

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Fall Semester 2024-25

BCSE306L – Artificial Intelligence E2+TE2

DIGITAL ASSIGNMENT-2

Ishika Agarwal 22BCE3179

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.

Answer:

The 8 Puzzle problem is a sliding puzzle consisting of a 3x3 grid with numbered tiles (1-8) and one blank space. The goal is to rearrange the tiles to match a specified configuration. Hill Climbing is a heuristic-based search algorithm that selects the best immediate move but may encounter issues such as local maxima and plateaus.

- Initial Configuration: The starting state of the puzzle.
- Goal State: The desired configuration.
- Heuristic Function: Manhattan Distance - to estimate puzzle state's proximity to goal by calculating total distance tiles need to move to reach correct positions. Lower heuristic value indicates better state.

Algorithm –

1. Set the current state to the initial configuration of the puzzle.
2. Compute the Manhattan Distance for the current state as the heuristic value.
3. Find all valid neighboring states by moving the empty tile (0) in each possible direction.
4. Calculate the heuristic for each neighboring state.
5. Choose the neighbor with the lowest heuristic value. If multiple neighbors have the same heuristic, pick one randomly.
6. If the neighbor's heuristic is lower than the current state's heuristic, move to the neighbor. Otherwise, stop the search (local maximum or plateau).
7. If the heuristic value is 0, the goal state has been reached. Terminate the algorithm.
8. Repeat steps 3–7 until the goal state is reached or the algorithm stops improving.
9. If the algorithm terminates without finding the goal state, report failure and analyze.

Implementation

```
1  import copy
2  import random
3  #Ishika Agarwal 22BCE3179
4  class EightPuzzle:
5      def __init__(self, initial_state):
6          self.state = initial_state
7          self.goal_state = [
8              [1, 2, 3],
9              [4, 5, 6],
10             [7, 8, 0]
11         ]
12
13     def manhattan_distance(self, state):
14         #Calculate Manhattan distance heuristic.
15         distance = 0
16         for i in range(3):
17             for j in range(3):
18                 if state[i][j] != 0:
19                     goal_pos = self.find_goal_position(state[i][j])
20                     distance += abs(i - goal_pos[0]) + abs(j - goal_pos[1])
21         return distance
22
23     def find_goal_position(self, num):
24         #Find goal position for a given number.
25         for i in range(3):
26             for j in range(3):
27                 if self.goal_state[i][j] == num:
28                     return (i, j)
29
30     def get_blank_position(self, state):
31         #Find position of blank (0) tile.
32         for i in range(3):
33             for j in range(3):
34                 if state[i][j] == 0:
35                     return (i, j)
36         return None
37
38     def get_possible_moves(self, state):
39         #Generate possible moves by moving blank tile.
40         moves = []
41         directions = [
42             (0, 1), # right
43             (0, -1), # left
44             (1, 0), # down
45             (-1, 0) # up
46         ]
47
48         blank_pos = self.get_blank_position(state)
49
50         for dx, dy in directions:
51             new_x, new_y = blank_pos[0] + dx, blank_pos[1] + dy
52
53             if 0 <= new_x < 3 and 0 <= new_y < 3:
54                 new_state = copy.deepcopy(state)
55                 # Swap blank with adjacent tile
56                 new_state[blank_pos[0]][blank_pos[1]], new_state[new_x][new_y] = \
57                     new_state[new_x][new_y], new_state[blank_pos[0]][blank_pos[1]]
58                 moves.append(new_state)
59
60         return moves
61
```

```

61
62     def hill_climbing(self, max_iterations=100):
63         #Hill Climbing algorithm to solve 8 Puzzle.
64         current_state = self.state
65         iterations = 0
66
67         while iterations < max_iterations:
68             # Check if current state is goal state
69             if current_state == self.goal_state:
70                 print(f"Solution found in {iterations} iterations!")
71                 return current_state
72
73             # Get possible moves
74             possible_moves = self.get_possible_moves(current_state)
75
76             # Evaluate moves and find best neighbor
77             best_move = None
78             best_heuristic = self.manhattan_distance(current_state)
79
80             for move in possible_moves:
81                 move_heuristic = self.manhattan_distance(move)
82
83                 # Hill Climbing: Choose move with lowest heuristic
84                 if move_heuristic < best_heuristic:
85                     best_move = move
86                     best_heuristic = move_heuristic
87
88             # Check for local maxima/plateau
89             if best_move is None or best_heuristic >= self.manhattan_distance(current_state):
90                 print(f"Local maximum or plateau reached after {iterations} iterations.")
91                 return None
92
93             current_state = best_move
94             iterations += 1
95
96             print("Maximum iterations reached without finding solution.")
97             return None
98
99     def print_state(self, state):
100         """Print puzzle state."""
101         for row in state:
102             print(row)
103         print()
104
105     def get_user_input():
106         #Get initial puzzle configuration from user.
107         print("Enter the initial 8-Puzzle configuration:")
108         print("Use numbers 0-8, with 0 representing the blank space")
109         print("Enter each row with space-separated numbers")
110

```

```

110
111     initial_state = []
112     for i in range(3):
113         while True:
114             try:
115                 row = list(map(int, input(f"Enter row {i+1} (space-separated): ").split()))
116                 if len(row) != 3 or any(num < 0 or num > 8 for num in row) or len(set(row)) != len(row):
117                     print("Invalid input. Ensure 3 unique numbers between 0-8.")
118                     continue
119                 initial_state.append(row)
120                 break
121             except ValueError:
122                 print("Invalid input. Use space-separated numbers.")
123
124     return initial_state
125
126 def main():
127     # Get user input for initial state
128     initial_state = get_user_input()
129
130     print("\nInitial State:")
131     puzzle = EightPuzzle(initial_state)
132     puzzle.print_state(initial_state)
133
134     solution = puzzle.hill_climbing()
135
136     if solution:
137         print("Goal State:")
138         puzzle.print_state(solution)
139     else:
140         print("No solution found.")
141
142 if __name__ == "__main__":
143     main()
144

```

Output

```

C:\Users\Ishika\Desktop>python aida.py
Enter the initial 8-Puzzle configuration:
Use numbers 0-8, with 0 representing the blank space
Enter each row with space-separated numbers
Enter row 1 (space-separated): 1 2 3
Enter row 2 (space-separated): 4 0 6
Enter row 3 (space-separated): 7 5 8

Initial State:
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]

Solution found in 2 iterations!
Goal State:
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

```

```
C:\Users\Ishika\Desktop>python aida.py
Enter the initial 8-Puzzle configuration:
Use numbers 0-8, with 0 representing the blank space
Enter each row with space-separated numbers
Enter row 1 (space-separated): 1 2 3
Enter row 2 (space-separated): 4 5 6
Enter row 3 (space-separated): 0 7 8
```

Initial State:

```
[1, 2, 3]
[4, 5, 6]
[0, 7, 8]
```

Solution found in 2 iterations!

Goal State:

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

```
C:\Users\Ishika\Desktop>python aida.py
Enter the initial 8-Puzzle configuration:
Use numbers 0-8, with 0 representing the blank space
Enter each row with space-separated numbers
Enter row 1 (space-separated): 8 7 6
Enter row 2 (space-separated): 5 4 3
Enter row 3 (space-separated): 2 1 0
```

Initial State:

```
[8, 7, 6]
[5, 4, 3]
[2, 1, 0]
```

Local maximum or plateau reached after 0 iterations.
No solution found.

Key Observations and Limitations

1. Local Maxima: The algorithm might reach a state where all neighboring states have worse heuristics, causing it to terminate prematurely.
 - Occurs when the algorithm reaches a state where no neighboring moves improve the heuristic.
 - Algorithm gets trapped because every possible move increase Manhattan distance.
2. Plateaus: In cases where multiple states have the same heuristic, the algorithm may "wander" without improvement.
 - Algorithm cannot distinguish which move might lead to solution
 - Creates "flat" search landscape where progress stalls
3. No Backtracking: Hill Climbing does not revisit previous states, limiting its ability to recover from poor decisions.
4. Incomplete: It is not guaranteed to find a solution, even if one exists.

Improvements

- Random Restarts: Restart the algorithm with a new random state when stuck.
- Beam Search: Explore multiple paths simultaneously.
- Use more advanced search algorithms like A* or Simulated Annealing

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.

Answer:

This project integrates **Prolog**, **REST API**, and a **web app** to evaluate student eligibility for scholarships and exams based on attendance and CGPA data stored in a CSV file.

Algorithm –

1. Create a folder for the project. Inside the folder, create the following files:

- `eligibility.pl` (rules and data loader).
- `server.pl` (HTTP server).
- `data.csv` (student data).
- `index.html` (web interface to interact with REST API)

2. Run the following commands in the terminal:

```
swipl -s eligibility.pl
?- load_data.
swipl -s server.pl
?- start_server(8080) .
```

3. Open a browser to test endpoints.

Check Scholarship Eligibility: http://127.0.0.1:8080/scholarship?student_id=1

Check Exam Permission: http://127.0.0.1:8080/exam?student_id=1

4. Open your browser and navigate to:

<http://127.0.0.1:8080/index.html>

The `index.html` file should load correctly, allowing you to interact with the eligibility system.

Implementation

data.csv

	A	B	C	D	E	F
1	Student_ID	Attendance_Percentage	CGPA			
2	1	80	9.2			
3	2	70	8.5			
4	3	85	9.5			
5	4	60	7.8			
6	5	75	8.7			
7	6	90	9.6			
8	7	65	8			
9	8	82	9.3			
10	9	72	8.4			
11	10	88	9.4			
12	11	68	7.9			
13	12	78	8.8			
14	13	92	9.7			
15	14	62	7.6			
16	15	85	9.1			
17	16	73	8.3			
18	17	80	8.9			
19	18	67	7.7			
20	19	87	9			
21	20	76	8.6			
22	21	83	9.2			
23	22	69	8.1			
24	23	91	9.5			
25	24	64	7.5			
26						
27						
28						

< >

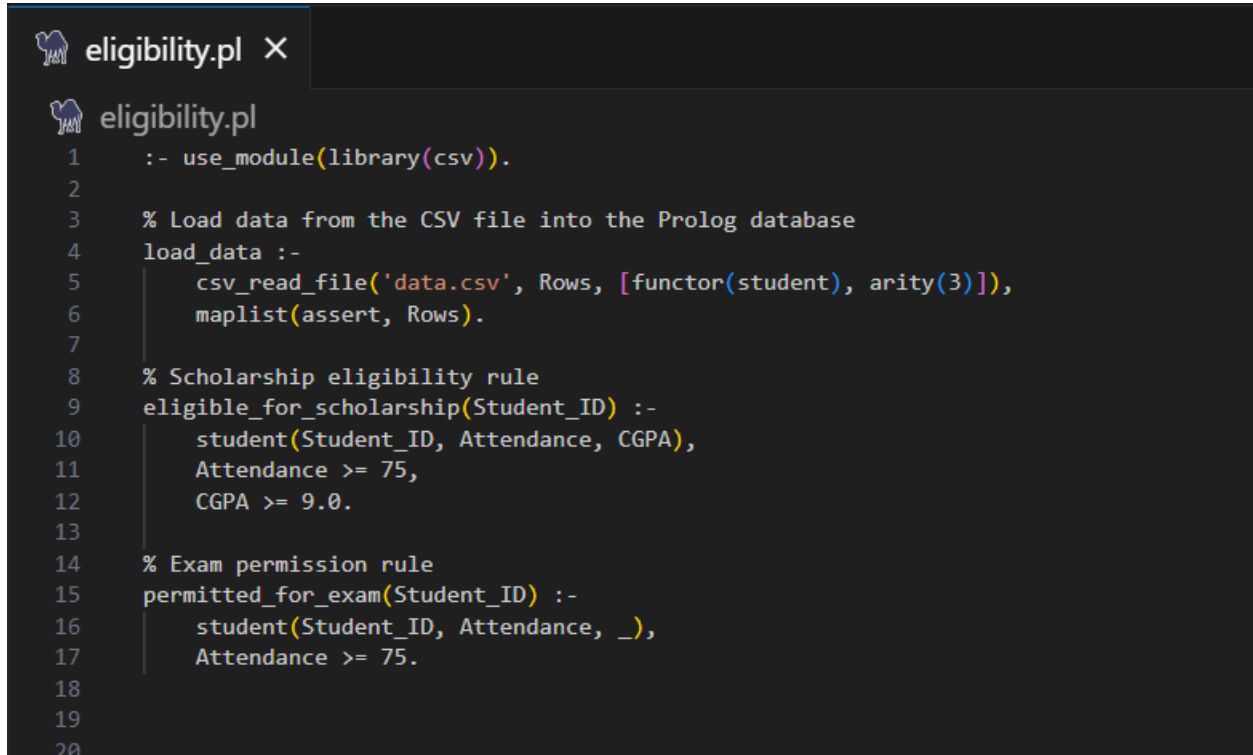
data

+

csv_read_file: Reads data from data.csv.

assert: Dynamically adds rows as facts in the Prolog database.

Define rules to evaluate eligibility for scholarships and exam permissions.



```
1  :- use_module(library(csv)).
2
3  % Load data from the CSV file into the Prolog database
4  load_data :-
5      csv_read_file('data.csv', Rows, [functor(student), arity(3)]),
6      maplist(assert, Rows).
7
8  % Scholarship eligibility rule
9  eligible_for_scholarship(Student_ID) :-
10     student(Student_ID, Attendance, CGPA),
11     Attendance >= 75,
12     CGPA >= 9.0.
13
14 % Exam permission rule
15 permitted_for_exam(Student_ID) :-
16     student(Student_ID, Attendance, _),
17     Attendance >= 75.
18
19
20
```

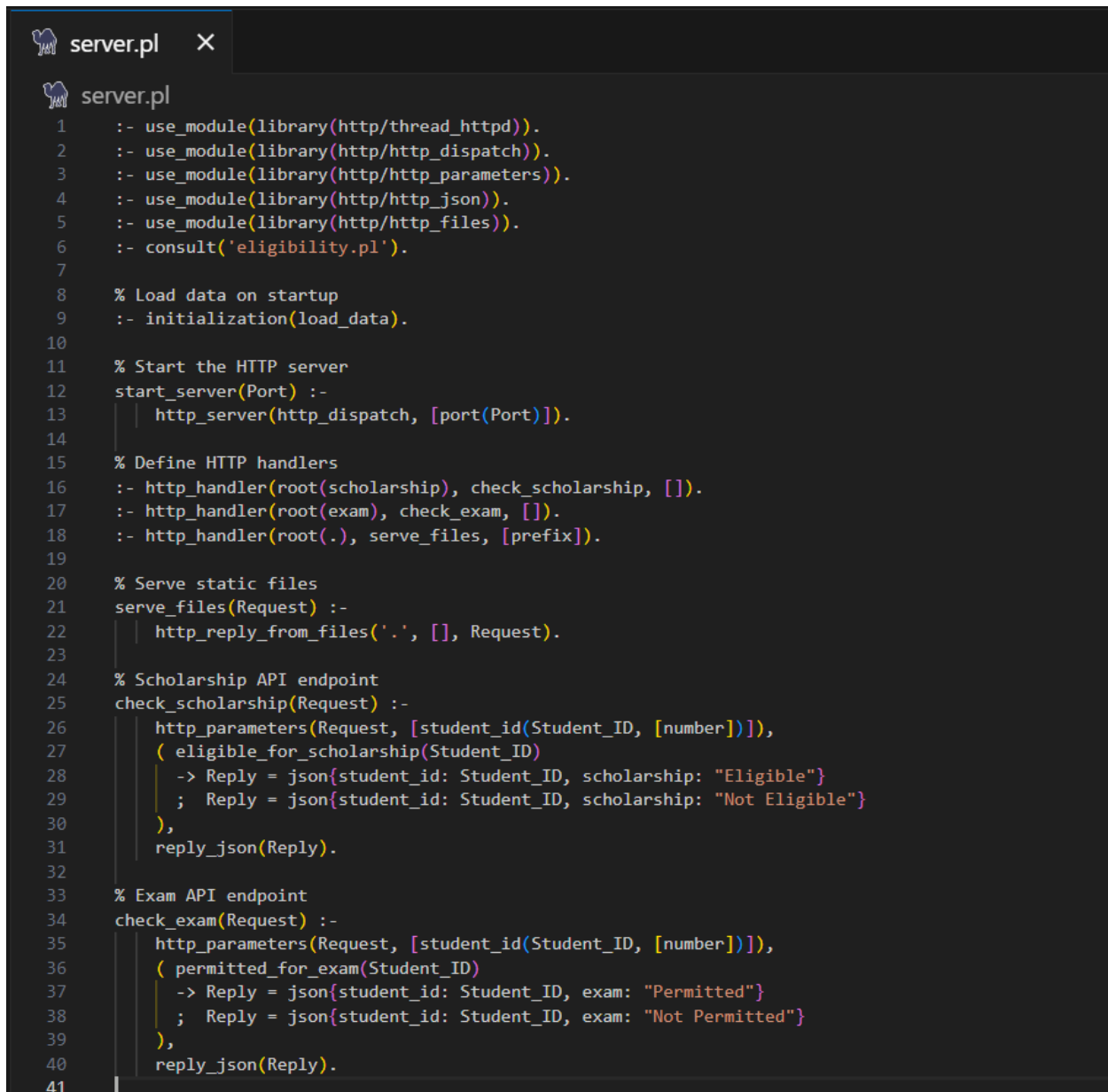
Using the **HTTP library**, create a REST API to query Prolog rules.

http_dispatch: Handles incoming requests and routes them to the correct handler.

reply_json: Responds to the client in JSON format.

Endpoints:

- /scholarship: Checks scholarship eligibility.
- /exam: Checks exam permission.



```
server.pl X
server.pl
1  :- use_module(library(http/thread_httpd)).
2  :- use_module(library(http/http_dispatch)).
3  :- use_module(library(http/http_parameters)).
4  :- use_module(library(http/http_json)).
5  :- use_module(library(http/http_files)).
6  :- consult('eligibility.pl').
7
8  % Load data on startup
9  :- initialization(load_data).
10
11 % Start the HTTP server
12 start_server(Port) :-
13     http_server(http_dispatch, [port(Port)]).
14
15 % Define HTTP handlers
16 :- http_handler(root(scholarship), check_scholarship, []).
17 :- http_handler(root(exam), check_exam, []).
18 :- http_handler(root(.), serve_files, [prefix]).
19
20 % Serve static files
21 serve_files(Request) :-
22     http_reply_from_files('.', [], Request).
23
24 % Scholarship API endpoint
25 check_scholarship(Request) :-
26     http_parameters(Request, [student_id(Student_ID, [number])]),
27     ( eligible_for_scholarship(Student_ID)
28     -> Reply = json{student_id: Student_ID, scholarship: "Eligible"}
29     ; Reply = json{student_id: Student_ID, scholarship: "Not Eligible"}
30     ),
31     reply_json(Reply).
32
33 % Exam API endpoint
34 check_exam(Request) :-
35     http_parameters(Request, [student_id(Student_ID, [number])]),
36     ( permitted_for_exam(Student_ID)
37     -> Reply = json{student_id: Student_ID, exam: "Permitted"}
38     ; Reply = json{student_id: Student_ID, exam: "Not Permitted"}
39     ),
40     reply_json(Reply).
41
```

A simple front-end web app calls the REST API and displays results.

```
index.html X
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>Eligibility Checker</title>
7      <style>
8        body {
9          font-family: Arial, sans-serif;
10         margin: 20px;
11       }
12       label,
13       input,
14       button {
15         margin: 5px 0;
16       }
17       .result {
18         margin-top: 20px;
19         padding: 10px;
20         border: 1px solid #ccc;
21       }
22     </style>
23   </head>
24   <body>
25     <h1>Student Eligibility Checker</h1>
26
27     <form id="eligibilityForm">
28       <label for="studentId">Enter Student ID:</label>
29       <input type="number" id="studentId" name="studentId" required />
30       <button type="button" onclick="checkEligibility()">Check</button>
31     </form>
32
33     <div id="result" class="result" style="display: none">
34       <h3>Result:</h3>
35       <p id="response"></p>
36     </div>
37
```

```

38     <script>
39         async function checkEligibility() {
40             const studentId = document.getElementById("studentId").value;
41             const resultDiv = document.getElementById("result");
42             const responseDiv = document.getElementById("response");
43
44             if (!studentId) {
45                 alert("Please enter a valid Student ID.");
46                 return;
47             }
48             try {
49                 const scholarshipResponse = await fetch(
50                     `http://127.0.0.1:8080/scholarship?student_id=${studentId}`
51                 );
52                 const examResponse = await fetch(
53                     `http://127.0.0.1:8080/exam?student_id=${studentId}`
54                 );
55
56                 if (!scholarshipResponse.ok || !examResponse.ok) {
57                     throw new Error("Failed to fetch data.");
58                 }
59
60                 const scholarshipData = await scholarshipResponse.json();
61                 const examData = await examResponse.json();
62
63                 responseDiv.innerHTML = `
64                     <strong>Student ID:</strong> ${studentId} <br>
65                     <strong>Scholarship Eligibility:</strong> ${scholarshipData.scholarship} <br>
66                     <strong>Exam Permission:</strong> ${examData.exam}
67                 `;
68                 resultDiv.style.display = "block";
69             } catch (error) {
70                 responseDiv.innerHTML = "An error occurred while fetching the data.";
71                 resultDiv.style.display = "block";
72                 console.error(error);
73             }
74         }
75     </script>
76 </body>
77 </html>

```

Output

```
C:\Users\Ishika\Desktop>swipl -s eligibility.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.15)
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).

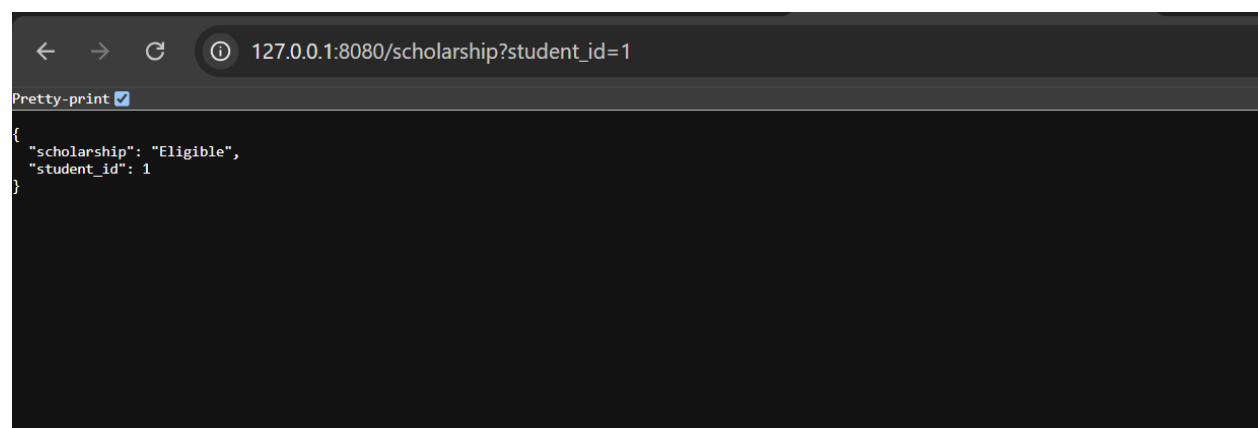
1 ?- load_data.
true.

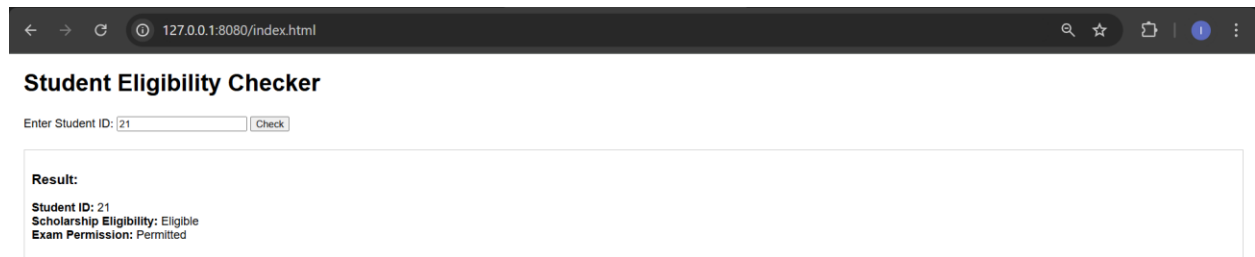
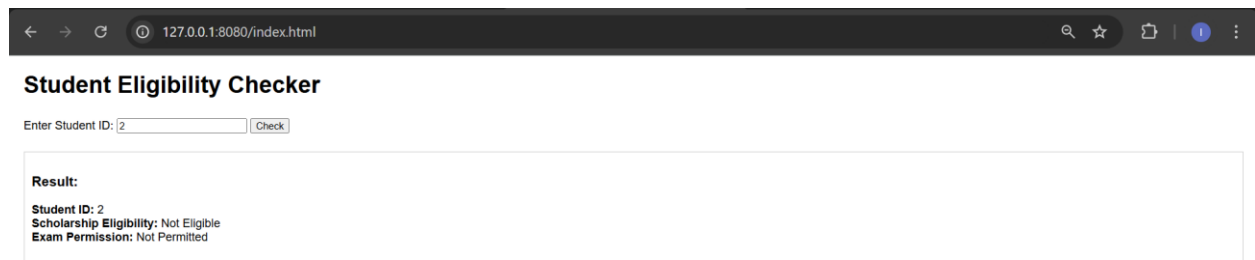
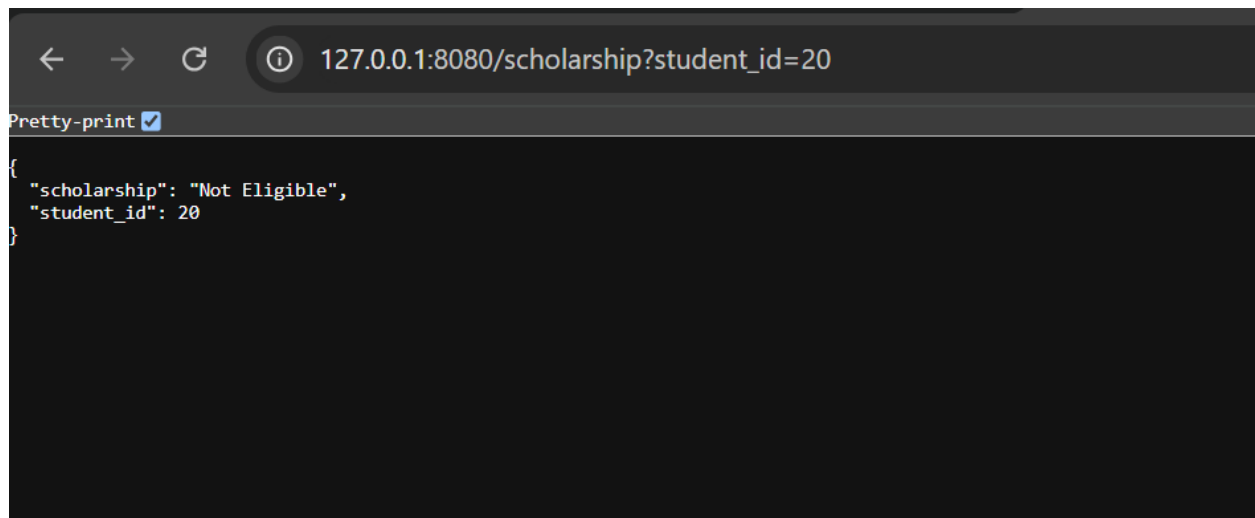
2 ?- eligible_for_scholarship(1).
true.
```

```
PS C:\Users\Ishika> cd Desktop
PS C:\Users\Ishika\Desktop> swipl -s server.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.15)
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 ?- start_server(8080).
% Started server at http://localhost:8080/
true.
```





Comparison of SQL vs Prolog

SQL and Prolog are fundamentally different tools designed for distinct purposes, each excelling in specific scenarios. SQL, a relational query language, is primarily used for managing structured, table-based data. It uses structured queries like `SELECT` statements to efficiently retrieve, update, and manipulate data, making it the ideal choice for database management systems that handle large datasets with frequent transactions and updates. SQL's design prioritizes scalability and performance, especially in environments where consistent and reliable data storage is critical.

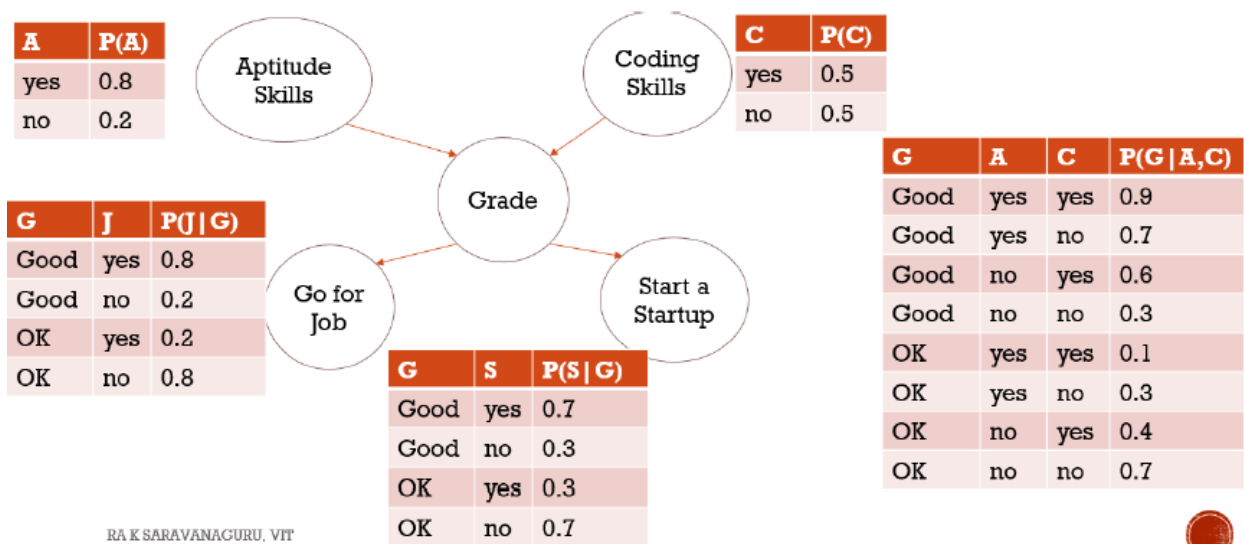
On the other hand, Prolog is a logic programming language that retrieves data based on facts and rules, evaluating queries through logical inference rather than direct table searches. This approach allows Prolog to excel in applications where reasoning and complex relationships need to be modeled, such as expert systems, decision-making applications, and symbolic problem-solving tasks. Unlike SQL, which requires techniques like nested queries or joins to define complex logic, Prolog inherently supports declarative logic. This allows simple, human-readable rules to drive sophisticated reasoning processes.

When comparing suitability, SQL is best for handling large-scale, high-volume transactional systems, such as financial databases or inventory management, where speed and data integrity are crucial. Prolog, however, shines in scenarios involving rule-based decision-making or artificial intelligence, such as eligibility systems, knowledge representation, or solving combinatorial problems.

In conclusion, SQL excels at managing structured data and transactional systems at scale, while Prolog offers unique advantages in logical reasoning and rule evaluation. Both languages have distinct strengths and are complementary, with their applicability depending on the nature of the problem being solved. Understanding their differences can help in choosing the right tool for the task at hand.

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.



Answer:

Algorithm

1. Understand the BBN Structure: The given BBN includes nodes:

- Aptitude Skills (A) and Coding Skills (C) as prior nodes.
- Grade (G) as the conditional node influenced by A and C.
- Go for Job (J) and Start a Startup (S), influenced by Grade (G).

You are tasked with estimating conditional probabilities, e.g., $P(J=\text{yes}|A=\text{yes},C=\text{yes})$

2. Define the Conditional Probabilities: Extract all the conditional probabilities from the tables in the BBN. This includes:

- Prior probabilities $P(A), P(C)$
- $P(G|A,C)$
- $P(J|G)$
- $P(S|G)$

3. Create Random Sampling: Use Monte Carlo sampling to generate random outcomes for each variable based on its probability distribution. Start from prior nodes (A, C), then sample G based on $P(G|A,C)$, and finally sample J and S.
4. Run Monte Carlo Simulation: For a large number of samples:
 - Count occurrences of evidence
 - Count how often the target node occurs given the evidence.
5. Calculate Conditional Probability: Estimate the conditional probability as:

$$P(J = \text{yes} | A = \text{yes}, C = \text{yes}) = \frac{\text{Count}(J = \text{yes and evidence})}{\text{Count}(\text{evidence})}$$

6. Analyze Convergence: Discuss how the accuracy improves as the number of samples increases.

Implementation

```
VIT > sem 5 > Artificial Intelligence > q3.py > ...

1  import numpy as np
2  #Ishika Agarwal 22BCE3179
3  # Define conditional probabilities
4  P_A = {'yes': 0.8, 'no': 0.2}
5  P_C = {'yes': 0.5, 'no': 0.5}
6  P_G_given_A_C = {
7      ('yes', 'yes'): {'Good': 0.9, 'OK': 0.1},
8      ('yes', 'no'): {'Good': 0.7, 'OK': 0.3},
9      ('no', 'yes'): {'Good': 0.6, 'OK': 0.4},
10     ('no', 'no'): {'Good': 0.3, 'OK': 0.7},
11 }
12 P_J_given_G = {'Good': {'yes': 0.8, 'no': 0.2}, 'OK': {'yes': 0.2, 'no': 0.8}}
13 P_S_given_G = {'Good': {'yes': 0.7, 'no': 0.3}, 'OK': {'yes': 0.3, 'no': 0.7}}
14
15 # Monte Carlo simulation
16 def monte_carlo_simulation(num_samples=10000):
17     count_target = 0
18     count_evidence = 0
19     for _ in range(num_samples):
20         # Sample A and C
21         A = 'yes' if np.random.rand() < P_A['yes'] else 'no'
22         C = 'yes' if np.random.rand() < P_C['yes'] else 'no'
23
24         # Sample G given A and C
25         G_probs = P_G_given_A_C[(A, C)]
26         G = 'Good' if np.random.rand() < G_probs['Good'] else 'OK'
27
28         # Sample J given G
29         J_probs = P_J_given_G[G]
30         J = 'yes' if np.random.rand() < J_probs['yes'] else 'no'
31
32         # Evidence check
33         if A == 'yes' and C == 'yes': # Evidence: A = yes, C = yes
34             count_evidence += 1
35             if J == 'yes': # Target: J = yes
36                 count_target += 1
37
38         # Calculate conditional probability
39         if count_evidence == 0:
40             return 0 # Avoid division by zero
41         return count_target / count_evidence
42
43 # Run simulation
44 estimated_probability = monte_carlo_simulation()
45 print(f"Estimated P(J=yes | A=yes, C=yes): {estimated_probability}")
46
47
```

Output

```
PS C:\Users\Ishika\desktop\vit\sem 5\Artificial Intelligence> py  
thon q3.py  
Estimated P(J=yes | A=yes, C=yes): 0.734007585335019
```

Results and Discussion

- **Result:** The output of the simulation provides an estimated probability of $P(J=\text{yes} | A=\text{yes}, C=\text{yes})$.
- **Accuracy:** The accuracy of the result depends on the number of samples. Increasing the sample size reduces random variation, leading to a more accurate estimate.
- **Convergence:** The simulation showcases how Monte Carlo methods approximate probabilities over multiple iterations. While the approach is computationally intensive, it is effective for inference in complex networks where exact computation is infeasible.

This implementation demonstrates the utility of Monte Carlo methods in reasoning over Bayesian networks.