SSN College of Engineering

Department of Computer Science and Engineering

CS1504 — Artificial Intelligence

2021 – 2022

Assignment — 04 (Additional)
(State Space Search — Mobile Robot)

August 25, 2021

Problem Statement

Solve 8-queens problem. Place 8 queens on a chessboard so that no queen is under attack from any other queen. Implement Hill climbing algorithm to find any one safe configuration

Response (*Link to* ***repl.it*** *implementation repository* *)*

State Space Formulation

State: An arrangement of all 8 queens on the board, one per column

State Representation: An 8-element tuple. Each value represents the row index of the queen's position in that column

Initial State: Randomly generated arrangement of 8 queens, one per column

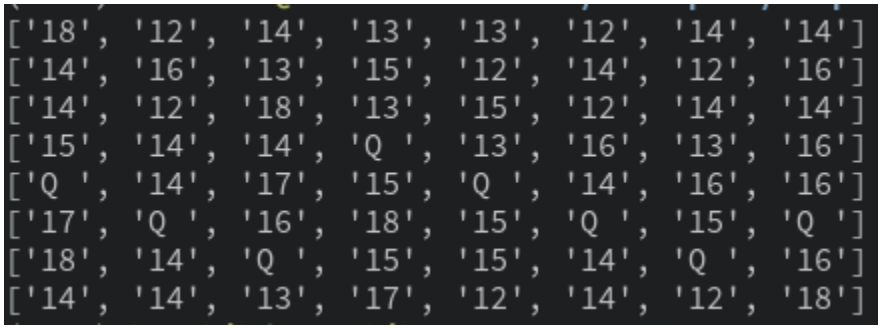
Actions: Move a queen from one row to another within its column

Heuristic / Cost: No. of pairs of attacking queens

Goal State: No attacking pairs of queens

Sample Case

No. of Attacking Pairs shown for each successor state



Python Program Code

1. StateFormulation.py - Script to formulate the state space and instantiate a problem case

```
import numpy.random as random
from copy import deepcopy

NUM_QUEENS = 8
MAX_POSSIBLE_ATTACKS =  ( NUM_QUEENS * (NUM_QUEENS-1) ) // 2  # i.e nC2

def generate_random_state():
    state = [
        random.randint(0, NUM_QUEENS)
        for x in range(NUM_QUEENS)
    ]
    return state

def count_attacks(state):
    num_attacks = 0
    # Count for each queen
    for column in range(NUM_QUEENS):
        # Count queens in same row. Exclude self
        num_attacks += state.count(state[column])-1
        # Try right and left diagonals. Exclude self
        num_attacks -= 2
        diag_sum = column + state[column]
        diag_diff = column - state[column]
        for i in range(NUM_QUEENS):
            if i+state[i]==diag_sum:
                num_attacks += 1
            if i-state[i]==diag_diff:
                num_attacks += 1
    # Each pair-attack was counted twice. Hence, return result/2
    return num_attacks//2

# Find the next states
def get_next_states(state, display=False):
    moves = list()
```

```
    attacks = list()
    if display:
        attacks_array = [
            [ 'Q ' if state[col]==row else None for col in
range(NUM_QUEENS) ]
            for row in range(NUM_QUEENS)
        ]
    # Try moving each queen to every other position in its column
    for column in range(NUM_QUEENS):
        for row in range(NUM_QUEENS):
            if row == state[column]:
                # If moving again to same row, skip
                continue
            else:
                moves.append((column, row))
                temp_state = deepcopy(state)
                temp_state[column] = row
                num_attacks = count_attacks(temp_state)
                attacks.append(num_attacks)
                if display:
                    attacks_array[row][column] =
str(num_attacks).ljust(2)
    if display:
        for disp_row in attacks_array:
            print(disp_row)
    return moves, attacks

# Find the lowest attack successor state
def get_next_best_move(state):
    min_attacks = MAX_POSSIBLE_ATTACKS
    min_attacks_move = None
    # Try moving each queen to every other position in its column
    for column in range(NUM_QUEENS):
        for row in range(NUM_QUEENS):
            if row == state[column]:
                # If moving again to same row, skip
                continue
            temp_state = deepcopy(state)
            temp_state[column] = row
            num_attacks = count_attacks(temp_state)
```

```
            if num_attacks<min_attacks:
                min_attacks_move = (column, row)
                min_attacks = num_attacks
    # Return the best move and its cost value
    return min_attacks_move, min_attacks

# Display the state, visually
def display_state(state):
    disp_array = [
        [ 'Q' if state[col]==row else '-' for row in
range(NUM_QUEENS) ]
        for col in range(NUM_QUEENS)
    ]
    for disp_row in disp_array:
        print(disp_row)

# Testing state generation
"""
sample_state = [4, 5, 6, 3, 4, 5, 6, 5]
print(get_next_states(sample_state))
"""
sample_state = [4, 5, 6, 3, 4, 5, 6, 5]
get_next_states(sample_state, display=True)
```

2. HillClimbing.py - Script to perform hill-climbing search

```
# A random-restart version of the Hill-Climbing search algorithm
# No sideways moves are allowed. If plateau is reacheed, restart is
applied

from StateFormulation import *

def search():
    restarts = 0
    while True:
        state = generate_random_state()
        # Reset at each restart
        transitions = [ state ]
        while True:
```

```
            curr_attacks = count_attacks(state)
            if curr_attacks == 0:
                # Goal reached
                return state, transitions, restarts
            # Generate next best state
            move, next_attacks = get_next_best_move(state)
            if next_attacks >= curr_attacks:
                # At some local maxima or plateau
                # Restart search
                restarts += 1
                break
            # Move to the best successor state
            in_col, to_row = move
            state[in_col] = to_row
            # Add this new state to set of transitions
            transitions.append(state)
        # Restart search with new random start
        state = generate_random_state()
```

3. main.py - Driver program to implement and summarize the search technique

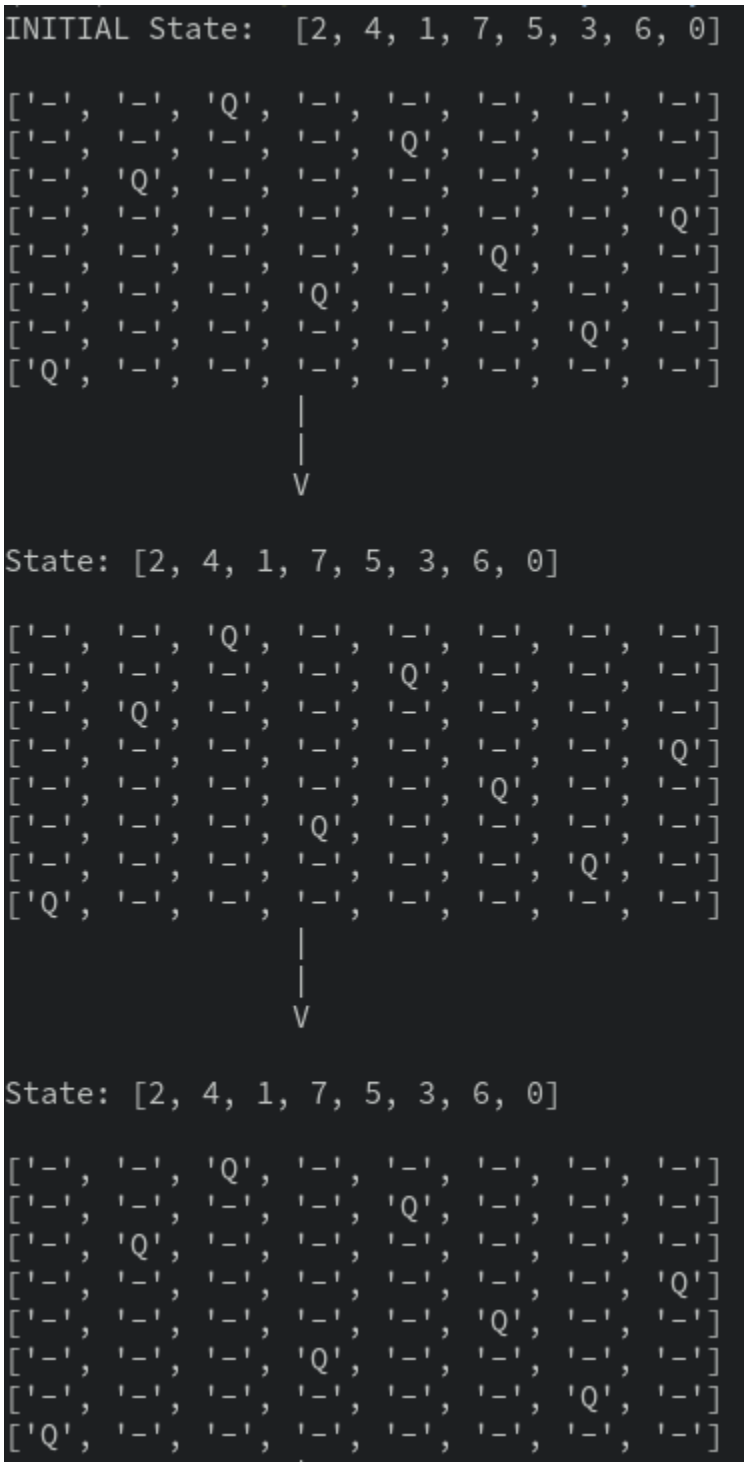
```
from HillClimbing import search as HC_search
from StateFormulation import display_state

goal, transitions, restarts = HC_search()

print("INITIAL State: ", transitions[0])
print()
display_state(transitions[0])
for state in transitions[1:-1]:
   print("\t\t|\n\t\t|\n\t\tV\n")
   print("State:", state)
   print()
   display_state(state)
print("\t\t|\n\t\t|\n\t\tV\n")
print("\nGOAL State:", transitions[-1])
display_state(goal)

print("\nNumber of Restarts:", restarts)
```

Sample Output




|
|
V
GOAL State: [2, 4, 1, 7, 5, 3, 6, 0]
['-', '-', 'Q', '-', '-', '-', '-', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']
['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['Q', '-', '-', '-', '-', '-', '-', '-']
Number of Restarts: 4