Experiment 02: AI Problem Formulation and Solving

Learning Objective: Student should be able to apply AI problem formulation approach to solve any problem.

Tools: Python under Windows or Linux environment/Online Platform

Theory:

Formulating problems : A problem is defined by four items:

A problem is defined by four items:

1. initial state e.g., "at Arad"
2. actions or successor function or operators

   S(x) = set of action–state pairs

   – e.g., *S(Arad)* = {*Arad* → *Zerind*, *Zerind* → , … }

3. goal test, can be

   – explicit, e.g., *x* = "at Bucharest"

   – implicit, e.g., *Checkmate(x)*

4. path cost function (additive)

   – e.g., sum of distances, number of actions executed, etc.

   – *c(x,a,y)* is the step cost, assumed to be ≥ 0

A solution is a sequence of actions leading from the initial state to a goal state

Implementaion:

Problem Formulation:

(1) Initial state:

(2) Actions or successor function or operators :

(3) Goal test

(4) Path cost function

(5) Solution (Please include state space search graph)

Result and discussion: ……

Learning Outcomes: Students should have the ability to

LO1: Formulate the problem using AI Approach.

LO2 : Solve the problem using AI Approach.

Course Outcomes: Upon completion of the course students will be able to evaluate various problem solving methods for an agent to find a sequence of actions to reach the goal state.

Conclusion: ……

Viva Questions:

Q. 1 Define problem solving agent with example.

Q.2 Illustrate 3 problems can be solved using problem formulation approach.

Q.3 Discuss various items or elements to be considered during problem formulation.

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				

Ashutosh A. Sharma
Class: TE - COMP C
Roll no: 13

Experiment 3(a):
Uninformed Search

Learning Objective: Students should be able to solve a given problem using uninformed search technique

Tools: Python under Windows or Linux Environment

Theory: Study and implement DLS or DFIDS uninformed search techniques.

Depth limited search: Depth first search has some desirable properties as space, but if wrong branch is expanded with no solution on it then it may not terminate. Thus, introduce limit on branches to be expanded. Hence, not expanded a branch below a particular depth. Hence, DLS will always terminate with solutions if one exists in the limit previously set before running the program.
Too small bounds misses on the solution and too large bound may find poor solution when there are better ones. It may also run for a very long time thus remaving it's advantage over DFS.

eg. Romania problem - only 20 cities on the map. Therefore, no path longer than 19 units.

Achutosh A. Sharma
COMPC -13

Depth first iterative deepening search: Choosing depth bound provides incomplete or poor solution. It may also give no solution. This variation is complete and finds the best possible solution

Algorithm (DFIDS)

```
// returns true if target is reachable
// from src within max-depth.
bool DFIDS (src, target, max-depth)
    for limit from zero to max_depth
        if DLS (src, target, limit) == True
            return True
    return false

bool DLS (src, target, limit)
    if (src == target)
        return true
    // if reached the maximum depth
    // stop recursing

    if (limit <= 0)
        return false

    for each adjacent i of src
        if DLS (i, target, limit-1)
            return true
    return false
```

Ashutosh A. Sharma
COMPC-13

Advantages:

1. DFIDS gives us the hope to find the solution if it exists in the tree
2. When the solutions are found at lower depths, then the algorithm proves to be efficient in time.
3. Though the work done here is more, performance of DFIDS is better than single BFS and DFS operating exclusively

Disadvantages:

1. The time taken is exponential to reach the goal node and it increases greatly as the depth increases.
2. The main problem with DFIDS is the time and wasted calculations at

Properties:

Complete: Depth first iterative deepening search algorithm is complete if the branching factor is finite

Time : Let's suppose 'b' is the branching factor and depth is 'd' then the worst case time- complexity of algorithm is $O(b^d)$

Space : The space complexity of DFIDS will be $O(bd)$

Optimal : DFIDS algorithm is optimal if path cost is a non-decreasing function of the depth of the node

Applications: DFIDS is used when we do not know the depth of our solution and have to search a very large state space. It may also be used as a slightly slower substitute for BFS if we are constrained by memory or space

Design:

Program and Output:

Ashutosh A. Shama
COMPC-13

Subject :- IIS Experiment / Tutorial / Assignment No. :- 3 Page :-

Experiment 3b:
Informed Search

Learning objectives: Students should be able to solve a given problem using informed search technique

Tools : Python under Windows or Linux environment / Online Platform

Theory: Study and implement Best First search or S* search under informed search techniques

Algorithm:

1. Create an empty Priority Queue
   Priority Queue pq;

2. Insert "start" in pq
   pq.insert(start)

3. Untill priority queue is empty
   u = PriorityQueue.deletemin
   If u is the goal
   Exit
   Else
   For each neighbour v of u
   If v "unvisited"
   Mark v "visited"
   pq.insert(v)
   Mark as examined
   End.

Ashutosh A. Sharma
COMP C-13

Subject :- IIS Experiment / Tutorial / Assignment No. :- 3 Page :-

Advantages:

1. Best First search can switch between BFS and DFS by gaining the advantage of both the algorithms
2. This algorithm is more efficient than BFS and DFS algorithms

Disadvantages:

1. It can behave as an unguided depth first search in the worst case scenario.
2. It can get stuck in a loop like DFS
3. It is not an optimal algorithm wrt time and space

Properties:

Complete: Best first search is incomplete even if the state space is finite.

Time: The worst case time complexity of best first search is $O(b^m)$ where b is the branching factor and m is the maximum depth of the search space

Space: The worst case space complexity of Best first search is $O(b^m)$

Optimal: Best First search is not an optimal algorithm

Ashutosh A. Sharma
COMPC 43

Subject :- IIS Experiment / Tutorial / Assignment No. :- 3 Page :-

Applications: Best First search or A$\S$* algorithm is used to predict the closeness of the end of the path and it's solution. It is used to decide which adjacent branch is most promising and then explore

Design:
Code and output

Result and Discussion: (i) Uninformed search algorithms do not know about the goal state. (ii) Informed search algorithms have some information such as distance of nodes wrt goal node to calculate the minimum distance in minimum time

Learning Outcomes: Students should have the ability to
LO1: identify a problem which can be solved using uninformed search methods
LO2: implement uninformed search methods
LO3: describe properties of uninformed search algorithm
LO4: identify advantage and disadvantage of the algorithm.

Course outcomes: Upon completion of the course students will be able to evaluate various problem solving methods for an agent to find a sequence of actions to reach the goal state

Conclusion: In this experiment different informed and uninformed algorithms were understood and implemented. Thus, the experiment was successfully completed.

Experiment 05 – Apply Genetic Algorithm on given problem

Learning Objective: Student should be able to solve an optimization problem using Genetic Algorithm.

Tools: Python under Windows or Linux environment

Theory: Study and implement Genetic Algorithm for optimization problem.

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

ALGORITHM: Genetic Algorithm

1. START
2. Generate the initial population
3. Compute fitness
4. REPEAT

   Selection

   Crossover

   Mutation

   Compute fitness

   UNTIL population has converged

5. STOP

ADVANTAGES:

1. Parallelism
2. Global optimization
3. A larger set of solution space
4. Requires less information
5. Provides multiple optimal solutions
6. Probabilistic in nature
7. Genetic representations using chromosomes

DISADVANTAGES:

1. The need for special definitions
2. Hyper-parameter tuning
3. Computational complexity

APPLICATIONS:

1. Feature Selection
2. Model Hyper-parameter Tuning
3. Machine Learning Pipeline Optimization

Design:

Experiment 05 – Apply Minimax with alpha beta pruning on given problem

Learning Objective: Student should be able to solve an Minimax with Alpha-Beta Pruning for any two player games.

Tools: Python under Windows or Linux environment

Theory: Study and implement Adversarial search for Minimax with Alpha-Beta Pruning.

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision. To illustrate this with a real-life example, suppose somebody is playing chess, and it is their turn. Move "A" will improve the player's position. The player continues to look for moves to make sure a better one hasn't been missed. Move "B" is also a good move, but the player then realizes that it will allow the opponent to force checkmate in two moves. Thus, other outcomes from playing move B no longer need to be considered since the opponent can force a win. The maximum score that the opponent could force after move "B" is negative infinity: a loss for the player. This is less than the minimum position that was previously found; move "A" does not result in a forced loss in two moves.

The algorithm maintains two values, alpha and beta, which respectively represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of. Initially, alpha is negative infinity and beta is positive infinity, i.e. both players start with their worst possible score. Whenever the maximum score that the minimizing player (i.e. the "beta" player) is assured of becomes less than the minimum score that the maximizing player (i.e., the "alpha" player) is assured of (i.e. beta < alpha), the maximizing

player need not consider further descendants of this node, as they will never be reached in the actual play.

ALGORITHM: Alpha-Beta Pruning Algorithm (English Steps)

- Initiate the search, down the tree, to the given depth.
- After reaching the bottom, determine the utilities of the higher nodes, using the utilities of the terminal nodes.
- Based on the branching factors of whether the move backtracked was made by the opponent or the computer, backtrack, propagate the values and path.
- Finally, when the search is complete, the Alpha value at the top node gives the maximum score that the player is guaranteed to score.

ADVANTAGES:

1. Allows elimination of the search tree branches.
2. Limits the search time to more promising sub-trees, which enables a deeper search.

DISADVANTAGES:

1. It does not solve all the problems associated with the original minimax algorithm.
2. Requires a set depth limit, as in most cases, it is not feasible to search the entire game tree.

APPLICATIONS:

It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Connect 4, etc.).

Design: Alpha-Beta Pruning Algorithm (Pseudo Code)

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):
   if node is a leaf node :
      return value of the node
   if isMaximizingPlayer :
      bestVal = -INFINITY
      for each child node :
         value = minimax(node, depth+1, false, alpha, beta)
```

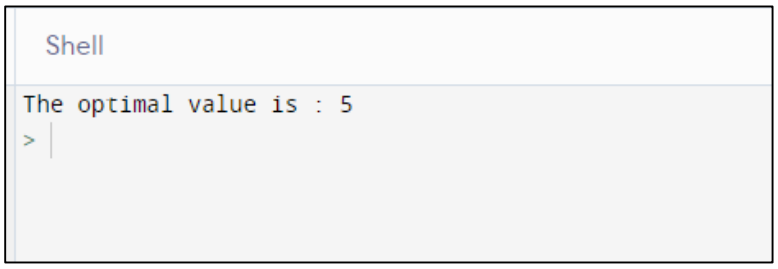
```
        bestVal = max( bestVal, value)
        alpha = max( alpha, bestVal)
        if beta <= alpha:
            break
    return bestVal

else :
    bestVal = +INFINITY
    for each child node :
        value = minimax(node, depth+1, true, alpha, beta)
        bestVal = min( bestVal, value)
        beta = min( beta, bestVal)
        if beta <= alpha:
            break
    return bestVal
```

CODE:

```
# Python3 program to demonstrate
# working of Alpha-Beta Pruning

# Initial values of Alpha and Beta
MAX, MIN = 1000, -1000

# Returns optimal value for current player
#(Initially called for root and maximizer)
def minimax(depth, nodeIndex,
maximizingPlayer,
                    values, alpha, beta):

    # Terminating condition. i.e
    # leaf node is reached
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:

        best = MIN

        # Recur for left and right
children
        for i in range(0, 2):

            val = minimax(depth
+ 1, nodeIndex * 2 + i,

                False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha,
best)

            # Alpha Beta Pruning
            if beta <= alpha:
                break

        return best

    else:
        best = MAX

        # Recur for left and
        # right children
        for i in range(0, 2):
```

```
                    val = minimax(depth
+ 1, nodeIndex * 2 + i,                                  return best

        True, values, alpha, beta)              # Driver Code
                    best = min(best, val)       if __name__ == "__main__":
                    beta = min(beta, best)

                                                        values = [3, 5, 6, 9, 1, 2, 0, -1]
                    # Alpha Beta Pruning                print("The optimal value is :",
                    if beta <= alpha:           minimax(0, 0, True, values, MIN, MAX))
                            break
```

OUTPUT:



Result and Discussion:

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.

Learning Outcomes: The student should have the ability to

LO6.1 Identify the need of Adversarial search.

LO6.2 Understand the constraints on which alpha beta pruning works.

LO6.3 Understand how alpha beta pruning optimizes mini max algorithms.

LO4.4 Identify an advantages and disadvantages of alpha beta pruning algorithm.

Course Outcomes: Understand and apply various AI search algorithms adversarial search algorithms to real-world problems (**Two Player Gaming**).

Conclusion: Thus, we are able to solve the min max algorithm using alpha beta pruning

Viva Questions:

1. List initial values alpha and beta. Justify your answer.
2. What kinds of problems are solved through Adversarial Search?
3. State the condition when pruning happens?

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				

Experiment 07: Local Search Algorithms

Learning Objective: Student should be able to solve a given problem using Local Search Technique.

Tools: Python under Windows or Linux environment

Theory: Study and implement Hill Climbing or Simulated Annealing Search techniques.

- **Local Search Algorithm:** Local Search in Artificial Intelligence is an optimizing algorithm to find the optimal solution more quickly. Local search algorithms are used when we care only about a solution but not the path to a solution. Local search is used in most of the models of AI to search for the optimal solution according to the cost function of that model. Local search is used in linear regression, neural networks, clustering models. Hill climbing, simulated annealing.

- **Hill Climbing Algorithm:** 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 neighbor has a higher value.

  It is a technique which is used for optimizing the mathematical problems. E.g. Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman**.**

- **Simulated Annealing:** The algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

ALGORITHM: (for selected search technique)

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
  a. If it is goal state, then return success and quit.
  b. Else if it is better than the current state then assign new state as a current state.
  c. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

ADVANTAGES:

1. Hill Climbing can be used in continuous as well as domains.
2. Hill climbing technique is very useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing.

DISADVANTAGES:

1. **Plateau:** It is a flat area of the search space in which a whole set of neighbouring states(nodes) have the same order. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.
2. **Ridge:** It is a special kind of local maximum. it is an area of the search space which is higher than the surrounding areas and that itself has a slope.

APPLICATIONS:

1. Hill Climbing technique can be used to solve many problems, such as Network-Flow, Travelling Salesman problem, 8-Queens problem, Integrated Circuit design, etc.
2. This technique is also used in robotics for coordinating multiple robots in a team.

Design/ Implementation:

CODE:

```
import random
def randomSolution(tsp):
    cities = list(range(len(tsp)))
    solution = []

    for i in range(len(tsp)):
        randomCity = cities[random.randint(0, len(cities) - 1)]
        solution.append(randomCity)
        cities.remove(randomCity)
    return solution
def routeLength(tsp, solution):
    routeLength = 0
    for i in range(len(solution)):
        routeLength += tsp[solution[i - 1]][solution[i]]
    return routeLength

def getNeighbours(solution):
    neighbours = []
    for i in range(len(solution)):
        for j in range(i + 1, len(solution)):
            neighbour = solution.copy()
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
            neighbours.append(neighbour)
    return neighbours
def getBestNeighbour(tsp, neighbours):
    bestRouteLength = routeLength(tsp, neighbours[0])
    bestNeighbour = neighbours[0]
    for neighbour in neighbours:
```

```
    currentRouteLength = routeLength(tsp, neighbour)
    if currentRouteLength < bestRouteLength:
        bestRouteLength = currentRouteLength
        bestNeighbour = neighbour
  return bestNeighbour, bestRouteLength

def hillClimbing(tsp):
  currentSolution = randomSolution(tsp)
  currentRouteLength = routeLength(tsp, currentSolution)
  neighbours = getNeighbours(currentSolution)
  bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp, neighbours)

  while bestNeighbourRouteLength < currentRouteLength:
    currentSolution = bestNeighbour
    currentRouteLength = bestNeighbourRouteLength
    neighbours = getNeighbours(currentSolution)
    bestNeighbour, bestNeighbourRouteLength = getBestNeighbour(tsp, neighbours)

  return currentSolution, currentRouteLength

def main():
  tsp = [
    [0, 400, 500, 300],
    [400, 0, 300, 500],
    [500, 300, 0, 400],
    [300, 500, 400, 0]
  ]

  print(hillClimbing(tsp))

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

OUTPUT

```
Shell
([2, 1, 0, 3], 1400)
>
```

Result and discussion:

Local Search in Artificial Intelligence is an optimizing algorithm to find the optimal solution more quickly. Local search algorithms are used when we care only about a solution but not the path to a solution.

Learning Outcomes: Students should have the ability to

LO1: identify a problem which can be solved using local search methods.

LO2: implement local search methods.

LO3: describe properties of local search algorithm.

LO4: identify advantage and disadvantage of the algorithm.

Course Outcomes: Upon completion of the course students will be able to evaluate various problem-solving methods for an agent to find a sequence of actions to reach the goal state.

Conclusion: Thus, we are able to solve the hill climbing algorithmn

Viva Questions:

1. Describe the drawbacks of hill climbing algorithm.
2. Name which algorithm overcomes drawback of hill climbing and how?
3. Compare and contrast local search strategies w.r.t Travelling Salesman Problem.

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				

Experiment 06 – Solve Constraint Statisfaction Problem

Learning Objective: Student should be able to solve Constraint Satisfaction problem.

Tools: Python under Windows or Linux environment

Theory:

A constraint satisfaction problem (CSP) is a problem that requires its solution to be within some limitations or conditions, also known as constraints, consisting of a finite variable set, a domain set and a finite constraint set. Given a partially filled 9×9 2D array 'grid[9][9]', the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.

Advantages:

Disadvantages:

Complexity Analysis:

- **Time complexity:** O(9^(n*n))
  For every unassigned index, there are 9 possible options so the time complexity is O(9^(n*n)).

- **Space Complexity:** O(n*n).
  To store the output array a matrix is needed.

Applications:

Result and Discussion:

Code:

```
N = 9
def printing(arr):
        for i in range(N):
                for j in range(N):
                        print(arr[i][j], end = " ")
                print()
# Checks whether it will be legal to assign num to the given row, col
def isSafe(grid, row, col, num):

        # Check if we find the same num in the similar row , we return false
```

```
        for x in range(9):
                if grid[row][x] == num:
                        return False

        # Check if we find the same num in the similar column , we return false
        for x in range(9):
                if grid[x][col] == num:
                        return False

        # Check if we find the same num in the particular 3*3 matrix, we return false
        startRow = row - row % 3
        startCol = col - col % 3
        for i in range(3):
                for j in range(3):
                        if grid[i + startRow][j + startCol] == num:
                                return False
        return True
# Takes a partially filled-in grid and attempts to assign values to all unassigned locations in such a way to meet the requirements for
def solveSudoku(grid, row, col):

        # Check if we have reached the 8th row and 9th column (0 indexed matrix)
        if (row == N - 1 and col == N):
                return True

        if col == N:
                row += 1
                col = 0

        # Check if the current position of the grid already contains value >0, we iterate for next column
        if grid[row][col] > 0:
                return solveSudoku(grid, row, col + 1)
        for num in range(1, N + 1, 1):

                if isSafe(grid, row, col, num):

                        grid[row][col] = num

                        if solveSudoku(grid, row, col + 1):
                                return True
                grid[row][col] = 0
        return False
grid = [[3, 0, 6, 5, 0, 8, 4, 0, 0],
      [5, 2, 0, 0, 0, 0, 0, 0, 0],
      [0, 8, 7, 0, 0, 0, 0, 3, 1],
      [0, 0, 3, 0, 1, 0, 0, 8, 0],
      [9, 0, 0, 8, 6, 3, 0, 0, 5],
      [0, 5, 0, 0, 9, 0, 6, 0, 0],
      [1, 3, 0, 0, 0, 0, 2, 5, 0],
      [0, 0, 0, 0, 0, 0, 0, 7, 4],
      [0, 0, 5, 2, 0, 6, 3, 0, 0]]

if (solveSudoku(grid, 0, 0)):
        printing(grid)
else:
        print("no solution exists ")
```

Output:

```
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
```

Learning Outcomes:

The student should have the ability to
LO4.1 identify the need of solving CSP
LO4.2 define CSP
LO4.3 apply search technique to solve CSP
LO4.4 identify advantages and disadvantages of the search algorithm used

Course Outcomes:

Evaluate search techniques to solve Constraint Satisfaction problem.

Conclusion:

Viva Questions:

1. Explain any Constraint Satisfaction problem.
2. What kind of CSP problems are solved through search techniques.
3. Explain how to solve a chosen CSP problem.

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				

Experiment 08– Apply Unification to solve a reasoning problem

Learning Objective: Solve a reasoning problem using unification.

Tools: Python under Windows or Linux environment.

Theory:

In logic and computer science, unification is an algorithmic process of solving equations between symbolic expressions. Depending on which expressions are allowed to occur in an equation set, and which expressions are considered equal, several frameworks of unification are distinguished. If higher-order variables, that is, variables representing functions, are allowed in an expression, the process is called **higher-order unification**, otherwise **first-order unification**. If a solution is required to make both sides of each equation literally equal, the process is called **syntactic** or **free unification**, otherwise **semantic** or **equational unification.**

Consider unifying the literal *P(x, g(x))* with:

1. *P(z,y)* : unifies with *{x/z, g(x)/y}*
2. *P(z, g(z))*: unifies with *{x/z}* or *{z/x}*
3. *P(Socrates, g(Socrates))* : unifies, *{Socrates/x}*
4. *P(z, g(y))*: unifies with *{x/z, x/y}* or *{z/x, z/y}*
5. *P(g(y), z)*: unifies with *{g(y)/x, g(g(y))/z}*
6. *P(Socrates, f(Socrates))* : does not unify: *f* and *g* do not match.
7. *P(g(y), y)* : does not unify: no substitution works.

Algorithm:

Step. 1: If Ψ_1 or Ψ_2 is a variable or constant, then:
- a) If Ψ_1 or Ψ_2 are identical, then return NIL.
- b) Else if $\Psi_1$is a variable,
  - a. then if Ψ_1 occurs in Ψ_2, then return FAILURE
  - b. Else return { (Ψ_2/ Ψ_1)}.
- c) Else if Ψ_2 is a variable,
  - a. If Ψ_2 occurs in Ψ_1 then return FAILURE,
  - b. Else return {(Ψ_1/ Ψ_2)}.
- d) Else return FAILURE.

Step.2: If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.
Step. 3: IF Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE.
Step. 4: Set Substitution set(SUBST) to NIL.
Step. 5: For i=1 to the number of elements in Ψ_1.
- a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2, and put the result into S.
- b) If S = failure then returns Failure

c) If S ≠ NIL then do,

a. Apply S to the remainder of both L1 and L2.

b. SUBST= APPEND(S, SUBST).

Step.6: Return SUBST.

Design:

```
function unify(E1, E2);
 begin
  case
    both E1 and E2 are constants or the empty list:  % base case
     if E1 = E2
       then return {}
       else return FAIL;
    E1 is a variable:
     if E1 occurs in E2
       then return FAIL;
       else return {E2/E1}
    E2 is a variable:
     if E2 occurs in E1
       then return FAIL;
       else return {E1/E2}
    otherwise:                          % both E1 and E2 are lists
     begin
       HE1 := first element of E1;
       HE2 := first element of E2;
       SUBS1 := unify(HE1, HE2);        %  recursion
       if SUBS1 = FAIL, then return FAIL;
       TE1 := apply(SUBS1, rest of E1);
       TE2 := apply(SUBS1, rest of E2);
       SUBS2 := unify(TE1, TE2);        %  recursion
       if SUBS2 = FAIL then return FAIL;
         else return composition(SUBS1, SUBS2)
     end
 end
```


$f(x_1, h(x_1), x_2\} = f(g(x_3), x_4, x_3)$

$б_1 = ((g(x_3), x_1), (x_3, x_2), (h(g(x_3), x_4)\}$

$б_2 = ((g(a), x_1), (a, x_2), (a, x_3), (h(g(a)), x_4)\}$.

Result and Discussion:

Learning Outcomes: The student should have the ability to

LO1 Ability to represent Problem

LO2 Ability to define Unification algorithm

LO3 Ability to apply Unification Algorithm

LO4 Ability to make inference from Problem representation

Course Outcomes: Upon completion of the course students will be able to review the strength and weakness of AI approaches to knowledge representation, reasoning and planning.

Conclusion:

Viva Questions:

Q.1 How propositional logic is different from predicate logic?
Q.2 Why unification is required in reasoning?
Q.3 Which problems are solved using unification?

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 09– Apply Decision Tree Learning on given problem

Learning Objective: Apply Decision Tree Learning on given problem.
Tools: Python under Windows or Linux environment.
Theory:
A **decision tree** is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm.
Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.
Advantages of decision trees:
Among decision support tools, decision trees (and influence diagrams) have several advantages.
Decision trees:
Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.
Have value even with little hard data. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes.
Allow the addition of new possible scenarios.
Help determine worst, best and expected values for different scenarios.
Use a white box model. If a given result is provided by a model.
Can be combined with other decision techniques.
Disadvantages of decision trees:
For data including categorical variables with different number of levels, information gain in decision trees is biased in favor of those attributes with more levels.[6]
Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.
Applications
`Agriculture
Astronomy

Algorithm:

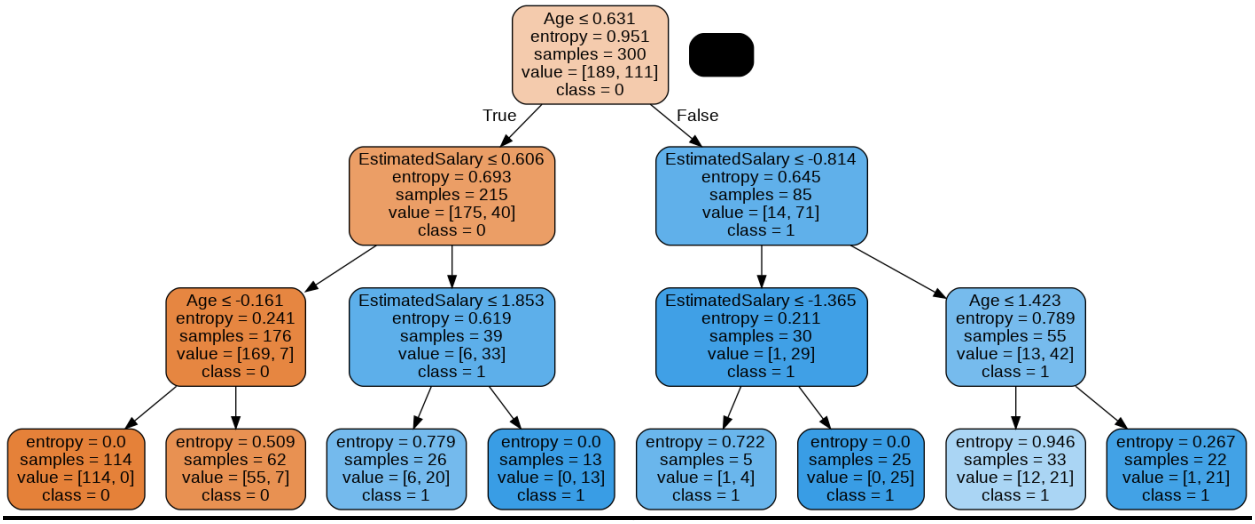
- It begins with the original set S as the root node.
- On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates **Entropy(H)** and **Information gain(IG)** of this attribute.
- It then selects the attribute which has the smallest Entropy or Largest Information gain.
- The set S is then split by the selected attribute to produce a subset of the data.
- The algorithm continues to recur on each subset, considering only attributes never selected before.

Design:

Code:-

```
#Importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
# Read the dataset
data = pd.read_csv('/content/suv_data.csv')
data.head()
feature_cols = ['Age','EstimatedSalary' ]
X = data.iloc[:,[2,3]].values
y = data.iloc[:,4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =  train_test_split(X,y,test_size = 0.25, random_state= 0)
#feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier = classifier.fit(X_train,y_train)
#prediction
y_pred = classifier.predict(X_test)#Accuracy
from sklearn import metrics
print('Accuracy Score:', metrics.accuracy_score(y_test,y_pred))
pip install pydotplus
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
# Create Decision Tree classifer object
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=3)# Train Decision Tree Classifer
classifier = classifier.fit(X_train,y_train)#Predict the response for test dataset
y_pred = classifier.predict(X_test)# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```


Output:-



Result and Discussion:

Learning Outcomes: The student should have the ability to
LO9.1 Ability to formulate Problem
LO9.2 Ability to represent problem
LO9.3 Ability to apply Learning algorithms
LO9.4 Ability to verify the program output to check values at specified location in Decision Tree

Course Outcomes:
Understand various forms of learning. Apply decision tree learning to a given problems.

Conclusion:

Viva Questions:
Q.1 Is decision tree learning is inductive or deductive learning?
Q.2 What are the basic types of decision tree algorithm?
Q.3 List advantages and disadvantages of decision tree algorithms?

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				