AI PROJECT REPORT

N-Queens Problem

Prepared by:

Name	Roll No.
Sarvang Jain (Team Leader)	18UCS062
Rohit Agarwal	18UCS094
Akshit Agarwal	18UCS045
Aditya Goyal	18UCS101

Submitted to:

Prof. Nitin Kumar
Dr. Poulami Dalapati
Dr. Roshni Chakraborty

INTRODUCTION

Problem Statement :

Generate a large number of n-queen instances and solve them by hill climbing, hill climbing with random restart and simulated annealing. Measure the search cost and percentage of solved problems and graph these against the optimal solution.

N-Queens Problem:

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.In other words, This problem is to find an arrangement of N queens on a chess board, such that no queen can attack any other queens on the board.
The chess queens can attack in any direction - horizontal, vertical, horizontal and diagonal.
A binary matrix is used to display the positions of N Queens, where no queens can attack other queens.

In this problem, we come up with three different algorithms to compute the solution of this problem as specified in the Problem Statement. The Algorithms are briefly explained:

- **Hill Climbing**: Hill Climbing is a **heuristic search which uses a greedy approach and ranks all the possible alternatives at any branching point and only traverses the best route possible**. It only moves to the next state if the next state has better heuristic value than the present state else the algorithm will stop and return the present value. The problem with this algorithm is that it will not move to a worse state than present state and terminate itself. The process will end even though a better solution may exist.

- **Hill Climbing with Random Restart:** hill climbing with random restart is a **meta-algorithm built on top of the hill climbing algorithm**. It iteratively does hill-climbing, each time with a random initial condition. This algorithm is used to solve the problem with the Hill Climbing algorithm mentioned before.Since we randomly select another starting point once a local optimum is reached, it eliminates the risk that you find a local optimum, but not the global optimum.

- **Simulated Annealing:**In Simulated Annealing, we define an initial temperature, often set as 1, and a minimum temperature, on the order of 10^-4. The current temperature is multiplied by some fraction alpha and thus decreased until it reaches the minimum temperature. For each distinct temperature value, we run the core optimization routine a fixed number of times.Simulated annealing is based on **metallurgical practices by which a material is heated to a high temperature and cooled**.

FUNCTIONS OR METHODS USED IN THE PROGRAM

1. ***Main Function - main()***
    a. Parameters: None
    b. Return type: int
    c. Function: start the program

2. ***Initial Setup - initialSetup()***
    a. Parameters: position, board
    b. Return type: void
    c. Function:initialize the chess board with random queen position

3. ***Print Board - printBoard()***
    a. Parameters:position, board
    b. Return type: void
    c. Function: prints the chess board

4. ***Copy state and board generation - copyPos_genBoard()***
    a. Parameters: position1,position2,board
    b. Return type: void
    c. Function: copy the position into new position vector and generate the board accordingly.

5. ***Calculate Objective - calculateObjective()***
    a. Parameters:board, position
    b. Return type: int
    c. Function: return the heuristic(objective value of the chessboard(no.of queen getting attacked)

6. ***Get Optimal - getOptimal()***
    a. Parameters:position, board
    b. Return type: void
    c. Function: finds the most optimal chess board we could get by changing any 1 queen's position.

7. ***Hill Climbing Algorithm - hillClimbing()***
    a. Parameters:position, board
    b. Return type: void
    c. Function: finds the optimal solution with the help of hill climbing algorithm.

8. ***Hill Climbing with Random Restart - hillClimbingWithRandomRestart()***
    a. Parameters:position, board
    b. Return type: void
    c. Function:finds the optimal solution with the help of random restart hill climbing algorithm.

9. ***Calculating Threatening positions - calTheats()***
    a. Parameters:board, numofQueens
    b. Return type: int
    c. Function:counts and returns the number of threatening position

10. ***Generating Board - generateboard()***
    a. Parameters: *board , numofQueens
    b. Return type: int*
    c. Function :makes a new board from the current state, so makes a new state in the form of a board

11. ***Find Next State - findNextState()***
    a. Parameters: *board , numofQueens
    b. Return type: bool
    c. Function :finds the next place to move to, if there is no place to move to then it will restart with a new board

12. ***Annealing board generator - generateboardAnneal()***
    a. Parameters: *board , numofQueens
    b. Return type: *int
    c. Function :makes a new board from the current state, so makes a new state in the form of a board

13. ***Finding next Annealing - findNextStateAnneal()***
    a. Parameters: *board , numofQueens
    b. Return type: bool
    c. Function :finds the next place to move to, if there is no place to move to then it will restart with a new board

14. Simulated Annealing - simAnnealing()

a. Parameters: numofQueens
b. Return type: void
c. Function :Simulated Annealing Algorithm to Solve N-Queens

INPUT AND OUTPUT OF THE PROGRAM:

Input:

- Input contains 1 line containing integer n
- N is the number of queens to be placed on an nxn board.

Output:

- 1st line prints “Initial Board”
- Next n lines contain nxn board showing the initial board randomly generated.
- Next line contains no. of threats in the initial board.
- Then a blank line prints.
- Next line prints “Hill climbing”
- Next n lines contain nxn board showing the resulting board by hill climb algorithm.
- Next line prints the time taken by the above process.
- Next line contains no. of threats remaining in the resulting board.
- Then a blank line prints.
- Next line prints “Random restart Hill climbing”
- Next n lines contain nxn board showing the resulting board by random restart hill climb algorithm.
- Next line prints the time taken by the above process.
- Next line contains no. of threats remaining in the resulting board.
- Then a blank line prints.
- Next line prints “Simulated Annealing”
- Next n lines contain nxn board showing the resulting board by Simulated Annealing algorithm.
- Next line prints the time taken by the above process.
- Next line contains no. of threats remaining in the resulting board.

End of output

SNAPSHOTS OF THE OUTPUTS (RESULTS)

Initial Board

```
Please enter the number of queens you want to place
8
Initial Board
. . Q . . . . .
. . . . . . . .
. . . . . Q . .
. . . . . . . .
. . . Q . . Q .
. Q . . . . . .
Q . . . . . . Q
. . . . Q . . .
Threats in this board are:5
```

Hill Climbing

```
Hill Climbing
. . Q . . . . .
. . . . . . . Q
. . . . . . . .
. Q . . . . . .
. . . Q . Q . .
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .

Time taken by program is : 0.000335 sec
Threats in this board are:1
```

Random Restart Hill Climbing

```
Random Restart Hill Climbing
. Q . . . . . .
. . . . . Q . .
Q . . . . . . .
. . . . . . Q .
. . . Q . . . .
. . . . . . . Q
. . Q . . . . .
. . . . Q . . .

Time taken by program is : 0.000779 sec
Threats in this board are:0
```

Simulated Annealing

```
simulated annealing
. . . . Q . . .
. . Q . . . . .
Q . . . . . . .
. . . . . . Q .
. Q . . . . . .
. . . . . . . Q
. . . . . Q . .
. . . Q . . . .

Threats in this board are: 0
Time taken by program is : 0.359328 sec
```

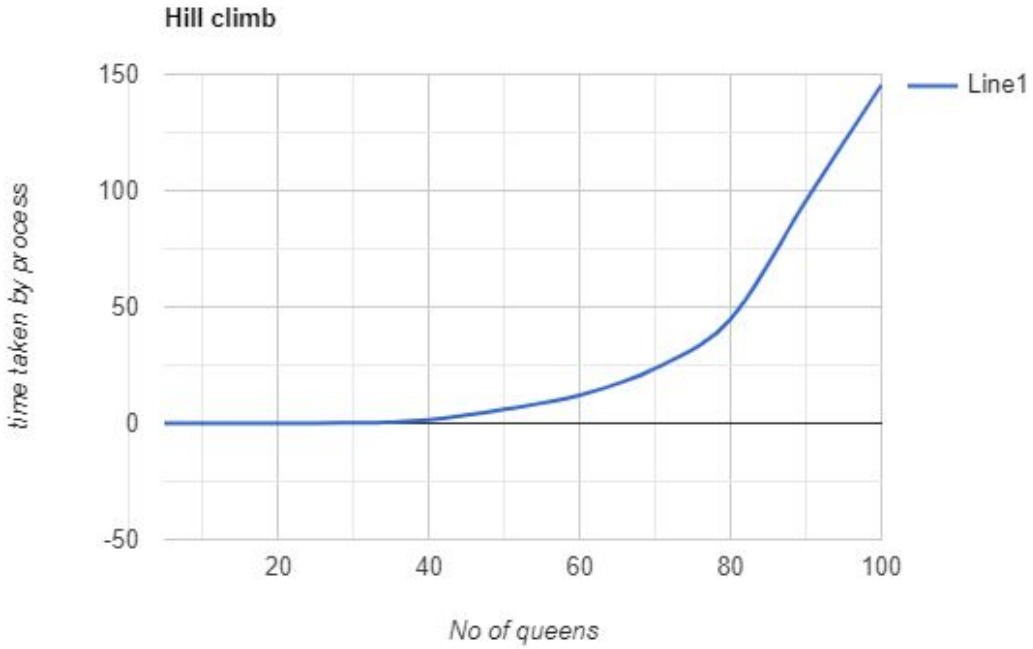
GRAPHS AND REPRESENTATIONS

X-Axis - No of queens ***Y-Axis*** - Time taken by Algorithm

Value table for hill climbing

No.	X-axis	Y-axis	No.	X-axis	Y-axis
1	5	0.000074	9	40	1.506660
2	8	0.000289	10	45	3.463179
3	10	0.000686	11	50	6.012999
4	15	0.003970	12	60	12.080467
5	20	0.013187	13	70	23.577417
6	25	0.093390	14	80	44.750851
7	30	0.258969	15	90	95.766576
8	35	0.422051	16	100	145.464428

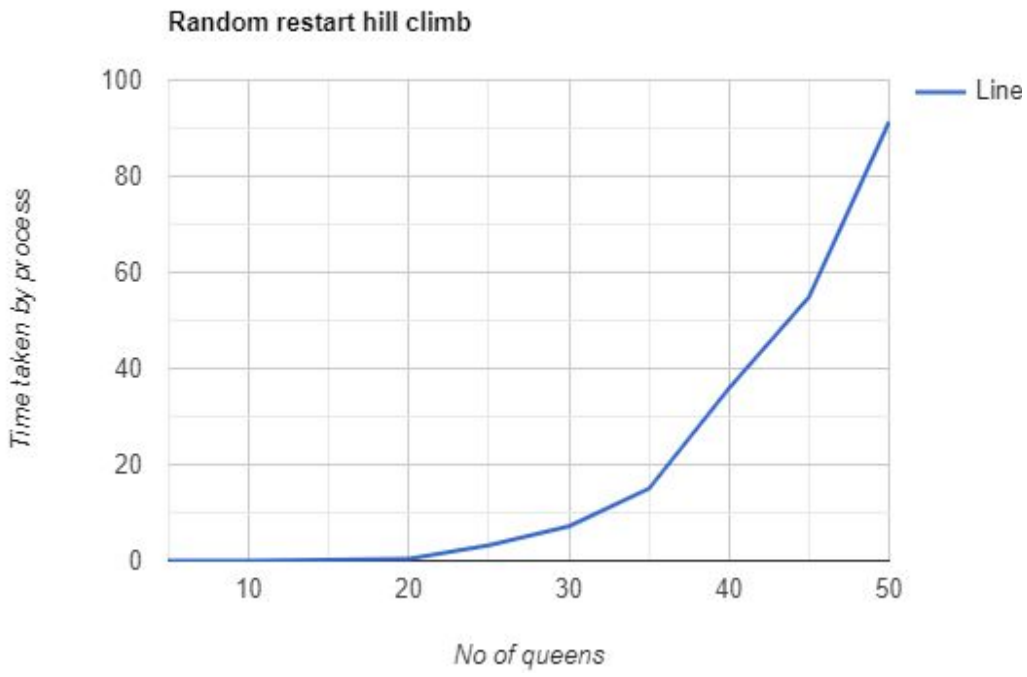
Graph for Hill Climbing



Value table for Random Restart hill climbing

No.	X-axis	Y-axis	No.	X-axis	Y-axis
1	5	0.000054	7	30	7.199211
2	8	0.004210	8	35	15.070515
3	10	0.010021	9	40	35.887138
4	15	0.191933	10	45	54.902195
5	20	0.467767	11	50	91.299024
6	25	3.196261	-	-	-

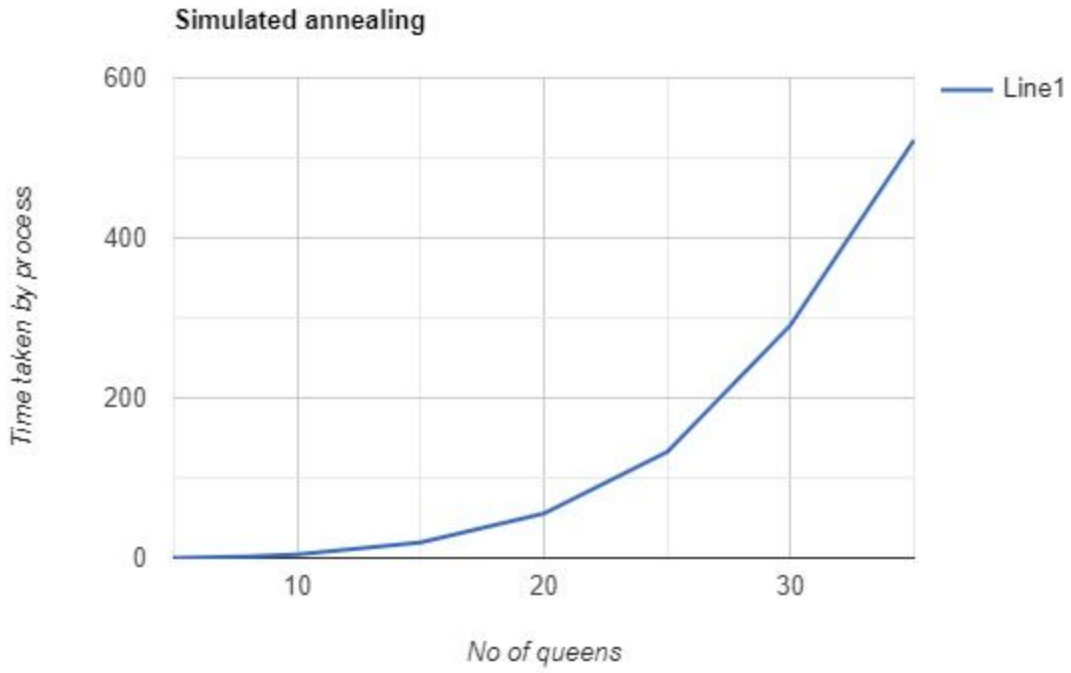
Graph for Random restart Hill climbing



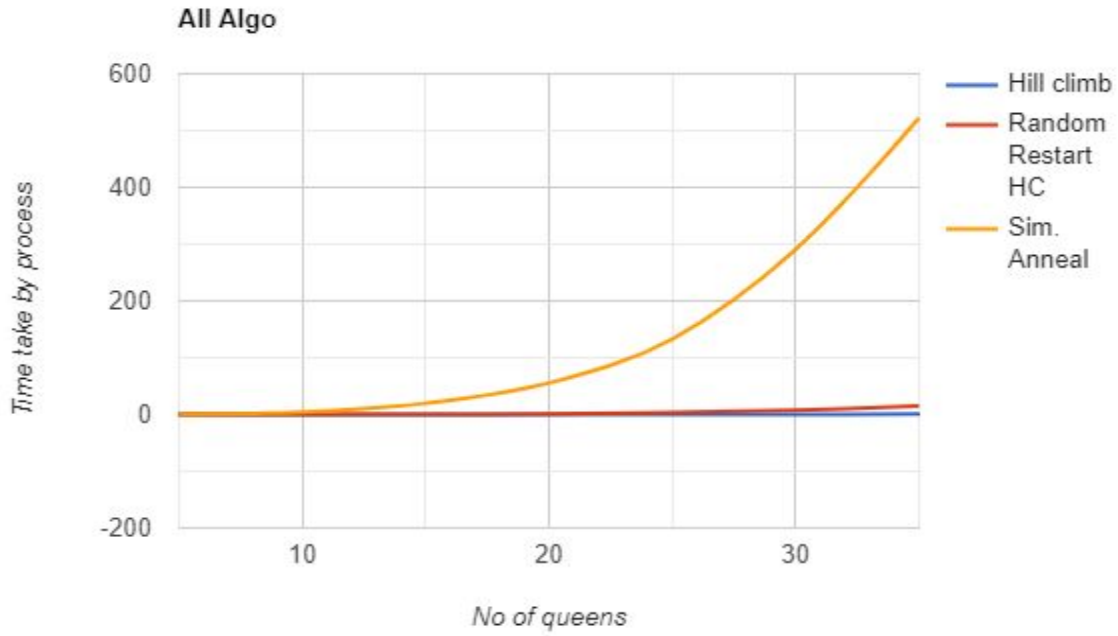
Value table for Simulated Annealing

No.	X-axis	Y-axis
1	5	0.078977
2	8	1.877046
3	10	4.081552
4	15	19.597022
5	20	55.511654
6	25	132.913176
7	30	290.909773
8	36	522.108279

Graph for Simulated Annealing



Combined graph of all three algorithm



As we can see from the graph that time required by the simulated annealing increases with the increment of queens.
On the other hand, both Hill Climbing and Random restart hill climbing takes much less time as compare to Simulated Annealing

OBSERVATION AND INFERENCES

On the Basis of Processing Time:

The Hill climbing method takes very little time, and the random restart hill climbing method and the simulated annealing algorithm take a lot of time. The relatively fast preferred hill climbing method is about six hundred times faster than the relatively slow simulated annealing algorithm. On the whole, the first choice is the best climbing method, and the other two methods can be eliminated in time.Although hill climbing with random restart and simulated annealing have better resolution rate and more probability of escaping local solution but hill climbing is much superior on the basis of processing time and is quick to come to an conclusion.(Irrespective of being optimal or not)

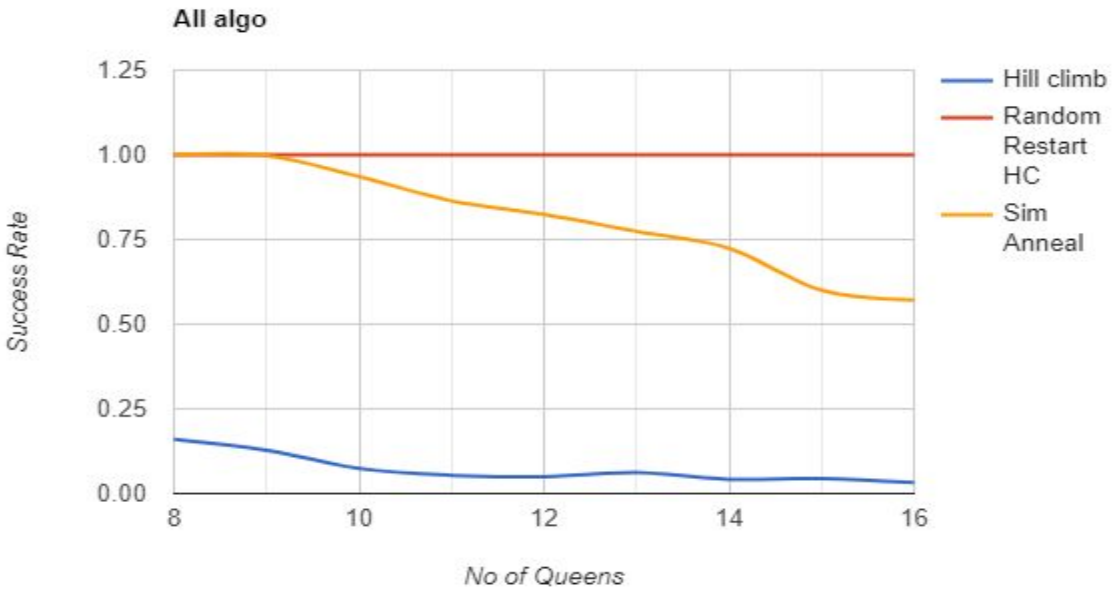
On the basis of Solved percentage:

As discussed in the previous point simulated annealing takes more time to give the output. So, for the large N simulated annealing fails to give the result or we can say it requires much more time. On comparing Hill Climbing and Hill Climbing with Random Restart, Hill Climbing with random restart has a large solving percentage but again for the large N Hill Climbing with random restart takes much more time than the Hill Climbing. Although, Hill Climbing with Random Restart is a decent solution for the local optima problem we face in Hill Climbing but our problem of N queens does not have a high number of local optima and hence some else less time consuming variation could be considered for this problem like a random neighbour if we have to find optimal solution with higher accuracy than Hill Climbing but with less processing time(and accuracy) than Hill Climbing with random restart. But Hill Climbing with random restart will be the most likely of all the three algorithms tested to give a global optima where no queen is being attacked by any other queen.

CONCLUSIONS

The data (as represented in the graphs before) shows that the three methods have a resolution rate of over 90% for the N Queens problem, and the difference is not obvious. The preferred hill climbing method is relatively superior, and the simulated annealing algorithm is relatively inferior. So, we can easily eliminate the simulated annealing on the basis of processing time.

Graph of Success Rate:



From the above graph, Hill Climbing with Random Restart is much more efficient than the Hill Climbing. Though Simulated Annealing also has a high success rate but we have already eliminated it on a time basis.

Overall: Hill Climbing with the Random Restart is the best algorithm among all three.

REFERENCES AND BIBLIOGRAPHY

We referred to online resources such as :-

- GeeksforGeeks - https://www.geeksforgeeks.org/ , for the understanding of different algorithms.

- Our Lecture Videos as well as Youtube videos of Artificial Intelligence course in UC Berkeley.
  https://www.youtube.com/playlist?list=PLsOUugYMBBJENfZ3XAToMsg44W7LeUVhF
  https://www.youtube.com/channel/UCKVFs2NL2yFKxasaDv3C9CA

Offline Resources such as books that included :-

- Stuart Russell, Peter Norvig, Artificial Intelligence – A Modern Approach (3rd Edition)

- Elaine Rich and Kevin Knight, Artificial Intelligence, 3rd Edition, McGraw-Hill, 2017.

CONTRIBUTION OF EACH MEMBER

1. **Sarvang Jain (18ucs062: Team Leader) -**
    a. Contributed to the functions: *simAnnealing(), findNextStateAnneal(), generateboardAnneal(), findNextState(), calThreats()*
    b. Contribution to the report: *Introduction along with References and Bibliography*

2. **Rohit Agrawal (18ucs094) -**
    a. Contributed to the functions: *initialSetup(), printBoard(), getOptimal(), hillClimbing(), generateboard()*
    b. Contribution to the report: *Functions and Method Used along with Snapshots of the Output*

3. **Akshit Agarwal (18ucs045) -**
    a. Contributed to the functions: *srand(), copyPos_genBoard(), calculateObjective(), main() and hillClimbingWithRandomRestart()*
    b. Contribution to the report: *Observation and Inferences along with Conclusion*
    c. Recorded the Video provided in the Link

4. **Aditya Goyal (18ucs101) -**
    a. Contributed in *calculating the total time taken by each algorithm, using: gettimeofday(), Plotted Graphs for all three algorithms and combined graph plotting.*
    b. Contribution to the report: *Graphs and Representations and Input and Output of the Program*

PROJECT LINKS

- ***GitHub Link*** : https://github.com/Rohitag14/N-Queen-AI-Project

- ***Google Drive Link :*** https://drive.google.com/drive/folders/1F1_QXd8tKOJ3nOnKTaIWgbf90f2zyUFv?usp=sharing