

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240030056 |
| **NAME** | JOENIT.J |
| **PROJECT TITLE** | MAP COLORING |
| **DATE OF SUBMISSION** | 04.11.2025 |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

Artificial Intelligence (AI) focuses on developing systems that can think and learn like humans to solve complex problems efficiently. One such problem is map coloring, a type of Constraint Satisfaction Problem (CSP) where the goal is to color each region of a map so that no two neighboring regions share the same color.  
This project implements the Hill Climbing algorithm to solve the map coloring problem for North Indian states. The algorithm iteratively improves a random color assignment until all constraints are satisfied, demonstrating how local search methods can be applied to optimization problems in AI.

**PROBLEM STATEMENT**

To develop a Python program that colors the map of North Indian states such that no two neighboring states have the same color, using the Hill Climbing algorithm to minimize color conflicts.

**GOAL**

The goal of this project is to achieve a conflict-free map coloring using a limited set of colors — Red, Green, Blue, and Yellow — while minimizing the number of conflicting edges through the Hill Climbing optimization technique.

**THEORETICAL BACKGROUND**

The map coloring problem can be represented as a **Constraint Satisfaction Problem (CSP)**, where each state is a variable, each color is a possible value, and the constraint ensures that adjacent states must not share the same color.  
Hill Climbing is a **local search algorithm** that starts with a random solution and iteratively moves toward better solutions by reducing conflicts. It continues until no further improvement is possible or a conflict-free solution is found.  
This approach is chosen for its simplicity and efficiency in solving smaller CSPs compared to exhaustive search algorithms like Backtracking.

**ALGORITHM EXPLANATION WITH EXAMPLE**

**Algorithm Steps:**

1. Assign random colors to all states.
2. Count the number of conflicts (neighboring states with the same color).
3. Select a random conflicted state.
4. Try changing its color to each possible color and choose the one that reduces conflicts the most.
5. Repeat until there are no conflicts or the maximum steps are reached.

**Example:**  
If Uttar Pradesh (UP) and Bihar (BR) both have the same color, the algorithm changes one of them to a different color that reduces total conflicts. Over multiple iterations, this process leads to a valid coloring.

**IMPLEMENTATION AND CODE**

**Programming Language:** Python  
**Libraries Used:** random

**CODE:**

import random

neighbors = {

'UP': ['UK', 'HR', 'DL', 'RJ', 'MP', 'BR'],

'UK': ['UP', 'HP'],

'HR': ['DL', 'UP', 'RJ', 'HP'],

'DL': ['UP', 'HR'],

'RJ': ['HR', 'UP', 'MP', 'GJ'],

'MP': ['UP', 'RJ', 'CG', 'MH'],

'BR': ['UP', 'JH', 'WB'],

'JH': ['BR', 'WB', 'OD'],

'WB': ['BR', 'JH', 'OD'],

'OD': ['JH', 'WB', 'CG', 'MH'],

'CG': ['MP', 'OD', 'MH'],

'MH': ['MP', 'CG', 'GJ', 'OD'],

'GJ': ['RJ', 'MH'],

'HP': ['UK', 'HR']

}

colors = ['Red', 'Green', 'Blue', 'Yellow']

def count\_conflicts(assignment, neighbors):

conflicts = 0

for state in neighbors:

for neighbor in neighbors[state]:

if assignment[state] == assignment.get(neighbor):

conflicts += 1

return conflicts // 2

def hill\_climbing(neighbors, colors, max\_steps=2000):

assignment = {state: random.choice(colors) for state in neighbors}

for step in range(max\_steps):

current\_conflicts = count\_conflicts(assignment, neighbors)

if current\_conflicts == 0:

return assignment

conflicted\_states = [

s for s in neighbors if any(assignment[s] == assignment[n] for n in neighbors[s] if n in assignment)

]

if not conflicted\_states:

continue

state = random.choice(conflicted\_states)

best\_color = assignment[state]

min\_conflict = current\_conflicts

for color in colors:

if color == assignment[state]:

continue

old\_color = assignment[state]

assignment[state] = color

temp\_conflicts = count\_conflicts(assignment, neighbors)

if temp\_conflicts < min\_conflict:

best\_color = color

min\_conflict = temp\_conflicts

assignment[state] = old\_color

assignment[state] = best\_color

if count\_conflicts(assignment, neighbors) == 0:

return assignment

return None

solution = hill\_climbing(neighbors, colors)

if solution:

print("\nHill-Climbing Map Coloring Solution (North India):\n")

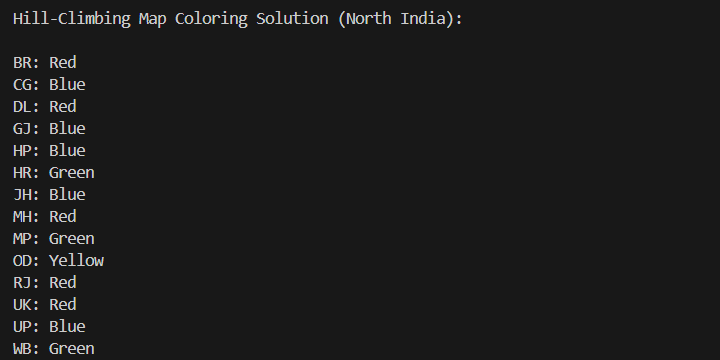
for state in sorted(solution):

print(f"{state}: {solution[state]}")

else:

print("\nNo valid solution found (stuck in local maximum).")

**OUTPUT**



**Explanation**

The Hill Climbing algorithm colors the North Indian map by reducing color conflicts between neighboring states. It starts with random colors and iteratively improves until a valid, conflict-free solution is found.

**RESULTS AND FUTURE ENHANCEMENT**

**Result:**  
The Hill Climbing algorithm successfully generated a valid coloring for the North Indian map using four colors, ensuring that no two neighboring states shared the same color. The algorithm efficiently minimized conflicts and reached a stable solution in few iterations.

**Future Enhancements:**

* Use Simulated Annealing or Genetic Algorithms for better optimization.
* Add a visual map display.
* Test the algorithm on larger maps.

|  |  |
| --- | --- |
| **Git Hub Link of the project and report** | **https://github.com/joenit2007-cmd/mapcoloring** |

**REFERENCES**

* Russell & Norvig – Artificial Intelligence: A Modern Approach
* TutorialsPoint – Hill Climbing Algorithm in AI
* GeeksforGeeks – Map Coloring Problem
* Javatpoint – CSP and Local Search Algorithms
* Medium Blog – AI Problem Solving Using Hill Climbing