.

The wumpus cannot move but eats the agent if he enters the room. If the agent enters the pit then it gets stuck. The goal of the agent is to get to the treasure and come out of the cave.

The agent is rewarded when the goal conditions are met. The agent is penalized, when it falls into pit or is eaten by the wumpus.

Some elements support the agent to explore the cave. Like the rooms next to the wumpus are stenchy. The agent is given one arrow with which it can kill the wumpus. Also the wumpus screams when it gets killed. The adjacent rooms of the pits are filled with breeze. The room with the treasure is always glittery.

Conclusion : We learnt about knowledge based agents & the wumpus world game.

EXPERIMENT 8

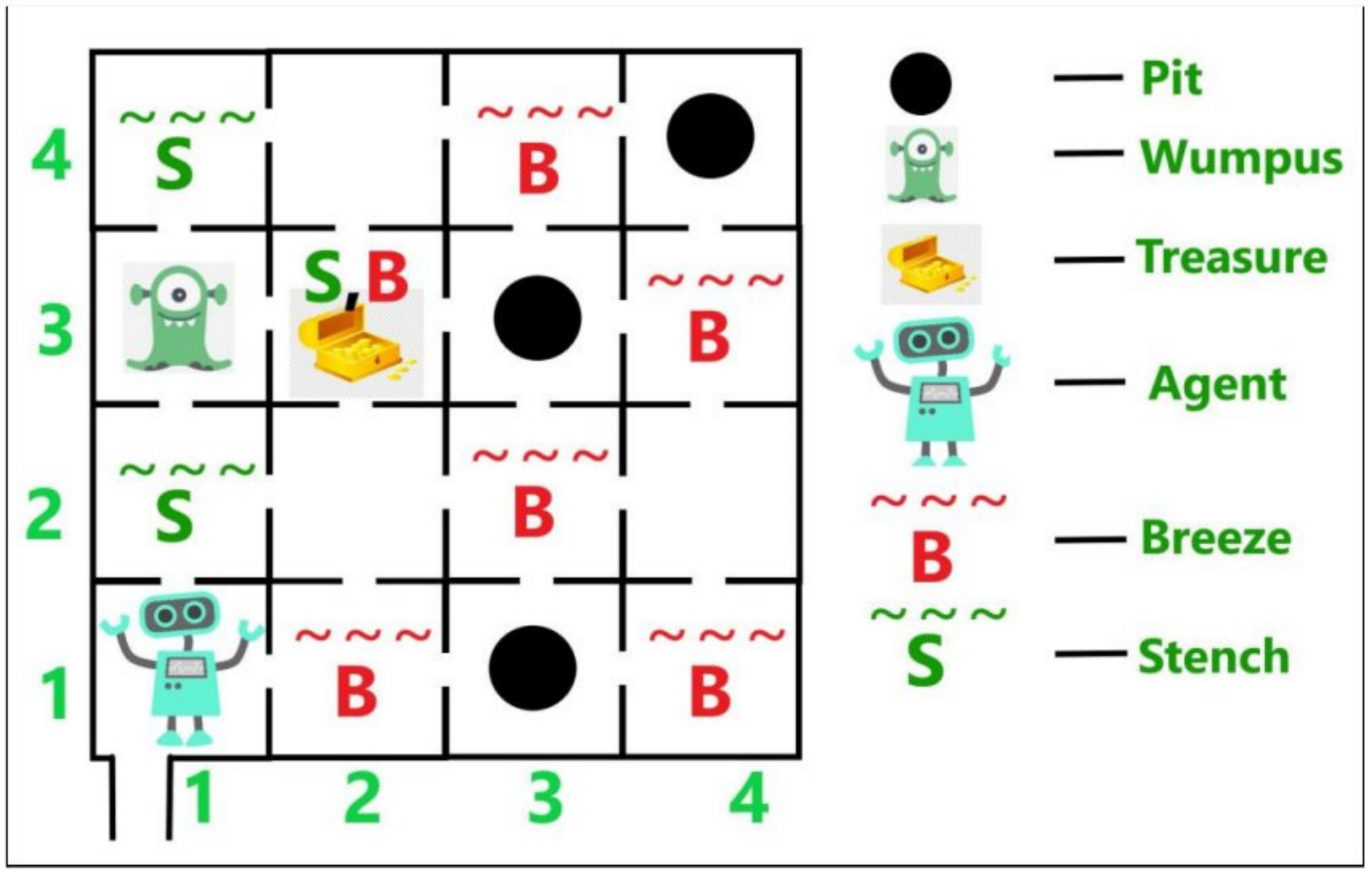
Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim: Implementation on any AI Problem : Wumpus world, Tic-tac-toe Theory:

The Wumpus world is a cave with 16 rooms (4×4). Each room is connected to others through walkways (no rooms are connected diagonally). The knowledge-based agent starts from Room[1, 1]. The cave has – some pits, a treasure and a beast named Wumpus. The Wumpus can not move but eats the one who enters its room. If the agent enters the pit, it gets stuck there. The goal of the agent is to take the treasure and come out of the cave. The agent is rewarded, when the goal conditions are met. The agent is penalized, when it falls into a pit or is eaten by the Wumpus.
Some elements support the agent to explore the cave, like -The wumpus's adjacent rooms are stenchy. -The agent is given one arrow which it can use to kill the wumpus when facing it (Wumpus screams when it is killed). – The adjacent rooms of the room with pits are filled with breeze. -The treasure room is always glittery.



PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below:

Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- -1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

Environment:

- A 4*4 grids of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square [1,1].
- Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release ● Shoot.

Sensors:

- The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive a breeze if he is in the room directly adjacent to the Pit.
- The agent will perceive the **glitter** in the room where the gold is present.
- The agent will perceive the **bump** if he walks into a wall.
- When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave.

- These precepts can be represented as a five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as:
  [Stench, Breeze, None, None, None].

The Wumpus world Properties:

- **Partially observable:** The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- **Deterministic:** It is deterministic, as the result and outcome of the world are already known.
- **Sequential:** The order is important, so it is sequential.
- **Static:** It is static as Wumpus and Pits are not moving.
- **Discrete:** The environment is discrete.
- **One agent:** The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

```
public class Block { public static final int
    NOT_PRESENT = 0; public static final int
    UNSURE = 1;

    public boolean hasBreeze, hasPit; public
    int pitStatus = UNSURE;

    public boolean hasStench, hasWumpus;

    public int wumpusStatus = UNSURE; public

    boolean hasGold;

    public boolean isVisited;
}



import java.util.Scanner;

class WumpusWorld {
    static Block[][] maze; static
    int n;

    public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);

            System.out.print("Enter the order of the maze: "); n =
            sc.nextInt();

            maze = new Block[n][n];
            for(int i=0; i<n; i++) {
                    maze[i] = new Block[n]; for(int
                    j=0; j<n; j++) maze[i][j] = new
                    Block();
    }

    System.out.print("\nEnter the number of pits: "); int
    pits = sc.nextInt();

    for(int i=0; i<pits; i++) {
```

```
System.out.print("Enter the location of pit " + (i+1) + ": "); addPit(n-sc.nextInt(),
sc.nextInt()-1);
```

Code:

```
}

System.out.print("\nEnter the location of wumpus: "); addWumpus(n-sc.nextInt(),
sc.nextInt()-1);

System.out.print("\nEnter the location of gold: "); addGold(n-sc.nextInt(),
sc.nextInt()-1);

System.out.print("\nEnter the starting location: ");
int r = n - sc.nextInt(); int c = sc.nextInt() - 1; int rPrev
= -1, cPrev = -1;

System.out.print("\nYour Position : *\nWumpus : X\nGold : $\nPit : O");

int moves = 0;
System.out.println("\nInitial state:"); printMaze(r,
c);

while(!maze[r][c].hasGold) {
        maze[r][c].isVisited = true; maze[r][c].pitStatus =
        Block.NOT_PRESENT; maze[r][c].wumpusStatus =
        Block.NOT_PRESENT;

if(!maze[r][c].hasBreeze) { if(r >= 1 && maze[r-1][c].pitStatus ==
        Block.UNSURE) maze[r-1][c].pitStatus = Block.NOT_PRESENT;
        if(r <= (n-2) && maze[r+1][c].pitStatus == Block.UNSURE)
        maze[r+1][c].pitStatus = Block.NOT_PRESENT; if(c >= 1 &&
        maze[r][c-1].pitStatus == Block.UNSURE) maze[r][c-
        1].pitStatus = Block.NOT_PRESENT; if(c <= (n-2) &&
        maze[r][c+1].pitStatus == Block.UNSURE)
        maze[r][c+1].pitStatus = Block.NOT_PRESENT;
}

if(!maze[r][c].hasStench) { if(r >= 1 && maze[r-1][c].wumpusStatus ==
        Block.UNSURE) maze[r-1][c].wumpusStatus = Block.NOT_PRESENT;
        if(r <= (n-2) && maze[r+1][c].wumpusStatus == Block.UNSURE)
        maze[r+1][c].wumpusStatus = Block.NOT_PRESENT; if(c >= 1 &&
        maze[r][c-1].wumpusStatus == Block.UNSURE) maze[r][c-
        1].wumpusStatus = Block.NOT_PRESENT; if(c <= (n-2) &&
        maze[r][c+1].wumpusStatus == Block.UNSURE)
        maze[r][c+1].wumpusStatus = Block.NOT_PRESENT;
}
```

60004220126

```
                boolean foundNewPath = false;

                if(r >= 1 && !((r-1) == rPrev && c == cPrev) && maze[r-1][c].isVisited == false &&
maze[r-1][c].pitStatus == Block.NOT_PRESENT && maze[r-1][c].wumpusStatus == Block.NOT_PRESENT) {
                        rPrev = r; cPrev
                        = c;

                        r--;
                        foundNewPath = true;
                }
                else if(r <= (n-2) && !((r+1) == rPrev && c == cPrev) && maze[r+1][c].isVisited == false &&
maze[r+1][c].pitStatus == Block.NOT_PRESENT && maze[r+1][c].wumpusStatus == Block.NOT_PRESENT) {
                        rPrev = r; cPrev
                        = c;

                        r++; foundNewPath =
                        true;
                }
                else if(c >= 1 && !(r == rPrev && (c-1) == cPrev) && maze[r][c-1].isVisited == false &&
maze[r][c-1].pitStatus == Block.NOT_PRESENT && maze[r][c-1].wumpusStatus == Block.NOT_PRESENT) {
                        rPrev = r; cPrev
                        = c;

                        c--;
                        foundNewPath = true;
                }
                else if(c <= (n-2) && !(r == rPrev && (c+1) == cPrev) && maze[r][c+1].isVisited == false &&
maze[r][c+1].pitStatus == Block.NOT_PRESENT && maze[r][c+1].wumpusStatus == Block.NOT_PRESENT) {
                        rPrev = r; cPrev
                        = c;

                        c++; foundNewPath =
                        true;
                }

                if(!foundNewPath) { int
                temp1 = rPrev; int
                temp2 = cPrev;

                rPrev = r; cPrev
                = c;

                r = temp1; c
                = temp2;
```

60004220126

```
        }

        moves++;

        System.out.println("\n\nMove " + moves + ":"); printMaze(r,
        c);

        if(moves > n*n) {
                System.out.println("\nNo solution found!"); break;
                }
        }

        if(moves <= n*n)
                System.out.println("\nFound gold in " + moves + " moves.");

        sc.close();
}

static void addPit(int r, int c) { maze[r][c].hasPit =
        true;

        if(r >= 1)
                maze[r-1][c].hasBreeze = true;
        if(r <= (n-2)) maze[r+1][c].hasBreeze =
        true; if(c >= 1) maze[r][c-1].hasBreeze =
        true;
        if(c <= (n-2)) maze[r][c+1].hasBreeze =
                true;
}

static void addWumpus(int r, int c) { maze[r][c].hasWumpus =
        true;

        if(r >= 1)
                maze[r-1][c].hasStench = true;
        if(r <= (n-2)) maze[r+1][c].hasStench =
        true; if(c >= 1) maze[r][c-1].hasStench =
        true;
        if(c <= (n-2)) maze[r][c+1].hasStench =
                true;
}
```

60004220126

```
static void addGold(int r, int c) { maze[r][c].hasGold =
        true;
}

static void printMaze(int r, int c) { for(int i=0;
        i<n; i++) { for(int j=0; j<n; j++) { char
        charToPrint = '-'; if(r == i && c == j)
        charToPrint = '*';
                else if(maze[i][j].hasPit)
                        charToPrint = 'O'; else
                if(maze[i][j].hasWumpus)
                charToPrint = 'X';
                else if(maze[i][j].hasGold) charToPrint =
                        '$';

                System.out.print(charToPrint + "\t");
            }
            System.out.println();
        }
    }
}
```

Output:

```
Enter the order of the maze: 4

Enter the number of pits: 2
Enter the location of pit 1: 1 4
Enter the location of pit 2: 3 4
```

```
Enter the location of wumpus: 2 4

Enter the location of gold: 2 3

Enter the starting location: 1 1

Your Position : *
Wumpus : X
Gold : $
Pit : O
```

Initial state:

```
-       -   -   -   -   -   O
-       -   $   X
*   -   -   O
```

Move 1:

```
-   -   -   -
-   -   O
*   -   $   X
-       -   -   O
```

Move 2:

```
-       -   -   *   -   -   O
-       -   $   X
-       -   -   O
```

Move 3:

```
*   -   -   -
-   -   O
-       -   $   X
-       -   -   O
```

Move 4:

```
-       *   -   -   -   -   O
-       -   $   X
-       -   -   O
```

Move 5:

```
-       -   -   -   *   -   O
-       -   $   X
-       -   -   O
```

Move 6:

```
-       -   -   -   -   -   O
```

```
-       *    $    X
-       -    -    O
```

Move 7:

```
-       -    -    -    -    -    O
-       -    $    X
-       *    -    O
```

Move 8:

```
-       -    -    -    -    -    O
-       -    $    X
-       -    *    O
```

Move 9:

```
-       -    -    -    -    -    O
-       -    *    X
-       -    -    O
```

Found gold in 9 moves.

Conclusion: We learnt about Knowledge based agents and about the Wumpus World game. We then explored the PEAS of an agent in the Wumpus World game and wrote a code in java to simulate the game.

Page No.
Date

Experiment 9

Shashwat Shah
60004220126
Div B C2-2

Aim: A study on planning problem in AI

Theory: Artificial Intelligence is a critical technology in the future. Whether it is intelligent robots or self driving cars or even 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 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 Eg. reaching a particular destination requires planning. Finding the best route is not only requirement in planning, but the actions to be done at a particular time and why they are done is also very important.

In other words planning is all about deciding the actions to be performed by the artificial intelligent system and the functioning of the system on its own in domain independent situations.

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

Experiment 9

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

Aim: A Study on Planning Problem in AI.

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.

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.

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

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

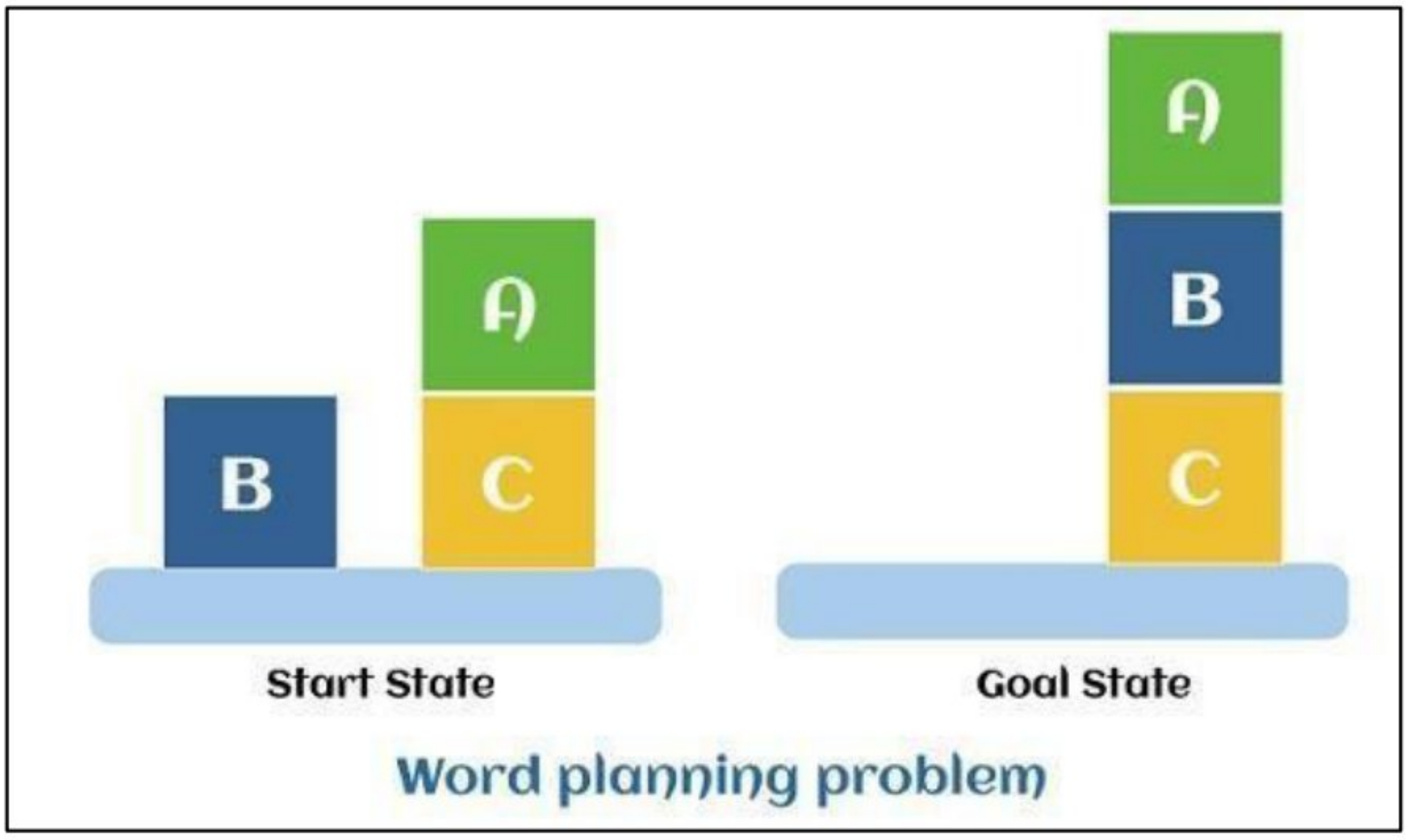
The block-world problem is known as the Sussmann anomaly.

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:

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.

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.

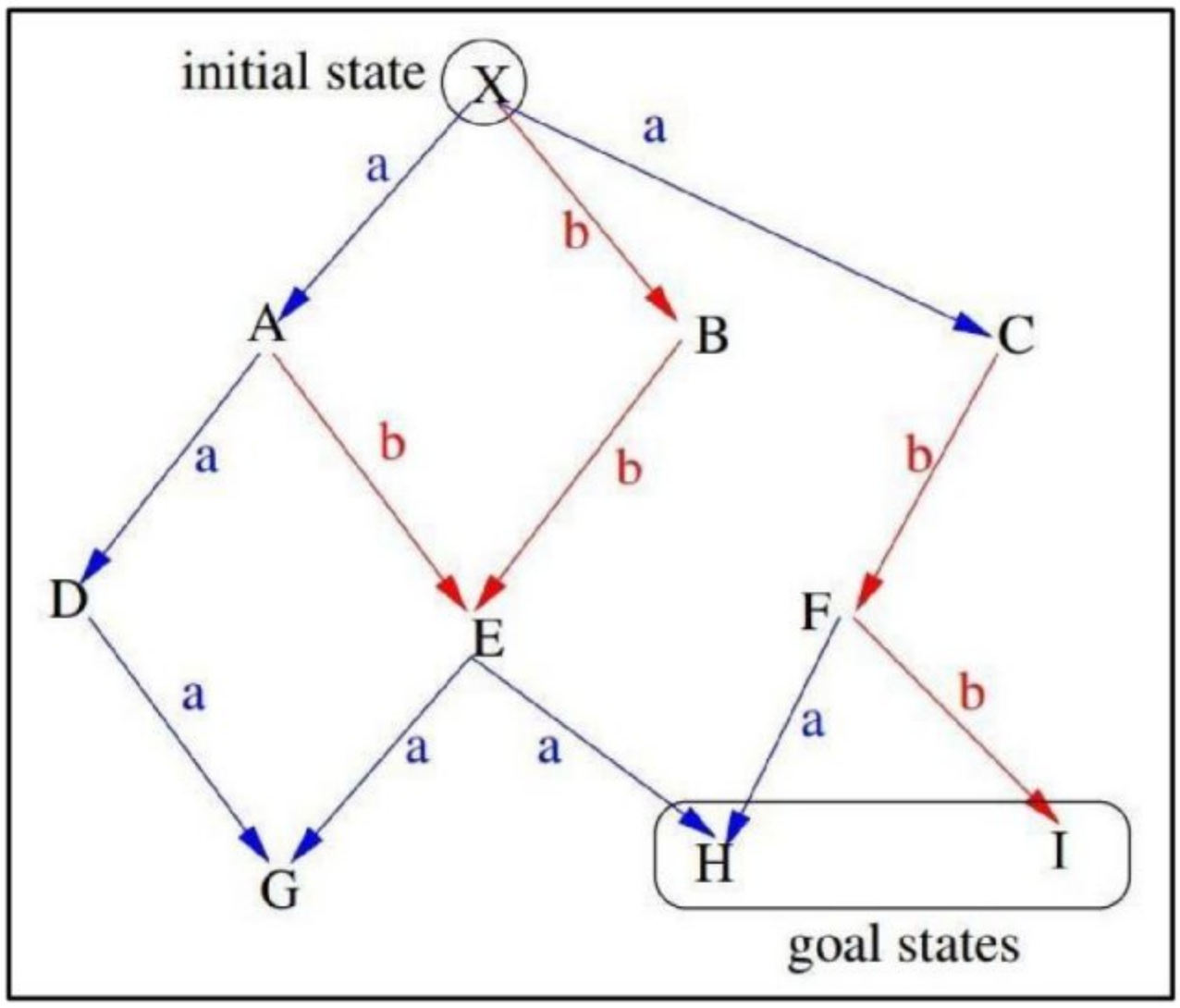
➢ Planning Through State Space Search

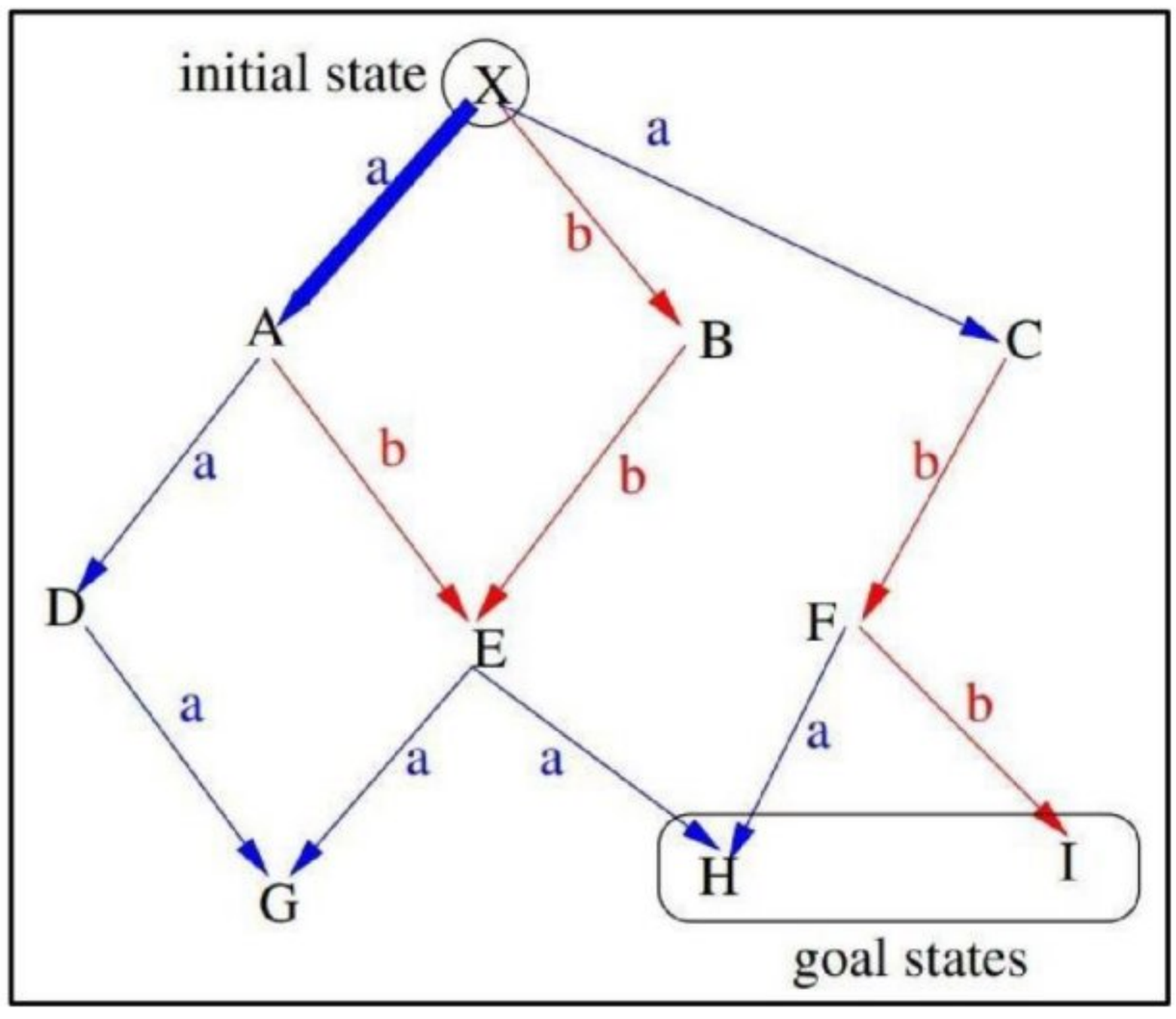
We can view planning problems as searching for goal nodes in a large labeled graph (transition system)

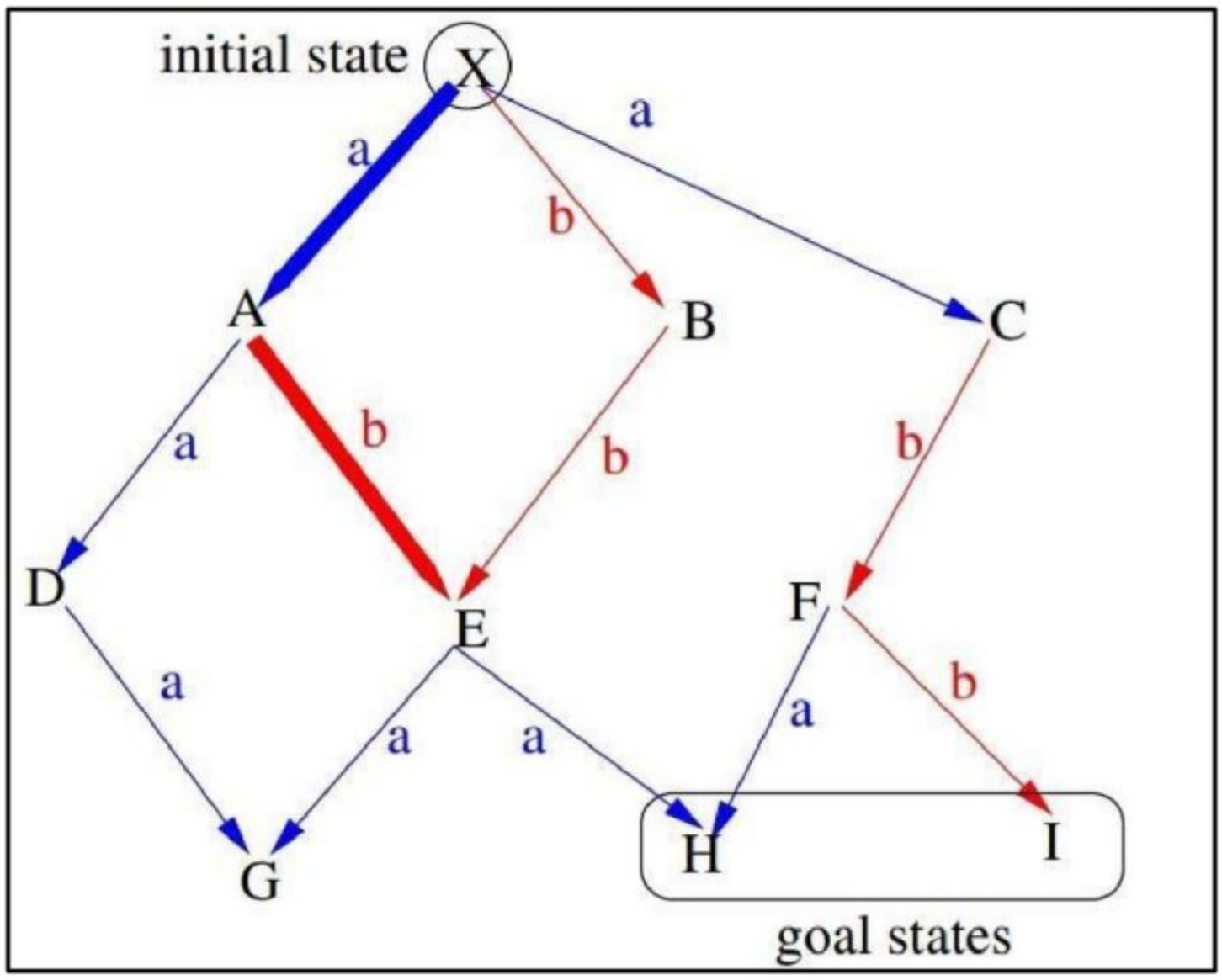
Nodes are defined by the value assignment to the fluents = states

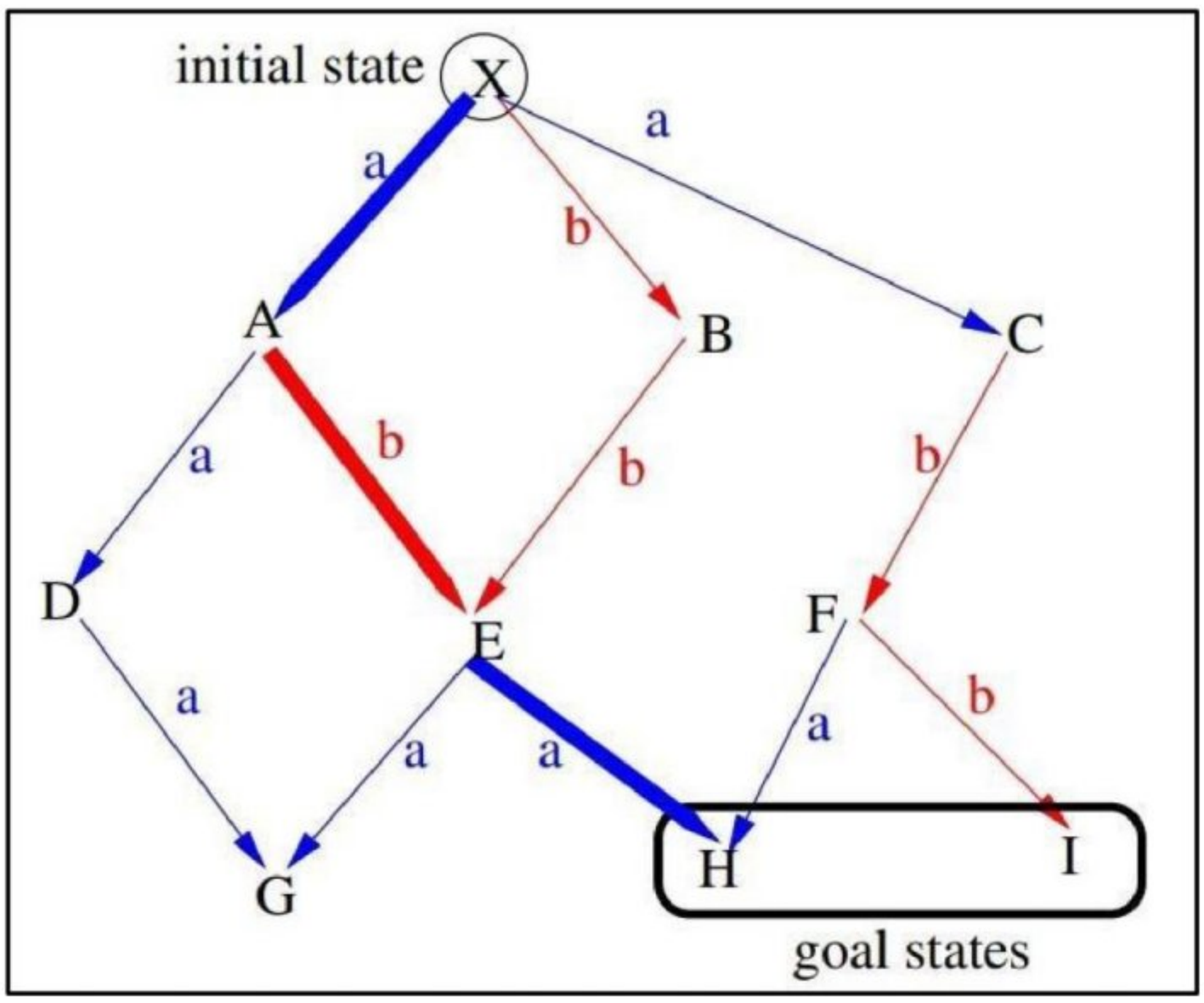
Labeled edges are defined by actions that change the appropriate fluents

Use graph search techniques to find a (shortest) path in this graph! Note: The graph can become huge: 50 Boolean variables lead to 250 = 1015 states Create the transition system on the fly and visit only the parts that are necessary









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 **Solution:**

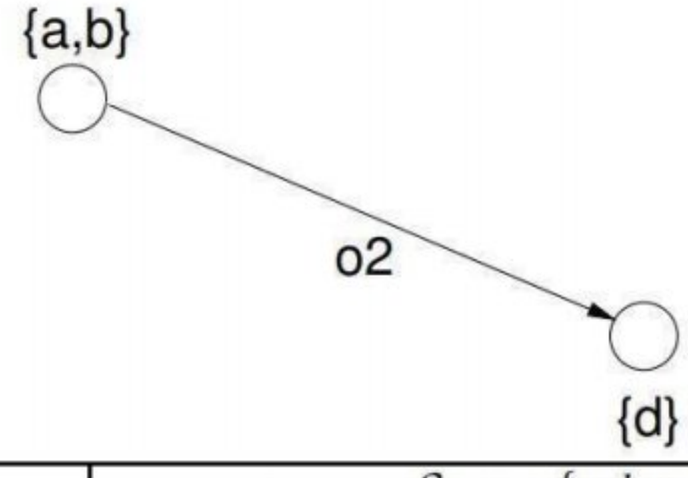
Search thr

1. Initialize
make it t
2. Test whe
return pl
3. Select o
   - compu
   - extend

Instead of
Progressio
planning la

Backward
state
and
space
search
planning

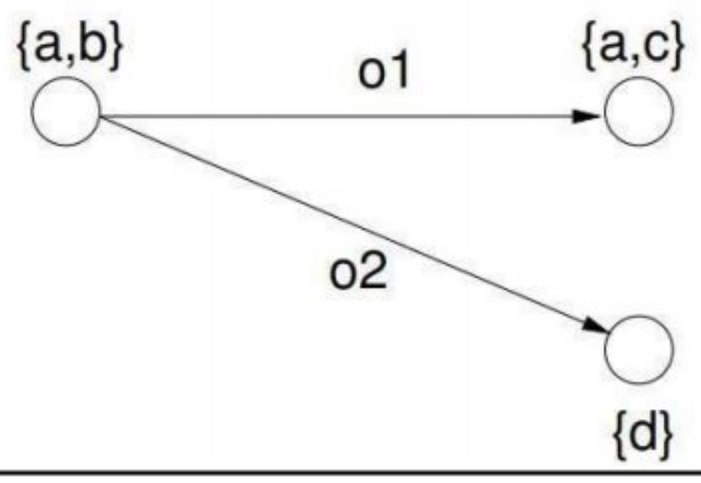
$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\
&\quad\ o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\
&\quad\ o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G}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



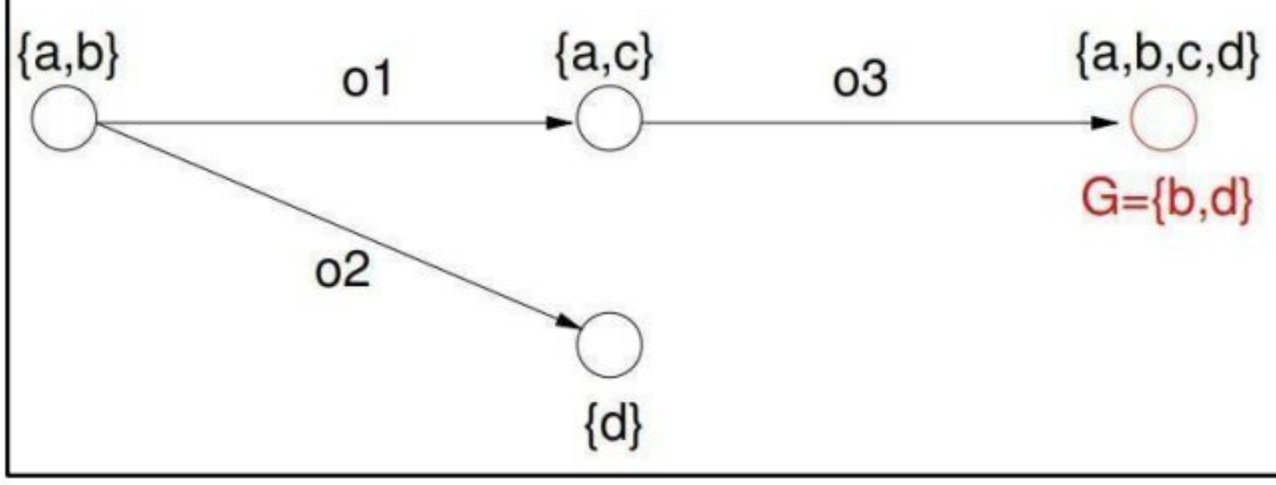
$\mathcal{S} = \{a, b, c, d\},$

Proble

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



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



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:

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

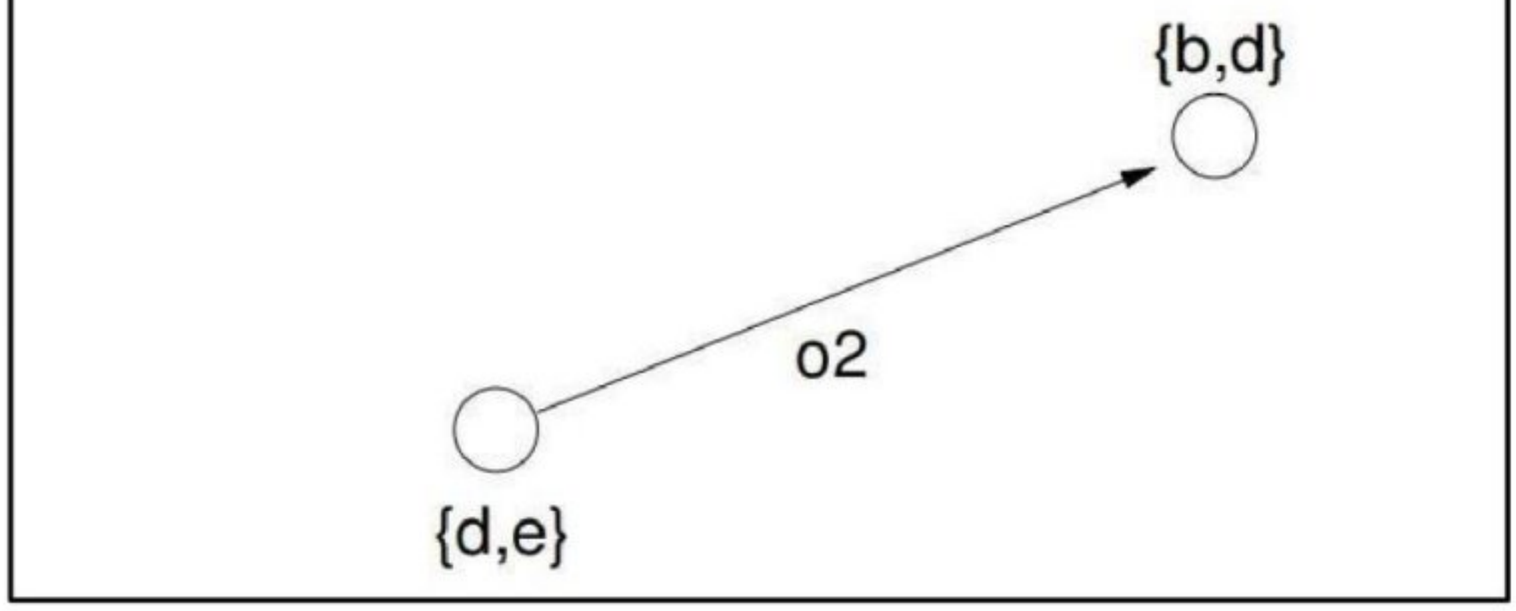
   - extend plan $\Delta := \langle o_i, \Delta \rangle$, and continue with step 2.

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

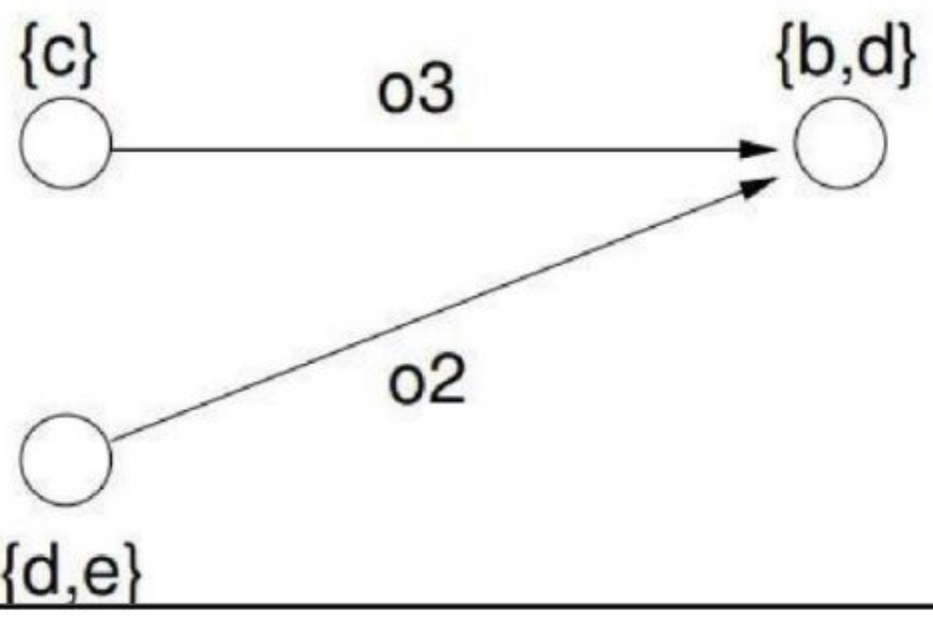
Solution:

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

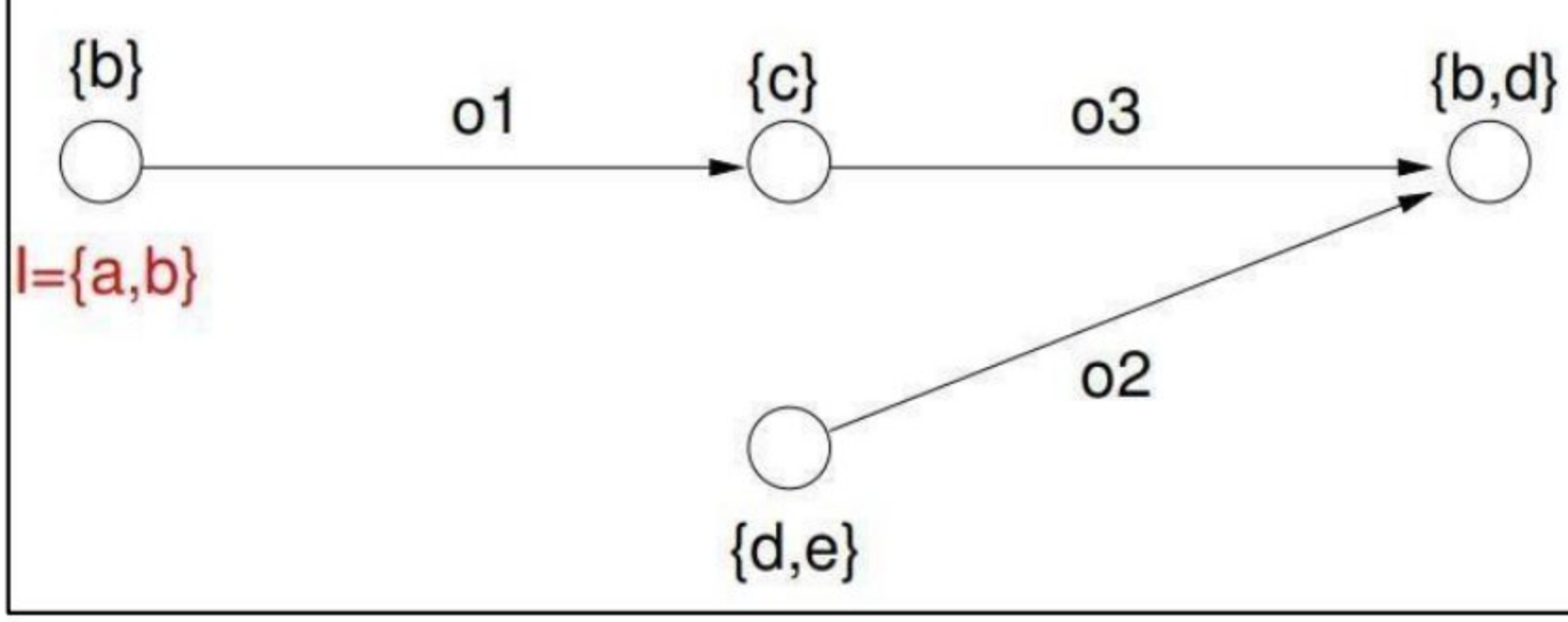
$$
\begin{aligned}
\mathcal{S} &= \{a, b, c, d, e\}, \\
\mathbf{O} &= \{\ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\
&\quad\ \ o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\
&\quad\ \ o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \\
\mathbf{I} &= \{a, b\} \\
\mathbf{G} &=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



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



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



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)

EXPERIMENT 10

Name: Shashwat Shah

SAP-ID: 60004220126

TY BTECH DIV B, Batch : C22

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.

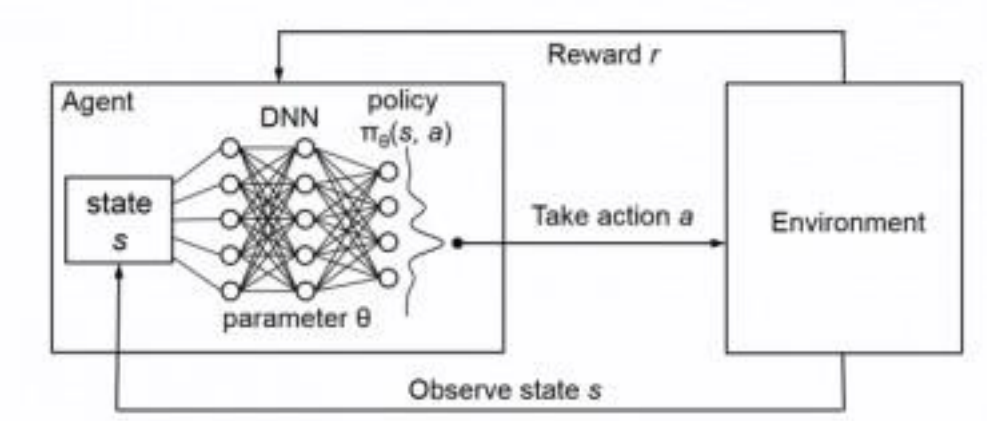
These long-term goals help prevent the agent from stalling on lesser goals. With time, the agent learns to avoid the negative and seek the positive. This learning method has been adopted in artificial intelligence (AI) as a way of directing unsupervised machine learning through rewards and penalties.

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.

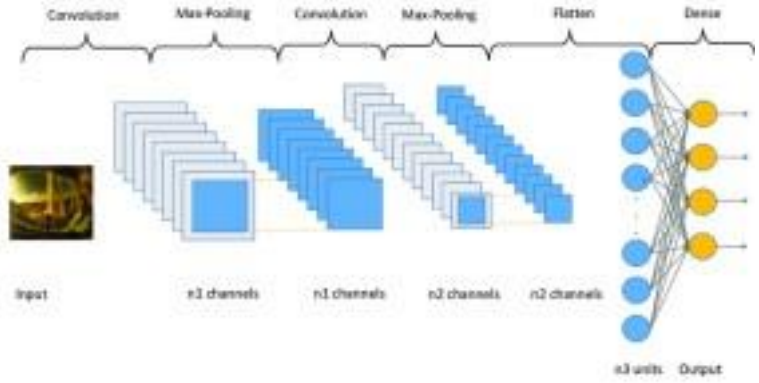


2) **CNN:** A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

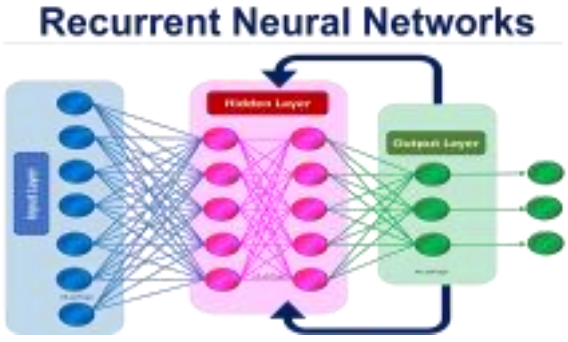
The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

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.



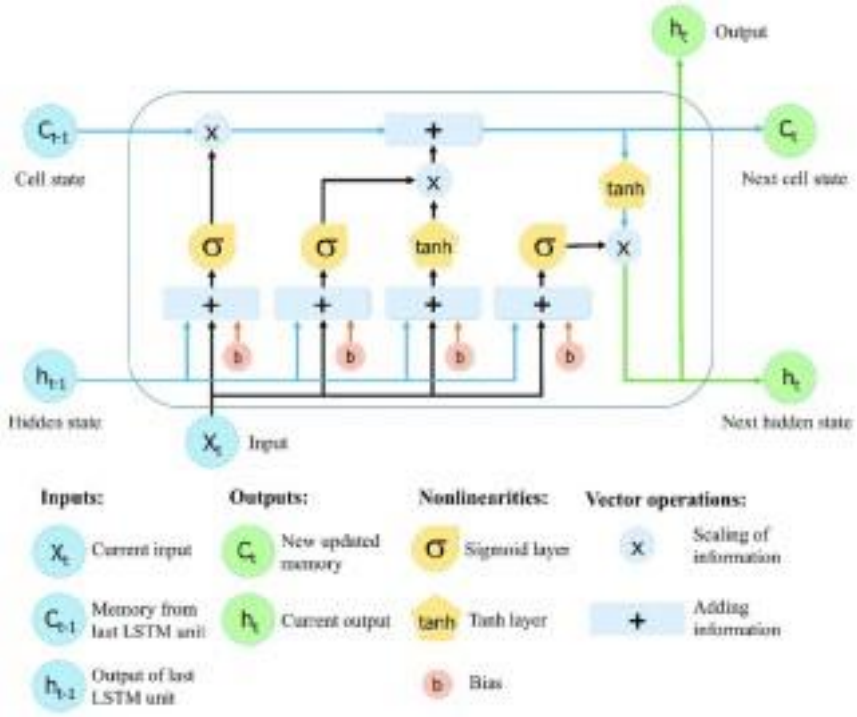
3) **RNN:** Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of 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.

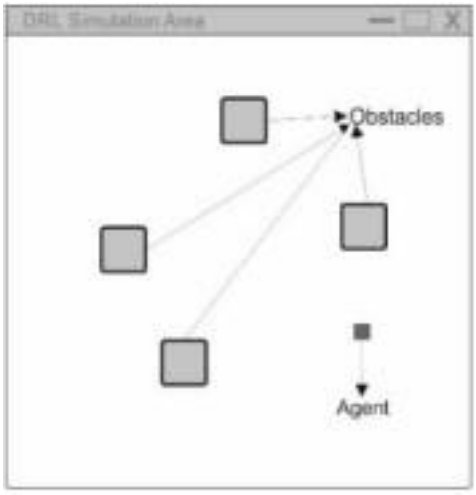
With LSTMs, there is no need to keep a finite number of states from beforehand as required in the hidden Markov model (HMM). LSTMs provide us with a large range of parameters such as learning rates, and input and output biases. Hence, no need for fine adjustments. The complexity to update each weight 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

For the model training, python was used as the software language and Keras, a deep learning library was used to create the neural networks. Mean Squared Error was used as the Loss function. Linear was preferred as the activation function. Sigmoid function is used as the activation function in the output layer. The training took 5 hours for CNN, 18 hours for RNN and 35 hours for LSTM on a standard equipped (core i5 Processor and 8GB RAM) computer

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.

Assignment 1

Shashwat Shah
500042201 26
Div B C22

1 Planning and acting in non deterministic domain.

In classical AI planning, and deterministic actions, allowing agents to plan beforehand and execute flawlessly. However real-world environments often introduce uncertainty, demanding agents to adapt during execution based on ongoing perceptions. Dealing with incomplete or incorrect information is vital.

Handling uncertainty relies on two factors bounded and unbounded indeterminacy. Bounded cases, like coin flipping with defined outcomes, permit agents to plan for all potential scenarios. Unbounded indeterminacy, found in complex domains like driving or economies, demands continual adaptation due to unknown or vast potential.

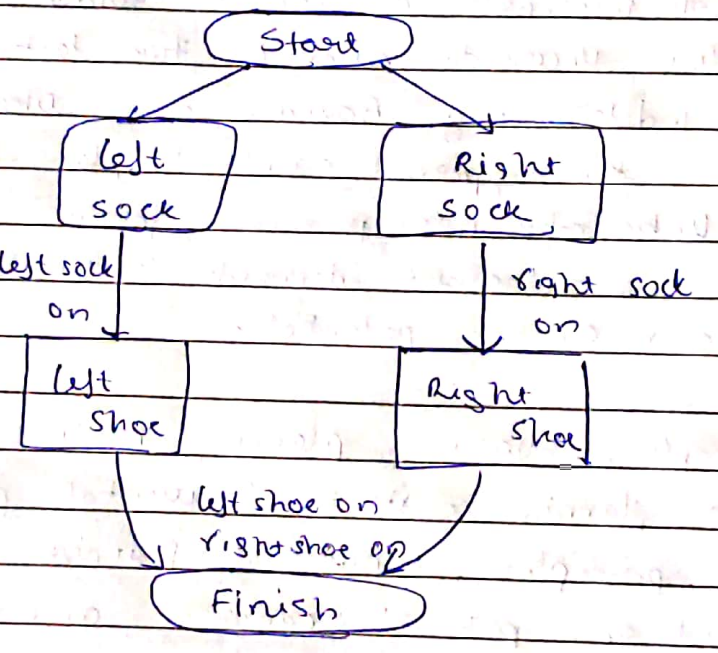
AI employes various planning method for indeterminacy sensorless planning constructs sequential plans without real time perception, conditional planning devises contingency plans based on potential scenarios, and execution monitoring and replanning dynamically adjust plans as situations unfold.

Continuous planning persists across changing environments, adapting goals and actions. Consider a scenario, where furniture needs painting to match colors, revealing how different agents might tackle this based on their planning methods. Classical planners falter due to incomplete initial state knowledge.

2 Partial order Planning & Hierarchical planning.

So here in POP (Partial Order Planning), ordering of the actions is partial. Also partial ordered planning dont specify which action will come first out of the two actions which are placed.

With partial order planning, problem can be decomposed, so it can work well in case the environment is non cooperative. It combines the two action seaences



Hierarchial Planning.

It basically refers to a problem solving approach that involves breaking down complex tasks into a hierarchial structure of smaller subtasks or actions that can be executed by an inteligent agent.

The agent decomposes the overall task into sub-task and generate a plan for each sub-task or actions dependencies, constraints and the goal of the overall task. Each sub plan is executed seaentially, will the results of each steps being used to guide subseaence steps

Components - High level goals. The overall objectives or tasks that the AI system aims to achieve

Task decomposition - Breaking down high level goals into lower level tasks or subgoals.

Planning Hierarchy - The organisation of tasks or subgoals into a hierarchy structure, such as a tree or a directed acyclic graph (DAG)

3 i) RDF (Resource Description Framework) - RDF in AI provides a structured way to represent information on the web, using subject predicate object triples to denote relationships between entities. It enables the creation of semantic connections between diverse data sources, allowing AI sources to understand complex relationships between entities.

ii) OWL - (web Ontology Language)

OWL is a part of the semantic web technology stack and provides a formal, standardized language for creating ontologies structures knowledge by defining concepts, their relationships and properties in a domain. It offers different levels of expressiveness.

It allows the creation of hierarchies among classes, permitting classification of entities based on the properties and relationships aiding in categorization and reasoning.

Discuss Dempster - Shafer theory

→ The theory is designed to deal with the distinction between uncertainty and ignorance. Rather than computing the probability of a proposition, it computes the probability that the evidence supports the proposition. This measure of belief is called a belief

function, written Bel(x). Suppose a shady character comes up to you and offers to bet you $10 that his coin will come up heads on the next flip

Assignment 2

Shashwat Shah
60004220126
Div B (22

Aim : Answer the questions asked accordingly.

Hierarchical planning.

So basically hierarchical planning refers to a problem solving approach that involves breaking down complex tasks into a hierarchical structure of smaller sub-task or action that can be executed by an intelligent agent. The agent decomposes the overall task into sub tasks and generates a plan for each task, taking into account dependencies, constraints, and the goals of the overall task. Each sub-plan is executed sequentially, with the results at each step being used to guide subsequent steps.

Components

High level goals - The overall objective or tasks that the AI system aims to achieve.

Task - decomposition - Breaking down high-level goals into lower-level task or subgoals.

Planning hierarchy - The organisation of tasks or subgoals into a hierarchical structure, such as a tree or a directed acyclic graph (DAG)

Eg. Top-level goal - How ultimate goal is to have a fantastic vacation

High level subgoals - Booking flights, arranging accom
planning activities

Subtasks - Under booking flights, there maybe subtask
like research airline, compare prices and purchas
tickets.

3 Goal stack planning.

It is one of the AI techniques where goals are organised in a stack. It involves decomposing high level goals into subgoals and creating a stack structure to manage their execution.

The system works by pursuing goals at the top of the stack and recursively achieving subgoals until the overall objective is met.

This approach helps in organizing and prioritizing tasks in complex problem solving scenarios.