Report on Hill Climbing

Submitted By:
Alekhya Majeti (801135655)
Preetham Shankare Gowda (801154091)
Sai Teja Sarangu (801100587)

Problem Formulation:

The N-queens problem is the problem of placing 'n' chess queens on an n * n chessboard so that no two queens are attacking each other. This means no queen can be in the same row, column or diagonal. We can find the solutions for all the natural numbers except for n =2 or 3. Here in this report, we are choosing to solve the 8 queens problem by taking a random state by placing 8 queens in the 8*8 chessboard by placing each queen in a column.

There are different types of hill-climbing search techniques that can be used to solve this problem. The general hill-climbing search has less percent of success that is around 14%. So to optimize this search there are a couple of updated searches like Hill Climbing sideways move and Random Restart Hill Climbing.

- Hill Climbing with a sideways move: This is an optimized version of the regular hill-climbing search algorithm. When a local minimum is reached, continuing search by non improving "sideways" moves will lead to a significant improvement in the performance of the algorithm.
- Random Restart Hill Climbing: This is built on top of the hill-climbing search algorithm. It iteratively does hill-climbing, each time with a random initial condition. The best state is kept; if a new run of hill-climbing produces a better state than the store state, it replaces the stored state. This is the most effective algorithm in most of the cases.

We are using a heuristic function to determine the steps each queen takes. The heuristic cost function h calculates the number of pairs of queens that are attacking each other, either directly or indirectly.

Program Structure:

Global Variables:

print_states: A boolean value which when set to “True” will print the states of the iterations.

Classes and methods:

- **HillClimbing:**
  - __init__ : Initializes the variables startState(Initial state of the n queens problem), side_max(Maximum steps allowed for sideways move), side_rem(Sideway moves remaining), total_steps(Total steps taken by the algorithm, including all the steps of every restart) and N(n value of the n queens problem).
  - get_diagonal_cells_to_right: Returns cells that are diagonal and are on the right side of the current cell.
  - get_horizaontal_cells_to_right: Returns cells that are horizontal and are on the right side of the current cell.
  - get_cells_to_right : Concatenates and returns the result of get_diagonal_cells_to_right and get_horizontal_cells_to_right
  - get_cells_of_state: Get cell positions of the queens of the given state
  - calculate_h : Get heuristic value for a given state
  - print_state : Display the n queen state in matrix format
  - get_h_matrix : Calculate heuristic values for all the cells to take the next step.
  - steepest_ascent : A recursive method implementation of steepest ascent. This method takes a step towards the lowest heuristic value with each recursion.
  - get_random_state: Generates and returns a random state everytime.
  - random_restart: A method that implements Random restart algorithm and uses the "steepest_ascent" method.

- **HillClimbingAnalysis:**
  - __init__ : Initializes the variables maxIter(Maximum number of iterations to perform), nValue(n value of the n queens problem), steep_climb_stat(Variable to store the statistics of Steepest ascent - hill climbing without sideways move), steep_climb_w_side_stat(n value of the n queens problem), random_restart_stat(Variable to store the statistics of random restart - hill climbing without sideways move) and random_restart_w_side_stat(Variable to store the statistics of random restart - hill climbing with sideways move).
  - is_notebook: Method to find out if the running environment is Jupyter notebook or not
  - update_progress: Method that displays the progress status(bar)
  - start_analysis: Starts iterating 'maxIter' number of times and performs steepest ascent and random restart hill climbing with and without sideways move
  - start_analysis: Starts iterating 'maxIter' number of times and performs steepest ascent and random restart hill climbing with and without sideways move
  - print_analysis: Calls respective methods to display the analysis/report for all 4 algorithms
  - print_rand_restart_stat: Displays the report for random restart algorithm with and without sideways move
  - print_steep_climb_stat: Displays the report for steepest ascent algorithm with and without sideways move

Sample Initial and Final configurations:

```
Hill climbing Search [Steepest Ascent] Analysis - Sample Run 1
Initial:
[(4, 0), (2, 1), (1, 2), (4, 3), (2, 4), (5, 5), (3, 6), (7, 7)]
| | | | | | | | |
| | |Q| | | | | |
| |Q| | |Q| | | |
| | | | | | |Q| |
|Q| | |Q| | | | |
| | | | | |Q| | |
| | | | | | | | |
| | | | | | | |Q|
Step:  2
[(4, 0), (7, 1), (1, 2), (4, 3), (2, 4), (5, 5), (3, 6), (7, 7)]
| | | | | | | | |
| | |Q| | | | | |
| | | | |Q| | | |
| | | | | | |Q| |
|Q| | |Q| | | | |
| | | | | |Q| | |
| | | | | | | | |
| |Q| | | | | |Q|
Step:  3
[(4, 0), (7, 1), (1, 2), (4, 3), (2, 4), (5, 5), (3, 6), (6, 7)]
| | | | | | | | |
| | |Q| | | | | |
| | | | |Q| | | |
| | | | | | |Q| |
|Q| | |Q| | | | |
| | | | | |Q| | |
| | | | | | | |Q|
| |Q| | | | | | |
Search Failed


Hill climbing Search [Steepest Ascent] Analysis - Sample Run 2
Initial:
[(6, 0), (3, 1), (4, 2), (3, 3), (2, 4), (1, 5), (6, 6), (2, 7)]
| | | | | | | | |
| | | | | |Q| | |
| | | | |Q| | |Q|
| |Q| |Q| | | | |
| | |Q| | | | | |
```

```
| | | | | | | | |
|Q| | | | | |Q| |
| | | | | | | | |
Step:  2
[(6, 0), (3, 1), (4, 2), (0, 3), (2, 4), (1, 5), (6, 6), (2, 7)]
| | | |Q| | | | |
| | | | | |Q| | |
| | | | |Q| | |Q|
| |Q| | | | | | |
| | |Q| | | | | |
| | | | | | | | |
|Q| | | | | |Q| |
| | | | | | | | |
Step:  3
[(5, 0), (3, 1), (4, 2), (0, 3), (2, 4), (1, 5), (6, 6), (2, 7)]
| | | |Q| | | | |
| | | | | |Q| | |
| | | | |Q| | |Q|
| |Q| | | | | | |
| | |Q| | | | | |
|Q| | | | | | | |
| | | | | | |Q| |
| | | | | | | | |
Step:  4
[(5, 0), (3, 1), (4, 2), (0, 3), (7, 4), (1, 5), (6, 6), (2, 7)]
| | | |Q| | | | |
| | | | | |Q| | |
| | | | | | | |Q|
| |Q| | | | | | |
| | |Q| | | | | |
|Q| | | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
Step:  5
[(5, 0), (3, 1), (0, 2), (0, 3), (7, 4), (1, 5), (6, 6), (2, 7)]
| | |Q|Q| | | | |
| | | | | |Q| | |
| | | | | | | |Q|
| |Q| | | | | | |
| | | | | | | | |
|Q| | | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
Success:
[(5, 0), (3, 1), (0, 2), (4, 3), (7, 4), (1, 5), (6, 6), (2, 7)]
```

```
| | |Q| | | | | |
| | | | | |Q| | |
| | | | | | | |Q|
| |Q| | | | | | |
| | | |Q| | | | |
|Q| | | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |

Hill climbing Search [Steepest Ascent] Analysis - Sample Run 3
Initial:
[(5, 0), (0, 1), (6, 2), (3, 3), (5, 4), (7, 5), (1, 6), (3, 7)]
| |Q| | | | | | |
| | | | | | |Q| |
| | | | | | | | |
| | | |Q| | | |Q|
| | | | | | | | |
|Q| | | |Q| | | |
| | |Q| | | | | |
| | | | | |Q| | |
Step:  2
[(5, 0), (0, 1), (6, 2), (3, 3), (5, 4), (7, 5), (1, 6), (4, 7)]
| |Q| | | | | | |
| | | | | | |Q| |
| | | | | | | | |
| | | |Q| | | | |
| | | | | | | |Q|
|Q| | | |Q| | | |
| | |Q| | | | | |
| | | | | |Q| | |
Search Failed

Hill climbing Search [Steepest Ascent] Analysis - Sample Run 4
Initial:
[(7, 0), (7, 1), (2, 2), (7, 3), (3, 4), (1, 5), (7, 6), (7, 7)]
| | | | | | | | |
| | | | | |Q| | |
| | |Q| | | | | |
| | | | |Q| | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
|Q|Q| |Q| | |Q|Q|
Step:  2
[(5, 0), (7, 1), (2, 2), (7, 3), (3, 4), (1, 5), (7, 6), (7, 7)]
```

```
| | | | | | | | |
| | | | | |Q| | |
| | |Q| | | | | |
| | | | |Q| | | |
| | | | | | | | |
|Q| | | | | | | |
| | | | | | | | |
| |Q| |Q| | |Q|Q|
Step:  3
[(5, 0), (7, 1), (2, 2), (7, 3), (3, 4), (1, 5), (7, 6), (4, 7)]
| | | | | | | | |
| | | | | |Q| | |
| | |Q| | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
|Q| | | | | | | |
| | | | | | | | |
| |Q| |Q| | |Q| |
Step:  4
[(5, 0), (7, 1), (2, 2), (6, 3), (3, 4), (1, 5), (7, 6), (4, 7)]
| | | | | | | | |
| | | | | |Q| | |
| | |Q| | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
|Q| | | | | | | |
| | | |Q| | | | |
| |Q| | | | |Q| |
Search Failed


Hill climbing Search [Steepest Ascent] with sideways Analysis - Sample Run
1
Initial:
[(3, 0), (0, 1), (6, 2), (5, 3), (7, 4), (6, 5), (7, 6), (7, 7)]
| |Q| | | | | | |
| | | | | | | | |
| | | | | | | | |
|Q| | | | | | | |
| | | | | | | | |
| | | |Q| | | | |
| | |Q| | |Q| | |
| | | | |Q| |Q|Q|
Step:  2
[(3, 0), (0, 1), (6, 2), (5, 3), (2, 4), (6, 5), (7, 6), (7, 7)]
```

```
| |Q| | | | | | |
| | | | | | | | |
| | | | |Q| | | |
|Q| | | | | | | |
| | | | | | | | |
| | | |Q| | | | |
| | |Q| | |Q| | |
| | | | | | |Q|Q|
Step:  3
[(3, 0), (0, 1), (6, 2), (5, 3), (2, 4), (6, 5), (1, 6), (7, 7)]
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
|Q| | | | | | | |
| | | | | | | | |
| | | |Q| | | | |
| | |Q| | |Q| | |
| | | | | | | |Q|
Step:  4
[(3, 0), (0, 1), (7, 2), (5, 3), (2, 4), (6, 5), (1, 6), (7, 7)]
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
|Q| | | | | | | |
| | | | | | | | |
| | | |Q| | | | |
| | | | | |Q| | |
| | |Q| | | | |Q|
Step:  5
[(4, 0), (0, 1), (7, 2), (5, 3), (2, 4), (6, 5), (1, 6), (7, 7)]
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | | |
|Q| | | | | | | |
| | | |Q| | | | |
| | | | | |Q| | |
| | |Q| | | | |Q|
Success:
[(4, 0), (0, 1), (7, 2), (5, 3), (2, 4), (6, 5), (1, 6), (3, 7)]
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | |Q|
|Q| | | | | | | |
```

```
| | | |Q| | | | |
| | | | | |Q| | |
| | |Q| | | | | |


Hill climbing Search [Steepest Ascent] with sideways Analysis - Sample Run
2
Initial:
[(4, 0), (0, 1), (5, 2), (3, 3), (7, 4), (1, 5), (6, 6), (2, 7)]
| |Q| | | | | | |
| | | | | |Q| | |
| | | | | | | |Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | |Q| | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
Step:  2
[(4, 0), (0, 1), (5, 2), (0, 3), (7, 4), (1, 5), (6, 6), (2, 7)]
| |Q| |Q| | | | |
| | | | | |Q| | |
| | | | | | | |Q|
| | | | | | | | |
|Q| | | | | | | |
| | |Q| | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
Step:  3
[(4, 0), (0, 1), (3, 2), (0, 3), (7, 4), (1, 5), (6, 6), (2, 7)]
| |Q| |Q| | | | |
| | | | | |Q| | |
| | | | | | | |Q|
| | |Q| | | | | |
|Q| | | | | | | |
| | | | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
Success:
[(4, 0), (0, 1), (3, 2), (5, 3), (7, 4), (1, 5), (6, 6), (2, 7)]
| |Q| | | | | | |
| | | | | |Q| | |
| | | | | | | |Q|
| | |Q| | | | | |
|Q| | | | | | | |
| | | |Q| | | | |
```

```
| | | | | | |Q| |
| | | | |Q| | | |
```

```
Hill climbing Search [Steepest Ascent] with sideways Analysis - Sample Run 3
Initial:
[(6, 0), (3, 1), (2, 2), (5, 3), (3, 4), (5, 5), (7, 6), (3, 7)]
| | | | | | | | |
| | | | | | | | |
| | |Q| | | | | |
| |Q| | |Q| | |Q|
| | | | | | | | |
| | | |Q| |Q| | |
|Q| | | | | | | |
| | | | | | |Q| |
Step:  2
[(6, 0), (4, 1), (2, 2), (5, 3), (3, 4), (5, 5), (7, 6), (3, 7)]
| | | | | | | | |
| | | | | | | | |
| | |Q| | | | | |
| | | | |Q| | |Q|
| |Q| | | | | | |
| | | |Q| |Q| | |
|Q| | | | | | | |
| | | | | | |Q| |
Step:  3
[(6, 0), (4, 1), (2, 2), (5, 3), (3, 4), (0, 5), (7, 6), (3, 7)]
| | | | | |Q| | |
| | | | | | | | |
| | |Q| | | | | |
| | | | |Q| | |Q|
| |Q| | | | | | |
| | | |Q| | | | |
|Q| | | | | | | |
| | | | | | |Q| |
Step:  4
[(6, 0), (1, 1), (2, 2), (5, 3), (3, 4), (0, 5), (7, 6), (3, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | |Q| | | | | |
| | | | |Q| | |Q|
| | | | | | | | |
| | | |Q| | | | |
|Q| | | | | | | |
```

```
| | | | | | |Q| |
Step:  5
[(6, 0), (1, 1), (2, 2), (5, 3), (3, 4), (0, 5), (7, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | |Q| | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | | | | | |Q| |
Step:  6
[(6, 0), (1, 1), (7, 2), (5, 3), (3, 4), (0, 5), (7, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | |Q| | | |Q| |
Step:  7
[(6, 0), (1, 1), (7, 2), (5, 3), (3, 4), (0, 5), (2, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | |Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | |Q| | | | | |
Step:  8
[(6, 0), (1, 1), (7, 2), (5, 3), (3, 4), (0, 5), (4, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | | | |
| | | | |Q| | | |
| | | | | | |Q|Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | |Q| | | | | |
Step:  9
[(6, 0), (2, 1), (7, 2), (5, 3), (3, 4), (0, 5), (4, 6), (4, 7)]
| | | | | |Q| | |
| | | | | | | | |
```

```
| |Q| | | | | | |
| | | | |Q| | | |
| | | | | | |Q|Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | |Q| | | | | |
Step:  10
[(6, 0), (1, 1), (7, 2), (5, 3), (3, 4), (0, 5), (4, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | | | |
| | | | |Q| | | |
| | | | | | |Q|Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | |Q| | | | | |
Step:  11
[(6, 0), (1, 1), (7, 2), (5, 3), (3, 4), (0, 5), (7, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | |Q| | | |Q| |
Step:  12
[(6, 0), (1, 1), (2, 2), (5, 3), (3, 4), (0, 5), (7, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | |Q| | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | | | | | |Q| |
Step:  13
[(6, 0), (1, 1), (7, 2), (5, 3), (3, 4), (0, 5), (7, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | | | |
| | | | |Q| | | |
| | | | | | | |Q|
| | | |Q| | | | |
|Q| | | | | | | |
```

```
| | |Q| | | |Q| |
Step:  14
[(6, 0), (1, 1), (7, 2), (5, 3), (3, 4), (0, 5), (2, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | |Q|
| | | |Q| | | | |
|Q| | | | | | | |
| | |Q| | | | | |
Step:  15
[(6, 0), (1, 1), (7, 2), (7, 3), (3, 4), (0, 5), (2, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | |Q|
| | | | | | | | |
|Q| | | | | | | |
| | |Q|Q| | | | |
Step:  16
[(6, 0), (1, 1), (5, 2), (7, 3), (3, 4), (0, 5), (2, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | |Q|
| | |Q| | | | | |
|Q| | | | | | | |
| | | |Q| | | | |
Step:  17
[(6, 0), (1, 1), (7, 2), (7, 3), (3, 4), (0, 5), (2, 6), (4, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | |Q|
| | | | | | | | |
|Q| | | | | | | |
| | |Q|Q| | | | |
Step:  18
[(6, 0), (1, 1), (7, 2), (7, 3), (3, 4), (0, 5), (2, 6), (5, 7)]
| | | | | |Q| | |
| |Q| | | | | | |
```

```
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | | |
| | | | | | | |Q|
|Q| | | | | | | |
| | |Q|Q| | | | |
Step:  19
[(6, 0), (1, 1), (7, 2), (0, 3), (3, 4), (0, 5), (2, 6), (5, 7)]
| | | |Q| |Q| | |
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | | |
| | | | | | | |Q|
|Q| | | | | | | |
| | |Q| | | | | |
Step:  20
[(6, 0), (1, 1), (7, 2), (0, 3), (3, 4), (6, 5), (2, 6), (5, 7)]
| | | |Q| | | | |
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
| | | | | | | | |
| | | | | | | |Q|
|Q| | | | |Q| | |
| | |Q| | | | | |
Success:
[(4, 0), (1, 1), (7, 2), (0, 3), (3, 4), (6, 5), (2, 6), (5, 7)]
| | | |Q| | | | |
| |Q| | | | | | |
| | | | | | |Q| |
| | | | |Q| | | |
|Q| | | | | | | |
| | | | | | | |Q|
| | | | | |Q| | |
| | |Q| | | | | |
```

```
Hill climbing Search [Steepest Ascent] with sideways Analysis - Sample Run
4
Initial:
[(3, 0), (3, 1), (0, 2), (4, 3), (4, 4), (3, 5), (7, 6), (3, 7)]
| | |Q| | | | | |
| | | | | | | | |
```

```
| | | | | | | | |
|Q|Q| | | |Q| |Q|
| | | |Q|Q| | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | |Q| |
Step:  2
[(3, 0), (3, 1), (0, 2), (4, 3), (4, 4), (1, 5), (7, 6), (3, 7)]
| | |Q| | | | | |
| | | | | |Q| | |
| | | | | | | | |
|Q|Q| | | | | |Q|
| | | |Q|Q| | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | |Q| |
Step:  3
[(3, 0), (3, 1), (0, 2), (4, 3), (4, 4), (1, 5), (7, 6), (2, 7)]
| | |Q| | | | | |
| | | | | |Q| | |
| | | | | | | |Q|
|Q|Q| | | | | | |
| | | |Q|Q| | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | |Q| |
Step:  4
[(3, 0), (3, 1), (0, 2), (4, 3), (4, 4), (1, 5), (5, 6), (2, 7)]
| | |Q| | | | | |
| | | | | |Q| | |
| | | | | | | |Q|
|Q|Q| | | | | | |
| | | |Q|Q| | | |
| | | | | | |Q| |
| | | | | | | | |
| | | | | | | | |
Step:  5
[(3, 0), (3, 1), (0, 2), (7, 3), (4, 4), (1, 5), (5, 6), (2, 7)]
| | |Q| | | | | |
| | | | | |Q| | |
| | | | | | | |Q|
|Q|Q| | | | | | |
| | | | |Q| | | |
| | | | | | |Q| |
| | | | | | | | |
```

```
| | | |Q| | | | |
Success:
[(3, 0), (6, 1), (0, 2), (7, 3), (4, 4), (1, 5), (5, 6), (2, 7)]
| | |Q| | | | | |
| | | | | |Q| | |
| | | | | | | |Q|
|Q| | | | | | | |
| | | | |Q| | | |
| | | | | | |Q| |
| |Q| | | | | | |
| | | |Q| | | | |
```

Output:

```
Hill climbing Search [Steepest Ascent] Analysis
===============================================

N value:  8  (i.e  8 x 8 )
Total Runs:  100

Success, Runs:  13
Success, Rate:  13.0 %
Success, Average Steps:  5.31

Failure, Runs:  87
Failure, Rate:  87.0 %
Failure, Average Steps:  3.99


Flat local maxima / Shoulder:  87


Hill climbing Search [Steepest Ascent] with sideways Analysis
=============================================================

N value:  8  (i.e  8 x 8 )
Total Runs:  100

Success, Runs:  98
Success, Rate:  98.0 %
Success, Average Steps:  21.38

Failure, Runs:  2
Failure, Rate:  2.0 %
Failure, Average Steps:  54.0


Flat local maxima / Shoulder:  1
```

```
Random - restart hill - climbing search
=======================================

Total Runs:  100

Average Restarts:  8.33
Average Steps (last restart):  5.24
Average steps (all restarts):  34.96


Random - restart hill - climbing search with sideways Analysis
==============================================================

Total Runs:  100

Average Restarts:  1.45
Average Steps (last restart):  20.81
Average steps (all restarts):  27.59
```

Analysis:

	HillClimbing	HillClimbing with sideways move	Random Restart HillClimbing	Random Restart with sideways HillClimbing
Success Rate	13	98	100	100
Average Steps	5.31	21.38	Last restart: 5.24 All restarts: 34.96	Last restart: 20.81 All restarts: 27.59
Number of restarts	NA	NA	8.33	1.45
Flat local maxima / Shoulder	87	1	NA	NA
Failure rate	87	2	0	0
Average Steps(Failure case)	3.99	54	NA	NA
Total Runs	100	100	100	100

Source Code:

```
import random
import copy
import numpy as np
from IPython.display import clear_output
from os import system, name

print_states = False

class HillClimbing:
    """
    Attributes
    ----------
    startState : list of integers
        Initial state of the n queens problem.

    side_max : int
        Maximum steps allowed for sideways move.

    side_rem : int
        Sideway moves remaining

    total_steps : int
          Total steps taken by the algorithm, including all the steps of every
restart.

    N: int
        n value of the n queens problem.
    """
    def __init__(self, state = None, side_max = 0, N = 0):
        self.startState = state

        if(state == None and N == 0):
            print("Invalid N value. Going with default: 8")
            self.N = 8
        elif(state == None and N):
            self.N = N
        else:
            self.N = len(state)

        self.side_max = side_max
        self.side_rem = side_max
        self.total_steps = 0

    """
    get_diagonal_cells_to_right(self, row, col)
```

```
        Returns cells that are diagonal and are on the right side of the current cell.
    """
    def get_diagonal_cells_to_right(self, row, col):
        i = col+1
        cells = []
        while i < self.N:
            if row-(i-col) >= 0: cells.append((row-(i-col), i)) # Top right diagonal cells
            if row+(i-col) <= self.N-1: cells.append((row+(i-col), i)) # Bottom right diagonal cells
            i+=1
        return cells

    """ get_horizontal_cells_to_right(self, row, col)
        Returns cells that are horizontal and are on the right side of the current cell."""
    def get_horizontal_cells_to_right(self, row, col):
        i = col+1
        cells = []
        while i < self.N:
            cells.append((row, i))
            i+=1
        return cells

    """
    get_cells_to_right(self, row, col)
        Concatenates and returns the result of get_diagonal_cells_to_right and get_horizontal_cells_to_right
    """
    def get_cells_to_right(self, row, col):
        return self.get_horizontal_cells_to_right(row,col) + self.get_diagonal_cells_to_right(row,col)

    """
    get_cells_of_state(self, state)
        Get cell positions of the queens of the given state
    """
    def get_cells_of_state(self, state):
        cells = []
        for col, row in enumerate(state):
            cells.append((row,col))
        return cells

    """
    calculate_h(self, cellsOfState)
        Get heuristic value for a given state
    """
```

```
def calculate_h(self, cellsOfState):
    hValue = 0

    for row,col in cellsOfState:
        a_set = set(cellsOfState)
        b_set = set(self.get_cells_to_right(row,col))

        common = a_set.intersection(b_set)
        hValue += len(common)

    return hValue

"""
print_state(self, cellsOfState)
    Display the n queen state in matrix format
"""
def print_state(self, cellsOfState):
    global print_states
    if(print_states):
        print(cellsOfState)
        for i in range(self.N):
            str = '|'
            for j in range(self.N):
                if((i,j) in cellsOfState):
                    str += 'Q|'
                else:
                    str += ' |'
            print(str)

'''
get_h_matrix(self, cellsOfState)
    Calculate heuristic values for all the cells to take the next step.
    Returns [
        'Matrix of Heuristics',
        'Least of Heuristics',
        '2 arrays containing rows and columns of cells containing hLeast'
    ]
'''
def get_h_matrix(self, cellsOfState):
    hMatrix = np.zeros((self.N,self.N), int) + -1
    hLeast = sum(range(self.N)) + 1
    hLeastState = None

    for (x,y) in cellsOfState:
        for i in range(self.N):
            if(x == i):
                pass
            else:
```

```
                    newState = copy.deepcopy(cellsOfState)
                    newState[y] = temp = (i, y)
                    hMatrix[i,y] = self.calculate_h(newState)

                    hLeast = min(hLeast, hMatrix[i,y])
                    hLeastState = newState

        return hMatrix, hLeast, np.where(hMatrix == hLeast)

    '''
    steepest_ascent(self, state = None, hValue = None, step = 0)
         A recursive method implementation of steepest ascent. This method takes
a step towards the lowest heuristic value with each recursion.
        Returns [result, step]
            result:
                1 -> Flat, shoulder or flat local maxima
                2 -> Local Maxima
                3 -> Success
    '''
    def steepest_ascent(self, state = None, hValue = None, step = 0):

        cellsOfState = None

        if(step == 0):
            state = self.startState
            cellsOfState = self.get_cells_of_state(state)
            hValue = self.calculate_h(cellsOfState)
        else:
            cellsOfState = self.get_cells_of_state(state)

        step+=1
        self.total_steps+=1

        if(hValue == 0):
            if(print_states): print("Success: ")
            self.print_state(cellsOfState)
            return 3, step

        if(step == 1):
            if(print_states): print("Initial: ")
            self.print_state(cellsOfState)
        else:
            if(print_states): print('Step: ', step)
            self.print_state(cellsOfState)

        hMatrix = self.get_h_matrix(cellsOfState)
        hLeast = hMatrix[1]
```

```
        randomChoice = random.randint(0, len(hMatrix[2][0])-1)
        choice_row = hMatrix[2][0][randomChoice]
        choice_col = hMatrix[2][1][randomChoice]

        newState = copy.deepcopy(state)
        newState[choice_col] = choice_row

        if(hLeast < hValue):
            return self.steepest_ascent(newState, hLeast, step)
          elif (hLeast > hValue): # Checking if the current state is in local
maxima
            if(print_states): print("Search Failed")
            return 2, step # result -> 2 (Local Maxima)
        elif (hLeast == hValue): # Checking if the current state is flat
            if(self.side_rem): # Checking if there are any side steps remaining
                self.side_rem-=1
                return self.steepest_ascent(newState, hLeast, step)
            else:
                if(print_states): print("Search Failed")
                    return 1, step # result -> 1 (flat, shoulder or flat local
maxima)
    """
    get_random_state(self)
        Generates and returns a random state everytime.
    """
    def get_random_state(self):
        state = []
        for i in range(self.N):
            state.append(random.randint(0,self.N-1))

        return state

    """
    random_restart(self)
             A method that implements Random restart algorithm and uses the
"steepest_ascent" method.

    """
    def random_restart(self):
        restart_count = 0

        while True:
            restart_count+=1
            self.startState = self.get_random_state()
            # print('self.startState: ', self.startState)
            result = self.steepest_ascent()
            if(result[0] == 3):
                return restart_count, result[1], self.total_steps
```

```
                break

class HillClimbAnalysis:
    """
    Attributes
    ----------
    maxIter : int
        Maximum number of iterations to perform.

    nValue: int
        n value of the n queens problem.

    steep_climb_stat : list of lists - [[0,[]],[0,[]],[0,[]],[0,[]]]
          Variable to store the statistics of Steepest ascent - hill climbing
without sideways move
    steep_climb_w_side_stat: list of lists - [[0,[]],[0,[]],[0,[]],[0,[]]]
          Variable to store the statistics of Steepest ascent - hill climbing
with sideways move
        1st List - [0,[]]: Stores total iterations.
           2nd List - [0,[]]: Stores count of shoulder or flat local maxima
encountered and the number of steps it took for each encounter
          3rd List - [0,[]]: Stores count of local maxima encountered and the
number of steps it took for each encounter
         4th List - [0,[]]: Stores count of successes encountered and the number
of steps it took for each encounter

    random_restart_stat: list - [0, [], [], []]
          Variable to store the statistics of random restart - hill climbing
without sideways move
    random_restart_w_side_stat
         Variable to store the statistics of random restart - hill climbing with
sideways move
        1st element - int: Stores total iterations.
        2nd element - list of int: Stores count of restarts for every iteration
          3rd element - list of int: Stores count of steps of last restart for
every iteration
          4th element - list of int: Stores count of steps of all restarts for
every iteration
    """
    def __init__(self, nValue, maxIter, sideMax = 0):
        self.nValue = nValue
        self.maxIter = maxIter
        self.sideMax = sideMax
        self.steep_climb_stat = [[0,[]],[0,[]],[0,[]],[0,[]]]
        self.steep_climb_w_side_stat = [[0,[]],[0,[]],[0,[]],[0,[]]]
        self.random_restart_stat = [0, [], [], []]
        self.random_restart_w_side_stat = [0, [], [], []]
```

```
    """
    is_notebook:
          Method to find out if the running environment is Jupyter notebook or
not.
                                                                  referred              from:
https://stackoverflow.com/questions/15411967/how-can-i-check-if-code-is-execute
d-in-the-ipython-notebook
    """
    def is_notebook(self):
        try:
            shell = get_ipython().__class__.__name__
            if shell == 'ZMQInteractiveShell':
                return True   # Jupyter notebook or qtconsole
            elif shell == 'TerminalInteractiveShell':
                return False  # Terminal running IPython
            else:
                return False  # Other type (?)
        except NameError:
            return False      # Probably standard Python interpreter

    """
    update_progress(self, progress)
        Method that displays the progress status(bar).

                                                                  referred              from:
https://www.mikulskibartosz.name/how-to-display-a-progress-bar-in-jupyter-noteb
ook/
    """
    def update_progress(self, progress):

        if(print_states == True): return

        import sys
#         if sys.stdin.isatty():
#             # running interactively
#             print "running interactively"

        if(sys.stdin.isatty() or self.is_notebook()):
            pass
        else:
            return

        bar_length = 20
        if isinstance(progress, int):
            progress = float(progress)
        if not isinstance(progress, float):
            progress = 0
        if progress < 0:
```

```
            progress = 0
        if progress >= 1:
            progress = 1

        block = int(round(bar_length * progress))

        if(self.is_notebook()):
            clear_output(wait = True)
        else:
            # for windows
            if name == 'nt':
                _ = system('cls')

            # for mac and linux(here, os.name is 'posix')
            else:
                _ = system('clear')

             text  =  "Progress:  [{0}]  {1:.1f}%".format(  "#"  *  block  +  "-"  *
(bar_length - block), progress * 100)
        print(text)

    """
    start_analysis(self)
         Starts iterating 'maxIter' number of times and performs steepest ascent
and random restart
        hill climbing with and without sideways move.

        Then calls print_analysis() method to print the found stats.
    """
    def start_analysis(self):

        if(self.nValue in range(4)):
            print('Invalid N value. Please provide a number above 3.')
            return

        if(self.maxIter < 1):
             print('Invalid iterations value. Please provide a number that is 1
or above.')
            return

        self.update_progress(0) # updates progress bar with the progress level

        for n in range(self.maxIter):
            self.steep_climb_stat[0][0]+=1
            self.steep_climb_w_side_stat[0][0]+=1
            self.random_restart_stat[0]+=1
            self.random_restart_w_side_stat[0]+=1
            state = []
```

```
            for i in range(self.nValue): # Generating random state
                state.append(random.randint(0,self.nValue-1))

            # state = [4,5,6,3,4,5,6,5]
            # print(state)
                if(print_states): print("Hill climbing Search [Steepest Ascent]
Analysis")
            hillClimbing = HillClimbing(state)
            result = hillClimbing.steepest_ascent()
                self.steep_climb_stat[result[0]][0]+=1 # Incrementing respective
result count
               self.steep_climb_stat[result[0]][1].append(result[1]) # appending
steps to corresponding list of result

                if(print_states): print("Hill climbing Search [Steepest Ascent]
with sideways Analysis")
            hillClimbing = HillClimbing(state, 100)
            result = hillClimbing.steepest_ascent()
                   self.steep_climb_w_side_stat[result[0]][0]+=1 # Incrementing
respective result(i.e flat local maxima, local maxima and success) count
                  self.steep_climb_w_side_stat[result[0]][1].append(result[1]) #
appending steps to corresponding list of result(i.e flat local maxima, local
maxima and success)

            if(print_states): print("Random - restart hill - climbing search")
            hillClimbing = HillClimbing(None, 0, self.nValue)
            result = hillClimbing.random_restart()
               self.random_restart_stat[1].append(result[0]) # appending restart
count to stat
                 self.random_restart_stat[2].append(result[1]) # appending step
count of last run to stat
                 self.random_restart_stat[3].append(result[2]) # appending total
step to stat

               if(print_states): print("Random - restart hill - climbing search
with sideways Analysis")
            hillClimbing = HillClimbing(None, 100, self.nValue)
            result = hillClimbing.random_restart()
                self.random_restart_w_side_stat[1].append(result[0]) # appending
restart count to stat
                self.random_restart_w_side_stat[2].append(result[1]) # appending
step count of last run to stat
                self.random_restart_w_side_stat[3].append(result[2]) # appending
total step to stat

              self.update_progress(self.random_restart_stat[0] / self.maxIter) #
updates progress bar with the progress level
```

```
        self.print_analysis()
        # print('random_restart_stat: ', self.random_restart_stat)
                                   #    print('random_restart_w_side_stat:    ',
self.random_restart_w_side_stat)

    """
    print_analysis(self)
           Calls respective methods to display the analysis/report for all 4
algorithms
    """
    def print_analysis(self):
             self.print_steep_climb_stat(self.steep_climb_stat, "Hill climbing
Search [Steepest Ascent] Analysis")
                self.print_steep_climb_stat(self.steep_climb_w_side_stat, "Hill
climbing Search [Steepest Ascent] with sideways Analysis")
              self.print_rand_restart_stat(self.random_restart_stat, "Random -
restart hill - climbing search")
         self.print_rand_restart_stat(self.random_restart_w_side_stat, "Random -
restart hill - climbing search with sideways Analysis")

    """
    print_rand_restart_stat(self)
           Displays the report for random restart algorithm with and without
sideways move.
    """
    def print_rand_restart_stat(self, result, title):

        totalRuns = result[0]
        averageRestarts = sum(result[1]) / totalRuns
        averageLastSteps = sum(result[2]) / totalRuns
        averageTotalSteps = sum(result[3]) / totalRuns

        print()
        print()
        print(title)
        t_underline = ''
        for i in range(len(title)): t_underline+="="
        print(t_underline)
        print()
                             print("N   value:   ",   self.nValue,   "   (i.e
",self.nValue,"x",self.nValue,")")
        print("Total Runs: ", totalRuns)
        print()
        print("Average Restarts: ", averageRestarts)
        print("Average Steps (last restart): ", averageLastSteps)
        print("Average steps (all restarts): ", averageTotalSteps)
```

```
    """
    print_steep_climb_stat(self)
            Displays the report for steepest ascent algorithm with and without
sideways move.
    """
    def print_steep_climb_stat(self, result, title):

        totalRuns = result[0][0]

        successRuns = result[3][0]

        if successRuns:
            successRate = round((successRuns/totalRuns)*100,2)
            successSteps = result[3][1]
            successAvgSteps = round(sum(successSteps)/successRuns, 2)
        else:
            successRate = successSteps = successAvgSteps = '-'

        failureRuns = result[1][0]+result[2][0]

        if failureRuns:
            failureRate = round((failureRuns/totalRuns)*100,2)
            failureSteps = result[1][1]+result[2][1]
            failureAvgSteps = round(sum(failureSteps)/failureRuns,2)
        else:
            failureRate = failureSteps = failureAvgSteps = '-'

        flatRuns = result[1][0]

        print()
        print()
        print(title)
        t_underline = ''
        for i in range(len(title)): t_underline+="="
        print(t_underline)
        print()
                            print("N value: ", self.nValue, " (i.e
",self.nValue,"x",self.nValue,")")
        print("Total Runs: ", totalRuns)
        print()
        print("Success, Runs: ", successRuns)
        print("Success, Rate: ", successRate, "%")
        # print("Success, Steps: ", successSteps)
        print("Success, Average Steps: ", successAvgSteps)
        print()
        print("Failure, Runs: ", failureRuns)
        print("Failure, Rate: ", failureRate, "%")
        # print("Failure, Steps: ", failureSteps)
```

```
        print("Failure, Average Steps: ", failureAvgSteps)
        print()
        print()
        print("Flat local maxima / Shoulder: ", flatRuns)

        return

input_N = 0
input_iterations = 0
input_sideways = 0

#Reading N value of N Queens problem
while(True):
    try:
        input_N = (int)(input("Please enter N value: "))
        if(input_N < 4):
            print("Please enter a number that is above 3! ")
        else:
            break
    except ValueError:
        print("Please enter a number!")

#Reading maximum iterations value
while(True):
    try:
        input_iterations = (int)(input("Please enter iterations value: "))
        if(input_iterations < 1):
            print("Please enter a number that is 1 or above! ")
        else:
            break
    except ValueError:
        print("Please enter a number!")

#Reading maximum sideways value
while(True):
    try:
           input_sideways = (int)(input("Please enter a value for the maximum
sideways move allowed: "))
        if(input_sideways < 1):
            print("Please enter a number that is 1 or above! ")
        else:
            break
    except ValueError:
        print("Please enter a number!")

hillClimbAnalysis = HillClimbAnalysis(input_N, input_iterations,
input_sideways)
hillClimbAnalysis.start_analysis()
```

Citation:

Progress bar:

https://www.mikulskibartosz.name/how-to-display-a-progress-bar-in-jupyter-notebook/

https://en.wikipedia.org/wiki/Hill_climbing