AI Assignment - 2 Report

Question 1: Sudoku Puzzle as a CSP Problem

Representing a Sudoku Puzzle Problem in a Constraint Satisfaction problem, includes defining the problem in terms of variables, domain and the constraints.

Variables: The variables for a Sudoku puzzle will be blank cells on the grid. Every cell will be considered as a variable.

Domain: The Domain holds all the values that can be assigned to the variables. In sudoku puzzle variables of the Sudoku Puzzle can hold any value in the range of 1 to 9, hence the domain = {1, 2, 3, 4, 5, 6, 7, 8, 9}

Constraints:

- The number to be filled in a cell should not be already assigned to another variable in that row.
- The number to be filled in a cell should not be already assigned to another variable in that column.
- The number to be filled in a cell should not be already assigned to another variable in the sub-grid of 3x3.

Generating Sudoku Puzzles Before Running the Algorithms
To test our algorithms, we needed 10 random Sudoku puzzles for logging the average time for comparison with other advanced algorithms. So, before starting with Backtracking, we created a bunch of random puzzle setups using a random seed value to create reproducibility among all the algorithms. While making them, we made sure that the initial numbers didn't break any of the basic Sudoku rules. Some of these puzzles are solvable, while some might not be. Just to mix things up, we also included one puzzle that actually has no solution at all.

Simple Backtracking Search:
The simple backtrack search is implemented as a recursive backtrack algorithm that checks for all assignments at every iteration and validates it on 10 randomly generated sudoku puzzle configuration setups.

Comparison Table for Fault Finding and Heuristic Algorithms

Algorithm	Average time	Explanation
Simple Backtracking	0.05899737 seconds	● This is a simple recursive backtrack algorithm ● It is basically a brute force approach which checks for all the possible values.
Forward Checking with Backtrack	0.07692020 seconds	● Forward checking helps by cutting down the number of possible values (domain) for each cell based on the rules. When we pick a cell to assign a value, the possible values for its neighboring cells also get updated right away in the same step. So basically, the number of options left to try for each cell becomes smaller compared to the basic backtracking, which makes the whole process faster and a bit easier. However, this is not always true. ● In our case the algorithm actually performed slower, this could be because of maintenance of the domain for the neighbours and large space values.
Arc Consistency With Backtrack	0.06788585 seconds	● AC-3 sets up arcs between neighboring cells and updates each cell's possible values (domain) based on constraints. ● Before running backtracking, it checks if the puzzle is arc consistent and only continues if it passes this check. ● It's faster than plain backtracking because it reduces domains early, but slower than forward checking due to the extra arc consistency step.
Minimum Remaining Values	0.01002169 seconds	● The algorithm uses minimum remaining values which picks the fewest possible values, which makes it faster. ● The time difference is significantly noticeable here.
Minimum Remaining Value	0.00735195 seconds	● Degree Heuristics picks cells with the

with Degree Heuristics		highest number of neighbors. In the sudoku case this will not work as the domain of the non neighbor elements should also be checked and updated. ● So, Used Degree Heuristics as a tie breaking heuristic with MRV algorithm. ● The time taken was significantly lesser than most of the algorithms.
Least Constraining Values	0.00665922 seconds	● **LCV (Least Constraining Value)** picks the value that rules out the fewest options for neighboring cells, helping keep more possibilities open for the rest of the puzzle. ● This algorithm has taken the least time for the same configuration.

Question 2: Simulated Annealing with N-Queens

Introduction

For the N-Queens problem, implemented a Simulated Annealing (SA) algorithm to place 8 queens on an 8x8 chessboard with no attacks (no shared rows, columns, or diagonals). I chose (N=8) since it’s challenging yet solvable. As required, we have to use the simulated annealing search algorithm and tweak it to act like Hill Climbing by setting (T=0).

Implementation

The Simulated Annealing algorithm works in a way that it starts off with high temperature (to get out from the local minima) and then settles down with some cooling rate with each iteration. So, simulated annealing randomly generates the neighbors and accepts worse neighbor states as well sometimes to get out from the minima, but the probability of acceptance reduces as the temperature reduces.

Acceptance Probability: e^(Δ *E/T)*
where , Δ *E = difference in the current state energy (or conflicts in our case) and new state energy.*
T = current temperature

Tweaking Simulated Annealing to behave like Hill Climbing:

- When the Temperature element is reaching near to zero, then the acceptance probability also tends to 0. This means that now the probability of accepting a worse state is close to 0 or 0, which makes it a Hill climbing algorithm as the Hill climbing algorithm also does not accept the worst state then the current state in any circumstances.
- In order to let Simulated Annealing act like a Hill climbing algorithm we need to just pass the temp parameter as 0 and does not have to update it (or cool down) as it is already zero, then the acceptance probability will be = *e^(∆ E/T).* Since, we put T = 0 the probability calculation was not happening and giving zerodivision error.
- So, to come up with this error, we have to tweak the algorithm and add a condition to check the temperature parameter and calculate probability only when the temperature is greater than 0. When the Temperature parameter will hold any value greater than 0 then the algorithm will behave like simulated annealing.
- Then to avoid an infinite loop situation in case of Hill climbing being stuck in local minima, added a maximum iteration loop and added that as a parameter, so the algo will run for a set number of loops to ensure there is a stopping condition.

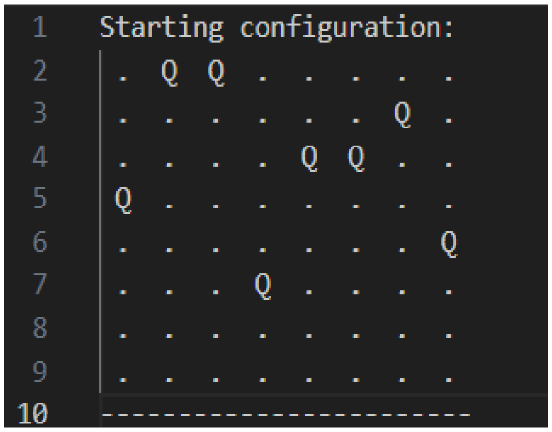
Behavioral Differences

The Behavioral difference in both the algorithm is Hill climbing generates the neighbors and only selects the best case neighbour always which leads to get it stuck at local minima most of the times, Simulated annealing generates the neighbors randomly and sometimes accept the worse states as well, to get out of the local minima. Below are 2 configurations for which both the algorithms were performed and the results are as:

Case 1: Scenario Without Local Minima

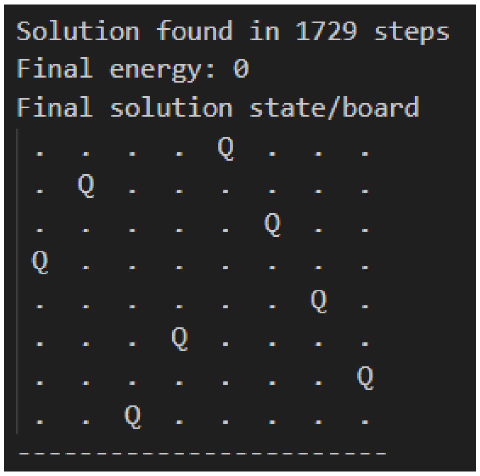
1. **Simulated Annealing:**
   In this case, the SA algorithm was configured with an initial temperature of 1.0 and a cooling rate of 0.95.

   Initial Configuration is [3, 0, 0, 5, 2, 2, 1, 4]



```
Starting configuration:
 .  Q  Q  .  .  .  .  .
 .  .  .  .  .  .  Q  .
 .  .  .  .  Q  Q  .  .
 Q  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  Q
 .  .  .  Q  .  .  .  .
 .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .
------------------------
```

Under these parameters, the algorithm successfully reached the goal state—where no queens are in conflict—after completing 1729 iterations.

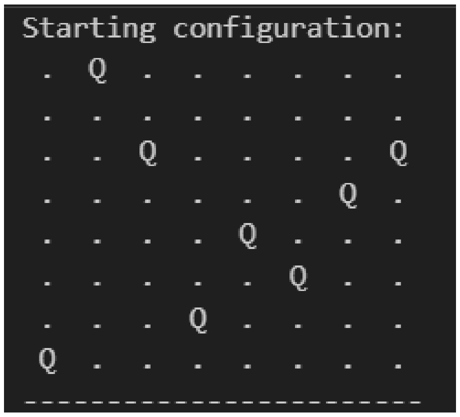


```
Solution found in 1729 steps
Final energy: 0
Final solution state/board
 .  .  .  .  Q  .  .  .
 .  Q  .  .  .  .  .  .
 .  .  .  .  .  Q  .  .
 Q  .  .  .  .  .  .  .
 .  .  .  .  .  .  Q  .
 .  .  .  Q  .  .  .  .
 .  .  .  .  .  .  .  Q
 .  .  Q  .  .  .  .  .
------------------------
```

The gradual cooling process allowed the algorithm to explore the search space effectively, eventually converging to an optimal solution.

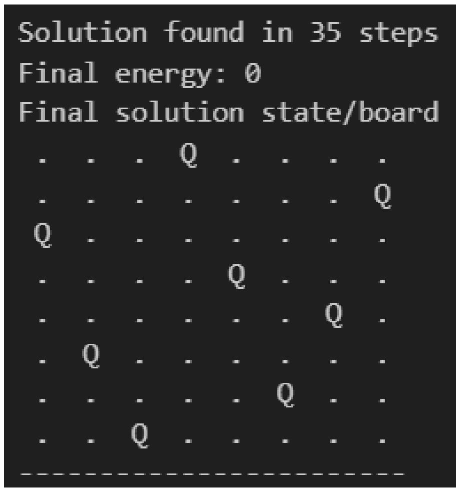
2. **Hill Climbing:**

   For HC, the parameters were set to mimic its standard behavior, with the temperature initialized to 0 and a cooling rate of 1.0.



```
Starting configuration:
 .  Q  .  .  .  .  .  .
 .  .  .  .  .  .  .  .
 .  .  Q  .  .  .  .  Q
 .  .  .  .  .  .  Q  .
 .  .  .  .  Q  .  .  .
 .  .  .  .  .  Q  .  .
 .  .  .  Q  .  .  .  .
 Q  .  .  .  .  .  .  .
------------------------
```

This setup led the algorithm to find the solution within 35 steps.



```
Solution found in 35 steps
Final energy: 0
Final solution state/board
 .  .  .  Q  .  .  .  .
 .  .  .  .  .  .  .  Q
 Q  .  .  .  .  .  .  .
 .  .  .  .  Q  .  .  .
 .  .  .  .  .  .  Q  .
 .  Q  .  .  .  .  .  .
 .  .  .  .  .  Q  .  .
 .  .  Q  .  .  .  .  .
------------------------
```

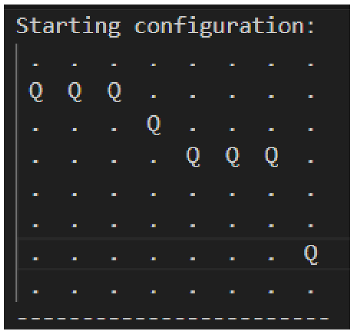
Since HC always moves towards a better state without accepting worse solutions, it quickly reached the global optimum in this straightforward landscape.

Observation:

- Both algorithms managed to find the solution in this scenario. However, HC was significantly faster than SA due to its deterministic nature and lack of randomness.
- SA, though slower, still consistently reached the goal because it allows exploration of the search space through random neighbor selection.
- It's important to note that while HC may outperform SA in such simple landscapes, it is also more susceptible to failure in the presence of local minima.

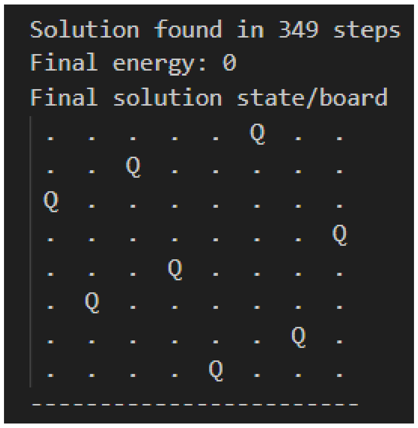
Case 2: Scenario With Local Minima

1. **Simulated Annealing:**
   Using the same SA configuration (temperature = 1.0, cooling rate = 0.95), the algorithm succeeded in reaching a conflict-free solution in 349 iterations.



```
Starting configuration:
 .  .  .  .  .  .  .  .
 Q  Q  Q  .  .  .  .  .
 .  .  .  Q  .  .  .  .
 .  .  .  .  Q  Q  Q  .
 .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  Q
 .  .  .  .  .  .  .  .
-----------------------
```

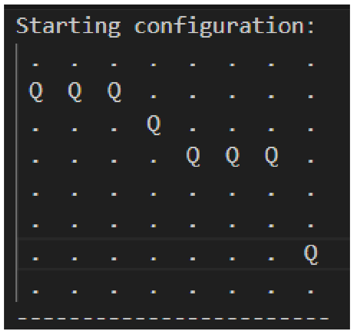
   The presence of local minima did not hinder the algorithm due to its probabilistic acceptance of worse states, which helps escape suboptimal regions.



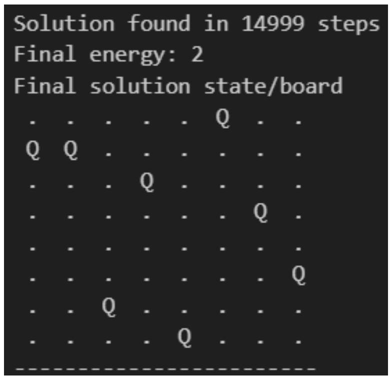
```
Solution found in 349 steps
Final energy: 0
Final solution state/board
 .  .  .  .  .  Q  .  .
 .  .  Q  .  .  .  .  .
 Q  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  Q
 .  .  .  Q  .  .  .  .
 .  Q  .  .  .  .  .  .
 .  .  .  .  .  .  Q  .
 .  .  .  .  Q  .  .  .
-----------------------
```

2. Hill Climbing:

   With the same HC configuration (temperature = 0, cooling rate = 1.0), the algorithm failed to reach the goal.



```
Starting configuration:
 .  .  .  .  .  .  .  .
 Q  Q  Q  .  .  .  .  .
 .  .  .  Q  .  .  .  .
 .  .  .  .  Q  Q  Q  .
 .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  Q
 .  .  .  .  .  .  .  .
------------------------
```



```
Solution found in 14999 steps
Final energy: 2
Final solution state/board
 .  .  .  .  .  Q  .  .
 Q  Q  .  .  .  .  .  .
 .  .  .  Q  .  .  .  .
 .  .  .  .  .  .  Q  .
 .  .  .  .  .  .  .  .
 .  .  .  .  .  .  .  Q
 .  .  Q  .  .  .  .  .
 .  .  .  .  Q  .  .  .
------------------------
```

It became trapped in a local minimum, unable to explore further due to its strict improvement-only strategy.

Observation:

- SA demonstrated robustness in this more complex scenario, highlighting its advantage in dealing with deceptive search spaces.
- In contrast, HC struggled due to its inability to accept non-improving moves, a common limitation in rugged landscapes with multiple peaks and valleys.
- Although rare, HC might occasionally succeed if it stumbles upon a favorable path during neighbor selection. However, such success is not guaranteed.