



**Solving N-queens problem using hill-climbing search and its variants**

**PROJECT DOCUMENTATION REPORT**

**PROGRAMMING PROJECT 2**

**ITCS-6150 Intelligent Systems**

**Submitted To**

**Dewan T. Ahmed, Ph.D.**

**Submitted By**

**Devyani Barde**

**801208619**

## Table of Contents

1	PROBLEM FORMULATION .....	3
1.1	INTRODUCTION .....	3
1.1.1	What is N-Queens Problem?.....	3
1.1.2	What is Hill Climbing Approach?.....	3
1.2	ALGORITHM PSEUDOCODE .....	4
2	PROGRAM STRUCTURE .....	5
2.1	PROCEDURES .....	5
2.1.1	Language Used .....	5
2.1.2	Input .....	5
2.1.3	Approach .....	5
2.1.4	Output .....	5
2.2	CLASSES AND METHODS .....	5
2.2.1	Classes .....	5
2.2.2	Global Methods.....	7
2.3	GLOBAL VARIABLES .....	7
3	CODE .....	7
4	SAMPLE OUTPUT.....	20
4.1	HILL CLIMBING.....	20
4.2	HILL CLIMBING WITH SIDEWAYS MOVES.....	21
4.3	RANDOM RESTART SEARCH .....	22
4.3.1	Random Restart Search without Sideways Moves .....	22
4.3.2	Random Restart Search without Sideways Moves .....	23
	CONCLUSION.....	24
	REFERENCES.....	24

## 1 PROBLEM FORMULATION

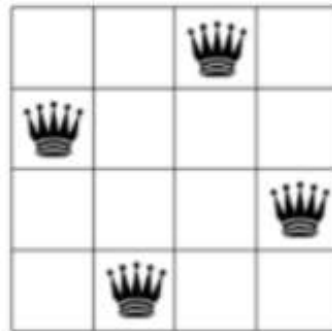
### 1.1 INTRODUCTION

#### 1.1.1 What is N-Queens Problem?

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. [1] The chess queens can attack in any direction as horizontal, vertical, horizontal and diagonal way. [1] That means no two queens can be placed in same row, same column or diagonal to each other.

The problem starts with randomly placing N number of queens on a NxN chess board. The problem is solved when all the queens are placed such that no two queens attack each other.

In this project, the N-queens problem will be solved using hill climbing approach. Following is the solution of a 4-queens problem-



#### 1.1.2 What is Hill Climbing Approach?

It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution. [2] Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value. [3]

Hill climbing is used when a good heuristic is available. In the case of N-queens problem, the heuristic used is number of pairs of queens attacking each other.

The variants of hill climbing search used in this project are-

- Hill climbing search
- Hill climbing search with sideways moves
- Random restart hill climbing search
  - Random restart hill climbing search without sideways moves
  - Random restart hill climbing search with sideways moves

## Solving N-queens problem using hill-climbing search and its variants

### 1.1.2.1 Hill Climbing Search

Simple hill climbing is the simplest way to implement a hill climbing algorithm. It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. [3]

Hill climbing search is less time consuming, but it gives less optimal solution and the solution is not guaranteed.

### 1.1.2.2 Hill Climbing Search with Sideways Moves

The algorithm halts if it reaches a plateau where the best successor has the same value as the current state. The algorithm allows sideways move in the hope that the plateau is a shoulder. An infinite loop may occur when the algorithm reaches a flat local that is not a shoulder. To avoid the infinite loop, the number of sideways moves allowed must be limited.

For example, if the number of consecutive sideways moves allowed are limited to 100, the percentage of problem instances solved by hill climbing raises from 14% to 94%.

### 1.1.2.3 Random Restart Hill Climbing Search

Random-restart hill-climbing conducts a series of hill-climbing searches from randomly generated initial states, running each until it halts or makes no discernible progress. [4] Random restart hill climbing overcomes local maxima.

The performance of random restart hill climbing search is improved even more by using sideways moves.

## 1.2 ALGORITHM PSEUDOCODE

**function** HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

**inputs:** *problem*, a problem

**local variables:** *current*, a node

*neighbor*, a node

*current*  $\leftarrow$  MAKE-NODE(INITIAL-STATE[*problem*])

**loop do**

*neighbor*  $\leftarrow$  a highest valued successor of *current*

**if** VALUE[*neighbor*]  $\leq$  VALUE[*current*] **then return** STATE[*current*]

*current*  $\leftarrow$  *neighbor*

**end**

## 2 PROGRAM STRUCTURE

### 2.1 PROCEDURES

#### 2.1.1 Language Used

The code is developed in python.

#### 2.1.2 Input

Puzzle Size: The user can input any value of N for N-queens problem.

#### 2.1.3 Approach

The user can choose from the following approaches to solve the N-queens problem

- Hill climbing search
- Hill climbing with sideways moves
- Random restart search (with and without sideways moves)

#### 2.1.4 Output

Output for hill climbing search and hill climbing with sideways moves has the following values-

- Total number of success
- Total number of fails
- Average of steps when it succeeds
- Average of steps when it fails
- Success rate
- Failure rate

Output for random restart has the following values-

- Number of random restarts
- Average number of random restarts
- Average number of steps

### 2.2 CLASSES AND METHODS

#### 2.2.1 Classes

I. **solve** – Contains the following methods

A. **\_\_init\_\_**

Objects are initialized. Creates a new board and calculates the number of iterations and heuristics.

B. **hillClimbing**

This method generates a successor with a better heuristic and calculates the target state; otherwise, the total steps are calculated. When no better successor is found, the run is marked as a failure. (There was no solution found)

**C. hillclimbingwithsideways**

Calls the method that returns a successor, which will determine the target state depending on the heuristic, or the total steps if the heuristic fails. It makes 100 sideways moves before marking the run as failed. (There was no solution found)

**D. randomRestart**

Performs a similar function, and if no better successor is found, the run will be marked as a failure, and the process will be restarted with a random initial state. For random restart with sideways moves same function is done with 100 sideways moves allowed.

**E. bestBoard**

The simple and restart without sideways step approaches use this form. It will look at all possible combinations and choose the best successor with the least amount of heuristic. If no best heuristic could be found, it will return the current heuristic value.

**F. Heuristic**

Calculate the heuristic by multiplying the number of attacks by the heuristic. Provides the final heuristic after calculating the horizontal, vertical, and diagonal attacks. For horizontal attacks, we pick each queen that is only found in one column and look for any other queens in the same row as the chosen queen. Any time a queen is found in the same row as the selected queen, the attacks are increased. Similarly, for each chosen queen, all four diagonal squares are tested for the presence of any contradicting queens, and the diagonal attacks are incremented for each queen found to be contradicting the chosen queen. The heuristic fun is calculated using the number of horizontal and diagonal attacks. As a heuristic tool, the number of horizontal and diagonal attacks is used.

**G. nextBoard**

Sideways and restart with sideways step approaches use this tool. It examines all possible combinations and selects the best successor with the fewest heuristics. If no best heuristic can be found, it can at least look for a successor heuristic and provide it. If none of the above are found, it will return the current heuristic value.

**H. printstats**

The outputs are printed depending on the method or variant chosen.

**II. randomboardgenerator** – Contains the following methods

**A. \_\_init\_\_**

The board object will be initialized. It will generate a new chess board at random.

## Solving N-queens problem using hill-climbing search and its variants

### 2.2.2 Global Methods

#### I. **printChessBoard**

The board will be printed with the proper column and row spacing.

### 2.3 GLOBAL VARIABLES

1. initialboard
2. sol
3. steps
4. randRes

## 3 CODE

```
import copy

import random

initialboard = None

sol = True

steps = 0

randRes = 0

class randomboardgenerator:

    def __init__(self, list=None):

        if list == None:

            self.randomboardgenerator = [["-" for i in range(0,n)] for j in range(0,n)]

            for j in range(0,n):

                rand_row = random.randint(0,n-1)

                if self.randomboardgenerator[rand_row][j] == "-":

                    self.randomboardgenerator[rand_row][j] = "Q"

            print("\nInitial placement of Queens on Chess Board:")

            printChessBoard(self.randomboardgenerator)
```

**#This method prints the chess Board**

## Solving N-queens problem using hill-climbing search and its variants

```
def printChessBoard(st):  
    for a in range(0,n):  
        for b in range(0,n):  
            if b < n-1:  
                print(st[a][b], end=" ")  
            elif(b == n-1):  
                print(st[a][b], end="\n")  
  
class solve:  
    def __init__(self, option, sol):  
        #intializing variables  
        self.Runs = 1000  
        self.Success = 0  
        self.Fails = 0  
        self.stepsSuccess = 0  
        self.stepsFail = 0  
        self.sideMoves = 0  
        self.sol = sol  
        if (option == 3):  
            print("1. Without side moves\n2. With side moves\n")  
            global x  
            x = int(input())  
            for i in range(0,1000):  
                if self.sol == True:  
                    print ("\n*****")
```



## Solving N-queens problem using hill-climbing search and its variants

```
print ("Run #",i+1)

print ("*****")

self.chessboard = randomboardgenerator(initialboard)

self.costh = self.Heuristic(self.chessboard)

#Calling the right variant of hill climbing

if (option == 1):

    self.hillClimbing()

elif (option == 2):

    self.hillclimbingwithsideways()

elif (option == 3):

    self.randomRestart()

#Simple Hill Climbing

def hillClimbing(self):

    totalSteps = 0

    while 1:

        curattacks = self.costh

        if self.costh == 0:

            break

        self.bestBoard()

        if (curattacks == self.costh):

            self.Fails += 1

            self.stepsFail += totalSteps

            if totalSteps == 0:

                self.stepsFail += 1
```

## Solving N-queens problem using hill-climbing search and its variants

```
        break

    totalSteps += 1

    if self.sol == True:

        print ("\nThe Number of attack pairs is", (int)(self.Heuristic(self.chessboard)))

        printChessBoard(self.chessboard.randomboardgenerator)

    if (self.costh == 0):

        break

    if self.costh != 0:

        if self.sol == True:

            print ("\n=====NO SOLUTION=====")

        else:

            if self.sol == True:

                print ("\n=====SOLUTION FOUND=====")

            self.Success += 1

            self.stepsSuccess += totalSteps

        return self.costh
```

### #Hill Climbing Search with sideways moves

```
def hillclimbingwithsideways(self):

    totalSteps = 0

    sideMoves = 0

    while 1:

        curattacks = self.costh

        curboard = self.chessboard

        if self.costh == 0:
```

## Solving N-queens problem using hill-climbing search and its variants

```
        break

    self.nextBoard()

    if curboard == self.chessboard:

        self.stepsFail += totalSteps

        self.Fails += 1

        if totalSteps == 0:

            self.stepsFail += 1

        break

    if curattacks == self.costh:

        sideMoves += 1

        if sideMoves == 100:

            self.stepsFail += totalSteps

            self.Fails += 1

            break

    elif(curattacks > self.costh):

        sideMoves = 0

    totalSteps += 1

    if self.sol == True:

        print ("\nThe Number of attack pairs is", (int)(self.Heuristic(self.chessboard)))

        printChessBoard(self.chessboard.randomboardgenerator)

    if self.costh == 0:

        break

    if self.costh != 0:

        if self.sol == True:
```

## Solving N-queens problem using hill-climbing search and its variants

```
        print ("\n=====NO SOLUTION=====")

    else:

        if self.sol == True:

            print ("\n=====SOLUTION FOUND=====")

            self.Success += 1

            self.stepsSuccess += totalSteps

        return self.costh
```

### **#Random Restart Hill Climbing Search**

```
def randomRestart(self):
```

```
    global randRes
```

```
    global steps
```

### **#Random Restart without sideways moves**

```
    if x == 1:
```

```
        while 1:
```

```
            curattacks = self.costh
```

```
            curboard = self.chessboard
```

```
            if self.costh == 0:
```

```
                break
```

```
            self.bestBoard()
```

```
            if (curboard == self.chessboard) or ((curattacks == self.costh) & (self.costh != 0)):
```

```
                self.chessboard = randomboardgenerator(initialboard)
```

```
                randRes += 1
```

```
                self.costh = self.Heuristic(self.chessboard)
```

```
            elif (self.costh < curattacks):
```

## Solving N-queens problem using hill-climbing search and its variants

```
        if self.sol == True:

            print ("\nAttack pairs:", (int)(self.Heuristic(self.chessboard)))

            printChessBoard(self.chessboard.randomboardgenerator)

        steps += 1

        if self.costh == 0:

            break

    if self.sol == True:

        print ("\n=====SOLUTION FOUND=====")

    self.Success += 1

    return self.costh
```

### **#Random Restart with sideways moves**

```
elif x == 2:

    sideMoves = 0

    while 1:

        curattacks = self.costh

        curboard = self.chessboard

        if self.costh == 0:

            break

        self.nextBoard()

        if curboard == self.chessboard:

            self.chessboard = randomboardgenerator(initialboard)

            randRes += 1

            self.costh = self.Heuristic(self.chessboard)

        if curattacks == self.costh:
```

## Solving N-queens problem using hill-climbing search and its variants

```
sideMoves += 1

if sideMoves == 100:

    self.chessboard = randomboardgenerator(initialboard)

    randRes += 1

    self.costh = self.Heuristic(self.chessboard)

elif (curattacks > self.costh):

    sideMoves = 0

steps += 1

if self.sol == True:

    print ("\nAttack pairs:", (int)(self.Heuristic(self.chessboard)))

    printChessBoard(self.chessboard.randomboardgenerator)

if self.costh == 0:

    break

if self.sol == True:

    print("\n=====SOLUTION FOUND=====")

self.Success += 1

return self.costh
```

**#This function tries to shift each queen to each position with just one move and returns the move with the fewest attack pairs.**

```
def bestBoard(self):

    mincost = self.Heuristic(self.chessboard)

    best_board = self.chessboard

    for acol in range(0,n):

        for arow in range(0,n):

            if self.chessboard.randomboardgenerator[arow][acol] == "Q":
```

## Solving N-queens problem using hill-climbing search and its variants

```
for brow in range(0,n):  
    if self.chessboard.randomboardgenerator[brow][acol] != "Q":  
        temp = copy.deepcopy(self.chessboard)  
        temp.randomboardgenerator[arow][acol] = "-"  
        temp.randomboardgenerator[brow][acol] = "Q"  
        tempcost = self.Heuristic(temp)  
        if tempcost < mincost:  
            mincost = tempcost  
            best_board = temp  
self.chessboard = best_board  
self.costh = mincost
```

### **#Finds the number of attack pairs**

```
def Heuristic(self, tempb):  
    straight = 0  
    diagonal = 0  
    for i in range(0,n):  
        for j in range(0,n):  
            if tempb.randomboardgenerator[i][j] == "Q":  
                straight -= 2  
                for k in range(0,n):  
                    if tempb.randomboardgenerator[i][k] == "Q":  
                        straight += 1  
                    if tempb.randomboardgenerator[k][j] == "Q":  
                        straight += 1
```

## Solving N-queens problem using hill-climbing search and its variants

```
k, l = i+1, j+1
```

```
while k < n and l < n:
```

```
    if tempb.randomboardgenerator[k][l] == "Q":
```

```
        diagonal += 1
```

```
    k +=1
```

```
    l +=1
```

```
k, l = i+1, j-1
```

```
while k < n and l >= 0:
```

```
    if tempb.randomboardgenerator[k][l] == "Q":
```

```
        diagonal += 1
```

```
    k +=1
```

```
    l -=1
```

```
k, l = i-1, j+1
```

```
while k >= 0 and l < n:
```

```
    if tempb.randomboardgenerator[k][l] == "Q":
```

```
        diagonal += 1
```

```
    k -=1
```

```
    l +=1
```

```
k, l = i-1, j-1
```

```
while k >= 0 and l >= 0:
```

```
    if tempb.randomboardgenerator[k][l] == "Q":
```

```
        diagonal += 1
```

```
    k -=1
```

```
    l -=1
```



## Solving N-queens problem using hill-climbing search and its variants

```
return ((diagonal + straight)/2)
```

### **#Finds the successor board**

```
def nextBoard(self):
```

```
    count = 0
```

```
    que = {}
```

```
    currentcostt = self.Heuristic(self.chessboard)
```

```
    mincost = self.Heuristic(self.chessboard)
```

```
    best_board = self.chessboard
```

```
    for q_col in range(0,n):
```

```
        for q_row in range(0,n):
```

```
            if self.chessboard.randomboardgenerator[q_row][q_col] == "Q":
```

```
                for m_row in range(0,n):
```

```
                    if self.chessboard.randomboardgenerator[m_row][q_col] != "Q":
```

```
                        test_board = copy.deepcopy(self.chessboard)
```

```
                        test_board.randomboardgenerator[q_row][q_col] = "-"
```

```
                        test_board.randomboardgenerator[m_row][q_col] = "Q"
```

```
                        test_board_cost = self.Heuristic(test_board)
```

```
                        if test_board_cost < mincost:
```

```
                            mincost = test_board_cost
```

```
                            best_board = test_board
```

```
                        if test_board_cost == currentcostt:
```

```
                            que[count] = test_board
```

```
                            count += 1
```

```
    if mincost == currentcostt:
```

## Solving N-queens problem using hill-climbing search and its variants

```
print("Successors with same heuristic value:", count)

if(count == 1):

    best_board = que[0]

elif(count > 1):

    rand_ind = random.randint(0,count - 1)

    print("Successors with same heuristic value:", rand_ind)

    best_board = que[rand_ind]

self.chessboard = best_board

self.cost = mincost

#Prints the output

def printstats(self):

    if (option == 1) or (option == 2):

        print("Total number of Success: ", self.Success)

        print("Total number of Fails: ", self.Fails)

        print("Average of steps when it succeeds: ",self.stepsSuccess/self.Success)

        print("Average of steps when it fails: ", self.stepsFail/self.Fails)

        print("Success Rate: ", (self.Success/self.Runs)*100, "%")

        print("Failure Rate: ", (self.Fails/self.Runs)*100, "%")

    else:

        print("Number of Random Restarts: ", randRes)

        print("Average number of random resarts: ", randRes/self.Runs)

        print("Average number of steps: ", steps/self.Runs)

print("Enter number of queens: ")

n = int(input())
```

## Solving N-queens problem using hill-climbing search and its variants

```
print("\nSelect the Search Strategy:\n1. Hill Climbing Search\n2. Hill Climbing Search with  
sideways moves\n3. Random restart hill climbing search\n")
```

```
option = int(input())
```

```
chessboard = solve(option, sol)
```

```
chessboard.printstats()
```

## 4 SAMPLE OUTPUT

### 4.1 HILL CLIMBING

```

1  #-*- coding: utf-8 -*-
2  """
3  Created on Wed Apr 21 14:22:07 2021
4
5  @author: Devyani Barde - 801208619
6  """
7
8  import copy
9  import random
10
11  initialboard = None
12  sol = True
13  steps = 0
14  randRes = 0
15
16  class randomboardgenerator:
17      def __init__(self, list=None):
18          if list == None:
19              self.randomboardgenerator = [{"-" for i in range(0,n)} for j in range(0,n)]
20              for j in range(0,n):
21                  rand_row = random.randint(0,n-1)
22                  if self.randomboardgenerator[rand_row][j] == "-":
23                      self.randomboardgenerator[rand_row][j] = "Q"
24              print("\nInitial placement of Queens on Chess Board:")
25              printChessBoard(self.randomboardgenerator)
26
27      #This method prints the chess Board
28      def printChessBoard(st):
29          for a in range(0,n):
30              for b in range(0,n):
31                  if b < n-1:
32                      print(st[a][b], end=" ")
33                  elif(b == n-1):
34                      print(st[a][b], end="\n")
35
36

```

Console 2/A:

```

Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/DELL/OneDrive/Desktop/Spring \'21/Intelligent Systems/Project2/Project2.py', wdir='C:/Users/DELL/OneDrive/Desktop/Spring \'21/Intelligent Systems/Project2')
Enter number of queens:

4

Select the Search Strategy:
1. Hill Climbing Search
2. Hill Climbing Search with sideways moves
3. Random restart hill climbing search

1

```

```

1  #-*- coding: utf-8 -*-
2  """
3  Created on Wed Apr 21 14:22:07 2021
4
5  @author: Devyani Barde - 801208619
6  """
7
8  import copy
9  import random
10
11  initialboard = None
12  sol = True
13  steps = 0
14  randRes = 0
15
16  class randomboardgenerator:
17      def __init__(self, list=None):
18          if list == None:
19              self.randomboardgenerator = [{"-" for i in range(0,n)} for j in range(0,n)]
20              for j in range(0,n):
21                  rand_row = random.randint(0,n-1)
22                  if self.randomboardgenerator[rand_row][j] == "-":
23                      self.randomboardgenerator[rand_row][j] = "Q"
24              print("\nInitial placement of Queens on Chess Board:")
25              printChessBoard(self.randomboardgenerator)
26
27      #This method prints the chess Board
28      def printChessBoard(st):
29          for a in range(0,n):
30              for b in range(0,n):
31                  if b < n-1:
32                      print(st[a][b], end=" ")
33                  elif(b == n-1):
34                      print(st[a][b], end="\n")
35
36

```

Console 2/A:

```

- - - Q
Q - -
- - Q

=====SOLUTION FOUND=====
+-----+
Run # 1000
+-----+

Initial placement of Queens on Chess Board:
Q - Q -
- - -
- Q -
- - Q

The Number of attack pairs is 1
- - Q -
- - -
- Q -
- - Q

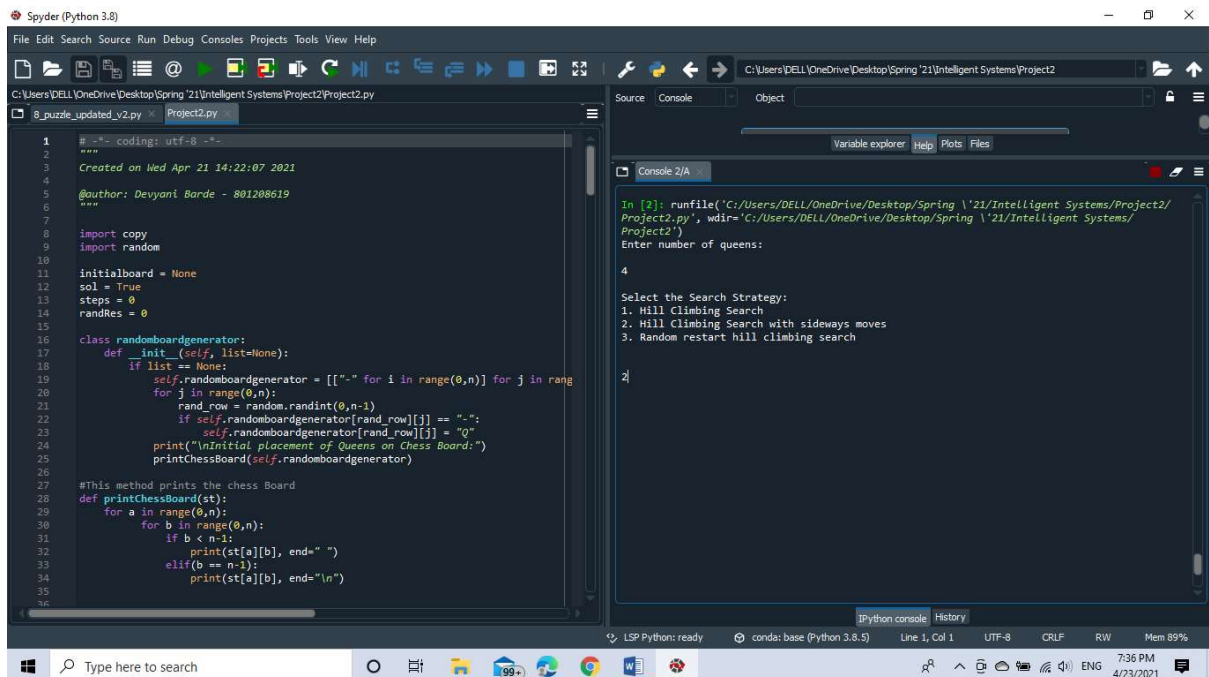
=====NO SOLUTION=====
Total number of Success: 408
Total number of Fails: 592
Average of steps when it succeeds: 1.9248196678431373
Average of steps when it fails: 1.3868243243243243
Success Rate: 40.8 %
Failure Rate: 59.199999999999996 %

In [2]:

```

# Solving N-queens problem using hill-climbing search and its variants

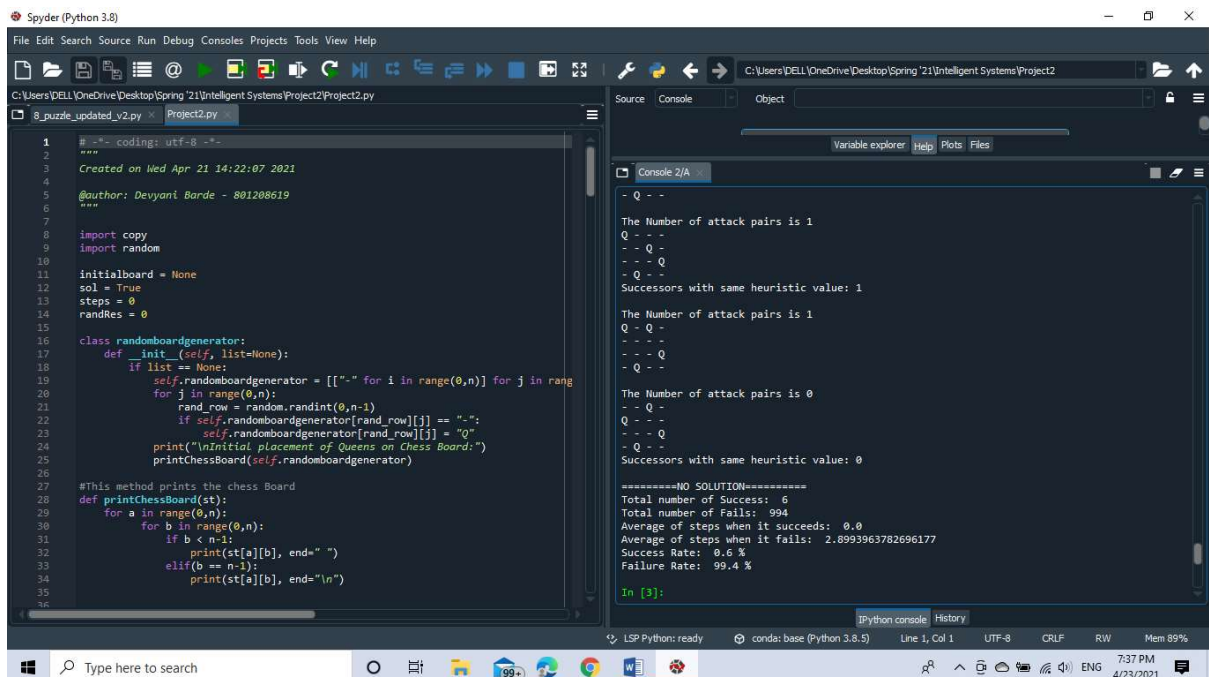
## 4.2 HILL CLIMBING WITH SIDEWAYS MOVES



```
1 #-*- coding: utf-8 -*-
2 """
3 Created on Wed Apr 21 14:22:07 2021
4
5 @author: Devyani Barde - 801208619
6 """
7
8 import copy
9 import random
10
11 initialboard = None
12 sol = True
13 steps = 0
14 randRes = 0
15
16 class randomboardgenerator:
17     def __init__(self, list=None):
18         if list == None:
19             self.randomboardgenerator = [[ "-" for i in range(0,n)] for j in range
20             for j in range(0,n):
21                 rand_row = random.randint(0,n-1)
22                 if self.randomboardgenerator[rand_row][j] == "-":
23                     self.randomboardgenerator[rand_row][j] = "Q"
24             print("\nInitial placement of Queens on Chess Board:")
25             printChessBoard(self.randomboardgenerator)
26
27 #This method prints the chess Board
28 def printChessBoard(st):
29     for a in range(0,n):
30         for b in range(0,n):
31             if b < n-1:
32                 print(st[a][b], end=" ")
33             elif(b == n-1):
34                 print(st[a][b], end="\n")
35
36
```

Console 2/A

```
In [2]: runfile('C:/Users/DELL/OneDrive/Desktop/Spring \'21/Intelligent Systems/Project2/
Project2.py', wdir='C:/Users/DELL/OneDrive/Desktop/Spring \'21/Intelligent Systems/
Project2')
Enter number of queens:
2
Select the Search Strategy:
1. Hill Climbing Search
2. Hill Climbing Search with sideways moves
3. Random restart hill climbing search
```



```
1 #-*- coding: utf-8 -*-
2 """
3 Created on Wed Apr 21 14:22:07 2021
4
5 @author: Devyani Barde - 801208619
6 """
7
8 import copy
9 import random
10
11 initialboard = None
12 sol = True
13 steps = 0
14 randRes = 0
15
16 class randomboardgenerator:
17     def __init__(self, list=None):
18         if list == None:
19             self.randomboardgenerator = [[ "-" for i in range(0,n)] for j in range
20             for j in range(0,n):
21                 rand_row = random.randint(0,n-1)
22                 if self.randomboardgenerator[rand_row][j] == "-":
23                     self.randomboardgenerator[rand_row][j] = "Q"
24             print("\nInitial placement of Queens on Chess Board:")
25             printChessBoard(self.randomboardgenerator)
26
27 #This method prints the chess Board
28 def printChessBoard(st):
29     for a in range(0,n):
30         for b in range(0,n):
31             if b < n-1:
32                 print(st[a][b], end=" ")
33             elif(b == n-1):
34                 print(st[a][b], end="\n")
35
36
```

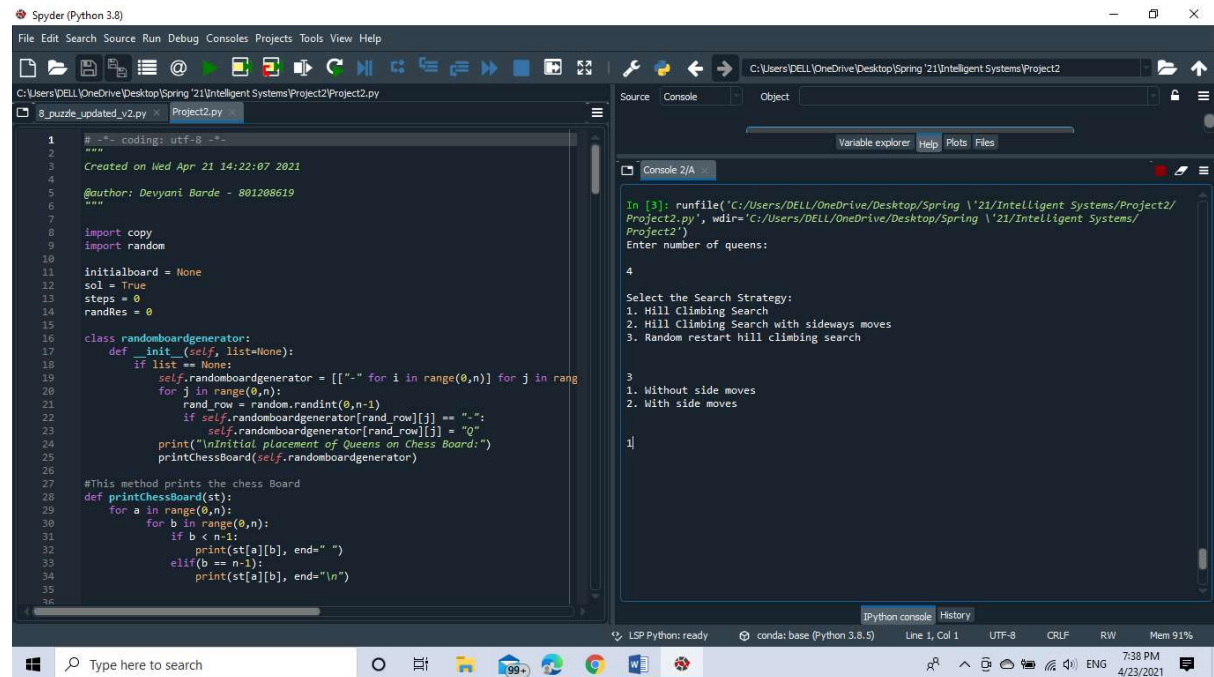
Console 2/A

```
- Q - -
The Number of attack pairs is 1
Q - - -
- - Q -
- - - Q
- Q - -
Successors with same heuristic value: 1
The Number of attack pairs is 1
Q - Q -
- - - -
- - - Q
- Q - -
The Number of attack pairs is 0
Q - Q -
- - - Q
- - - Q
Successors with same heuristic value: 0
=====NO SOLUTION=====
Total number of Success: 6
Total number of Fails: 994
Average of steps when it succeeds: 0.0
Average of steps when it fails: 2.8993963782696177
Success Rate: 0.6 %
Failure Rate: 99.4 %
In [3]:
```

# Solving N-queens problem using hill-climbing search and its variants

## 4.3 RANDOM RESTART SEARCH

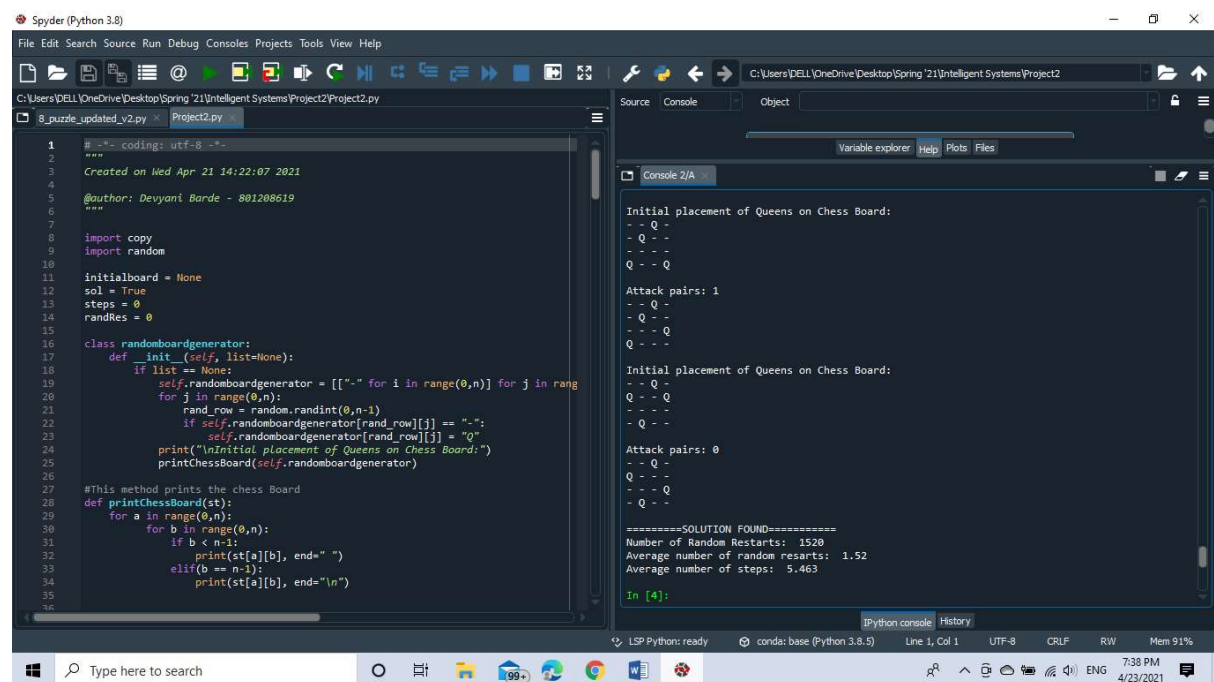
### 4.3.1 Random Restart Search without Sideways Moves



```
1 #-*- coding: utf-8 -*-
2 """
3 Created on Wed Apr 21 14:22:07 2021
4
5 @author: Devyani Barde - 801208619
6 """
7
8 import copy
9 import random
10
11 initialboard = None
12 sol = True
13 steps = 0
14 randRes = 0
15
16 class randomboardgenerator:
17     def __init__(self, list=None):
18         if list == None:
19             self.randomboardgenerator = [[ "-" for i in range(0,n)] for j in range(0,n)]
20             for j in range(0,n):
21                 rand_row = random.randint(0,n-1)
22                 if self.randomboardgenerator[rand_row][j] == "-":
23                     self.randomboardgenerator[rand_row][j] = "Q"
24             print("Initial placement of Queens on Chess Board:")
25             printChessBoard(self.randomboardgenerator)
26
27 #This method prints the chess Board
28 def printChessBoard(st):
29     for a in range(0,n):
30         for b in range(0,n):
31             if b < n-1:
32                 print(st[a][b], end=" ")
33             elif(b == n-1):
34                 print(st[a][b], end="\n")
35
36
```

Console 2/A

```
In [3]: runfile('C:/Users/DELL/OneDrive/Desktop/Spring \'21/Intelligent Systems/Project2/Project2.py', wdir='C:/Users/DELL/OneDrive/Desktop/Spring \'21/Intelligent Systems/Project2')
Enter number of queens:
4
Select the Search Strategy:
1. Hill Climbing Search
2. Hill Climbing Search with sideways moves
3. Random restart hill climbing search
3
1. Without side moves
2. With side moves
1
```



```
1 #-*- coding: utf-8 -*-
2 """
3 Created on Wed Apr 21 14:22:07 2021
4
5 @author: Devyani Barde - 801208619
6 """
7
8 import copy
9 import random
10
11 initialboard = None
12 sol = True
13 steps = 0
14 randRes = 0
15
16 class randomboardgenerator:
17     def __init__(self, list=None):
18         if list == None:
19             self.randomboardgenerator = [[ "-" for i in range(0,n)] for j in range(0,n)]
20             for j in range(0,n):
21                 rand_row = random.randint(0,n-1)
22                 if self.randomboardgenerator[rand_row][j] == "-":
23                     self.randomboardgenerator[rand_row][j] = "Q"
24             print("Initial placement of Queens on Chess Board:")
25             printChessBoard(self.randomboardgenerator)
26
27 #This method prints the chess Board
28 def printChessBoard(st):
29     for a in range(0,n):
30         for b in range(0,n):
31             if b < n-1:
32                 print(st[a][b], end=" ")
33             elif(b == n-1):
34                 print(st[a][b], end="\n")
35
36
```

Console 2/A

```
Initial placement of Queens on Chess Board:
- - Q -
-Q - -
- - - -
Q - - -

Attack pairs: 1
- - Q -
-Q - -
- - - -
Q - - -

Initial placement of Queens on Chess Board:
- - Q -
Q - - Q
- - - -
- - - -

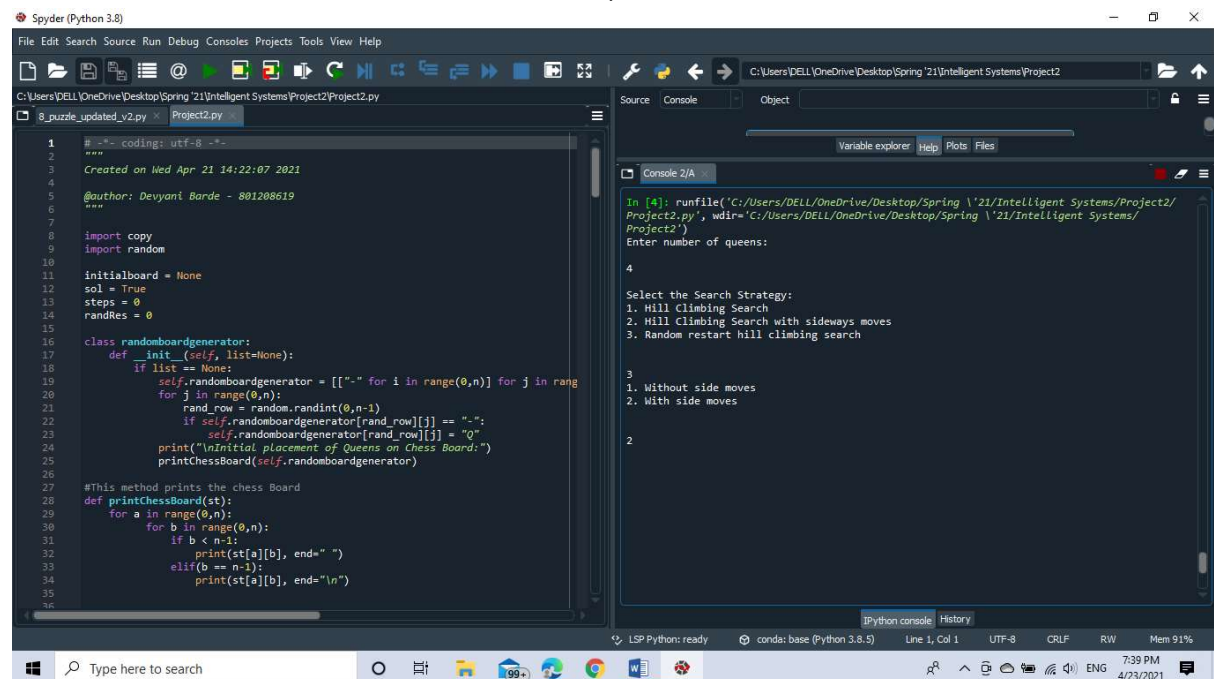
Attack pairs: 0
- - Q -
Q - - -
- - - -
- - - -

=====SOLUTION FOUND=====
Number of Random Restarts: 1520
Average number of random restarts: 1.52
Average number of steps: 5.463

In [4]:
```

# Solving N-queens problem using hill-climbing search and its variants

## 4.3.2 Random Restart Search without Sideways Moves



```
1  -*- coding: utf-8 -*-
2  """
3  Created on Wed Apr 21 14:22:07 2021
4
5  @author: Devyani Barde - 801208619
6  """
7
8  import copy
9  import random
10
11  initialboard = None
12  sol = True
13  steps = 0
14  randRes = 0
15
16  class randomboardgenerator:
17      def __init__(self, list=None):
18          if list == None:
19              self.randomboardgenerator = [[ "-" for i in range(0,n)] for j in range(0,n)]
20              for j in range(0,n):
21                  rand_row = random.randint(0,n-1)
22                  if self.randomboardgenerator[rand_row][j] == "-":
23                      self.randomboardgenerator[rand_row][j] = "Q"
24              print("\nInitial placement of Queens on Chess Board:")
25              printChessBoard(self.randomboardgenerator)
26
27      #This method prints the chess Board
28      def printChessBoard(st):
29          for a in range(0,n):
30              for b in range(0,n):
31                  if b < n-1:
32                      print(st[a][b], end=" ")
33                  elif(b == n-1):
34                      print(st[a][b], end="\n")
35
36
```

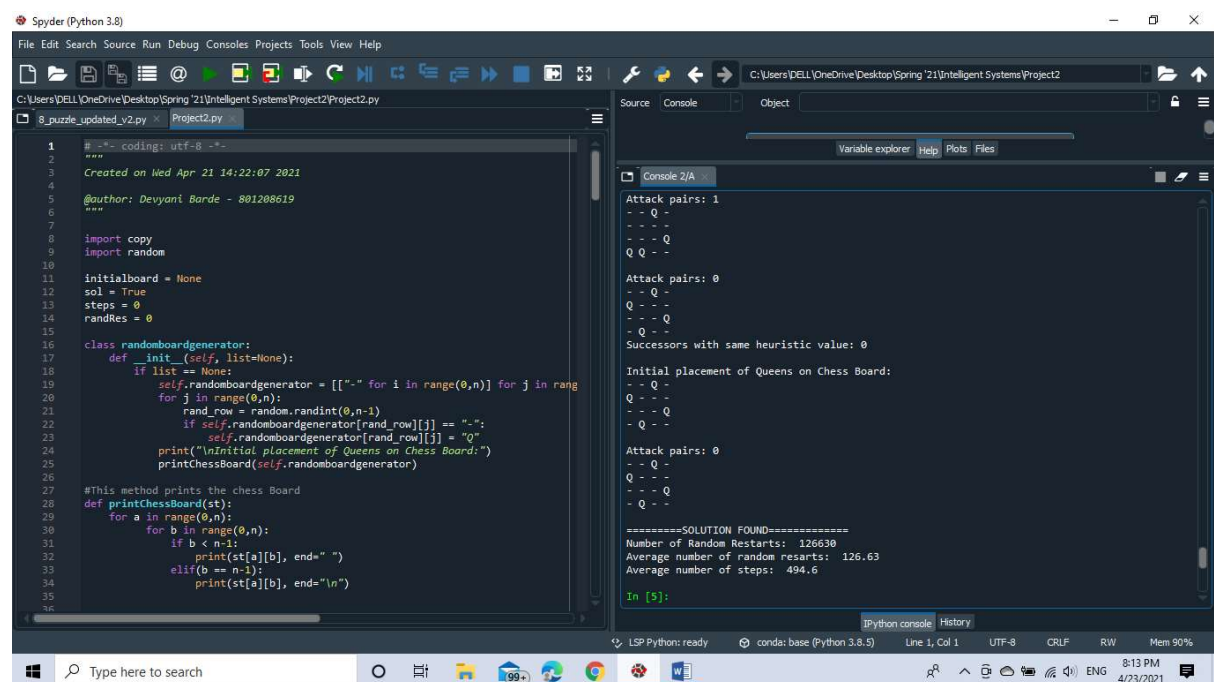
Console 2/A

```
In [4]: runfile('C:/Users/DELL/OneDrive/Desktop/Spring \'21/Intelligent Systems/Project2/Project2.py', wdir='C:/Users/DELL/OneDrive/Desktop/Spring \'21/Intelligent Systems/Project2')
Enter number of queens:
4

Select the Search Strategy:
1. Hill Climbing Search
2. Hill Climbing Search with sideways moves
3. Random restart hill climbing search

3
1. Without side moves
2. With side moves

2
```



```
1  -*- coding: utf-8 -*-
2  """
3  Created on Wed Apr 21 14:22:07 2021
4
5  @author: Devyani Barde - 801208619
6  """
7
8  import copy
9  import random
10
11  initialboard = None
12  sol = True
13  steps = 0
14  randRes = 0
15
16  class randomboardgenerator:
17      def __init__(self, list=None):
18          if list == None:
19              self.randomboardgenerator = [[ "-" for i in range(0,n)] for j in range(0,n)]
20              for j in range(0,n):
21                  rand_row = random.randint(0,n-1)
22                  if self.randomboardgenerator[rand_row][j] == "-":
23                      self.randomboardgenerator[rand_row][j] = "Q"
24              print("\nInitial placement of Queens on Chess Board:")
25              printChessBoard(self.randomboardgenerator)
26
27      #This method prints the chess Board
28      def printChessBoard(st):
29          for a in range(0,n):
30              for b in range(0,n):
31                  if b < n-1:
32                      print(st[a][b], end=" ")
33                  elif(b == n-1):
34                      print(st[a][b], end="\n")
35
36
```

Console 2/A

```
Attack pairs: 1
- - Q -
- - - -
- - - Q
Q Q - -

Attack pairs: 0
- - Q -
Q - - -
- - - Q
- Q - -

Successors with same heuristic value: 0

Initial placement of Queens on Chess Board:
- - Q -
Q - - -
- - - Q
- Q - -

Attack pairs: 0
- - Q -
Q - - -
- - - Q
- Q - -

=====SOLUTION FOUND=====
Number of Random Restarts: 126630
Average number of random resarts: 126.63
Average number of steps: 494.6

In [5]:
```



## CONCLUSION

By implementation of this project, we learnt that hill climbing approach for solving N-queens problem can get stuck on a shoulder and fail to find the solution. Therefore, hill climbing with sideways moves is used to overcome shoulder. Hill climbing approach with and without sideways moves can still fail to find the solution and get stuck on local maxima, to overcome this problem random restart hill climbing is used.

## REFERENCES

- [1] "N-Queen-Problem," [Online]. Available: <https://www.tutorialspoint.com/N-Queen-Problem>.  
[Accessed 22 April 2021].
- [2] "Wikipedia," [Online]. Available: [https://en.wikipedia.org/wiki/Hill\\_climbing](https://en.wikipedia.org/wiki/Hill_climbing).  
[Accessed 04 April 2021].
- [3] "JavaTPoint," [Online]. Available: <https://www.javatpoint.com/hill-climbing-algorithm-in-ai>.  
[Accessed 04 April 2021].
- [4] "ArtificialIntelligence,"[Online].Available:[https://en.wikibooks.org/wiki/Artificial\\_Intelligence/Search/Iterative\\_Improvement/Hill\\_Climbing](https://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Iterative_Improvement/Hill_Climbing). [Accessed 23 April 2021].