CHARLOTTE

PROJECT DOCUMENTATION REPORT
ON
Solving N-Queens Problem by Hill-Climbing and its variants

PROJECT 2

ITCS 6150 - Intelligent Systems

DEPARTMENT OF COMPUTER SCIENCE
SUBMITTED TO
Dewan T. Ahmed, Ph.D.

SUBMITTED BY:

Anoosh Guddehithlu Prathap Kumar **801200789**
Sharat Sindoor **801203539**

Table of Contents

PROBLEM FORMULATION

INTRODUCTION

What is N-Queen Problem?

The N-Queen problem was introduced by Carl Gauss in 1850. The goal of this problem is to place N queens that can take each other. Queens can move in three directions vertical, horizontal and diagonal, which means, there can be only one queen per row and one per column, and two queens cannot find themselves on the same diagonal.

It is a computationally expensive problem – NP complete, which makes it very popular problem in computer science.

Hill Climbing Search

In Hill-Climbing search method, we start at the base of the hill and walk uphill until we reach the top of the hill. In other words, we start with initial state and we keep improving the solution until it is optimum.

It's a variation of a generate-and-test algorithm which discards all states which do not look capable or seem questionable to lead us to the goal state. To take such decisions, it uses heuristics (an evaluation function) which indicates how nearby the current state is to the goal state.

Hill Climbing with Sideways move

The hill climbing search algorithms stops if it reaches plateau where the best successor has same value as the current state.

Then we allow algorithm to move sideways in the hope that the plateau is actually is a shoulder.

The algorithm will go in an infinite loop if the algorithm reaches to a flat local maximum and that is not a shoulder.

The only solution of it is to put limit on the number of consecutive sideways move allowed.

Random Restart Hill Climbing Without Sideways Move

Random restart hill climbing conducts series of hill climbing searches from randomly generated initial states, until a goal state is found.

The probability of this technique approaches 1, because sometimes it will eventually generate 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.

For 8-queens instances with no sideways moves allowed, $p \approx 0.14$, so we need roughly 7 iterations to find a goal (6 failures and 1 success). Roughly 22 steps in all.

Random Restart Hill Climbing With Sideways Move

Random restart hill climbing conducts series of hill climbing searches from randomly generated initial states, until a goal state is found.

The probability of this technique approaches 1, because sometimes it will eventually generate goal state as initial state.

If each hill climbing search has a probability p of success, then the expected number of restarts required is 1/p.

For 8-queens instances, when we allow sideways moves, $1/0.94 \sim 1.06$ iterations are needed on average and $(1 * 21) + (0.06/0.94) * 64 \sim 25$ steps.

EXPECTED OUTPUT

Hill climbing

Success rate is: 40.4 % and Failure rate is: 59.6 %

The average number of steps when the algorithm succeeds: 2.02

The average number of steps when the algorithm fails: 2.27

Hill climbing with sideways

Success rate is: 100.0 % and Failure rate is: 0.0 %

The average number of steps when the algorithm succeeds: 2.77

Random restart without sideways

The average number of random restarts required without sideways move 1.22

The average number of steps required without sideways move 4.75

Random with sideways

The average number of random restarts required with sideways move 0.0

The average number of steps required with sideways move 2.79

PROGRAM STRUCTURE

GLOBAL / LOCAL VARIABLES

We use these various global/local variables in our program:

- rr and ri : Random range and Random Integer.
- uu, vv and ww: variables for the range
- n: Number of Queens.
- ran_row: variable that stores index-row.
- ran_col: variable that stores index-column.
- state_s: Current state.
- the_board: Actual N queens board.
- expense and lowest_expense: total cost and minimum cost.
- iteration_number: Total number o iterations/count.
- step_number: Total number of steps.
- no_of_restarts
- successful_steps_count
- failure_step_count
- successful_iterations_count
- fail_iterations_count
- final_restarting_count and many more.

FUNCTIONS / PROCEDURES

Our code implements n queens problem using Hill Climbing local search method and its variants – Hill Climbing, Hill Climbing with Sideways move and Random restart Hill Climbing using Python.

We use the following functions within the code:

- __init__(): Initialization.
- display_state(): This function prints state of N cross N board.
- eval_h_values(): Used to evaluate heuristic values.
- eval_state_min(): Used to calculate minimum cost.

- simple_hill_climbing_algo(): This Function runs the hill climbing algorithm.
- hill_climbing_algo_with_sideways(): This function is for hill climbing using sideways movement.
- hill_climbing_algo_random_restart(): This Function runs the hill climbing algorithm with random restart without sideways movement.
- random_restart_hill_climbing_algo_with_sideways(): This Function runs the hill climbing algorithm with sideways movement.
- main(): The main function running the Hill climbing algorithm for N-queens solution.

LOGIC

We are running the code in python 3 in Anaconda package Spyder. We first generate a random board with n queens in it. The functions row collisions and column collisions count the number of attacks and keep a track of it. The movements are made as per the hill climbing technique used (Hill Climbing or Sideways moves) and collisions are compared and accordingly the next step is taken. With random restart technique, on failure, again a random board gets generated and it starts looking for a solution, hence a solution is guaranteed. We print the steps, success failure rates at the end. For Sideways moves technique, we put a limitation over sideways moves to stop it from entering an infinite loop.

Commenting the print of the sequences as there are minimum 20 moves on success and 64 moves on failure in Sideway moves, might let the system to crash.

Calculation of average steps for success – Total number of success moves/ Total number of success

Calculation of average steps for failure – Total number of failure moves/ Total number of failure

Success/Failure rate = (Success/Failure)/Total * 100

PROGRAM IMPLEMENTATION

Source Code:

Importing Packages and Libraries:

```
import copy
from random import randint as ri
from random import randrange as rr

class state_board():
    def __init__(self, n):
        self.n = n
        self.state_board= [[0 for uu in range (0,self.n)] for vv in range (0,self.n)]
        for uu in range(0, n):
            while 1:
                ran_row = ri(0,n-1)
                ran_col = uu
                #insert queens in different columns
                if self.state_board[ran_row][ran_col] == 0:
                    self.state_board[ran_row][ran_col] = 'Q'
                    break

#This function prints state of N cross N board
def display_state(state_s):
        for uu in range(n):
            table = ""
            for vv in range(n):
                table += str(state_s[uu][vv])+" "
            print(table)
        print("")

def eval_h_values(the_board,n):
    expense = 0
    for uu in range(0,n):
        for vv in range(0,n):
            if the_board.state_board[uu][vv]=='Q':
                #checking in rows
                for ww in range(vv+1,n):
                    if the_board.state_board[uu][ww] == 'Q' :
                        expense+=1
                uuu,vvv=uu+1,vv+1
                #checking the diagonals
                while (uuu<n and vvv<n):
                    if the_board.state_board[uuu][vvv]  == 'Q':
                        expense+=1
                    uuu=uuu+1
```

```
                vvv=vvv+1
            uuu,vvv=uu-1,vv+1
            while (uuu>=0 and vvv<n):
                if the_board.state_board[uuu][vvv] == 'Q':
                    expense+=1
                uuu=uuu-1
                vvv=vvv+1
    return expense

def eval_state_min(the_board,n):
    temp_list = []
    lowest_expense=eval_h_values(the_board,n)
    for uu in range(0,n):
        for vv in range(0,n):
            if the_board.state_board[vv][uu]=='Q':
                #Trying various arrangements by shifting the queen from the column
                for uuu in range(0,n):
                        if the_board.state_board[uuu][uu]!='Q' :
                            next_board_state = copy.deepcopy(the_board)
                            next_board_state.state_board[vv][uu]=0
                            next_board_state.state_board[uuu][uu]='Q'
                            expense=eval_h_values(next_board_state, n)
                            if expense < lowest_expense:
                                temp_list.clear()
                                lowest_expense = expense
                                temp_list.append([uuu,uu])
                            elif expense == lowest_expense:
                                temp_list.append([uuu,uu])

    return temp_list,lowest_expense

# This Function runs the hill climbing algorithm
def simple_hill_climbing_algo(state_board,iteration_number):
    step_number=0
    valuation = eval_h_values(state_board,n)
    if (iteration_number < 4):
        print('\nThe searching sequence for random configuration: ', iteration_number + 1)
        iteration_count = 0
    while 1:
        if (iteration_number < 4):
            display_state(state_board.state_board)
            iteration_count += 1
        if valuation == 0:
            break
        else:
            step_number += 1
            temp_list,expense = eval_state_min(state_board,n)
```

```
        if valuation <= expense or len(temp_list) == 0:
            break
        else:
            ran_indexing = rr(0,len(temp_list))
            index = temp_list[ran_indexing]
            valuation = expense
            for uu in range (0,n):
                    state_board.state_board[uu][index[1]]=0
            state_board.state_board[index[0]][index[1]]='Q'

    if (iteration_number < 4):
        #Printing if calculation is a failure or success
        if valuation == 0:
            print("Success")
        else:
            print("Failure")
        print('Number of steps: ',iteration_count-1)
        print('~~~~~~~~~~~~~~~~~~~~~')
    if valuation == 0:
        return 1, step_number
    return 0, step_number

# This function is for hill climbing using sideways movement
def hill_climbing_algo_with_sideways(the_board, iteration_number):
    step_number = 0
    side_ways_count=0
    bs = the_board
    cost_current = eval_h_values(bs, n)
    if (iteration_number < 4):
        print('\nThe searching sequence for random configuration: ', iteration_number + 1)
        iteration_count = 0
    while step_number<cent:
        if (iteration_number < 4):
            display_state(bs.state_board)
            iteration_count += 1
        if cost_current == 0:
            break
        else:
            step_number += 1
            temp_list, expense = eval_state_min(bs, n)
            if cost_current < expense:
                break
            if len(temp_list) == 0:
                break
            else:
                if cost_current == expense:
                    side_ways_count+=1
```

```
            else:
                side_ways_count=0
            ran_indexing  = rr(0, len(temp_list))
            index = temp_list[ran_indexing]
            cost_current = expense
            for uu in range(0, n):
                bs.state_board[uu][index[1]]  = 0
            bs.state_board[index[0]][index[1]]  = 'Q'
```
#printing if calculation is a failure or success
```
    if (iteration_number  < 4):
        if cost_current == 0:
            print("Success")
        else:
            print("Failure")
        print('Number  of steps: ',iteration_count-1)
        print('~~~~~~~~~~~~~~~~~~~~')
    if cost_current == 0:
        return 0, step_number
    return 1, step_number
```

 # This Function runs the hill climbing algorithm with random restart without sideways movement
```
def hill_climbing_algo_random_restart(the_board):

    no_of_restarts=0
    step_number=0
    bs= the_board
    previous_h_val = eval_h_values(bs, n)
    while 1:
        if previous_h_val == 0:
            break
        else:
            step_number += 1
            temp_list,h_val  = eval_state_min(bs,n)
            if previous_h_val <= h_val or len(temp_list)  == 0:
                no_of_restarts += 1
                bs = state_board(n)
                previous_h_val = eval_h_values(bs, n)
                continue

            ran_indexing  = rr(0,len(temp_list))
            index = temp_list[ran_indexing]
            previous_h_val = h_val
            for uu in range (0,n):
                    bs.state_board[uu][index[1]]=0
            bs.state_board[index[0]][index[1]]='Q'
```

```
    if previous_h_val == 0:
        return 0, step_number, no_of_restarts
    return 1, step_number, no_of_restarts
```

 # This Function runs the hill climbing algorithm with sideways movement

```
def random_restart_hill_climbing_algo_with_sideways(the_board):
    step_number = 0
    side_ways_count=0
    no_of_restarts = 0
    bs = the_board
    previous_h_val = eval_h_values(bs, n)
    while 1:
        if previous_h_val == 0:
            break
        else:
            step_number += 1
            temp_list, h_val = eval_state_min(bs, n)
            if previous_h_val < h_val or len(temp_list) == 0:
                bs=state_board(n)
                previous_h_val = eval_h_values(bs, n)
                no_of_restarts += 1
                side_ways_count=0
                continue

            if previous_h_val == h_val:
                side_ways_count+=1
                if step_number >= cent:
                    bs=state_board(n)
                    previous_h_val = eval_h_values(bs, n)
                    no_of_restarts += 1
                    side_ways_count=0
            else:
                side_ways_count=0

            ran_indexing = rr(0, len(temp_list))
            index = temp_list[ran_indexing]
            previous_h_val = h_val
            for uu in range(0, n):
                bs.state_board[uu][index[1]] = 0
            bs.state_board[index[0]][index[1]] = 'Q'

    if previous_h_val == 0:
        return 0, step_number, no_of_restarts
    return 1, step_number, no_of_restarts
```

#The main function running the Hill climbing algorithm for N-queens solution

```
def main():
   global n
   global iterations
   global cent
   z=0
   cent=100
   successful_steps_count = 0
   failure_step_count = 0
   successful_iterations_count = 0
   fail_iterations_count = 0
   try:
      iterations=100
      print("\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~     N-Queens Problem
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
      n = int(input("\nPLease enter the Number of Queens: "))

print("______________________________________________________________________________________________
_______________")
      print("\n\t\t\t~~~~:Hill Climbing:~~~~  ")
      print("\nCalculating...")
      for uu in range(0,iterations):
         bs = state_board(n)
         valuation,step_number = simple_hill_climbing_algo(bs,uu)
         if valuation == 0:
            fail_iterations_count +=1
            failure_step_count += step_number
         else:
            successful_iterations_count +=1
            successful_steps_count += step_number


rate_of_success=(successful_iterations_count/(successful_iterations_count+fail_iterations_count))*cen
t
      rate_of_failure=(fail_iterations_count/(successful_iterations_count+fail_iterations_count))*cent
      print("\nSuccess rate is: ",round(rate_of_success,2),"% and Failure rate is:
",round(rate_of_failure,2),"%")
      if successful_iterations_count != 0:
         print("\nThe average number of steps when the algorithm succeeds: ",
round(successful_steps_count / successful_iterations_count, 2))
      if fail_iterations_count != 0:
         print("\nThe average number of steps when the algorithm fails: ", round(failure_step_count /
fail_iterations_count, 2))

print("______________________________________________________________________________________________
_______________")
      print("\n\t\t\t~~~~:Hill-climbing search with sideways move:~~~~  ")
```

```
    print("\n  Calculating...")
    successful_steps_count = z
    failure_step_count  = z
    successful_iterations_count  = z
    fail_iterations_count  = z
    for uu in range(0, iterations):
        bs = state_board(n)
        valuation,  step_number  = hill_climbing_algo_with_sideways(bs,uu)
        if valuation  == 1:
            fail_iterations_count  += 1
            failure_step_count += step_number
        else:
            successful_iterations_count  += 1
            successful_steps_count += step_number
    rate_of_success = (successful_iterations_count  / (successful_iterations_count  +
fail_iterations_count))  * cent
    rate_of_failure = (fail_iterations_count  / (successful_iterations_count  + fail_iterations_count))  *
cent
    print("\nSuccess  rate is: ", round(rate_of_success,2), "% and Failure  rate is: ",
round(rate_of_failure,2),  "%")
    if successful_iterations_count!=0:
        print("\nThe  average number of steps when the algorithm  succeeds: ",
round(successful_steps_count  / successful_iterations_count,2))
    if fail_iterations_count!=0:
        print("\nThe  average number of steps when the algorithm  fails: ", round(failure_step_count  /
fail_iterations_count,2))

print("____________________________________________________________________________
________________")
    print("\n\t\t\t~~~~:Random-restart  hill-climbing  search without  sideways move:~~~~  ")
    print("\n  Calculating...")
    successful_steps_count = z
    failure_step_count  = z
    successful_iterations_count  = z
    fail_iterations_count  = z
    final_restarting_count  = z
    for uu in range(0,iterations):
        bs = state_board(n)
        valuation,  step_number, final_restart_count  = hill_climbing_algo_random_restart(bs)
        if valuation  == 1:
            fail_iterations_count  +=1
            failure_step_count += step_number
        else:
            successful_iterations_count  +=1
            successful_steps_count += step_number
        final_restarting_count  += final_restart_count
```

```
rate_of_success=(successful_iterations_count/(successful_iterations_count+fail_iterations_count))*cen
t
        rate_of_failure=(fail_iterations_count/(successful_iterations_count+fail_iterations_count))*cent
        print("\nSuccess rate is: ",round(rate_of_success,2),"% and Failure rate is:
",round(rate_of_failure,2),"%")
        print("\nThe average number of random restarts required without sideways move",
final_restarting_count/(successful_iterations_count+fail_iterations_count))
        print("\nThe average number of steps required without sideways move",
successful_steps_count/(successful_iterations_count+fail_iterations_count))

print("_____________________________________________________________________________________
________________")
        print("\n\t\t\t~~~~:Random-Restart hill-climbing search with sideways move:~~~~ ")
        print("Calculating...")
        successful_steps_count = z
        failure_step_count = z
        successful_iterations_count = z
        fail_iterations_count = z
        final_restarting_count = z
        for uu in range(0, iterations):
            bs = state_board(n)
            valuation, step_number, final_restart_count =
random_restart_hill_climbing_algo_with_sideways(bs)
            if valuation == 1:
                fail_iterations_count += 1
                failure_step_count += step_number
            else:
                successful_iterations_count += 1
                successful_steps_count += step_number
            final_restarting_count += final_restart_count

rate_of_success=(successful_iterations_count/(successful_iterations_count+fail_iterations_count))*cen
t
        rate_of_failure=(fail_iterations_count/(successful_iterations_count+fail_iterations_count))*cent
        print("\nSuccess rate is: ",round(rate_of_success,2),"% and Failure rate is:
",round(rate_of_failure,2),"%")
        print("\nThe average number of random restarts required with sideways move",
final_restarting_count / (successful_iterations_count + fail_iterations_count))
        print("\nThe average number of steps required with sideways move", successful_steps_count /
(successful_iterations_count + fail_iterations_count))


    except ValueError:
        print("Please enter size of the Board N (integer).")

main()
```

Sample Output (1):

```
~~~~~~~~~~~~~~~~~~~~~~~ N-Queens Problem ~~~~~~~~~~~~~~~~~~~~~~~~~~

Please enter the Number of Queens: 4
____________________________________________________________________________________

                     ~~~~:Hill Climbing:~~~~

Calculating...

The searching sequence for random configuration:  1
0 0 0 0
0 Q Q 0
Q 0 0 Q
0 0 0 0

0 0 0 0
0 Q Q 0
0 0 0 Q
Q 0 0 0

0 0 Q 0
0 Q 0 0
0 0 0 Q
Q 0 0 0

Failure
Number of steps:  2
~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  2
Q Q 0 0
0 0 Q 0
0 0 0 Q
0 0 0 0

Q 0 0 0
0 0 Q 0
0 0 0 Q
0 Q 0 0

Failure
Number of steps:  1
~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  3
Q 0 0 0
0 0 Q 0
0 Q 0 Q
0 0 0 0
```

```
Q 0 0 0
0 0 Q 0
0 0 0 Q
0 Q 0 0

Failure
Number of steps:  1
~~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  4
0 0 0 0
0 0 0 Q
0 Q 0 0
Q 0 Q 0

0 Q 0 0
0 0 0 Q
0 0 0 0
Q 0 Q 0

0 Q 0 0
0 0 0 Q
Q 0 0 0
0 0 Q 0

Success
Number of steps:  2
~~~~~~~~~~~~~~~~~~~~

Success rate is:  36.0 % and Failure rate is:  64.0 %

The average number of steps when the algorithm succeeds:  2.06

The average number of steps when the algorithm fails:  2.25
```

```
               ~~~~:Hill-climbing search with sideways move:~~~~

 Calculating...

The searching sequence for random configuration:  1
0 Q 0 0
0 0 Q Q
Q 0 0 0
0 0 0 0

0 Q 0 0
0 0 0 Q
Q 0 0 0
0 0 Q 0

Success
Number of steps:  1
~~~~~~~~~~~~~~~~~~~~
```

```
The searching sequence for random configuration:  2
0 0 Q 0
0 Q 0 0
Q 0 0 Q
0 0 0 0

0 0 Q 0
0 Q 0 0
0 0 0 Q
Q 0 0 0

0 0 Q 0
0 0 0 0
0 0 0 Q
Q Q 0 0

0 0 Q 0
Q 0 0 0
0 0 0 Q
0 Q 0 0

Success
Number of steps:  3
~~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  3
0 Q 0 0
Q 0 0 Q
0 0 0 0
0 0 Q 0

0 Q 0 0
0 0 0 Q
Q 0 0 0
0 0 Q 0

Success
Number of steps:  1
~~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  4
Q 0 0 Q
0 0 0 0
0 Q 0 0
0 0 Q 0

Q 0 0 0
0 0 0 Q
0 Q 0 0
0 0 Q 0

Q Q 0 0
0 0 0 Q
0 0 0 0
```

```
0 0 Q 0

0 Q 0 0
0 0 0 Q
Q 0 0 0
0 0 Q 0

Success
Number of steps:  3
~~~~~~~~~~~~~~~~~~~~

Success rate is:  100.0 % and Failure rate is:  0.0 %

The average number of steps when the algorithm succeeds:  2.81
```

```
       ~~~~:Random-restart hill-climbing search without sideways move:~~~~

 Calculating...

Success rate is:  100.0 % and Failure rate is:  0.0 %

The average number of random restarts required without sideways move 1.24

The average number of steps required without sideways move 4.71
```

```
       ~~~~:Random-Restart hill-climbing search with sideways move:~~~~
Calculating...

Success rate is:  100.0 % and Failure rate is:  0.0 %

The average number of random restarts required with sideways move 0.0

The average number of steps required with sideways move 2.75
```

PERFORMANCE MEASURE

Hill Climbing Search	Success rate	Failure Rate	Average number of steps when it succeeds	Average number of steps when it fails
	36	64	2.06	2.25

Hill Climbing Search with sideways move	Success rate	Failure Rate	Average number of steps when it succeeds	Average number of steps when it fails
	100	0	2.81	0

Random Restart hill climbing search with 100% Success rate	Average number of random restarts required without sideways move	Average number of steps required without sideways move	Average number of random restarts required with sideways move	Average number of steps required with sideways move
	1.24	4.71	0.0	2.75

Sample Output (2):

```
~~~~~~~~~~~~~~~~~~~~~~~~~ N-Queens Problem ~~~~~~~~~~~~~~~~~~~~~~~~~~~~

PLease enter the Number of Queens: 8
______________________________________________________________________________________

                        ~~~~:Hill Climbing:~~~~

Calculating...

The searching sequence for random configuration:  1
0 0 0 0 0 0 0 0
Q 0 0 0 0 Q Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0

Q 0 0 0 0 0 0 0
0 0 0 0 0 Q Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0

Q 0 0 0 0 0 0 0
0 0 0 0 0 Q Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 Q 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

Q 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
```

```
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

Success
Number of steps:  5
~~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  2
0 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 Q 0 0 0
Q 0 0 Q 0 0 Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0

0 0 0 Q 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 Q 0 0 0
Q 0 0 0 0 0 Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0

0 0 0 Q 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 Q 0 0 0
Q 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 Q 0 0 0 0 0 0

0 0 0 Q 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 0 0 0 0
Q 0 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 Q
0 Q 0 0 0 0 0 0

0 0 0 Q 0 0 0 0
0 0 0 0 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 Q
```

```
0 Q 0 0 0 0 0 0

Failure
Number of steps:  4
~~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  3
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 0 0 Q 0
Q 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 0 0 0 Q 0 0 0
0 Q 0 Q 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 0 0 Q 0
Q 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0

0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 0 Q 0 0 0 0 0
Q 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0

0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 0 Q 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 Q 0 0 Q
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0

0 0 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0
0 0 Q 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0

Success
Number of steps:  4
~~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  4
0 Q 0 0 0 0 0 0
```

```
0 0 0 0 0 0 Q 0
0 0 Q 0 0 Q 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 Q

0 Q 0 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 Q

0 Q 0 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

0 Q 0 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

Failure
Number of steps:  3
~~~~~~~~~~~~~~~~~~~~

Success rate is:  17.0 % and Failure rate is:  83.0 %

The average number of steps when the algorithm succeeds:  4.06

The average number of steps when the algorithm fails:  4.06
```

```
                ~~~~:Hill-climbing search with sideways move:~~~~

 Calculating...

The searching sequence for random configuration:  1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
Q Q 0 Q 0 Q Q Q
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0
```

0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
Q Q 0 Q 0 0 Q Q
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q Q 0 0 0 0 Q Q
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 Q Q
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0
0 0 Q 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 Q 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 Q 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

```
0 0 Q 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 0
0 0 Q Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
```

```
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
Q Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
```

```
0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0

0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0

Success
Number of steps:  25
~~~~~~~~~~~~~~~~~~~
```

The searching sequence for random configuration: 2

```
0 0 0 0 0 0 0 0
0 Q 0 0 Q 0 0 0
Q 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
Q 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0
Q 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0

0 0 Q 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0

0 0 Q 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
```

```
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 Q
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0
Q 0 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 Q
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 Q
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 Q
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

0 0 0 0 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 Q
```

```
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 Q
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 Q
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 Q
0 0 0 0 0 0 0 0
0 Q 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 Q 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0

0 0 0 0 Q 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 0

Success
```

```
Number of steps:  18
~~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  3
0 Q Q 0 0 Q 0 Q
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

0 Q Q 0 0 0 0 Q
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 Q Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 Q 0 0

0 Q Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 Q 0 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 0 Q 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 Q 0 0

Success
Number of steps:  4
~~~~~~~~~~~~~~~~~~~~

The searching sequence for random configuration:  4
0 0 Q 0 0 0 0 0
0 0 0 Q 0 Q 0 0
0 Q 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
```

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 Q 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q

0 0 0 0 Q 0 0 0
Q Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

Q 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q Q 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 Q Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0

```
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 Q 0 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 Q 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 Q Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 Q 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 0 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 Q 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 Q 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
```

```
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 Q 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 0 0
0 0 0 0 Q 0 0 0

0 0 Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 Q 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0

0 Q Q 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 0 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0

0 Q 0 0 0 0 0 0
0 0 0 0 0 Q 0 0
0 0 0 0 0 0 0 Q
0 0 Q 0 0 0 0 0
Q 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
0 0 0 0 0 0 Q 0
0 0 0 0 Q 0 0 0

Success
Number of steps:  29
~~~~~~~~~~~~~~~~~~~~

Success rate is:  89.0 % and Failure rate is:  11.0 %

The average number of steps when the algorithm succeeds:  18.47

The average number of steps when the algorithm fails:  65.55
```

```
        ~~~~:Random-restart hill-climbing search without sideways move:~~~~

 Calculating...

Success rate is:  100.0 % and Failure rate is:  0.0 %

The average number of random restarts required without sideways move 6.06

The average number of steps required without sideways move 28.63
```

```
        ~~~~:Random-Restart hill-climbing search with sideways move:~~~~
Calculating...

Success rate is:  100.0 % and Failure rate is:  0.0 %

The average number of random restarts required with sideways move 0.81

The average number of steps required with sideways move 26.41
```

PERFORMANCE MEASURE

Hill Climbing Search	Success rate	Failure Rate	Average number of steps when it succeeds	Average number of steps when it fails
	17	83	4.06	4.06

Hill Climbing Search with sideways move	Success rate	Failure Rate	Average number of steps when it succeeds	Average number of steps when it fails
	89	11	18.47	65.55

Random Restart hill climbing search with 100% Success rate	Average number of random restarts required without sideways move	Average number of steps required without sideways move	Average number of random restarts required with sideways move	Average number of steps required with sideways move
	6.06	28.63	0.81	26.41

CONCLUSION

- In this project we can observe that when N queens problem is implemented using hill climbing method, it succeeds only 17% of the time whereas 83% of the time it gets stuck at a local minimum. However, it takes only 4 steps on average when it succeeds and 4 on average when it gets stuck
- It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau.

- In order to escape these problems, we implement n queens using sideways move. For 8-queens, we allow sideways moves with a limit of 100 which raises the percentage of problem instances solved from 17 to 89%. However, it takes around 18 steps for every successful solution and 65 steps for each failure.

- Both the methods –Hill Climbing and Sideways move Hill climbing are incomplete, i.e they often fail to find a goal when one exists because they get stuck on local maxima. Therefore, we implement the N queen problem using Random restart Hill climbing method. The algorithm conducts various hill climbing searches from randomly generated initial states, until a goal is found. It is complete with probability approaching 1, because it will eventually generate a goal state as the initial state.