Department of Computer Science and Engineering (Data Science)

Subject: Artificial Intelligence (DJ19DSC502)

AY: 2023-24

Experiment 10

(Planning)

60009210105

Amitesh Sawarkar

D - 12

Aim: Implement a plan using AO*.

Theory:

The Depth-first search and Breadth-first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined since all goals following an AND node must be realized; whereas a single goal node following an OR node will do. So for this purpose, we are using AO* algorithm. Like A* algorithm here we will use two arrays and one heuristic function.

OPEN: It contains the nodes that have been traversed but yet not been marked solvable or unsolvable.

CLOSE: It contains the nodes that have already been processed.

AO* Search Algorithm

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Lab Assignment to do:

Consider the use case of a plan to travel from Mumbai to Goa to attend a wedding at Taj Aguada. The plan needs to be decided based on the cost. You can either travel by train or bus or flight and stay in a hotel near or far to the wedding venue. The three options of the venues are Westin, Kennel Worth and Maria Rica hotels. You can choose between a two days package for stay and meal together or separately. Other option for your travel and stay will be a vanity van. There you need to decide if you want to cook or eat outside.
Implement AO* to find the most suitable plan in terms of cost.

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
        # INITIALIZING THE GRAH OBJECT WITH GRAPH TOPOLOGY, HEURISTIC VALUES AND START NODE
        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode

        self.parent={}
        ''' IF NODE IS EXPLORED, THEN STATUS = -1
            AT START, STATUS = 0
            ELSE STATUS = NUMBER OF CHILD NODES OF v '''
        self.status={}
        self.solutionGraph={}


    def applyAOStar(self):
        '''
            APPLIES RECERSIVE AO* WITH BACKTRACKING FLAG
            IF FLAG = TRUE, UPDATE THE PREVIOUS H VALUES
            ELSE EXPLORE THE GRAPH FORWARD '''
        self.aoStar(self.start, False)
```

```
    def getNeighbors(self, v):
        # GET NEIGHBOURS OF NODE v
        return self.graph.get(v,'')


    def getStatus(self,v):
        # RETURN STATUS OF NODE v
        return self.status.get(v,0)


    def setStatus(self, v, val):
        # SETS THE STATUS OF NODE v
        self.status[v]=val


    def getHeuristicNodeValue(self, n):
        # RETURNS H VALUE OF NODE v
        return self.H.get(n, 0)
```

```
def setHeuristicNodeValue(self, n, value):
    # SET THE NEW H VALUE OF NODE v
    self.H[n]=value


def printSolution(self):
    print("\nFOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE:", self.start)
    print("------------------------------------------------------------")
    print(self.solutionGraph)
    print("------------------------------------------------------------")
```

```
    def computeMinimumCostChildNodes(self, v):
        # COMPUTES THE PATH WITH MINIMUM COST
        minimumCost = 0
        minimumPath = []
        first = True
        for childNodesInTuples in self.getNeighbors(v):
            # ITERATING OVER ALL CHILD NODES
            cost=0
            nodeList=[]
            for c, weight in childNodesInTuples:
                cost = cost + self.getHeuristicNodeValue(c) + weight
                nodeList.append(c)

            # INITIALIZING MINIMUM COST = FIRST CHILD COST
            if first:
                minimumCost=cost
                minimumPath=nodeList
                first = False


             # UPDATING MINIMUM COST WITH CURRECT MINIMUM COST
             elif minimumCost > cost:
                    minimumCost = cost
                    minimumPath = nodeList

         # RETURN MINIMUM COST WITH PATH
         return minimumCost, minimumPath
```

```
# AO* ALGORITHM FOR START NODE WITH BACKTRACKING FLAG
def aoStar(self, v, backTracking):
    print("HEURISTIC VALUES :", self.H)
    print("SOLUTION GRAPH :", self.solutionGraph)
    print("PROCESSING NODE :", v)
    if v == self.start:
        print('START: ', v)
    print("------------------------------------------------------------------------------------------")

    # IF STATUS OF NODE n >= 0, THEN COMPUTE ITS MINIMUM COST
    print(f'STATUS OF {v} = {self.getStatus(v)}')
    if self.getStatus(v) >= 0:
        minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
        print(f'\nSELECTED PATH {childNodeList} FROM {v} WITH COST = {minimumCost}')
        self.setHeuristicNodeValue(v, minimumCost)
        self.setStatus(v,len(childNodeList))
        # CHECK WHETHER THE MINIMUM COST NODE FROM NODE v ARE SOLVED
        solved=True
```

```
            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved = False

            # IF MINIMUM COST NODE v ARE SOLVED, THEN SET STATUS OF CURRENT NODE AS -1 (SOLVED)
            # print('NODE V =', v, 'IS SOLVED =', solved, 'DO BACKTRACKING =', backTracking)
            if solved==True:
                self.setStatus(v, -1)
                # UPDATE THE SOLUTION GRAPH WITH SOLVED NODES
                self.solutionGraph[v] = childNodeList

            # IF CURRENT NODE IS NOT START NODE, THEN BACKTRACK FOR H VALUE UPDATION
            if v != self.start:
                self.aoStar(self.parent[v], True)
```

```
            # IF BACKTRACKING IS FALSE, THEN EXPLORE THE GRAPH AHEAD
            if not backTracking:
                # FOR EACH NODE IN MINIMUM COST PATH
                for childNode in childNodeList:
                    # SETTING THE STATUS OF CHILD NODE = 0
                    self.setStatus(childNode,0)
                    # EXPLORE THE MINIMUM COST CHILD NODE WITH NO BACKTRACKING
                    self.aoStar(childNode, False)
```

```
# HEURISTIC VALUE OF ALL THE NODES
H = {'MUMBAI': 0,
     'TRAVEL': 0,
     'TRAIN': 2200, 'BUS': 2000, 'AIR': 7000,
     'HOTEL BOOKING': 0,
     'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0,
     'STAY': 0, 'MEAL': 0, 'PACKAGE':0,
     'VANITY VAN': 0,
     'COOK': 2000, 'COST OF VANITY VAN': 30000, 'EAT OUTSIDE': 3000}


# DIRECTED GRAPH OF THE AND/OR TREE
graph = {
    'MUMBAI': [[('TRAVEL', 0), ('HOTEL BOOKING', 0)], [('VANITY VAN', 100)]],
    'TRAVEL': [[('TRAIN', 4090)], [('BUS', 1180)], [('AIR', 4050)]],
    'HOTEL BOOKING': [[('MARIA RIO', 600)],[('WESTERN', 1340)],[('KENNEL WORTH', 4030)]],
    'MARIA RIO': [ [('STAY', 8000), ('MEAL', 4000)], [('PACKAGE', 40000)]],
    'WESTERN': [[('STAY', 34000), ('MEAL', 6000)],[('PACKAGE', 14000)]],
    'KENNEL WORTH': [[('STAY', 20000), ('MEAL', 6000)],[('PACKAGE', 32000)]],
    'VANITY VAN': [[('COOK', 0), ('COST OF VANITY VAN', 30000)],[('EAT OUTSIDE', 0), ('COST OF VANITY VAN', 30000)]]
}
```

```
G = Graph(graph, H, 'MUMBAI')
print('APPLYING THE AO* ALGORTIHM:\n')
G.applyAOStar()
G.printSolution()
```

```
APPLYING THE AO* ALGORITHM:

HEURISTIC VALUES : {'MUMBAI': 0, 'TRAVEL': 0, 'TRAIN': 2200, 'BUS': 2000, 'AIR': 7000, 'HOTEL BOOKING': 0, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'COST O
SOLUTION GRAPH : {}
PROCESSING NODE : MUMBAI
START:  MUMBAI
--------------------------------------------------------------------------------
STATUS OF MUMBAI = 0

SELECTED PATH ['TRAVEL', 'HOTEL BOOKING'] FROM MUMBAI WITH COST = 0
HEURISTIC VALUES : {'MUMBAI': 0, 'TRAVEL': 0, 'TRAIN': 2200, 'BUS': 2000, 'AIR': 7000, 'HOTEL BOOKING': 0, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'COST O
SOLUTION GRAPH : {}
PROCESSING NODE : TRAVEL
--------------------------------------------------------------------------------
STATUS OF TRAVEL = 0

SELECTED PATH ['BUS'] FROM TRAVEL WITH COST = 3180
HEURISTIC VALUES : {'MUMBAI': 0, 'TRAVEL': 3180, 'TRAIN': 2200, 'BUS': 2000, 'AIR': 7000, 'HOTEL BOOKING': 0, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'COS
SOLUTION GRAPH : {}
PROCESSING NODE : MUMBAI
START:  MUMBAI
--------------------------------------------------------------------------------
STATUS OF MUMBAI = 2

SELECTED PATH ['VANITY VAN'] FROM MUMBAI WITH COST = 100
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 3180, 'TRAIN': 2200, 'BUS': 2000, 'AIR': 7000, 'HOTEL BOOKING': 0, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'C
SOLUTION GRAPH : {}
PROCESSING NODE : BUS
--------------------------------------------------------------------------------
STATUS OF BUS = 0

SELECTED PATH [] FROM BUS WITH COST = 0
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 3180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 0, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'COS
SOLUTION GRAPH : {'BUS': []}
PROCESSING NODE : TRAVEL
--------------------------------------------------------------------------------
STATUS OF TRAVEL = 1

SELECTED PATH ['BUS'] FROM TRAVEL WITH COST = 1180
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 0, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'COS
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS']}
PROCESSING NODE : MUMBAI
START:  MUMBAI
--------------------------------------------------------------------------------
STATUS OF MUMBAI = 1

SELECTED PATH ['VANITY VAN'] FROM MUMBAI WITH COST = 100
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 0, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'COS
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS']}
PROCESSING NODE : HOTEL BOOKING
--------------------------------------------------------------------------------
STATUS OF HOTEL BOOKING = 0

SELECTED PATH ['MARIA RIO'] FROM HOTEL BOOKING WITH COST = 600
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 600, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'CO
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS']}
PROCESSING NODE : MUMBAI
START:  MUMBAI
--------------------------------------------------------------------------------
STATUS OF MUMBAI = 1

SELECTED PATH ['VANITY VAN'] FROM MUMBAI WITH COST = 100
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 600, 'MARIA RIO': 0, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000, 'CO
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS']}
PROCESSING NODE : MARIA RIO
--------------------------------------------------------------------------------
STATUS OF MARIA RIO = 0

SELECTED PATH ['STAY', 'MEAL'] FROM MARIA RIO WITH COST = 12000
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 600, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000,
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS']}
PROCESSING NODE : HOTEL BOOKING
--------------------------------------------------------------------------------
STATUS OF HOTEL BOOKING = 1

SELECTED PATH ['WESTERN'] FROM HOTEL BOOKING WITH COST = 1340
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS']}
PROCESSING NODE : MUMBAI
START:  MUMBAI
--------------------------------------------------------------------------------
STATUS OF MUMBAI = 1

SELECTED PATH ['VANITY VAN'] FROM MUMBAI WITH COST = 100
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000,
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS']}
PROCESSING NODE : STAY
--------------------------------------------------------------------------------
STATUS OF STAY = 0

SELECTED PATH [] FROM STAY WITH COST = 0
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000,
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS'], 'STAY': []}
PROCESSING NODE : MARIA RIO
--------------------------------------------------------------------------------
STATUS OF MARIA RIO = 2

SELECTED PATH ['STAY', 'MEAL'] FROM MARIA RIO WITH COST = 12000
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000,
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS'], 'STAY': []}
PROCESSING NODE : HOTEL BOOKING
--------------------------------------------------------------------------------
STATUS OF HOTEL BOOKING = 1

SELECTED PATH ['WESTERN'] FROM HOTEL BOOKING WITH COST = 1340
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000,
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS'], 'STAY': []}
PROCESSING NODE : MUMBAI
START:  MUMBAI
--------------------------------------------------------------------------------
STATUS OF MUMBAI = 1
```

```
SELECTED PATH ['VANITY VAN'] FROM MUMBAI WITH COST = 100
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS'], 'STAY': []}
PROCESSING NODE : MEAL
--------------------------------------------------------------------------------
STATUS OF MEAL = 0

SELECTED PATH [] FROM MEAL WITH COST = 0
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS'], 'STAY': [], 'MEAL': []}
PROCESSING NODE : MARIA RIO
--------------------------------------------------------------------------------
STATUS OF MARIA RIO = 2

SELECTED PATH ['STAY', 'MEAL'] FROM MARIA RIO WITH COST = 12000
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS'], 'STAY': [], 'MEAL': [], 'MARIA RIO': ['STAY', 'MEAL']}
PROCESSING NODE : HOTEL BOOKING
--------------------------------------------------------------------------------
STATUS OF HOTEL BOOKING = 1

SELECTED PATH ['WESTERN'] FROM HOTEL BOOKING WITH COST = 1340
HEURISTIC VALUES : {'MUMBAI': 100, 'TRAVEL': 1180, 'TRAIN': 2200, 'BUS': 0, 'AIR': 7000, 'HOTEL BOOKING': 1340, 'MARIA RIO': 12000, 'WESTERN': 0, 'KENNEL WORTH': 0, 'STAY': 0, 'MEAL': 0, 'PACKAGE': 0, 'VANITY VAN': 0, 'COOK': 2000
SOLUTION GRAPH : {'BUS': [], 'TRAVEL': ['BUS'], 'STAY': [], 'MEAL': [], 'MARIA RIO': ['STAY', 'MEAL']}
PROCESSING NODE : MUMBAI
START:  MUMBAI
--------------------------------------------------------------------------------
STATUS OF MUMBAI = 1

SELECTED PATH ['VANITY VAN'] FROM MUMBAI WITH COST = 100

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: MUMBAI
------------------------------------------------------------
{'BUS': [], 'TRAVEL': ['BUS'], 'STAY': [], 'MEAL': [], 'MARIA RIO': ['STAY', 'MEAL']}
------------------------------------------------------------
```

THUS, ALGORITHM SUGGEST TO TRAVEL GOA BY BUS, AND BOOK HOTEL MARIA RIO WITH STAY AND MEAL