# Intelligent Systems Project-2 Report

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

**Team Members:**

**Shreya Vinodh 801149383**

**Utkarsha Gurjar 801149356**

**Afreen 801136632**

**Hill-Climbing Search:**

The hill-climbing search algorithm is a loop that continually moves in the direction of the best successor value and terminates when there are no neighboring states with better values. The hill climbing algorithm usually gets stuck because of local maxima, ridges and plateaus.

In case of 8-queens problem, hill climbing gets stuck around 86% of the time and solves about 14% of problem instances. It takes 4 steps on average when it succeeds and 3 when it gets stuck.

To improve the results when stuck on a plateau, sideways move is implemented to check if the plateau is a shoulder i.e. to keep moving forward on the plateau state; however, an infinite loop can occur when the algorithm reaches a flat local maximum/minimum that is not a shoulder. To solve this problem, a limit of say 100 consecutive sideways moves can be implemented.

This raises the percentage of problem instances solved by hill climbing from 14% to 94%. With this addition, the algorithm averages roughly 21 steps for each successful instance and 64 for each failure.

The hill-climbing algorithms are incomplete—they often fail to find a goal when one exists because they can get stuck on local maxima. Random-restart hill climbing conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

This implementation completes 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. For 8-queens instances with no sideways moves allowed, p ≈ 0.14, so we need roughly 7 iterations to find a goal (6 failures and 1 success). The expected number of steps is the cost of one successful iteration plus (1−p)/p times the cost of failure, or roughly 22 steps in all. When we allow sideways moves, 1/0.94 ≈ 1.06 iterations are needed on average and (1×21)+(0.06/0.94)×64 ≈ 25 steps.

**n-Queens Formulation**:

Initial state:

An n\*n matrix with n queens placed randomly one per row in the matrix. For eg. 8\*8 matrix with 8 queens placed randomly one per row in the matrix.

Heuristic:

The heuristic cost h, is the number of pairs of queens that are attacking each other, either directly or indirectly.

Goal state:

An n\*n matrix where each queen is placed on one position per row, such that no queen attacks another either horizontally, vertically or diagonally.

Goal Test :

When the heuristic h of a state is zero, that state is considered to be the goal.

Successor function:

Each successor is obtained by adding one queen in an empty square

**Program Structure:**

Create an object of class board start

Class board initializes variables for calculating the total number of steps, successes, failures etc.

start calls main function defined in class board main()

main() function asks the user for the number of queens to be placed on the board, and the number of runs to be executed.

After getting these inputs, implement hill-climbing algorithm. calc\_board()

Implement hill-climbing algorithm with sideways move. calc\_board\_wSideway()

Implement Random-Restart algorithm without sideways move and with sideways move.

calc\_RandomRestart()

Class state initializes the size of the board, i.e. no. of rows and no.of columns (same as the no. of queens to be placed on the board), ‘h’ value, and data to a board state.

This class has functions to:

create a random initial board state generate\_initial()

generate successor board state generate\_successor()

calculate h value for each board state calc\_h()

**Global variables used:**

n – Number of queens to be placed on the board; input by the user.

h\_dash – A variable that stores the minimum heuristic value of a state of the board.

**Functions/Procedure used to compute Heuristic Function:**

calc\_h(): counts the number of queens that are placed on the attacking positions from a particular position. A helper function findAttckPos() is used to find the attacking positions, i.e vertically and diagonally.

findAttckPos(): finds the attacking positions from a particular position on the board. Returns two lists - one with vertical attack positions and another with diagonal attack positions.

**Other functions/procedures used:**

Apart from the above mentioned heuristic functions, there are 2 constructors and 8 methods:

1. The constructor of class “state” initializes the size of the board, board data containing the queens and the blank spaces.
2. The function “generate\_initial” generates the board randomly.
3. The function “generate\_successor” generates the successor board.
4. The helper function “copy” is used to copy the parent board in order to generate the successor board.
5. The constructor of class “board” initializes the success steps, failure steps, total no.of steps, total success steps, total failure steps.
6. The function “find\_rowPos” is used to find the position of the queen of a particular row on the initial board.
7. The function “main” asks the user for the number of queens to be placed on the board, and the number of runs to be executed.
8. The function “calc\_board” performs hill climbing algorithm.
9. The function “calc\_board\_wSideway” performs hill climbing with sideways move.
10. The function “calc\_RandomRestart” performs hill climbing with random restart method.

**Program Code:**

import random

import math

class state:

# global variable to store the h value of each board state.

global h\_dash

# constructor of class state that initializes size, data, and h field of the board.

def \_\_init\_\_(self, size, data=None):

self.size = size

self.data = data

self.h = -1 #initializing with a non-positive number

# function that generates the initial configuration of the board.

def generate\_initial(self):

board\_s=[]

for i in range(int(self.size)):

col=[None]\*int(self.size)

rand = random.randint(0,int(self.size)-1)

for j in range(int(self.size)):

if j == rand:

col[j]='Q'

else:

col[j]='\_'

board\_s.append(col)

self.data = board\_s

return self.data

# function to generate a successor board by moving the queen from its current location(old\_pos) to a

# specified location(new\_pos) on the board.

def generate\_successor(self, old\_pos, new\_pos):

x = new\_pos[0]

y = new\_pos[1]

old\_x = old\_pos[0]

old\_y = old\_pos[1]

nxt = self.copy(self.data)

for i in range(0, len(nxt)):

for j in range(0, len(nxt)):

if i==x and j==y:

nxt[i][j]='Q'

if i==old\_x and j==old\_y:

nxt[i][j]='\_'

return nxt

# helper function to copy the data of the board of which a successor needs to be generated,

# onto a new board.

def copy(self,prev):

copy\_board=[]

for i in prev:

squares=[]

for j in i:

squares.append(j)

copy\_board.append(squares)

return copy\_board

# function to calculate the h value of a board.

def calc\_h(self):

count=0

for i in range(len(self.data)):

for j in range(len(self.data)):

if self.data[i][j]=='Q':

pos = [None]\*2

pos[0]=i

pos[1]=j

v,d = self.findAttckPos(pos)

for k in range(len(self.data)):

for l in range(len(self.data)):

if self.data[k][l]=='Q':

for e in v:

if e[0] == k and e[1] == l:

count+=1

for e in d:

if e[0] == k and e[1] == l:

count+=1

return math.ceil(count/2)

# helper function to calculate the attack positions(vertical and diagonal) from a

# particular position(x,y) on the board.

def findAttckPos(self,pos):

ver = [] #an empty list to store the vertical attack positions

x=pos[0]

y=pos[1]

if x==0:

while(x < len(self.data)-1):

attck\_v=[None]\*2

x+=1

attck\_v[0]=x

attck\_v[1]=y

ver.append(attck\_v)

elif x==len(self.data):

while(x>=0):

attck\_v=[None]\*2

x-=1

attck\_v[0]=x

attck\_v[1]=y

ver.append(attck\_v)

else:

while(x < len(self.data)-1):

attck\_v=[None]\*2

x+=1

attck\_v[0]=x

attck\_v[1]=y

ver.append(attck\_v)

x=pos[0]

y=pos[1]

while(x>=0):

attck\_v=[None]\*2

x-=1

attck\_v[0]=x

attck\_v[1]=y

ver.append(attck\_v)

diag = [] #an empty list to store the diagonal attack positions

x=pos[0]

y=pos[1]

#1. Checking upper left squares

while x>0 and x<len(self.data) and y>0 and y<len(self.data):

attck\_d=[None]\*2

x-=1

y-=1

attck\_d[0]=x

attck\_d[1]=y

diag.append(attck\_d)

x=pos[0]

y=pos[1]

#2. Checking lower left squares

while x>=0 and x<len(self.data)-1 and y>0 and y<len(self.data):

attck\_d=[None]\*2

x+=1

y-=1

attck\_d[0]=x

attck\_d[1]=y

diag.append(attck\_d)

x=pos[0]

y=pos[1]

#3. Checking upper right squares

while x>0 and x<len(self.data) and y>=0 and y<len(self.data)-1:

attck\_d=[None]\*2

x-=1

y+=1

attck\_d[0]=x

attck\_d[1]=y

diag.append(attck\_d)

x=pos[0]

y=pos[1]

#4. Checking lower right squares

while x>=0 and x<len(self.data)-1 and y>=0 and y<len(self.data)-1:

attck\_d=[None]\*2

x+=1

y+=1

attck\_d[0]=x

attck\_d[1]=y

diag.append(attck\_d)

return ver, diag

class board:

# global variable that stores the input of number of queens to be placed on the board from a user.

global n

def \_\_init\_\_(self):

self.successes = 0

self.failures = 0

self.steps = 0

self.tot\_sSteps = 0

self.tot\_fSteps = 0

self.count\_init = 0

# function to find the position of the queen on a particular row on the initial board.

def find\_rowPos(self,data,row\_num):

for j in range(len(data)):

if data[row\_num][j] == 'Q':

pos = [None]\*2

pos[0]= row\_num

pos[1]= j

break

return pos

# main function

def main(self):

global n

print("How many Queens do you want to place on the board?")

n = input()

print("How many runs?")

r = input()

print()

# Call for Hill-Climbing search.

print("Hill-Climbing search:")

for x in range(int(r)):

self.steps = 0

self.calc\_board(x)

print()

print("....")

print("....")

print("....")

print("....")

print("....")

print()

print("Hill-Climbing search stats for " + str(n) + "-Queens problem for " + str(r) + " runs:")

print("Success Rate: ", 100\*self.successes/int(r),"%")

print("Failure Rate: ", 100\*self.failures/int(r), "%")

print("Average number of steps for successes: ", self.tot\_sSteps/self.successes)

print("Average number of steps for failures: ", self.tot\_fSteps/self.failures)

print()

# call for Hill-climbing search with sideways move.

print("Hill-Climbing search with sideways move:")

self.successes = 0

self.failures = 0

self.steps = 0

self.tot\_sSteps = 0

self.tot\_fSteps = 0

for x in range(int(r)):

self.steps = 0

self.calc\_board\_wSideway(x)

print()

print("....")

print("....")

print("....")

print("....")

print("....")

print()

print("Hill-Climbing search with sideways move stats for " + str(n) + "-Queens problem for " + str(r) + " runs:")

print("Success Rate: ", 100\*self.successes/int(r),"%")

print("Failure Rate: ", 100\*self.failures/int(r), "%")

print("Average number of steps for successes: ", self.tot\_sSteps/self.successes)

print("Average number of steps for failures: ", self.tot\_fSteps/self.failures)

print()

# call for random-restart without sideways move.

print("Random-Restart Hill-Climbing search:")

self.steps = 0

self.count\_init = 0

for x in range(int(r)):

y=self.steps

self.calc\_RandomRestart(0)

print()

print("Random-Restart Hill-Climbing search without sideways move stats for " + str(n) + "-Queens problem for " + str(r) + " runs:")

print("Average number of random-restarts without sideways move: ", self.count\_init/int(r))

print("Average number of steps required without sideways move: ", self.steps/int(r))

print()

# call for random-restart with sideways move.

self.steps = 0

self.count\_init = 0

for x in range(int(r)):

self.calc\_RandomRestart(1)

print()

print("Random-Restart Hill-Climbing search with sideways move stats for " + str(n) + "-Queens problem for " + str(r) + " runs:")

print("Average number of random-restarts with sideways move: ", self.count\_init/int(r))

print("Average number of steps required with sideways move: ", self.steps/int(r))

print()

# Hill-Climbing search.

def calc\_board(self, call):

h\_dash=-1

initial\_board = state(n)

initial\_board.data = initial\_board.generate\_initial()

initial\_board.h = initial\_board.calc\_h()

if call < 4:

print("Search Sequence " + str(call+1) + ":")

print("Initial state:")

for x in initial\_board.data:

for y in x:

print(y, end=" ")

print()

print("h value:", initial\_board.h)

print()

min\_board = initial\_board.data

while h\_dash !=0:

store\_min\_hPos = []

previous\_board = state(n, min\_board)

previous\_board.h = previous\_board.calc\_h()

h\_dash = previous\_board.h

for i in range(int(n)):

pos = self.find\_rowPos(previous\_board.data,i)

for j in range(int(n)):

new\_pos = [None]\*2

new\_pos[0]=i

new\_pos[1]=j

if new\_pos == pos:

continue

successor\_board = state(n)

successor\_board.data = previous\_board.generate\_successor(pos, new\_pos)

successor\_board.h = successor\_board.calc\_h()

if successor\_board.h <= h\_dash:

h\_dash = successor\_board.h

store\_pos =new\_pos

store\_pos.append(successor\_board.h)

store\_min\_hPos.append(store\_pos)

if store\_min\_hPos:

l = len(store\_min\_hPos)-1

while l>=0:

y = store\_min\_hPos[l]

if y[2] != h\_dash:

del store\_min\_hPos[l]

l-=1

rand = random.randint(0,len(store\_min\_hPos)-1)

pos\_dash = store\_min\_hPos[rand]

del pos\_dash[2]

pos\_parent = self.find\_rowPos(previous\_board.data,pos\_dash[0])

min\_board = previous\_board.generate\_successor(pos\_parent, pos\_dash)

if h\_dash == previous\_board.h:

if h\_dash == 0:

if call < 4:

print("Solution found.")

self.successes +=1

else:

self.tot\_fSteps +=self.steps

if call < 4:

for x in min\_board:

for y in x:

print(y, end=" ")

print()

print("h value: " + str(h\_dash))

print()

print("Solution not found.")

print()

self.failures +=1

break

else:

self.steps +=1

if call < 4:

print("Next state:")

for x in min\_board:

for y in x:

print(y, end=" ")

print()

print("h value:", h\_dash)

print()

if h\_dash==0:

self.tot\_sSteps += self.steps

if call < 4:

print("Solution found.")

print()

self.successes += 1

# Hill-climbing search with sideways move.

def calc\_board\_wSideway(self, call):

try\_ = 0

h\_dash=-1

initial\_board = state(n)

initial\_board.data = initial\_board.generate\_initial()

initial\_board.h = initial\_board.calc\_h()

if call < 4:

print("Search Sequence " + str(call+1) + ":")

print("Initial state:")

for x in initial\_board.data:

for y in x:

print(y, end=" ")

print()

print("h value:", initial\_board.h)

print()

min\_board = initial\_board.data

while h\_dash !=0:

store\_min\_hPos = []

self.steps +=1

check\_hHigh = 0

previous\_board = state(n, min\_board)

previous\_board.h = previous\_board.calc\_h()

h\_dash = previous\_board.h

for i in range(int(n)):

pos = self.find\_rowPos(previous\_board.data,i)

for j in range(int(n)):

new\_pos = [None]\*2

new\_pos[0]=i

new\_pos[1]=j

if new\_pos == pos:

continue

successor\_board = state(n)

successor\_board.data = previous\_board.generate\_successor(pos, new\_pos)

successor\_board.h = successor\_board.calc\_h()

if successor\_board.h <= h\_dash:

h\_dash = successor\_board.h

store\_pos =new\_pos

check\_hHigh = 1

if successor\_board.h < h\_dash:

try\_ = 0

store\_pos.append(successor\_board.h)

store\_min\_hPos.append(store\_pos)

if store\_min\_hPos:

l = len(store\_min\_hPos)-1

while l>=0:

y = store\_min\_hPos[l]

if y[2] != h\_dash:

del store\_min\_hPos[l]

l-=1

rand = random.randint(0,len(store\_min\_hPos)-1)

pos\_dash = store\_min\_hPos[rand]

del pos\_dash[2]

pos\_parent = self.find\_rowPos(previous\_board.data,pos\_dash[0])

min\_board = previous\_board.generate\_successor(pos\_parent, pos\_dash)

if h\_dash == previous\_board.h:

if h\_dash == 0:

if call < 4:

print("Solution found.")

print()

self.tot\_sSteps += self.steps

self.successes +=1

else:

if check\_hHigh != 0:

try\_ +=1

if call < 4:

print("Next state:")

for x in min\_board:

for y in x:

print(y, end=" ")

print()

print("h value:", h\_dash)

print()

else:

self.tot\_fSteps +=self.steps

self.failures +=1

if call < 4:

print("Solution not found.")

print()

break

if try\_ >=100:

self.tot\_fSteps +=self.steps

self.failures +=1

if call < 4:

for x in min\_board:

for y in x:

print(y, end=" ")

print()

print("h value:", h\_dash)

print()

print("Solution not found after 100 sideways move.")

print()

break

else:

if call < 4:

print("Next state:")

for x in min\_board:

for y in x:

print(y, end=" ")

print()

print("h value:", h\_dash)

print()

if h\_dash==0:

self.tot\_sSteps += self.steps

if call < 4:

print("Solution found.")

print()

self.successes += 1

# Random-restart.

def calc\_RandomRestart(self, sideway):

try\_=0

h\_dash=-1

while h\_dash != 0:

self.count\_init +=1

initial\_board = state(n)

initial\_board.data = initial\_board.generate\_initial()

initial\_board.h = initial\_board.calc\_h()

if initial\_board.h == 0:

self.successes +=1

break

min\_board = initial\_board.data

while h\_dash !=0:

store\_min\_hPos = []

check\_hHigh = 0

previous\_board = state(n, min\_board)

previous\_board.h = previous\_board.calc\_h()

h\_dash = previous\_board.h

for i in range(int(n)):

pos = self.find\_rowPos(previous\_board.data,i)

for j in range(int(n)):

new\_pos = [None]\*2

new\_pos[0]=i

new\_pos[1]=j

if new\_pos == pos:

continue

successor\_board = state(n)

successor\_board.data = previous\_board.generate\_successor(pos, new\_pos)

successor\_board.h = successor\_board.calc\_h()

if successor\_board.h <= h\_dash:

if successor\_board.h < h\_dash:

try\_ = 0

h\_dash = successor\_board.h

store\_pos =new\_pos

store\_pos.append(successor\_board.h)

store\_min\_hPos.append(store\_pos)

check\_hHigh = 1

if h\_dash == 0 or check\_hHigh == 0:

if h\_dash == 0:

self.steps+=1

self.tot\_sSteps += self.steps

self.successes +=1

if check\_hHigh == 0:

self.tot\_fSteps +=self.steps

self.failures +=1

break

if store\_min\_hPos:

l = len(store\_min\_hPos)-1

while l>=0:

y = store\_min\_hPos[l]

if y[2] != h\_dash:

del store\_min\_hPos[l]

l-=1

rand = random.randint(0,len(store\_min\_hPos)-1)

pos\_dash = store\_min\_hPos[rand]

del pos\_dash[2]

pos\_parent = self.find\_rowPos(previous\_board.data,pos\_dash[0])

min\_board = previous\_board.generate\_successor(pos\_parent, pos\_dash)

if h\_dash == previous\_board.h:

if sideway == 0:

break

if sideway == 1:

self.steps+=1

if check\_hHigh != 0:

try\_ +=1

if try\_ >=100:

break

else:

self.steps+=1

start = board()

start.main()

**Execution Results:**

For 8-Queens problem:

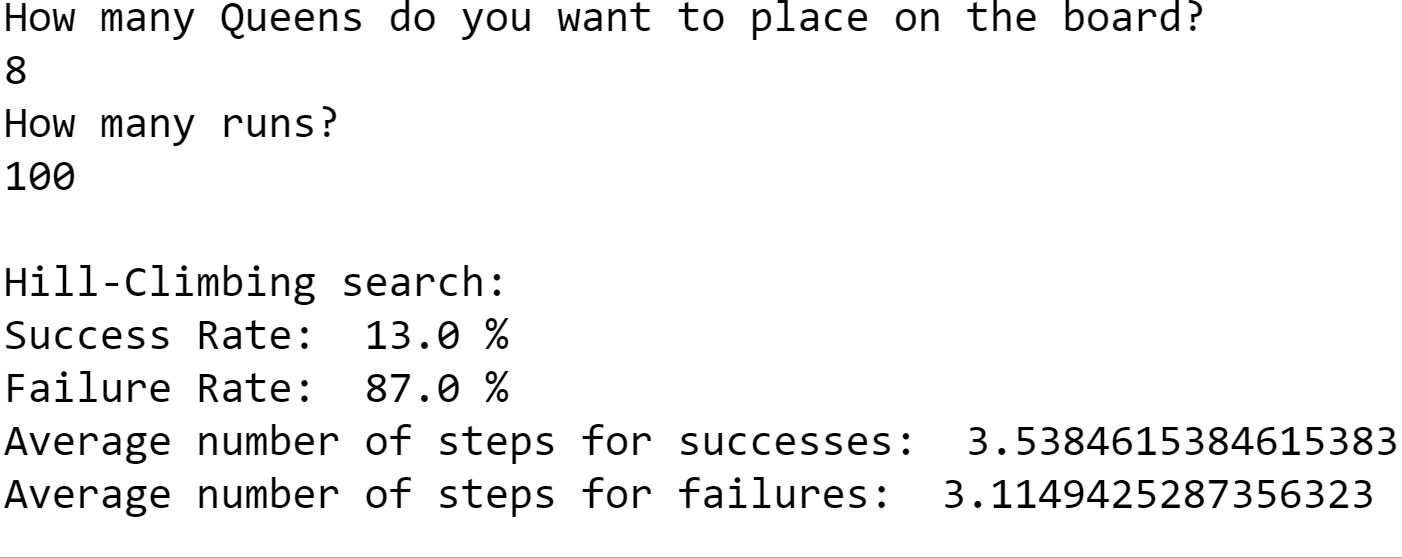
* **Hill-Climbing search**

Expected output:

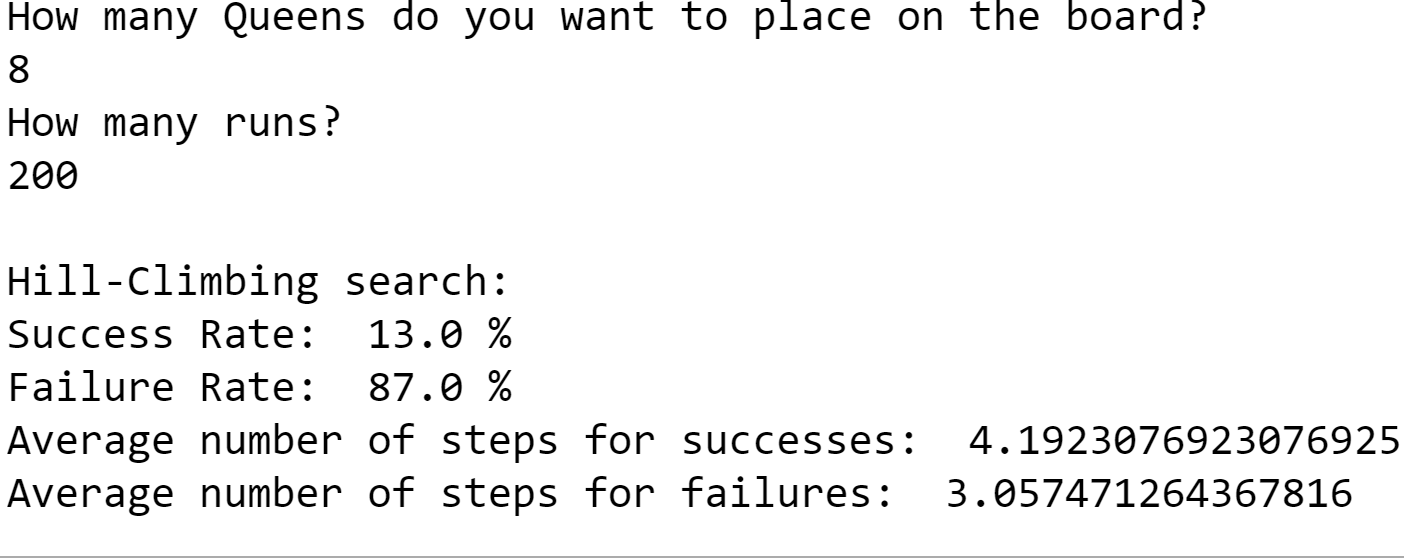
* Solves 14% of problem instances.
* Takes 4 steps on average when it succeeds and 3 when it gets stuck.

Actual output:

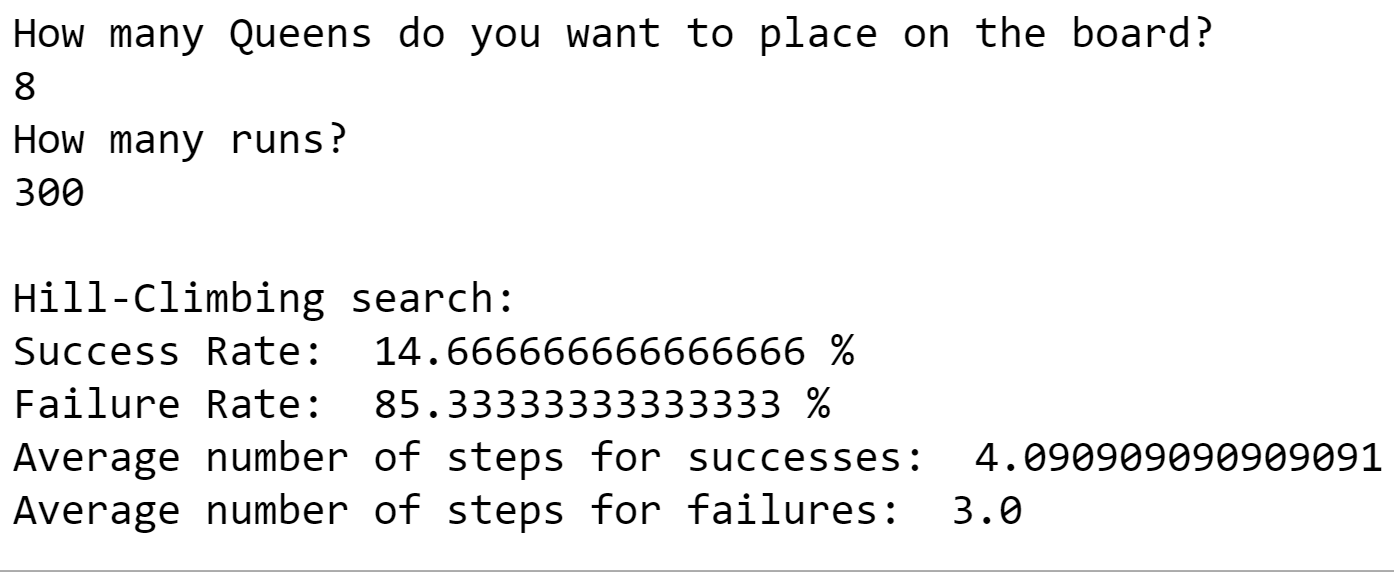
For 100 runs:



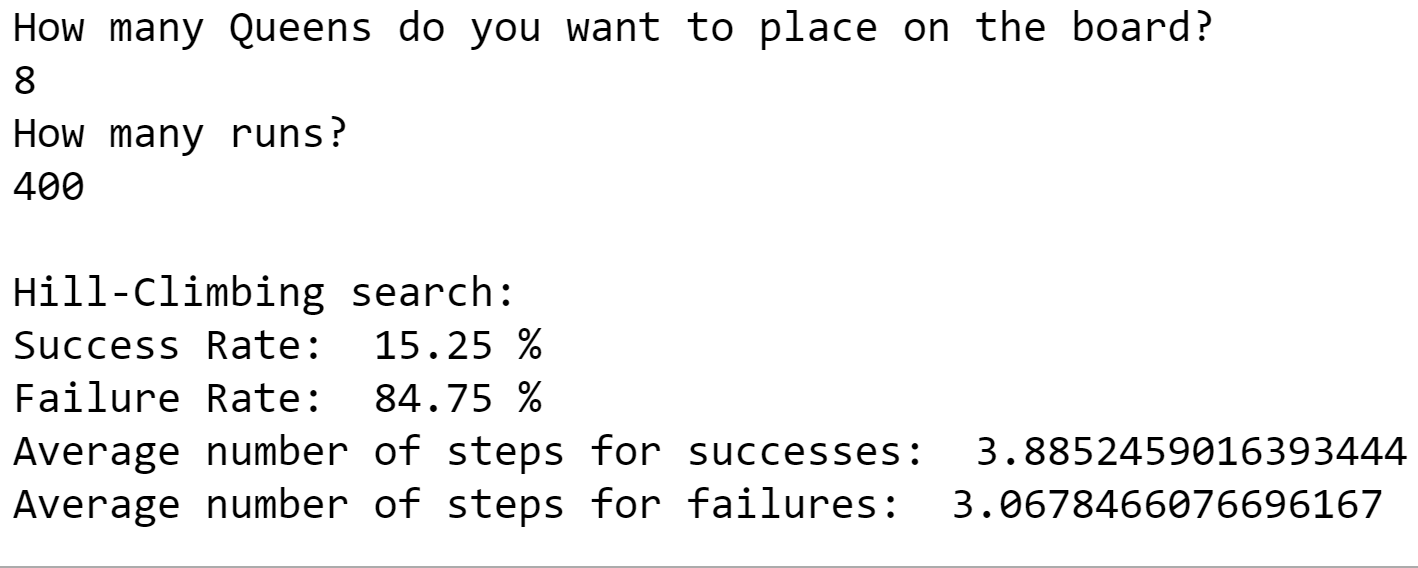
For 200 runs:



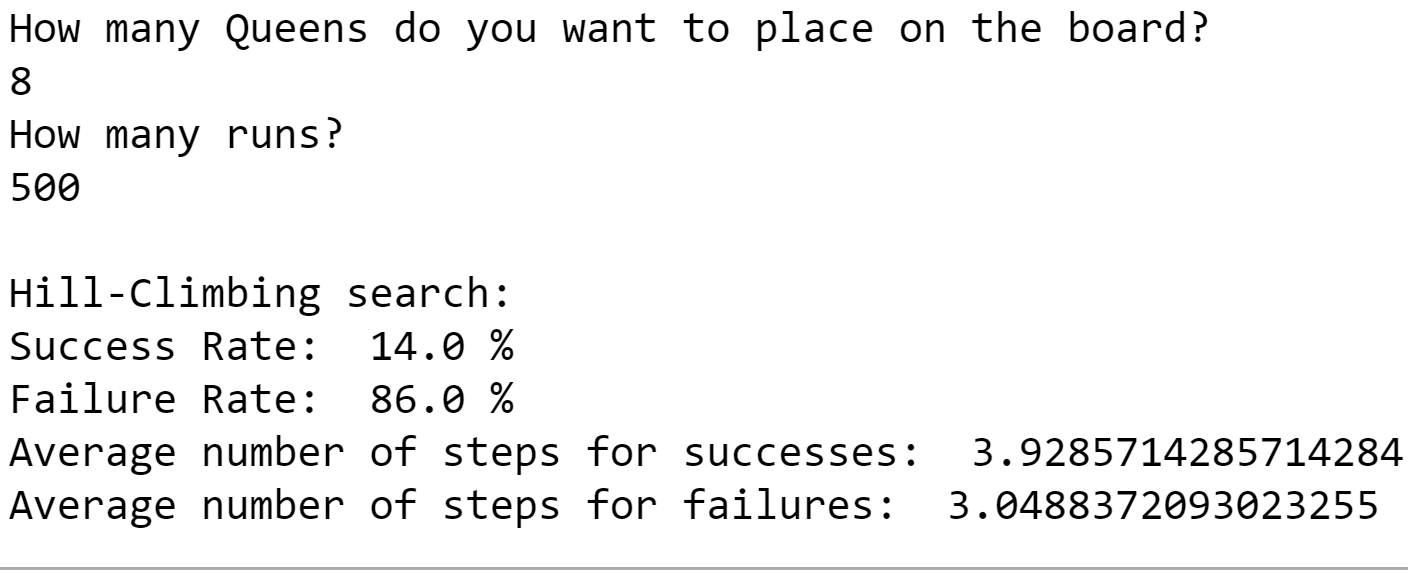
For 300 runs:



For 400 runs:



For 500 runs:



Search sequences for Hill-Climbing search:

Output for 8 queens on the board-

|  |  |  |  |
| --- | --- | --- | --- |
| Search sequence 1 | Search sequence 2 | Search sequence 3 | Search sequence 4 |
| Initial state:  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ Q \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  h value: 7  Next state:  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ Q \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  h value: 3  Next state:  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ Q \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  h value: 2  Next state:  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ Q \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  h value: 1  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ Q \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 1  Solution not found. | Initial state:  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 9  Next state:  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ \_ Q \_ \_  h value: 6  Next state:  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  h value: 4  Next state:  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  h value: 1  \_ \_ \_ \_ Q \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  h value: 1  Solution not found. | Initial state:  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 5  Next state:  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 3  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 1  Solution not found. | Initial state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 7  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ Q \_ \_ \_ \_  h value: 4  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ Q \_ \_ \_ \_  h value: 2  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ Q \_ \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  h value: 0  Solution found. |

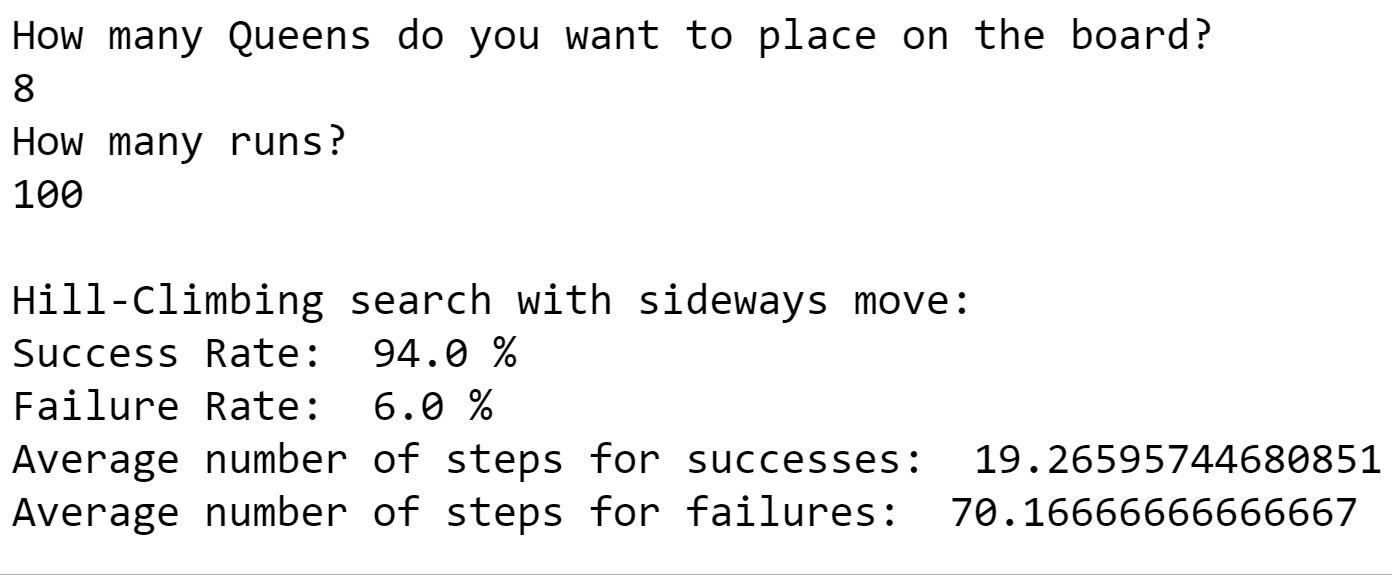
* **Hill-Climbing search with sideways move**

Expected output:

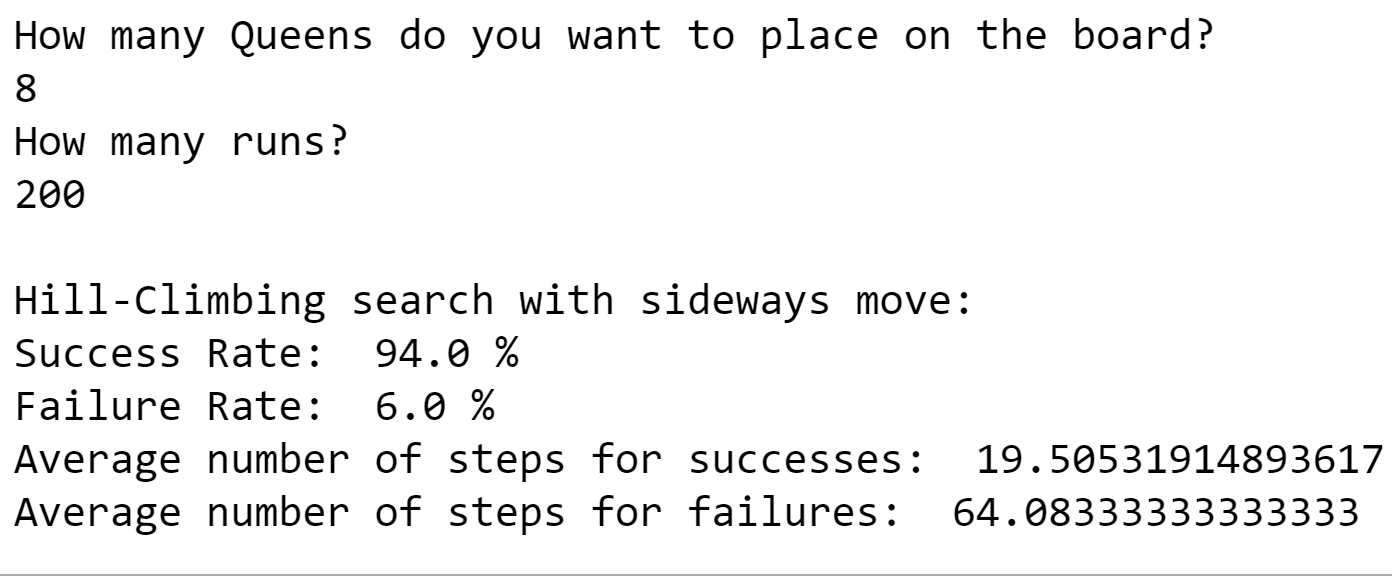
* Solves 94% of problem instances.
* Takes 21 steps on average when it succeeds and 64 on failure.

Actual output:

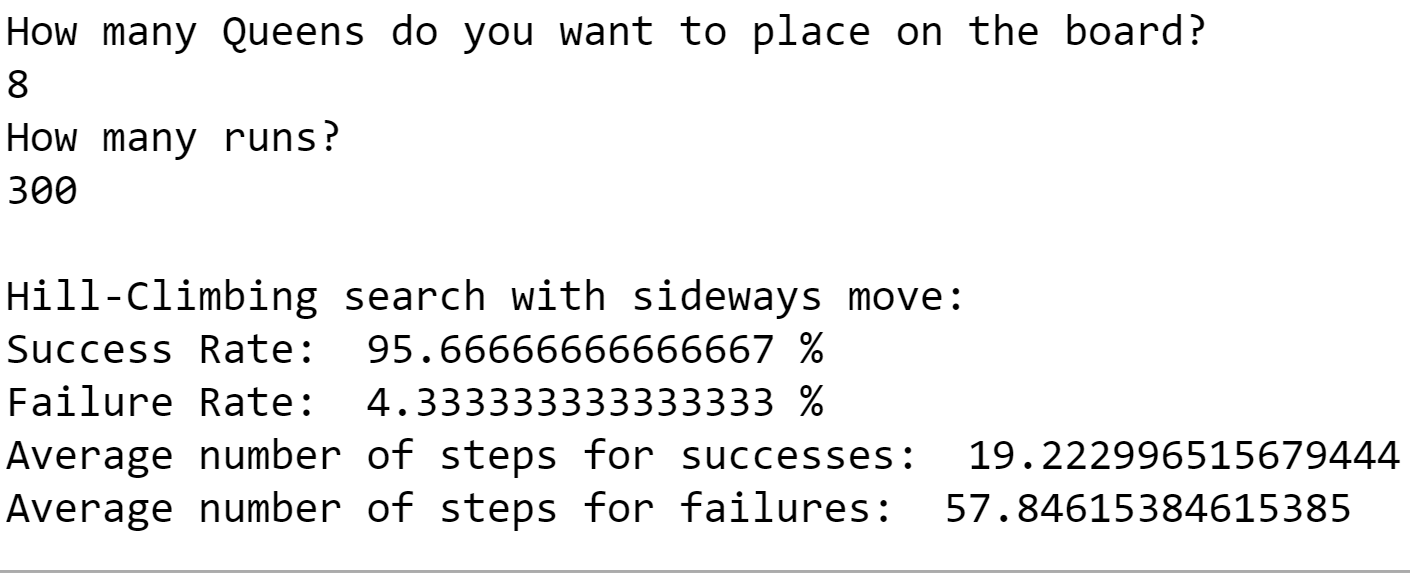
100 runs



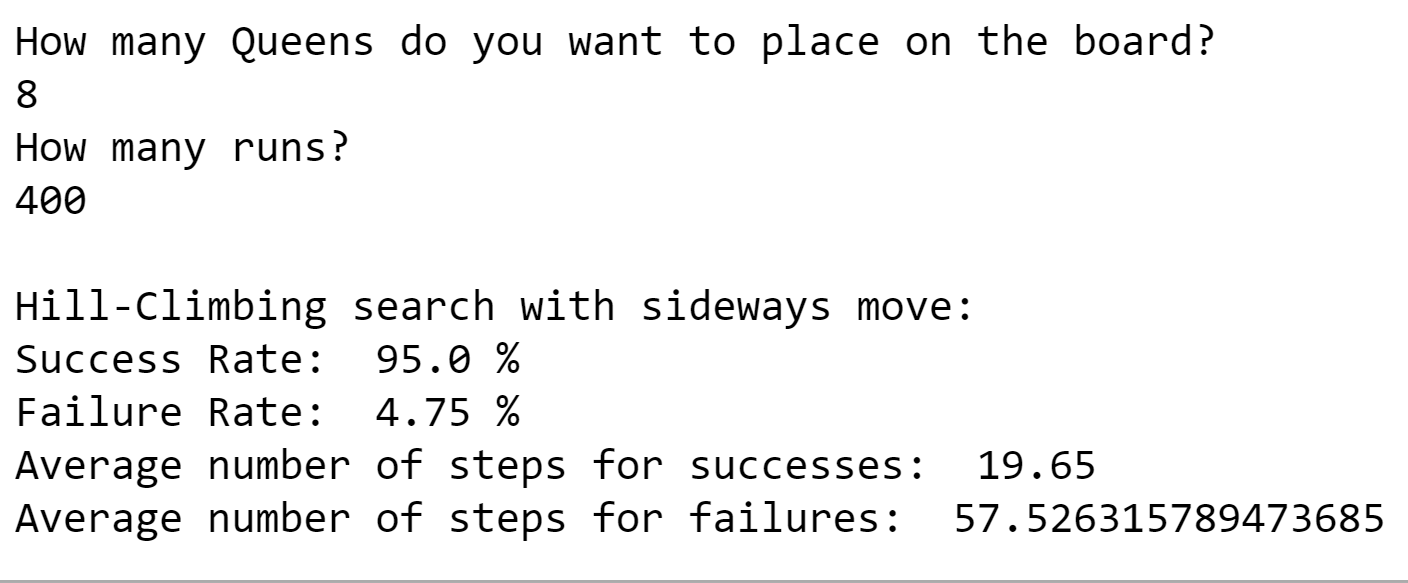
200 runs



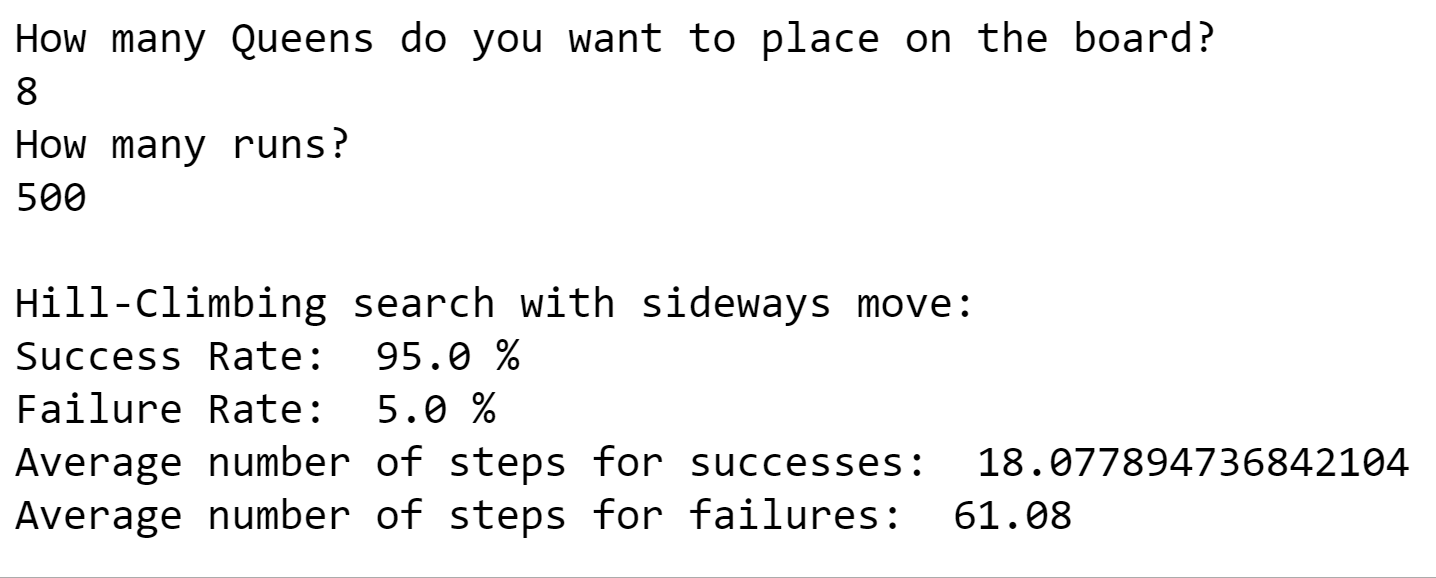
300 runs



400 runs



500 runs



Search sequences for Hill-Climbing with sideways move implemented-

Output for 8 queens on the board-

|  |  |  |  |
| --- | --- | --- | --- |
| Search sequence 1 | Search sequence 2 | Search sequence 3 | Search sequence 4 |
| Initial state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 8  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 5  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 3  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  h value: 2  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  h value: 2  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  h value: 0  Solution found. | Initial state:  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  h value: 13  Next state:  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  h value: 8  Next state:  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  h value: 4  Next state:  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  h value: 2  Next state:  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  h value: 2  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 1  Next state:  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  h value: 1  Next state:  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 1  Next state:  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 1  Next state:  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 0  Solution found. | Initial state:  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 14  Next state:  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 9  Next state:  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  h value: 5  Next state:  \_ \_ \_ \_ \_ \_ Q \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  h value: 3  Next state:  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  h value: 2  Next state:  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  h value: 1  Next state:  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  h value: 0  Solution found. | Initial state:  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 7  Next state:  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 3  Next state:  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 2  Next state:  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ \_ Q \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 2  Next state:  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ Q \_ \_ \_ \_ \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 2  Next state:  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ \_ \_ \_ \_ \_ Q  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ Q \_ \_  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 1  Next state:  \_ \_ \_ \_ \_ Q \_ \_  \_ \_ Q \_ \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ Q \_  \_ Q \_ \_ \_ \_ \_ \_  \_ \_ \_ Q \_ \_ \_ \_  \_ \_ \_ \_ \_ \_ \_ Q  Q \_ \_ \_ \_ \_ \_ \_  \_ \_ \_ \_ Q \_ \_ \_  h value: 0  Solution found. |

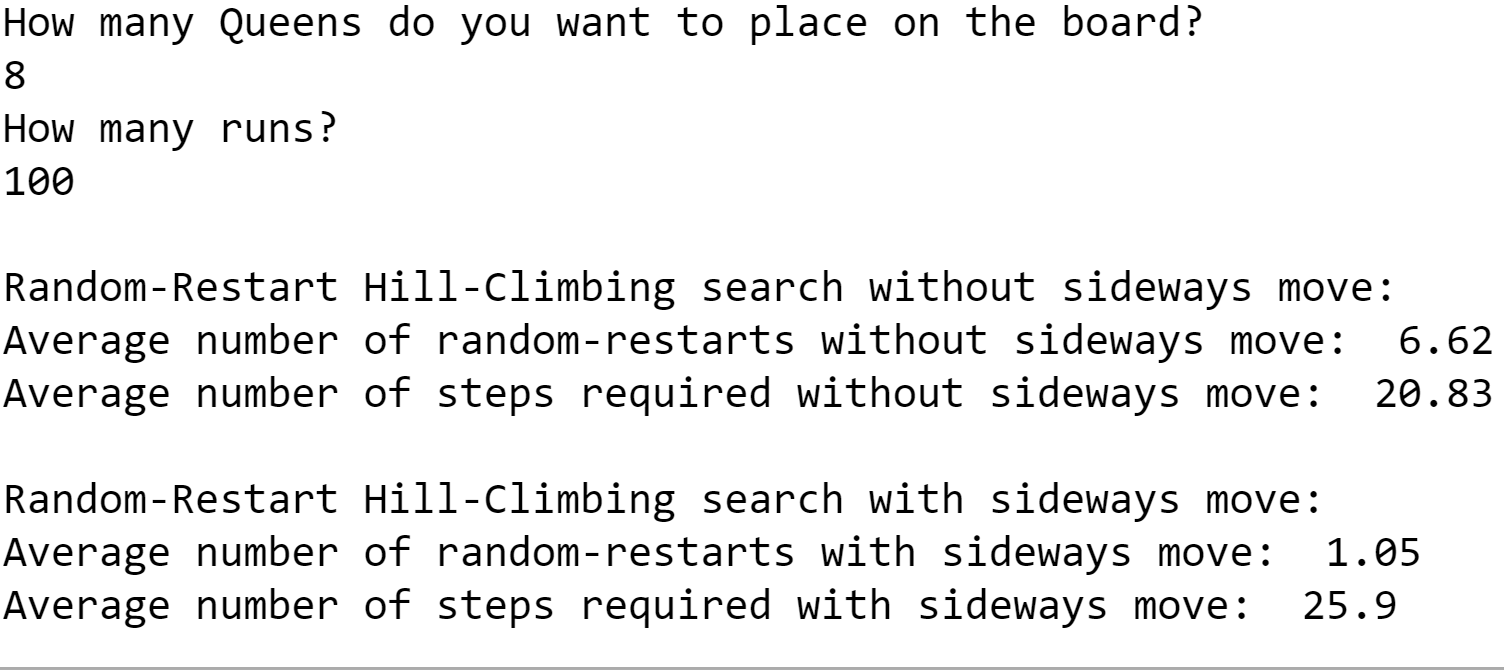
* **Random-Restart without sideways move and with sideways move**

Expected output:

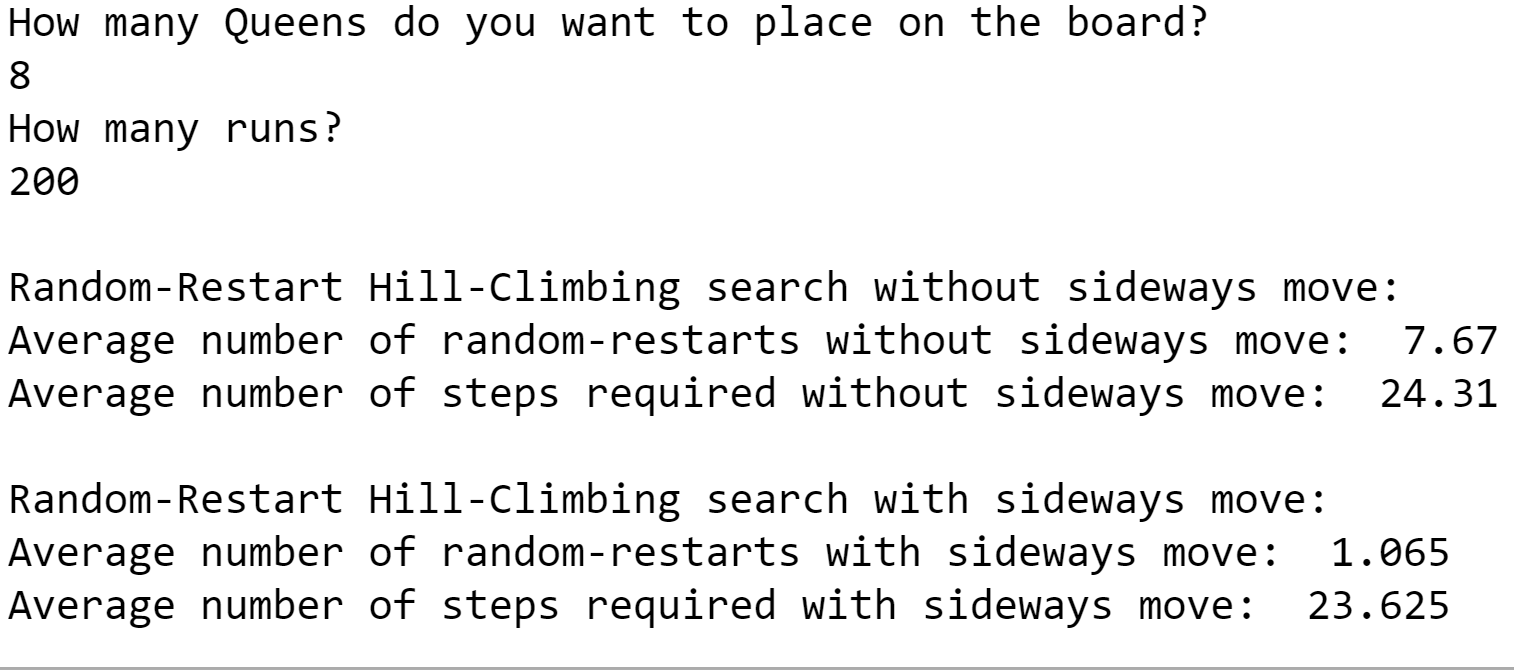
* Without sideways move, takes roughly 7 iterations to find a goal and around 22 steps in all.
* With sideways move, 1.06 iterations are needed on average to find a goal and around 25 steps in all.

Actual Output:

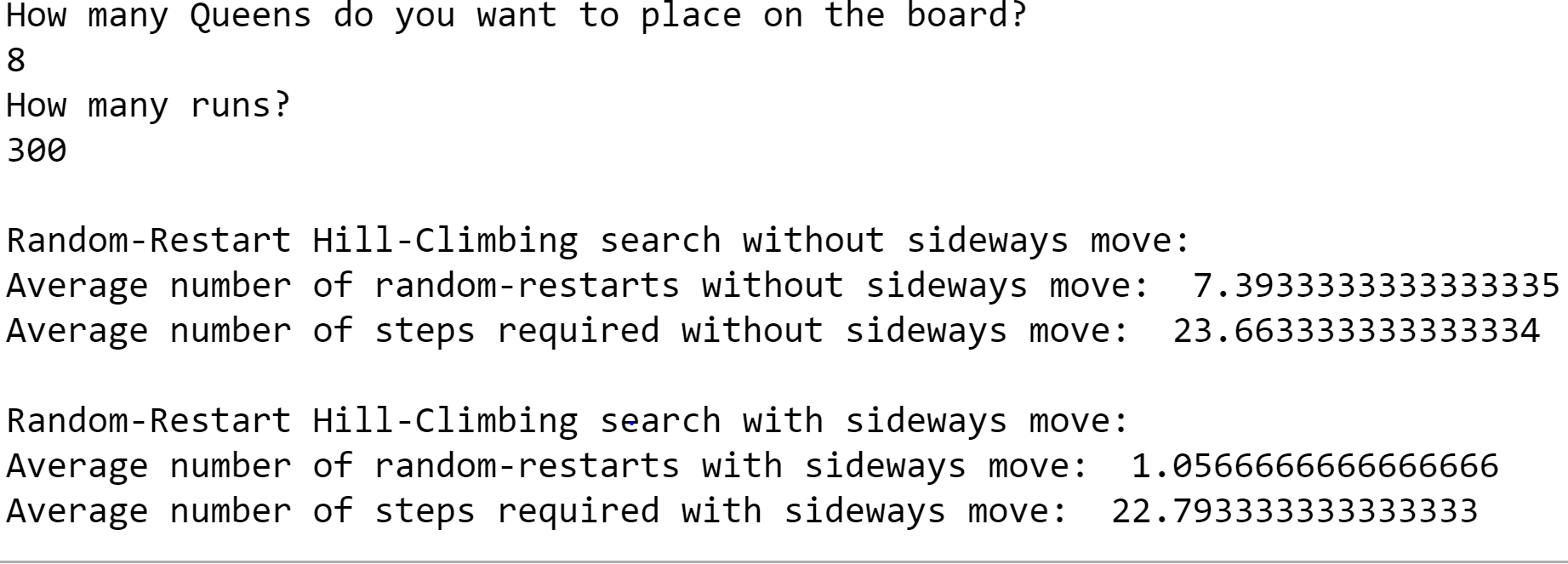
100 runs



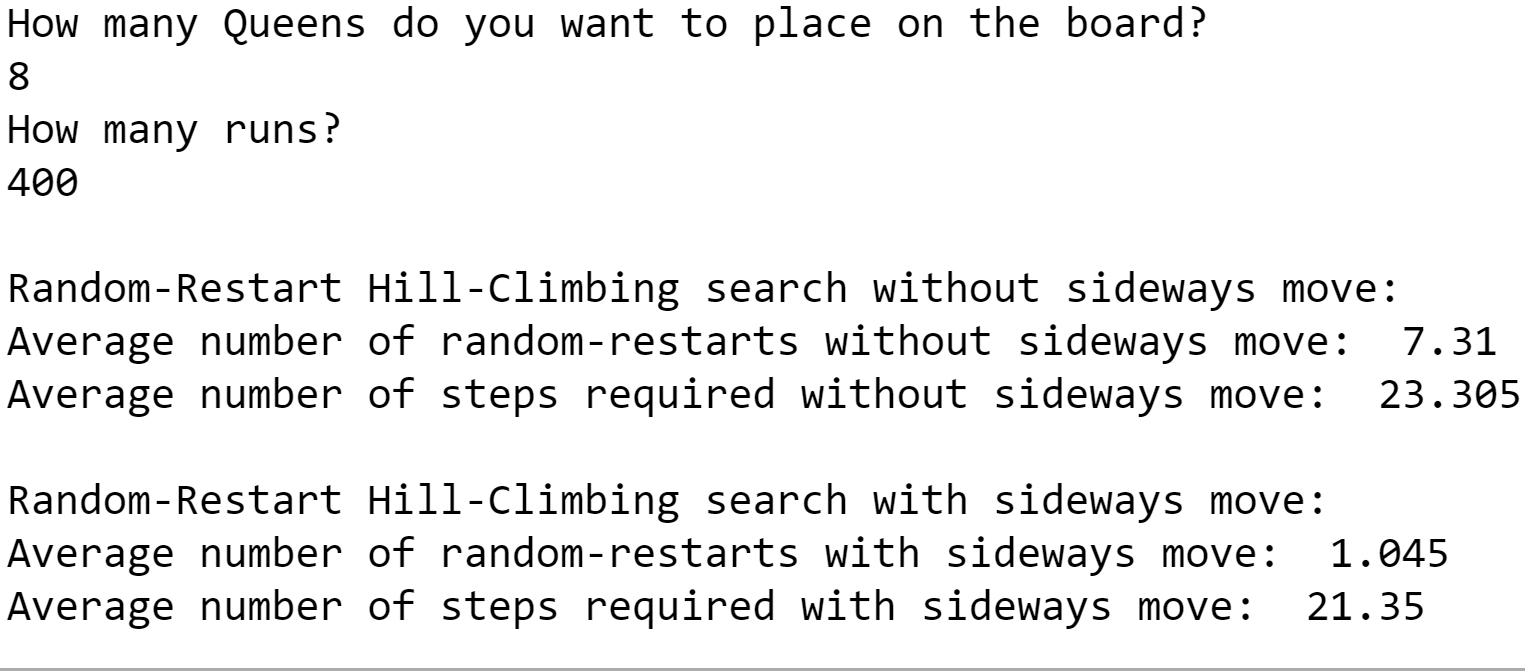
200 runs



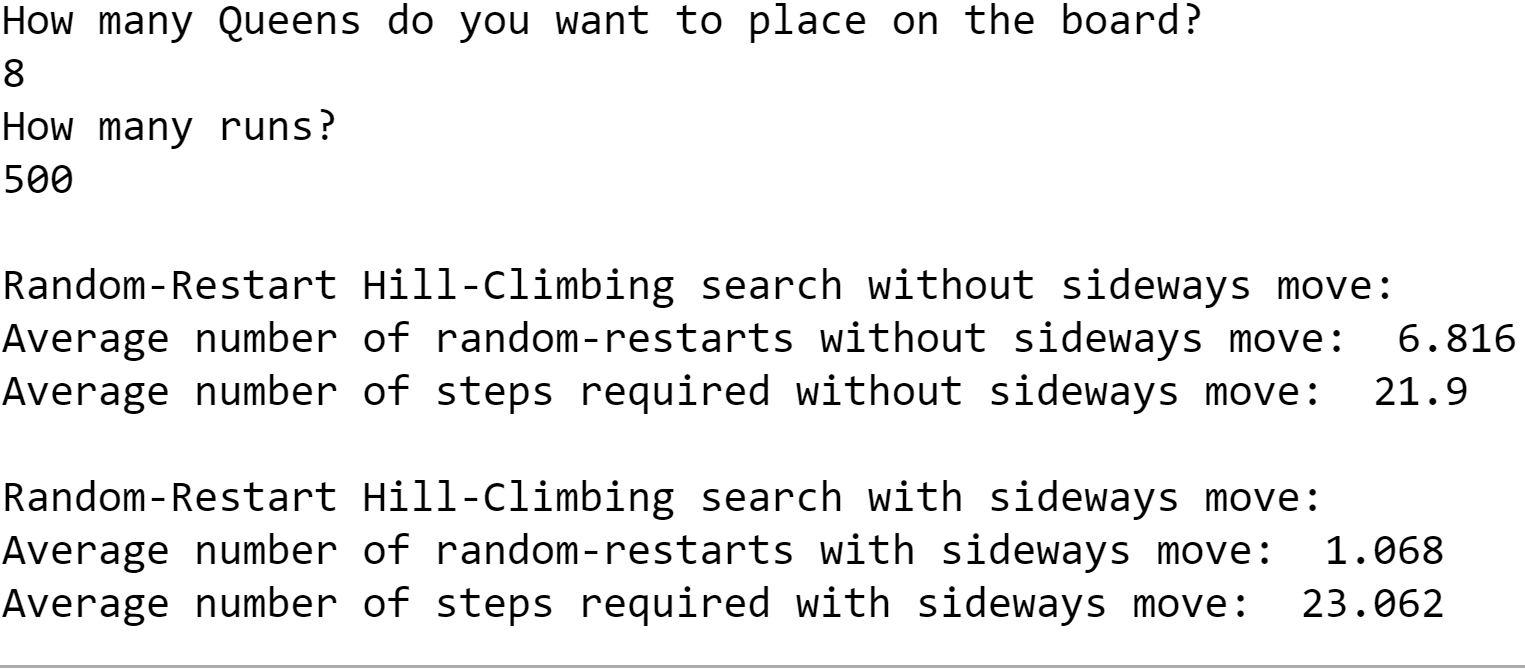
300 runs



400 runs



500 runs



Reference:

* Artificial Intelligence -A Modern Approach, Third Edition, Stuart J. Russell and Peter Norvig
* Lecture slides