CHARLOTTE

ITCS 6150-Intelligent Systems

Project 2

N queens problem using Hill Climbing

Submitted By

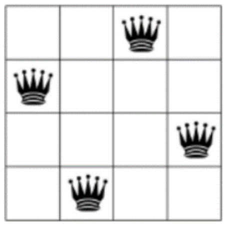
Srushti Khot(801203532)
Sanket Revadigar(801203510)

Instructor

Prof. Dewan T. Ahmed, Ph.D.

Problem Statement:

The N-Queens Problem consists of placing N queens on an NxN chessboard so that no two queens can capture each other. That is, no two queens are allowed to be placed on the same row, the same column or the same diagonal. The following figure illustrates a solution to the 4-Queens Problem.



The problem of finding all solutions to the 8-queens problem can be quite computationally expensive, as there are 64C864C8 =4,426,165,368=4,426,165,368 possible arrangements of eight queens on an 8×88×8board, but only 9292 solutions.

Using a programming language of your choice, implement the followings:

1. Hill climbing search
2. Hill-climbing search with sideways move
3. Random-restart hill-climbing with sideways move.
4. Random-restart hill-climbing without sideways move .

Hill climbing search:

Hill Climbing is a heuristic search algorithm. Mathematical optimization problems use this in the field of Artificial Intelligence. Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem.

Hill climbing search with sideways move:

It is a variant of Hill Climbing Search. if no uphill moves, allow moving to a state with the same value as the current one (escape shoulders)

Random Restart Hill Climbing:

“If at first you don’t succeed, try, try again” is adopted by Random-restart hill climbing.

Series of hill-climbing searches are performed from randomly generated initial states,1 until a goal is found. It is trivially complete with probability approaching 1, because it will eventually generate a goal state as the initial state. If each hill-climbing search has a probability p of success, then the expected number of restarts required is 1/p.

Classes and methods:

Class: Hill_Climbing_Search

Methods:

- __init__ : It acts like constructor. A new object is created of class Hill_Climbing_Search whenever the __int__ method is used.

  It initializes the following variables:

  - first_state: It is the initial state.
  - m_sideways: It is maximum number of sideways steps can be performed.
  - r_sides: It contains the number of remaining sideways moves.
  - total_number_steps : Total number of steps used by the algorithm.
  - nqueens: number of queens present in the matrix.

- retrive_right_diagonal: Returns the cells that are present diagonally from the right of current cell.
- retrieve_right_horizontal - Outputs the horizontal cells that are to the right of the current cell
- combine - Combines right horizontal cells and right diagonal cells to get all cells that are to the right of current cell.
- queen_cell - retrieve the position of the cell of the queen of the specified tree.
- heuristic_n_queens - Evaluate the heuristic value for specified state.

- print_state - Print the problem as matrix.
- cal_heuristic - Calculate heuristic values for all the right to take the next step. Returns ['matrix of heuristics value','least value of Heuristics','two arrays containing row and col of right cells containing lower value of Heuristics']
- steepest_ascent_algo -Calculates the least value of heuristic and executes.we get 1 if there is Flat,shoulder or flat local maxima. 2-if local maxima occurred. 3- if success occurs.
- random_state - new random state is generated whenever this method is called.
- random_restart_hc - Using Steepest Ascent as the base ,Random restart hill climbing search is implemented.

Class Hill_Climbing_check:

Methods:

- __init__: It acts like constructor.A new object is created of class Hill_Climbing_Search whenever the __int__ method is used.

  n_value: number of queens

  max_iter: (Maximum number of iterations)

  m_sideways:

  steepest_ascent_config: (Store the statistics of Steepest ascent hill climbing without sideways move.

  steepest_ascent_side_config: Store the statistics of Steepest ascent hill climbing with sideways move.

  random_restart_config: Store the statistics of random restart hill climbing without sideways move.

  random_restart_side_config: Store the statistics of random restart hill climbing with sideways move.

- explore - Executes steepest Ascent algo and random restart hill climbing for max number of iterations times
- final_outcomes -End analysis of all 4 algorithms are printed.
- display_random_restart_config(self) - Prints the Random Restart hill climbing search inspection.

- display_steepest_ascent_config -The report for steepest ascent hill climb with sideways and without sideways move is displayed.

Final Output for 8 queens' problem for iterations 100,200,400,600,800,1000:

100 iterations:

```
HILL CLIMBING SEARCH

====================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  100

SUCCESS RUNS:  16

SUCCESS RATES:  16.0 %

SUCCESS AVERAGE STEPS:  5.25

NUMBER_OF_FAILURE RUNS:  84

FAILURE RATE:  84.0 %

FAILURE AVERAGE STEPS:  4.14

FLAT LOCAL MINIMA:  82

HILL CLIMBING WITH SIDEWAYS

===========================

 VALUE OF N:  8 (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  100

SUCCESS RUNS:  20

SUCCESS RATES:  20.0 %
SUCCESS AVERAGE STEPS:  5.95

NUMBER_OF_FAILURE RUNS:  80

FAILURE RATE:  80.0 %

FAILURE AVERAGE STEPS:  5.28

FLAT LOCAL MINIMA:  78

RANDOM RESTART HILLCLIMBING

---------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  100

AVERAGE NUMBER OF RESTARTS:  7.62

AVERAGE NUMBER OF STEPS OF LAST RESTART 5.2

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 31.91

RANDOM RESTART WITH SIDEWAYS

----------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  100

AVERAGE NUMBER OF RESTARTS:  7.29

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.17

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 30.92
```

FOR 200 iterations:

```
HILL CLIMBING SEARCH

====================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  200

SUCCESS RUNS:  24

SUCCESS RATES:  12.0 %

SUCCESS AVERAGE STEPS:  5.04

NUMBER_OF_FAILURE RUNS:  176

FAILURE RATE:  88.0 %

FAILURE AVERAGE STEPS:  3.96

FLAT LOCAL MINIMA:  172

HILL CLIMBING WITH SIDEWAYS

===========================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  200

SUCCESS RUNS:  41

SUCCESS RATES:  20.5 %

SUCCESS AVERAGE STEPS:  5.66

NUMBER_OF_FAILURE RUNS:  159

FAILURE RATE:  79.5 %
FAILURE AVERAGE STEPS:  5.22

FLAT LOCAL MINIMA:  156

RANDOM RESTART HILLCLIMBING

---------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  200

AVERAGE NUMBER OF RESTARTS:  7.495

AVERAGE NUMBER OF STEPS OF LAST RESTART  4.98

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 31.2

RANDOM RESTART WITH SIDEWAYS

----------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  200

AVERAGE NUMBER OF RESTARTS:  6.14

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.235

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 27.085
```

FOR 400 ierations:

```
HILL CLIMBING SEARCH

====================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  400
```

```
SUCCESS RUNS:  63

SUCCESS RATES:  15.75 %

SUCCESS AVERAGE STEPS:  5.25

NUMBER_OF_FAILURE RUNS:  337

FAILURE RATE:  84.25 %

FAILURE AVERAGE STEPS:  4.01

FLAT LOCAL MINIMA:  334

HILL CLIMBING WITH SIDEWAYS

===========================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  400

SUCCESS RUNS:  113

SUCCESS RATES:  28.25 %

SUCCESS AVERAGE STEPS:  5.61

NUMBER_OF_FAILURE RUNS:  287

FAILURE RATE:  71.75 %

FAILURE AVERAGE STEPS:  5.27

FLAT LOCAL MINIMA:  280

RANDOM RESTART HILLCLIMBING

---------------------------

n is  8  (i.e  8 x 8 )
```

```
TOTAL NUMBER OF RUNS:  400

AVERAGE NUMBER OF RESTARTS:  6.79

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.03

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 28.475

RANDOM RESTART WITH SIDEWAYS

---------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  400

AVERAGE NUMBER OF RESTARTS:  6.575

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.225

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 28.79
```

For 600 iterations:

```
HILL CLIMBING SEARCH

====================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  600

SUCCESS RUNS:  88

SUCCESS RATES:  14.67 %

SUCCESS AVERAGE STEPS:  5.05

NUMBER_OF_FAILURE RUNS:  512

FAILURE RATE:  85.33 %
FAILURE AVERAGE STEPS:  4.01

FLAT LOCAL MINIMA:  507

HILL CLIMBING WITH SIDEWAYS

===========================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  600

SUCCESS RUNS:  151

SUCCESS RATES:  25.17 %

SUCCESS AVERAGE STEPS:  5.57

NUMBER_OF_FAILURE RUNS:  449

FAILURE RATE:  74.83 %

FAILURE AVERAGE STEPS:  5.24

FLAT LOCAL MINIMA:  437

RANDOM RESTART HILLCLIMBING

---------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  600

AVERAGE NUMBER OF RESTARTS:  7.475

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.11

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 31.276666666666667
```

```
RANDOM RESTART WITH SIDEWAYS
----------------------------
n is  8  (i.e  8 x 8 )
TOTAL NUMBER OF RUNS:  600
AVERAGE NUMBER OF RESTARTS:  6.136666666666667
AVERAGE NUMBER OF STEPS OF LAST RESTART  5.236666666666666
AVERAGE NUMBER OF STEPS OF ALL RESTARTS 26.998333333333335
```

For 800 iterations:

```
HILL CLIMBING SEARCH
====================
 VALUE OF N:  8  (i.e  8 x 8 )
TOTAL NUMBER OF RUNS:  800
SUCCESS RUNS:  104
SUCCESS RATES:  13.0 %
SUCCESS AVERAGE STEPS:  5.05
NUMBER_OF_FAILURE RUNS:  696
FAILURE RATE:  87.0 %
FAILURE AVERAGE STEPS:  4.05
FLAT LOCAL MINIMA:  685
```

```
HILL CLIMBING WITH SIDEWAYS

===========================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  800

SUCCESS RUNS:  172

SUCCESS RATES:  21.5 %

SUCCESS AVERAGE STEPS:  5.48

NUMBER_OF_FAILURE RUNS:  628

FAILURE RATE:  78.5 %

FAILURE AVERAGE STEPS:  5.29

FLAT LOCAL MINIMA:  611

RANDOM RESTART HILLCLIMBING

---------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  800

AVERAGE NUMBER OF RESTARTS:  7.18875

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.06625

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 30.2575
```

```
RANDOM RESTART WITH SIDEWAYS

----------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  800

AVERAGE NUMBER OF RESTARTS:  6.61875

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.19

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 28.80875
```

For 1000 iterations:

```
HILL CLIMBING SEARCH

====================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  1000

SUCCESS RUNS:  127

SUCCESS RATES:  12.7 %

SUCCESS AVERAGE STEPS:  5.08

NUMBER_OF_FAILURE RUNS:  873

FAILURE RATE:  87.3 %

FAILURE AVERAGE STEPS:  4.09

FLAT LOCAL MINIMA:  863
```

```
HILL CLIMBING WITH SIDEWAYS

============================

 VALUE OF N:  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  1000

SUCCESS RUNS:  238

SUCCESS RATES:  23.8 %

SUCCESS AVERAGE STEPS:  5.78

NUMBER_OF_FAILURE RUNS:  762

FAILURE RATE:  76.2 %

FAILURE AVERAGE STEPS:  5.27

FLAT LOCAL MINIMA:  743

RANDOM RESTART HILLCLIMBING

---------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  1000

AVERAGE NUMBER OF RESTARTS:  7.396

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.063

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 30.951
```

```
RANDOM RESTART WITH SIDEWAYS

----------------------------

n is  8  (i.e  8 x 8 )

TOTAL NUMBER OF RUNS:  1000

AVERAGE NUMBER OF RESTARTS:  6.237

AVERAGE NUMBER OF STEPS OF LAST RESTART  5.2

AVERAGE NUMBER OF STEPS OF ALL RESTARTS 27.55
```

OUTPUT ANALYSIS:

	Hill Climbing	Hill Climbing with sideways move	Random restart hill climbing	Random restart hill climbing with sideways move
Success rate	16%	20%	100%	100%
Average number of steps (Success case)	5.25	5.95	Last Restart – 5.2 All Restart – 31.91	Last Restart – 5.17 All Restart – 30.92
Number of restarts	N/A	N/A	7.62	7.29
Flat local maxima/shoulder	82	78	N/A	N/A
Failure rate	84%	80%	0	0
Average number of steps (Failure case	4.14	5.28	N/A	N/A
Total Runs	100	100	100	100

	Hill Climbing	Hill Climbing with sideways move	Random restart hill climbing	Random restart hill climbing with sideways move
Success rate	12%	20.05%	100%	100%
Average number of steps (Success case)	5.04	5.66	Last Restart – 4.98 All Restart – 31.2	Last Restart – 5.23 All Restart – 27.08
Number of restarts	N/A	N/A	7.495	6.14
Flat local maxima/shoulder	172	156	N/A	N/A
Failure rate	88%	79.5%	0	0
Average number of steps (Failure case	3.96	5.22	N/A	N/A
Total Runs	200	200	200	200

	Hill Climbing	Hill Climbing with sideways move	Random restart hill climbing	Random restart hill climbing with sideways move
Success rate	15.75%	28.25%	100%	100%
Average number of steps (Success case)	5.25	5.61	Last Restart – 5.03 All Restart – 28.47	Last Restart – 5.22 All Restart – 28.79
Number of restarts	N/A	N/A	6.79	6.575
Flat local maxima/shoulder	334	280	N/A	N/A
Failure rate	84.25%	71.25%	0	0
Average number of steps (Failure case	4.01	5.27	N/A	N/A
Total Runs	400	400	400	400

	Hill Climbing	Hill Climbing with sideways move	Random restart hill climbing	Random restart hill climbing with sideways move
Success rate	14.67%	25.17%	100%	100%
Average number of steps (Success case)	5.05	5.57	Last Restart – 5.11 All Restart – 31.27	Last Restart – 5.23 All Restart – 26.99
Number of restarts	N/A	N/A	7.475	6.136
Flat local maxima/shoulder	507	437	N/A	N/A
Failure rate	85.33%	74.83%	0	0
Average number of steps (Failure case	4.01	5.24	N/A	N/A
Total Runs	600	600	600	600

	Hill Climbing	Hill Climbing with sideways move	Random restart hill climbing	Random restart hill climbing with sideways move
Success rate	13%	21.5%	100%	100%
Average number of steps (Success case)	5.05	5.48	Last Restart – 5.06 All Restart – 30.25	Last Restart – 5.19 All Restart – 28.80
Number of restarts	N/A	N/A	7.18	6.61
Flat local maxima/shoulder	685	611	N/A	N/A
Failure rate	87%	78.5%	0	0
Average number of steps (Failure case	4.05	5.29	N/A	N/A
Total Runs	800	800	800	800

	Hill Climbing	Hill Climbing with sideways move	Random restart hill climbing	Random restart hill climbing with sideways move
Success rate	12.7%	23.8%	100%	100%
Average number of steps (Success case)	5.08	5.78	Last Restart – 5.06 All Restart – 30.95	Last Restart – 5.2 All Restart – 27.55
Number of restarts	N/A	N/A	7.39	6.23
Flat local maxima/shoulder	863	743	N/A	N/A
Failure rate	87.3%	76.2%	0	0
Average number of steps (Failure case	4.09	5.27	N/A	N/A
Total Runs	1000	1000	1000	1000

Source Code:

```
import random
import numpy as np
import copy
class hillclimb:
    def __init__(self, state = None, m_sideways = 0, nqueens = 0):
        self.first_state = state
        if(state == None and nqueens):
            self.nqueens = nqueens
        else:
            l_s=len(state)
            self.nqueens =l_s
        self.total_number_steps = 0
        self.m_sideways = m_sideways
        self.r_sides = m_sideways
    # retrive_right_diagonal -  Outputs the diagonal cells that are to the right of the current cell.
    def retrive_right_diagonal(self, r, c):
        i = c+1
        right_di_cells = []
        while i < self.nqueens:
            c1=r-(i-c)
            if c1 >= 0:
                right_di_cells.append((c1,i))
            c2= r+(i-c)
            if c2 <= self.nqueens-1:
                right_di_cells.append((c2, i))
            i=i+1
        return right_di_cells
    # retrieve_right_horizontal - Outputs the horizontal cells that are to the right of
the current cell
    def retrieve_right_horizontal(self, r, c):
        i = c+1
        right_h_cells = []
        while i < self.nqueens:
            right_h_cells.append((r, i))
            i+=1
        return right_h_cells
    #combine - Combines right horizontal cells and right diagonal cells to get all cells
that are to the right of current cell.
    def combine(self, r, c):
        rrh=self.retrieve_right_horizontal(r,c)
        rrd=self.retrive_right_diagonal(r,c)
        return_ele=rrh+rrd
        return return_ele
    # queen_cell - retrieve the position of the cell of the queen of the specified tree.
    def queen_cell(self, state):
        q_celllls = []
        for columns, r in enumerate(state):
            q_celllls.append((r,columns))
        return q_celllls
```

```
# heuristic_n_queens - Evaluate  the heuristic value for specified state.
def heuristic_n_queens(self, cell_of_state):
    hv = 0
    for row,column in cell_of_state:
        x_coord = set(cell_of_state)
        y_coord = set(self.combine(row,column))
        coord_intersection = x_coord.intersection(y_coord)
        l_i=len(coord_intersection)
        hv =hv+ l_i
    return hv
# print_state - Print the problem as matrix.
def print_state(self, cell_of_state):
    print(cell_of_state)
    for i in range(self.nqueens):
        mat = '|'
        for j in range(self.nqueens):
            if((i,j) in cell_of_state):
                mat =mat+ 'Q|'
            else:
                mat =mat+ '-|'
        print(mat)
```

```
'''
cal_heuristic - Calculate heuristic values for all the right to take the next step.Returns ['matrix of heuristics value','least value of Heuristics','two arrays containing row and col of right cells containing lower value of Heuristics' ]
'''
def cal_heuristic(self, cell_of_state):
    mzero= np.zeros((self.nqueens,self.nqueens), int)
    matix_of_heuristic =mzero + -1
    least_of_heiristic = sum(range(self.nqueens)) + 1
    for (x_coord,y_coord) in cell_of_state:
        for i in range(self.nqueens):
            if(x_coord == i):
                pass
            else:
                next_state = copy.deepcopy(cell_of_state)
                tup1=(i,y_coord)
                next_state[y_coord] =tup1
                mh=self.heuristic_n_queens(next_state)
                matix_of_heuristic[i,y_coord] = mh
                min_val= min(least_of_heiristic, matix_of_heuristic[i,y_coord])
                least_of_heiristic =min_val
                ind=np.where(matix_of_heuristic == least_of_heiristic)
    return matix_of_heuristic, least_of_heiristic,ind
    '''
    steepest_ascent_algo -Calculates the least value of heuristic and executes.we get
1 if there is Flat,shoulder
    or flat local maxima. 2-if local maxima occurred. 3- if success occurs.
    '''
    def steepest_ascent_algo(self, state = None, hv = None, steps = 0):
        cell_of_state = None
        if(steps == 0):
            state = self.first_state
            cell_of_state = self.queen_cell(state)
            hv = self.heuristic_n_queens(cell_of_state)
        else:
            cell_of_state = self.queen_cell(state)
        steps=steps+1
        self.total_number_steps+=1
        if(hv == 0):
            print("***Success achieved***")
            self.print_state(cell_of_state)
            return 3, steps
        if(steps == 1):
            print("***Initial representation***")
            self.print_state(cell_of_state)
        else:
            print('Step number-->', steps)
            self.print_state(cell_of_state)
        matix_of_heuristic = self.cal_heuristic(cell_of_state)
        least_of_heiristic = matix_of_heuristic[1]
        l_mh=len(matix_of_heuristic[2][0])
        random_int = random.randint(0, l_mh-1)
        row = matix_of_heuristic[2][0][random_int]
        column = matix_of_heuristic[2][1][random_int]
        next_state = copy.deepcopy(state)
        next_state[column] = row
        if(least_of_heiristic < hv):
            return self.steepest_ascent_algo(next_state, least_of_heiristic, steps)
        elif (least_of_heiristic > hv):
            print("Search procedure Failed")
            return 2, steps
        elif (least_of_heiristic == hv):
            if(self.r_sides):
                self.r_sides=self.r_sides-1
                return self.steepest_ascent_algo(next_state, least_of_heiristic, steps)
            else:
                print("Search procedure Failed")
                return 1, steps
    # random_state - new random state is generated whenever this method  is called.
    def random_state(self):
        rset = []
        for i in range(self.nqueens):
            r_int=random.randint(0,self.nqueens-1)
            rset.append(r_int)
        return rset
    # random_restart_hc - Using Steepest Ascent as the base ,Random restart hill
climbing search is implemented.
    def random_restart_hc(self):
        r_val = 0
        while True:
            r_val=r_val+1
            self.first_state = self.random_state()
            output = self.steepest_ascent_algo()
            if(output[0] == 3):
                t_steps=self.total_number_steps
                return r_val, output[1], t_steps
                break
class Hill_Climbing_check:
    def __init__(self, n_value, max_iter, m_sideways = 0):
        self.n_value = n_value
        self.max_iter = max_iter
        self.m_sideways = m_sideways
        self.steepest_ascent_config = [[0,[]],[0,[]],[0,[]],[0,[]]]
        self.steepest_ascent_side_config = [[0,[]],[0,[]],[0,[]],[0,[]]]
        self.random_restart_config = [0, [], [], []]
        self.random_restart_side_config = [0, [], [], []]
    # explore - Executes steepest Ascent algo and random restart hill climbing for max number of iterations times.
    def explore(self):
        if(self.n_value in range(4)):
            print('VALUE MUST BE GREATER THAN 3.')
            return
        if(self.max_iter < 1):
            print('VALUE MUST BE GREATER THAN 1.')
            return
        for n in range(self.max_iter):
            self.steepest_ascent_config[0][0]=self.steepest_ascent_config[0][0]+1
self.steepest_ascent_side_config[0][0]=self.steepest_ascent_side_config[0][0]+1
            self.random_restart_config[0]=self.random_restart_config[0]+1
            self.random_restart_side_config[0]= self.random_restart_side_config[0]+1
            s = []
            for i in range(self.n_value):
                rand_int1=random.randint(0,self.n_value-1)
                s.append(rand_int1)
            print("HILL CLIMBING SEARCH")
            hillClimbing = hillclimb(s)
            outcome = hillClimbing.steepest_ascent_algo()
self.steepest_ascent_config[outcome[0]][0]=self.steepest_ascent_config[outcome[0]][0]+1
            self.steepest_ascent_config[outcome[0]][1].append(outcome[1])
            print("HILL CLIMBING WITH SIDEWAYS")
            hillClimbing = hillclimb(s, self.m_sideways)
            outcome = hillClimbing.steepest_ascent_algo()
            self.steepest_ascent_side_config[outcome[0]][0]=
self.steepest_ascent_side_config[outcome[0]][0]+1
            self.steepest_ascent_side_config[outcome[0]][1].append(outcome[1])
            print("RANDOM RESTART HILLCLIMBING")
            hillClimbing = hillclimb(None, 0, self.n_value)
            outcome = hillClimbing.random_restart_hc()
            self.random_restart_config[1].append(outcome[0])
            self.random_restart_config[2].append(outcome[1])
            self.random_restart_config[3].append(outcome[2])
            print("RANDOM RESTART WITH SIDEWAYS")
            hillClimbing = hillclimb(None, self.m_sideways, self.n_value)
            outcome = hillClimbing.random_restart_hc()
            self.random_restart_side_config[1].append(outcome[0])
            self.random_restart_side_config[2].append(outcome[1])
            self.random_restart_side_config[3].append(outcome[2])
        self.final_outcomes()
    # final_outcomes -End analysis of all 4 algorithms are printed.
    def final_outcomes(self):
        self.display_steepest_ascent_config(self.steepest_ascent_config,         "HILL
CLIMBING SEARCH")
        self.display_steepest_ascent_config(self.steepest_ascent_side_config,   "HILL
CLIMBING WITH SIDEWAYS")
        self.display_random_restart_config(self.random_restart_config,    "RANDOM
RESTART HILLCLIMBING")
        self.display_random_restart_config(self.random_restart_side_config,
"RANDOM RESTART WITH SIDEWAYS")
```

```
    #display_random_restart_config(self) - Prints the Random Restart hill climbing search inspection
    def display_random_restart_config(self, result, first):
        sum_runs = result[0]
        s1=sum(result[1])
        average_of_restart = s1 / sum_runs
        s2= sum(result[2])
        avg_last_steps = s2/ sum_runs
        s3=sum(result[3])
        avg_total = s3 / sum_runs
        print("\n\n"+first)
        border = ''
        lenf=len(first)
        for i in range(lenf):
            border+="-"
        print(border)
        print()
        print("n is ", self.n_value, " (i.e ",self.n_value,"x_coord",self.n_value,")")
        print("TOTAL NUMBER OF RUNS: ", sum_runs)
        print()
        print("AVERAGE NUMBER OF RESTARTS: ", average_of_restart)
        print("AVERAGE NUMBER OF STEPS OF LAST RESTART ", avg_last_steps)
        print("AVERAGE NUMBER OF STEPS OF ALL RESTARTS", avg_total)
    # display_steepest_ascent_config -The report for steepest ascent hill climb  with
sideways and without sideways move is displayed

    def display_steepest_ascent_config(self, result, first):

        sum_runs = result[0][0]

        succ = result[3][0]

        if succ:

            succ1=(succ/sum_runs)

            succ_rate = round((succ1)*100,2)

            success_steps = result[3][1]

            avearge_succ_steps = round(sum(success_steps)/succ, 2)

        else:

            succ_rate = success_steps = avearge_succ_steps = '-'

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

        if fail:

            fail1=(fail/sum_runs)

            fail_rate = round((fail1)*100,2)

            fail_steps = result[1][1]+result[2][1]

            fail_average_steps = round(sum(fail_steps)/fail,2)

        else:

            fail_rate = fail_steps = fail_average_steps = '-'

        num_flat_runs = result[1][0]

        print("\n\n"+first)

        border = ''

        l_f=len(first)

        for i in range(l_f): border+="="

        print(border)
        print("\n VALUE OF N: ", self.n_value, " (i.e ",self.n_value,"x",self.n_value,")")
        print("TOTAL NUMBER OF RUNS: ", sum_runs)
        print("\nSUCCESS RUNS: ", succ)
        print("SUCCESS RATES: ", succ_rate, "%")
        print("SUCCESS AVERAGE STEPS: ", avearge_succ_steps)
        print("\nNUMBER_OF_FAILURE RUNS: ", fail)
        print("FAILURE RATE: ", fail_rate, "%")
        print("FAILURE AVERAGE STEPS: ", fail_average_steps)
        print("\n\nFLAT LOCAL MINIMA: ", num_flat_runs)
        return
n_input = (int)(input("ENTER THE N VALUE GREATER THAN 3: "))
input_iterations = (int)(input("ENTER NUMBER OF ITERATIONS REQUIRED(MUST BE GREATER THAN OR EQUAL TO 1: "))
input_sideways = (int)(input("ENTER NUMBER OF MAXIMUM SIDEWAYS ALLOWED(MUST BE >=1): "))
if __name__ == "__main__":
    hill_climbing_analysis = Hill_Climbing_check(n_input, input_iterations, input_sideways)
    hill_climbing_analysis.explore()
```