# **Backtracking and Forward-Checking | N-Queens.**

Thomas Mulvey

3/11/19

Dr. Doboli

## BACKTRACKING PSEUDO-CODE

```
FUNCTION backtrack(self, col): # col is current col being checked
        IF  board size is 2 or 3:
            OUTPUT "NO SOLUTIONS EXIST FOR THE GIVEN N"
            RETURN False
        ENDIF

        IF  N=col:
            RETURN True #done
        ENDIF
        FOR r in range(0,  N): #loop over every row trying to get a valid solution
            queen_pos[col] <- r
            IF  checkPositions(r, col) = True: # current board setup is valid
                IF  backtrack(col+1) = True:
                    RETURN True
                ENDIF
            ELSE:
                 queen_pos[col] <- -2* N; #failed, go back to default value
            ENDIF
    ENDFUNCTION
```

## FORWARD-CHECKING PSEUDO-CODE

```
FUNCTION forwardtrack(self,col, domains):
        # invalid game size 2 OR 3.
        IF   N = 2 OR  N = 3:
            RETURN False
        # IF N = col, N queens finished
        ENDIF
        IF  N = col:
            RETURN True
        # IF curent columns domain size is zero, backtrack
        ENDIF
        IF len(domains[col])==0:
            RETURN False
        # while there is still a row to try in current domain
        ENDIF
        WHILE length of (domains[col]) > 0 :
            r <- (domains[col]).pop() # top of set, and pops it off
            queen_pos[col] <- r
            IF  current board setup is ok:
                # domain wipeout!
                new_domains <-  domain_wipeout(board setup)
                IF a queen domain size == 0 :
                    break
                ENDIF
                IF  forwardtrack(col+1, new_domains) = True:
                        ENDFOR
                    RETURN True
                ENDIF
            ELSE:
                queen_pos[col] <- -2* N
            ENDIF
    ENDFUNCTION
```

## CODE:

```
import time
import random

class n_queens:
    def __init__(self, N, queen_pos):
        self.N = N  # SIZE OF BOARD
        self.queen_pos = queen_pos #ARR[i] RETURNS ROW POS OF COL I QUEEN
        self.nodes_visited = 0

        # fill queen pos array
        for i in range(0,N):
            queen_pos.append(-2*N) #fill array with junk for now


    def checkPositions(self, row, col):
        # given a queen in column 'col' and in row 'row', is it valid vs the rest of the board
        row_to_check = row
        diag1_check = row - col
        diag2_check = row + col

        # loop thru queen pos array to validate positions
        for c in range(0, len(self.queen_pos) ):
            if c == col:
                continue #dont want to validate against self
            cur_row = self.queen_pos[c]
            cur_diag1 = self.queen_pos[c] - c
            cur_diag2 = self.queen_pos[c] + c
            if row_to_check == cur_row or diag1_check == cur_diag1 or diag2_check==cur_diag2:
                return False
        return True

    def print_sol(self):
        res = [ [0] * self.N for _ in range(self.N) ]
        for i in range(0, self.N):
            res[i][self.queen_pos[i]] = self.queen_pos[i]+1
        for j in range(0, self.N):
            s = ""
            for k in range(0, self.N):
                if res[j][k] == 0:
                    s += "- " #print("- ")
                else:
                    s+= str(res[j][k]) + " " #print(str(res[j][k]) + " ")
            print(s)
class backtracking(n_queens):
    def __init__(self, N, queen_pos):
        n_queens.__init__(self, N, queen_pos)

    def backtrack(self, col): # col is current col being checked
        self.nodes_visited += 1
        if self.N==2 or self.N==3:
            print("NO SOLUTIONS EXIST FOR THE GIVEN N")
            return False #no sols. exist for n=2 or 3
        if self.N==col:
            self.print_sol()
            return True #done

        for r in range(0, self.N,): #loop over every row trying to get a valid solution
            self.queen_pos[col] = r
            if self.checkPositions(r, col) == True:
                if self.backtrack(col+1) == True:
                    return True
            else:
                self.queen_pos[col] = -2*self.N; #failed, go back to def value

class forwardtracking(n_queens):
    def __init__(self, N, queen_pos):
```

```python
        n_queens.__init__(self, N, queen_pos)
        self.domains = dict()
        temp = [x for x in range(N)]
        for i in range(N):
            random.shuffle(temp)
            self.domains[i]=set(temp[:]) # domains is a dict int->set,
                # set is int of rows left in domain

    def domain_wipeout(self, old_domains, row, col):
        #1. eliminate rows
        #2. eliminate r-c, diag1
        #3. eliminate r+c, diag2

        # old_domains is a copy of current stacks current domains

        #1. rows
        for i in range(self.N):
            if i==col:
                continue
            old_domains[i] = (old_domains[i]).difference({row})


        #2. diag1, row-col
        dif = row - col
        for i in range(self.N):
            if i==col:
                continue
            old_domains[i] = (old_domains[i]).difference({i+dif})

        #3. diag2, row+col
        add = row + col
        for i in range(self.N):
            if i==col:
                continue
            old_domains[i] = (old_domains[i]).difference({add-i})

        return old_domains


    def forwardcheck(self,col, domains):
        self.nodes_visited += 1
        # invalid game size 2 or 3.
        if  self.N == 2 or self.N == 3:
            return False
        # if N == col, N queens finished
        if self.N == col:
            self.print_sol()
            return True
        # if curent columns domain size is zero, backtrack
        if len(domains[col])==0:
            return False

        # while there is still a row to try in current domain
        while (domains[col]).__len__() >0 :
            r = (domains[col]).pop()
            # assume current row is good
            self.queen_pos[col] = r
            # if valid
            if self.checkPositions(r, col) == True:
                # domain wipeout!
                temp = domains.copy()
                new_domains = self.domain_wipeout(temp, r, col )
                # if any of the domains are 0, kill stack
                valid = True
                for i in range(col+1,self.N):
                    if new_domains[i].__len__() == 0 :
                        valid = False
                if not valid:
```

```
                    break
                # print("CUR COL : " + str(col))
                if self.forwardcheck(col+1, new_domains) == True:
                    return True
            else:
                self.queen_pos[col] = -2*self.N



def main():
    print("n_queens mulvey\n")
    n = int(input("ENTER size of board ? : "))
    print("\n====!!BACKTRACKING!!====")
    start_time = time.time()
    a = backtracking(n, [])
    a.backtrack(0)
    end_time = time.time() - start_time
    print("BACKTRACKING took " + str(end_time) + "secs to run (and print) with " + str(a.nodes_visited) +
" nodes visited")
    print("\n\n====!!FORWARD-CHECING!!====")
    start_time = time.time()
    b = forwardtracking(n, [])
    b.forwardcheck(0, b.domains)
    end_time = time.time() - start_time
    print("FORWARDCHECKING took " + str(end_time) + "secs to run (and print) with " + str(b.nodes_visited)
+ " nodes visited")
    return

#-----#
main()
```

# SAMPLE OUTPUTS

```
tom@tmulvey-LT0:/mnt/c/Users/tmulvey/Documents/skool/cs_ai/hw2$ python3 hw2.py
n_queens mulvey

ENTER size of board ? : 10

====!!BACKTRACKING!!====
1 - - - - - - - - -
- - - - - 6 - - - -
- - - - - - - 8 - -
- 2 - - - - - - - -
- - - - - - 7 - - -
- - - - - - - - 9 -
- - 3 - - - - - - -
- - - - 5 - - - - -
- - - - - - - - - 10
- - - 4 - - - - - -
BACKTRACKING took 0.011760234832763672secs to run (and print) with 148 nodes visited


====!!FORWARD-CHECING!!====
- - - - - - - 8 - -
- - 3 - - - - - - -
- - - - - - - - 9 -
- - - 4 - - - - - -
- - - - - - - - - 10
1 - - - - - - - - -
- - - - - 6 - - - -
- 2 - - - - - - - -
- - - - 5 - - - - -
- - - - - - 7 - - -
FORWARDCHECKING took 0.005118131637573242secs to run (and print) with 72 nodes visited
```

```
tom@tmulvey-LT0:/mnt/c/Users/tmulvey/Documents/skool/cs_ai/hw2$ python3 hw2.py
n_queens mulvey

ENTER size of board ? : 20

====!!BACKTRACKING!!====
1 - - - - - - - - - - - - - - - - - - -
- - 3 - - - - - - - - - - - - - - - - -
- - - - 5 - - - - - - - - - - - - - - -
- 2 - - - - - - - - - - - - - - - - - -
- - - - - - - - 9 - - - - - - - - - - -
- - - - - - - - - - 11 - - - - - - - - -
- - - - - - - - - - - - - - 15 - - - - -
- - - - - - - - - - - - - - - - 17 - - -
- - - - - - - - - - - - - - - - - - 19 -
- - - - - - - - - - - - - - - 16 - - - -
- - - - - - - - - - - - - - - - - 18 - -
- - - - - 6 - - - - - - - - - - - - - -
- - - - - - - 8 - - - - - - - - - - - -
- - - - - - - - - - - - 13 - - - - - - -
- - - 4 - - - - - - - - - - - - - - - -
- - - - - - 7 - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - 20
- - - - - - - - - - - - - 14 - - - - - -
- - - - - - - - - - - 12 - - - - - - - -
- - - - - - - - - 10 - - - - - - - - - -
BACKTRACKING took 2.160827875137329secs to run (and print) with 24814 nodes visited


====!!FORWARD-CHECING!!====
1 - - - - - - - - - - - - - - - - - - -
- - 3 - - - - - - - - - - - - - - - - -
- - - - 5 - - - - - - - - - - - - - - -
- - - - - - - - - 10 - - - - - - - - - -
- - - - - - - - - - - 12 - - - - - - - -
- - - - - - - - - - - - - - - 16 - - - -
- - - - - - - - - - 11 - - - - - - - - -
- - - - - - - - - - - - - - - - - - - 20
- - - - - - - - - - - - - - - - 17 - - -
- - - - - - - - - - - - - - - - - - 19 -
- - - - - - - - 9 - - - - - - - - - - -
- - - - - - - - - - - - - - 15 - - - - -
- - - - - - - - - - - - - - - - - 18 - -
- 2 - - - - - - - - - - - - - - - - - -
- - - - - 6 - - - - - - - - - - - - - -
- - - - - - - 8 - - - - - - - - - - - -
- - - - - - - - - - - - 13 - - - - - - -
- - - - - - 7 - - - - - - - - - - - - -
- - - 4 - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - 14 - - - - - -
FORWARDCHECKING took 0.13434100151062012secs to run (and print) with 1020 nodes visited
```

```
ENTER size of board ? : 30

====!!BACKTRACKING!!====
1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - 3 - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - 5 - - - - - - - - - - - - - - - - - - - - - - - - -
- 2 - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - 4 - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - 9 - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - 11 - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - 13 - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - 15 - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - 18 - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - 23 - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - 25 - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - 27 - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - 29 -
- - - - - - - - - - - - - - - - - - - - - - - 24 - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - 26 - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - 28 - -
- - - - - - 7 - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - 12 - - - - - - - - - - - - - - - - - -
- - - - - 6 - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - 17 - - - - - - - - - - - - -
- - - - - - - - - - - - - 14 - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - 30
- - - - - - - 8 - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - 10 - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - 20 - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - 22 - - - - - - - -
- - - - - - - - - - - - - - - 16 - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - 19 - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - 21 - - - - - - - - -
BACKTRACKING took 624.1868057250977secs to run (and print) with 4164548 nodes visited
====!!FORWARD-CHECING!!====
1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - 3 - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - 5 - - - - - - - - - - - - - - - - - - - - - - - - -
- 2 - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - 4 - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - 11 - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - 24 - - - - - -
- - - - - - - - - - - - - - - - - - 19 - - - - - - - - - - -
- - - - - - - - - - - - 13 - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - 16 - - - - - - - - - - - - - -
- - - - - - - - - - - - - 14 - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - 25 - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - 27 - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - 29 -
- - - - - - - - - - - - - - - - - - - - - - 23 - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - 28 - -
- - - - - - 7 - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - 15 - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - 26 - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - 30
- - - - - - - - 9 - - - - - - - - - - - - - - - - - - - - -
- - - - - 6 - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - 12 - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - 18 - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - 21 - - - - - - - - -
- - - - - - - 8 - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - 22 - - - - - - - -
- - - - - - - - - - - - - - - - - - - 20 - - - - - - - - - -
- - - - - - - - - 10 - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - 17 - - - - - - - - - - - - -
FORWARDCHECKING took 1.6253600120544434secs to run (and print) with 20797 nodes visited
```

```
ENTER size of board ? : 40


====!!FORWARD-CHECING!!====
1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - 3 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - 5 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- 2 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - 4 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - 9 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - 11 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - 13 - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - 15 - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - 20 - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 33 - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 38 - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - 28 - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 34 - - - - - -
- - - - - - - - - - - - - - - - - - - - - 22 - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - 27 - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 36 - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 31 - - - - - - - - -
- - - - - - - - - - - - - 14 - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 37 - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 35 - - - - -
- - - - - - - - - - - 12 - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 32 - - - - - - - -
- - - - - - - - - - - - - - - - - - - - 21 - - - - - - - - - - - - - - - - - - -
- - - - - - - - - 10 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - 6 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 39 -
- - - - - - - - - - - - - - - - - - 19 - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - 17 - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - 24 - - - - - - - - - - - - - - - -
- - - - - - 7 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 40
- - - - - - - - - - - - - - - - - - - - - - - - - - - - 29 - - - - - - - - - - -
- - - - - - - 8 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - 16 - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - 23 - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - 26 - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - 30 - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - 25 - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - 18 - - - - - - - - - - - - - - - - - - - - - -
FORWARDCHECKING took 25.280693769454956secs to run (and print) with 176500 nodes visited
```

**COMPARISONS:**

For Backtracking vs Forward-Checking, I only ran tests for sizes 10, 20, and 30. Since backtracking took over 10 minutes for N=30, I decided that was enough of that. I ran My Forward-Checking for N=40 and N=50. Here is a table of N, Time Elapsed, and Nodes Visited (with a 1 CPU core and 0.5GB RAM VM).

| ALGORITHM | SIZE, N | TIME RAN (S) | NODES VISITED |
|---|---|---|---|
| BackTracking | 10 | 0.011760235 | 148 |
| Forward-Checking | 10 | 0.005118132 | 72 |
| BackTracking | 20 | 2.160827875 | 24814 |
| Forward-Checking | 20 | 0.134341002 | 1020 |
| BackTracking | 30 | 624.1868057 | 4164548 |
| Forward-Checking | 30 | 1.625360012 | 20797 |
| Forward-Checking | 40 | 25.28069377 | 176500 |
| Forward-Checking | 50 | 2878.868532 | 15426815 |

For smaller Ns, N<10, Backtracking seems to visit about 2x as many nodes as the forward-checking will. When N got a little larger, 20, backtracking visited 24x more nodes than forward checking. At the largest tested N, 30, backtracking visited over **200x** more nodes than its counterpart! With my forward-checking algorithm, 40x40 board seemed to be the limit because N=50 took 48 minutes to run and visited 15,426,815 nodes. The total # of possible nodes for a NxN board is $1 + n + n^2 + n^3 + \ldots + n^n = \frac{n^{n+1}-1}{n-1}$. Considering for N=50, there are 9.0630451*10^84 nodes, and only 1.54x10^7 were visited, that is pretty good, but can be improved upon.

**CONCLUSION:**

Forward checking was always faster than backtracking due to the fewer nodes it had to visit. Much more calculations had to be done for forward-checking, but they obviously paid off as you can see in the chart above. Backtracking basically brute forced its way through the NxN chessboard, while forward-checking did the same but was able to tell when a future Queen would have nowhere to go, and it would go back before going through many other possibilities where that same queen would also fail. The forward checking can still be improved upon by implementing a Priority Queue of the domain lengths of each queen. The priority queue would try the smallest domain (most constrained variable). Within that domain, it would choose the value/row that would eliminate the least amount of values from other domains (Least constraining value heuristic). The most constrained variable and least constraining value heuristic would make the forward-checking even faster.

Both backtracking and forward-checking are exponential in terms of time, and the forward-checking with the two heuristics is also exponential time complexity. Space complexity wise, backtracking will be better off than its forward-checking counterparts because they will require extra space for domains and advanced data structures. The extra space used by forward-checking does result in many fewer nodes visited than backtracking though, and in CSP like N-queens, every solution is a valid one, so why not go get the fastest one? When implementing N-Queens recursively, it does not explicitly show the backtracking. Following the stack traces though, does show giving up on a certain queen spot and trying a different one. Like mentioned earlier, the Forward-checking can be improved upon with a PriorityQueue().

There is also another approach called "Hill Climbing" which is a very rudimentary heuristic. It goes as follows:

- Select an initial (random or pre-determined) assignment of Queens
- WHILE there are threatened queens
  - Select Random Threatened Queen
  - Move that queen to another row that minimizes conflicts

The heuristic is # of queens threatening that piece, and obviously the lower the better. This is much faster than the forward checking or backtracking but is a heuristic-based approach. Russel and Norvig claim in their book "Artificial Intelligence: A Modern Approach" that they had an average of 50 moves for 1 million queens with this method. I would assume they used more efficient methods (for example, break a recursive stack after x iterations of trying to move the queen), but this method would be the fastest to solve many queens.

My forward-checking algorithm was not too great, and it would probably benefit greatly with a priority queue implemented with the two heuristics. It would still be slower than hill climbing. Hill climbing will "pick best random" and forward-checking will brute-force, but take some of the branching depth and width down. A simple Hill-Climbing algorithm for me was able to perform N=50 in 27 seconds and N=100 in 521 seconds, so not as good as Russel and Norvig's implementation.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 0 0 0 0 0 1 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 1 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 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 0 0 0 0 1 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

Total number of Steps Climbed: 4871
Number of random restarts: 217
Steps Climbed after last restart: 25
Took 27 Secs to run
```

```
0 0 0 0 0 0 0 0 0 0 0 0 1 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 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 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 1 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 0 0 0 0 0 0 0 0 0 0

Total number of Steps Climbed: 5854
Number of random restarts: 130
Steps Climbed after last restart: 51
Took 521 Secs to run
```