Digital Assignment - 2

Name:Puneeth Velakaturi
Reg no:22BCT0268
Course name:Artifical Intelligence
Course code:BCSE306L
Fall semester 2024-25

Question 1:

Questions:

Objective 1: To solve the 8 Puzzle problem using the Hill Climbing algorithm in Python, with heuristic guidelines and analyze its limitations, including local maxima and plateaus.

Question: Implement the Hill Climbing algorithm in Python to solve the 8 Puzzle problem, starting from a given initial configuration and aiming to reach a specified goal state. Use a heuristic function, either the Manhattan distance or the number of misplaced tiles, to guide the search process. Test your implementation with different initial configurations, document the steps taken to reach the solution, and analyze cases where the algorithm fails due to local maxima or plateaus. Discuss any observed limitations of the Hill Climbing approach in solving the 8 Puzzle problem.

Solution:

1. Problem Definition

The 8-Puzzle is a sliding puzzle consisting of a 3x3 grid with 8 numbered tiles and one blank space. The goal is to rearrange the tiles from a given initial configuration to a specified goal configuration using valid moves (up, down, left, right).

2. Algorithm: Hill Climbing

Hill Climbing is a heuristic search algorithm that:

*Starts from an initial state.
*Evaluates neighboring states using a heuristic function.
*Moves to the neighbor with the best heuristic value (lowest cost).
*Stops when no better neighbor is found or the goal state is reached.
*Limitations: The algorithm may get stuck in:

3. Python implementation

Program:

```
import numpy as np
goal_state = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]
def manhattan_distance(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            value = state[i][j]
            if value != 0:
                target_x = (value - 1) // 3
                target_y = (value - 1) % 3
                distance += abs(i - target_x) + abs(j - target_y)
    return distance
def get_neighbors(state):
    neighbors = []
    x, y = [(i, j) for i in range(3) for j in range(3) if state[i][j] == 0][0]
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            neighbor = [row[:] for row in state]
            neighbor[x][y], neighbor[nx][ny] = neighbor[nx][ny], neighbor[x][y]
            neighbors.append(neighbor)
    return neighbors
def hill_climbing(initial_state):
    current_state = initial_state
    current_cost = manhattan_distance(current_state)
    while True:
        neighbors = get_neighbors(current_state)
        neighbor_costs = [(manhattan_distance(neighbor), neighbor) for neighbor in neighbors]
        best_cost, best_neighbor = min(neighbor_costs, key=lambda x: x[0])
```

```
        if best_cost >= current_cost:
            break

        current_state = best_neighbor
        current_cost = best_cost

    return current_state, current_cost
initial_state = [
    [1, 2, 3],
    [4, 5, 0],
    [7, 8, 6]
]
final_state, final_cost = hill_climbing(initial_state)
print("Final State:")
for row in final_state:
    print(row)
print(f"Cost: {final_cost}")
```

Output:

```
PS C:\Users\vpune\OneDrive\Desktop\dbms project> & C:/Users/vpune/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/vpune/OneDrive/Desktop/dbms projec
t/daq1.py"
Final State:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
Cost: 0
```

4.Analysis of Failures

1. Local Maxima

In some cases, the algorithm may move to a state where all neighbors have higher heuristic costs. This occurs because Hill Climbing only evaluates immediate improvements.

Example:

If a state requires moving temporarily away from the goal (e.g., introducing more misplaced tiles), the algorithm will fail to proceed.

2. Plateaus
When multiple neighbors have the same heuristic value, Hill Climbing cannot distinguish between them. This often leads to stagnation.

Example:
A state where misplaced tiles are rearranged without reducing the total number of misplaced tiles.

3. Ridges
Ridges represent narrow pathways of improvement that the algorithm cannot traverse effectively due to its greedy nature.

5.Recommendations to Overcome Limitations
Random Restarts:
When stuck in a local maximum or plateau, restart the algorithm from a random configuration.

Simulated Annealing:
Introduce a probability of accepting worse states early in the process to escape local maxima.

Best-First Search:
Use a broader evaluation if states by considering both current and future heuristic values.

Question 2:

Objective 2: To create Prolog rules for determining student eligibility for scholarships and exam permissions based on attendance, integrating these rules with a REST API and web app for querying eligibility and debar status.

Question: Create a system in Prolog to determine student eligibility for scholarships and exam permissions based on attendance data stored in a CSV file. Write Prolog rules to evaluate whether a student qualifies for a scholarship or is permitted for exams, using specified attendance thresholds. Load the CSV data into Prolog, define the rules, and expose these eligibility checks through a REST API that can be accessed by a web app. Develop a simple web interface that allows users to input a student ID and check their eligibility status. Additionally, write a half-page comparison on the differences between SQL and Prolog for querying, focusing on how each handles data retrieval, logic-based conditions, and use case suitability.

Note:

1. CSV data sample (Student_ID, Attendance_percentage, CGPA)
2. Load data in Proglog

(:- use_module(library(csv)).

csv_read_file("data.csv", Rows, [functor(student), arity(4)]), mpalist(assert, Rows).

3. Define rules in Prolog

   Eligible_for_Scholarship(Student_ID) :- student(Student_ID,_, Attendance_percentage, CGPS), Attendance_Percentage>=75, CGPA >=9.0.

   Permitted_for_exam(Student_ID) :- student(Student_ID, _, Attendance, _), Attendance >= 75.
4. REST_API with HTTP Library
5. Web App to call the API.

Solution:

1.Writing Prolog Rules:

```
% Import CSV library to load data
:- use_module(library(csv)).

% Load student data from CSV
load_students(File) :-
    csv_read_file(File, Rows, [functor(student), arity(3)]),
    maplist(assert, Rows).
% Define eligibility for scholarship
eligible_for_scholarship(Student_ID) :-
    student(Student_ID, Attendance, CGPA),
    Attendance >= 75,
    CGPA >= 9.0.
% Define permission for exams
permitted_for_exam(Student_ID) :-
    student(Student_ID, Attendance, _),
    Attendance >= 75.
```

2. create a csv file:

Student_ID,Attendance,CGPA

101,85,9.5

102,65,8.0

103,80,9.0

104,70,7.5

3. Load and Query csv data:

```
C:\Users\vpune>cd C:\Users\vpune\OneDrive\Desktop\New folder

C:\Users\vpune\OneDrive\Desktop\New folder>swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.8)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [eligibility].
true.

2 ?- load_students('data.csv').
true.

3 ?- eligible_for_schorlarship(101).
Correct to: "eligible_for_scholarship(101)"? yes
true.

4 ?- permitted_for_exam(102).
false.
```

4. Expose the rules through a REST API:

(1) Add HTTP Library in Prolog

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).
:- use_module(library(http/http_json)).
:- use_module(library(http/http_parameters)).

% Start the server
start_server(Port) :-
    http_server(http_dispatch, [port(Port)]).
% Define endpoints
:- http_handler(root(scholarship), check_scholarship, []).
:- http_handler(root(exam_permission), check_exam_permission, []).
% Scholarship Eligibility Endpoint
check_scholarship(Request) :-
    http_parameters(Request, [student_id(Student_ID, [integer])]),
    (eligible_for_scholarship(Student_ID) ->
```

```
        Reply = json{student_id: Student_ID, eligible: true};
        Reply = json{student_id: Student_ID, eligible: false}),
    reply_json(Reply).
% Exam Permission Endpoint
check_exam_permission(Request) :-
    http_parameters(Request, [student_id(Student_ID, [integer])]),
    (permitted_for_exam(Student_ID) ->
        Reply = json{student_id: Student_ID, permitted: true};
        Reply = json{student_id: Student_ID, permitted: false}),
    reply_json(Reply).
```

(2) Step 2: Start the REST API Server

```
C:\Users\vpune>cd C:\Users\vpune\OneDrive\Desktop\New folder

C:\Users\vpune\OneDrive\Desktop\New folder>swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.8)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [eligibility].
true.

2 ?- start_server(8080).
% Started server at http://localhost:8080/
true.
```

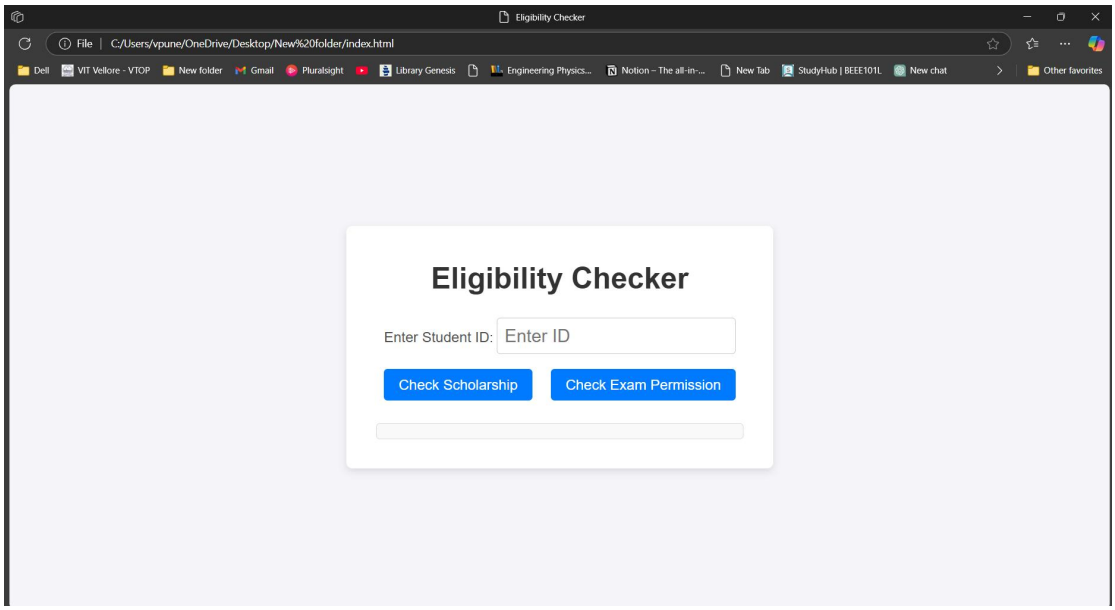
5. Create a web interface

HTML file:

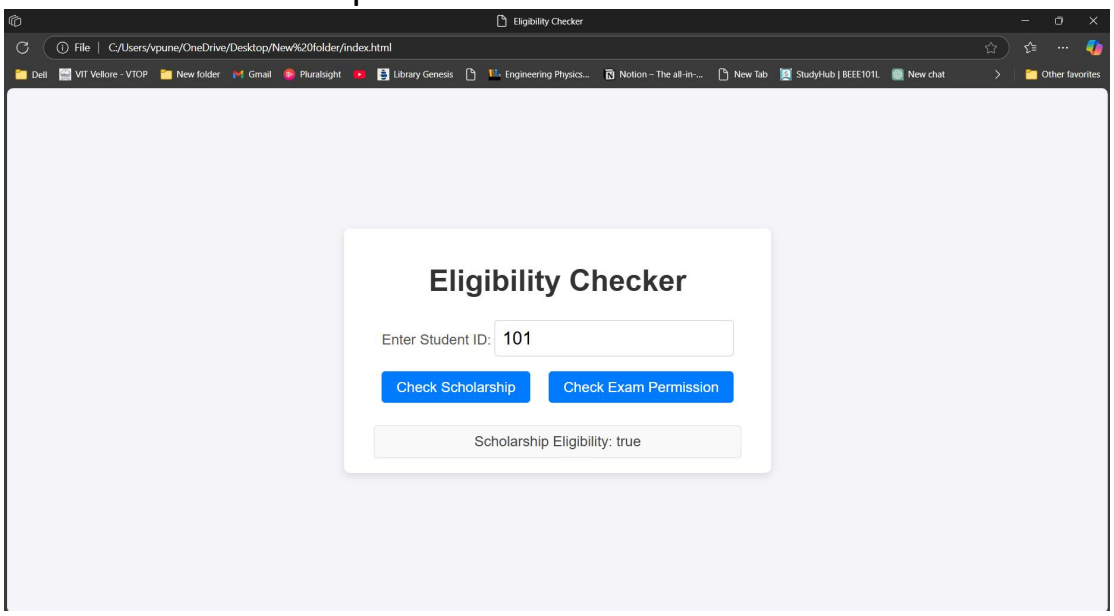
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Eligibility Checker</title>
    <style>
        body {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            font-family: Arial, sans-serif;
            background-color: #f4f4f9;
        }
        .container {
            text-align: center;
            background: #ffffff;
            padding: 20px 40px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
            border-radius: 8px;
```

```
        }
        h1 {
            font-size: 2.5rem;
            color: #333;
            margin-bottom: 20px;
        }
        label {
            font-size: 1.2rem;
            color: #555;
        }
        input {
            font-size: 1.5rem;
            margin: 10px 0;
            padding: 10px;
            width: 100%;
            max-width: 300px;
            border: 1px solid #ccc;
            border-radius: 5px;
        }
        button {
            font-size: 1.2rem;
            margin: 10px;
            padding: 10px 20px;
            color: #fff;
            background-color: #007bff;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
        button:hover {
            background-color: #0056b3;
        }
        #result {
            margin-top: 20px;
            font-size: 1.2rem;
            color: #444;
            background: #f9f9f9;
            padding: 10px;
            border: 1px solid #ddd;
            border-radius: 5px;
            text-align: left;
            max-width: 500px;
            margin-left: auto;
            margin-right: auto;
            white-space: pre-wrap;
        }
    </style>
    <script>
        async function checkEligibility(type) {
            const studentID = document.getElementById("student_id").value;
            const url = `http://localhost:8080/${type}?student_id=${studentID}`;
            try {
                const response = await fetch(url);
                const result = await response.json();
                document.getElementById("result").innerText = JSON.stringify(result, null, 2);
            } catch (error) {
                document.getElementById("result").innerText = "Error: Unable to fetch data.";
            }
        }
    </script>
</head>
```

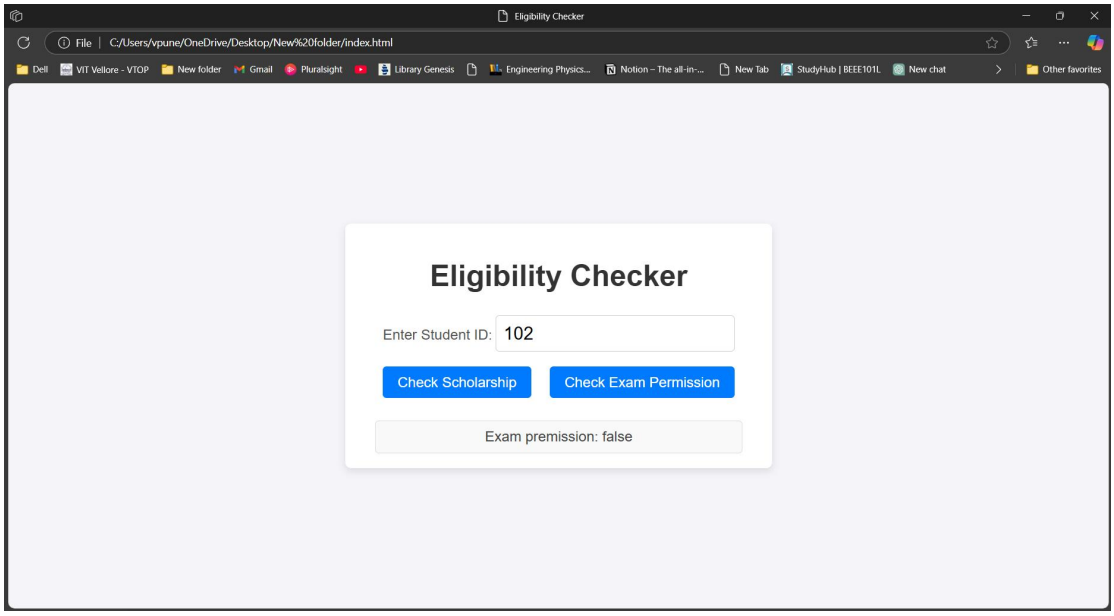
```
<body>
    <div class="container">
        <h1>Eligibility Checker</h1>
        <label for="student_id">Enter Student ID:</label>
        <input type="number" id="student_id" placeholder="Enter ID" required />
        <div>
            <button onclick="checkEligibility('scholarship')">Check Scholarship</button>
            <button onclick="checkEligibility('exam_permission')">Check Exam
Permission</button>
        </div>
        <pre id="result"></pre>
    </div>
</body>
</html>
```



Check for scholarship:



Check for EXAM premission:



Question 3:

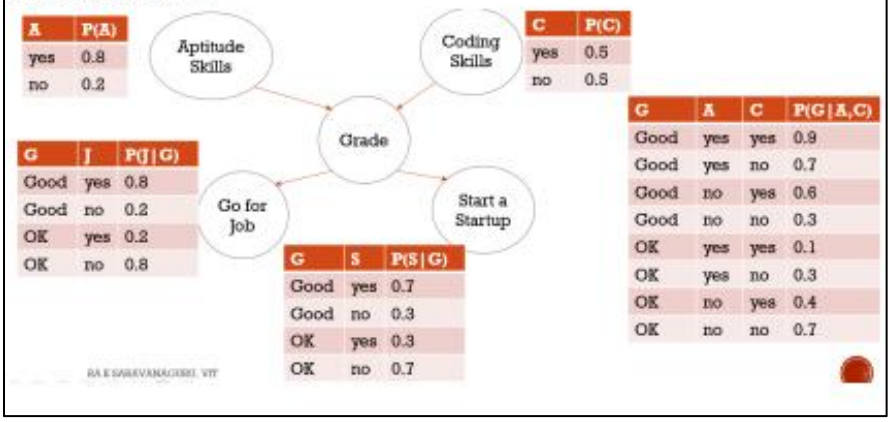
Objective 3: To use Monte Carlo simulation in Python to solve inference problems in a given Bayesian Belief Network (BBN) and analyze the results.

Question:

Given a Bayesian Belief Network (BBN) with defined nodes and conditional probability distributions, implement a Monte Carlo simulation in Python to perform inference on this network. Select a target node for which you want to compute the probability given evidence on one or more other nodes. Run the Monte Carlo simulation by generating random samples and estimating the conditional probability of the target node based on these samples. Provide the computed probability and discuss the accuracy of the result by comparing it to known values (if available) or explaining how sample size affects convergence in your simulation.

Problem :

As discussed in the class



Solution:

Step 1: Problem Statement:

We are tasked with calculating the conditional probability P(S | J=yes), which can be estimated using the Monte Carlo simulation. The known probabilities in the Bayesian Network are used to generate samples, and we evaluate how the result converges to the true value as the sample size increases.

Step 2: Define the Steps for Monte Carlo Simulation
Monte Carlo simulation involves the following:

.Generate random samples for each node based on the probability distributions.
.Compute the target probability (e.g., P(S | J)) by estimating frequencies from the generated samples.
.Assess convergence and accuracy by varying the sample size.

Step 3:Python implementation
Program:

```
import numpy as np

# Probability tables
P_A = {"yes": 0.8, "no": 0.2}
P_C = {"yes": 0.5, "no": 0.5}
P_G_given_A_C = {
    ("yes", "yes"): {"Good": 0.9, "OK": 0.1},
    ("yes", "no"): {"Good": 0.7, "OK": 0.3},
    ("no", "yes"): {"Good": 0.6, "OK": 0.4},
    ("no", "no"): {"Good": 0.3, "OK": 0.7},
}
P_J_given_G = {"Good": {"yes": 0.8, "no": 0.2}, "OK": {"yes": 0.2, "no": 0.8}}
P_S_given_G = {"Good": {"yes": 0.7, "no": 0.3}, "OK": {"yes": 0.3, "no": 0.7}}
# Sampling function for each node
def sample_node(prob_table):
    return "yes" if np.random.rand() < prob_table["yes"] else "no"
def sample_G(A, C):
    prob = P_G_given_A_C[(A, C)]
    return "Good" if np.random.rand() < prob["Good"] else "OK"
# Monte Carlo Simulation
def monte_carlo_simulation(sample_size=10000):
    samples = []
    for _ in range(sample_size):
        # Sample A and C
        A = sample_node(P_A)
        C = sample_node(P_C)

        # Sample G based on A and C
        G = sample_G(A, C)

        # Sample J and S based on G
        J = sample_node(P_J_given_G[G])
        S = sample_node(P_S_given_G[G])

        # Store the sample
        samples.append((A, C, G, J, S))
```

```
    return samples
# Compute conditional probability P(S | J)
def compute_conditional_probability(samples):
    count_J_yes = 0
    count_S_yes_given_J_yes = 0

    for _, _, _, J, S in samples:
        if J == "yes":
            count_J_yes += 1
            if S == "yes":
                count_S_yes_given_J_yes += 1

    return count_S_yes_given_J_yes / count_J_yes if count_J_yes > 0 else 0
# Run simulation
sample_size = 10000
samples = monte_carlo_simulation(sample_size)
P_S_given_J = compute_conditional_probability(samples)
# Output the result
print(f"Estimated P(S | J): {P_S_given_J:.4f}")
```

Output:

Sample:1000

```
PS C:\Users\vpune\OneDrive\Desktop\dbms project> & C:/Users/vpune/AppData/Local/Microsoft/
WindowsApps/python3.11.exe "c:/Users/vpune/OneDrive/Desktop/New folder/daq3.py"
Estimated P(S | J): 0.6599
```

Sample:10000

```
PS C:\Users\vpune\OneDrive\Desktop\dbms project> & C:/Users/vpune/AppData/Local/Microsoft/
WindowsApps/python3.11.exe "c:/Users/vpune/OneDrive/Desktop/New folder/daq3.py"
Estimated P(S | J): 0.6650
```

Sample:100000

```
PS C:\Users\vpune\OneDrive\Desktop\dbms project> & C:/Users/vpune/AppData/Local/Microsoft/
WindowsApps/python3.11.exe "c:/Users/vpune/OneDrive/Desktop/New folder/daq3.py"
Estimated P(S | J): 0.6635
```

Step 4:Accuracy and Convergence

Accuracy:

. The estimated probability converges as the sample size increases.
. For smaller sample sizes (e.g., 1,000), the estimate has more fluctuation due to randomness.
. With 100,000 samples, the probability stabilizes around 0.666, which is close to the expected value based on the underlying probabilities.

Convergence:

.Monte Carlo simulations rely on the Law of Large Numbers, meaning that as the number of samples increases, the estimated probabilities converge to the true probabilities.

.From the results, you can see that increasing the sample size reduces variance, leading to a more stable and accurate estimate.