SSN College of Engineering

Department of Computer Science and Engineering

CS1504 — Artificial Intelligence

2021 – 2022

Assignment — 04 (Additional)
(State Space Search — Genetic Algorithm)

September 11, 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 the Genetic Algorithm to find any one such 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

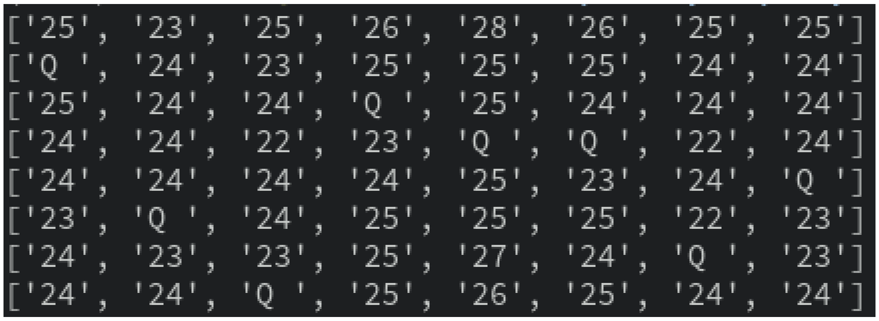
Actions: Move a queen from one row to another within its column

Heuristic / Cost: No. of pairs of attacking queens

Goal State: No attacking pairs of queens

Sample Case

No. of Non-Attacking Pairs shown for each successor state



```
['25', '23', '25', '26', '28', '26', '25', '25']
['Q ', '24', '23', '25', '25', '25', '24', '24']
['25', '24', '24', 'Q ', '25', '24', '24', '24']
['24', '24', '22', '23', 'Q ', 'Q ', '22', '24']
['24', '24', '24', '24', '25', '23', '24', 'Q ']
['23', 'Q ', '24', '25', '25', '25', '22', '23']
['24', '23', '23', '25', '27', '24', 'Q ', '23']
['24', '24', 'Q ', '25', '26', '25', '24', '24']
```

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
MIN_POSSIBLE_ATTACKS = 0

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
```

```
def count_non_attacks(state):
    return MAX_POSSIBLE_ATTACKS - count_attacks(state)


# Find the next states
def get_next_states(state):
    moves = list()
    attacks = list()
    disp_arr = [[None for x in range(NUM_QUEENS)] for y 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
                disp_arr[row][column] = 'Q '
                continue
            else:
                temp_state = deepcopy(state)
                temp_state[column] = row
                disp_arr[row][column] =
str(count_non_attacks(temp_state)).ljust(2)
    for i in disp_arr:
        print(i)
    return moves, attacks


def is_goal_reached(state):
    return count_attacks(state)==0


# 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]
sample_state = [1, 5, 7, 2, 3, 3, 6, 4]
print(get_next_states(sample_state))
"""
```

2. GeneticAlgorithm.py - Script to perform genetic alogorithm search

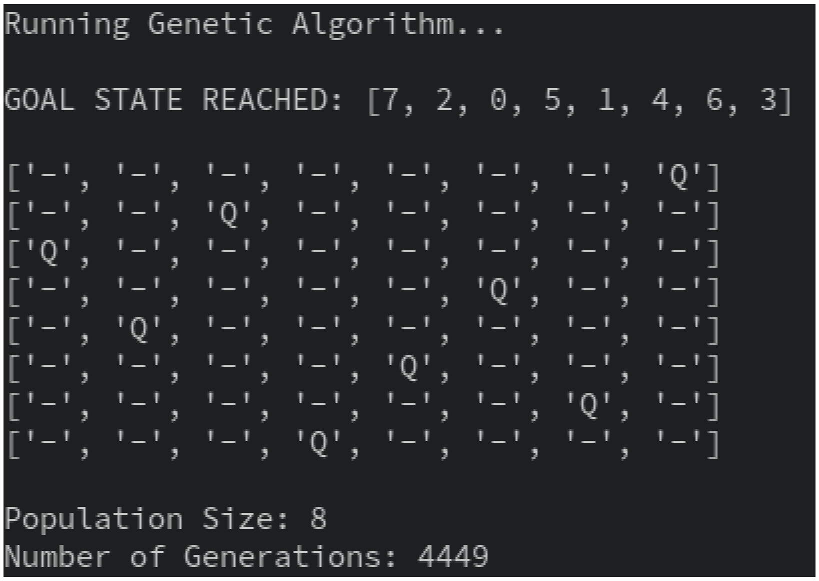
```
from copy import copy
from StateFormulation import *
import numpy as np

def sample_population(population):
   num_draws = len(population)
   # Create array with elements as per probability of choice
   population_fitness = [ count_non_attacks(state)
                           for state in population ]
   # Create a population sample
   population_sample = []
   for idx, state in enumerate(population):
       population_sample.extend([
           state for k in range(population_fitness[idx])
           ])
   np.random.shuffle(population_sample)
   # Select parents and pair them up
   population_idx = [ x for x in range(len(population_sample)) ]
   parent_pairs_idx = np.random.choice(population_idx, num_draws)
   parent_pairs = [ population_sample[idx] for idx in
parent_pairs_idx ]
   parent_pairs = [ (parent_pairs[i], parent_pairs[i+1])
                       for i in range(num_draws//2) ]
   return parent_pairs



def crossover_pair(pair, population_size):
   crossover_idx = np.random.randint(1, population_size-1)
   result = []
   parent_1, parent_2 = pair
```

```
    result.append(parent_1[:crossover_idx] +
parent_2[crossover_idx:])
    result.append(parent_2[:crossover_idx] +
parent_1[crossover_idx:])
    return tuple(result)


def mutate_state(state):
    mutation_idx = np.random.randint(0, len(state)+1)
    if mutation_idx == len(state):
        # No mutation
        pass
    else:
        mutated_state = np.random.randint(0, NUM_QUEENS)
        state[mutation_idx] = mutated_state
    return state

def search(population_size=8):

    # Check if the population has a goal state
    def scan_goal_state(population):
        for state in population:
            if is_goal_reached(state):
                return True, state
        return False, None

    # Generate random population
    population = [ generate_random_state() for k in
range(population_size) ]

    # Get parents, crossover and mutate till goal is generated
    num_generations = 0
    has_goal_state, goal_state = scan_goal_state(population)
    while not has_goal_state:
        num_generations += 1
        # Select parent-pairs from population as per probability of
selection
        parent_pairs = sample_population(population)
        # Generate descendant population
        population_next = list()
        for pair in parent_pairs:
            population_next.extend(list(map(mutate_state,
crossover_pair(pair, population_size))))
        population = copy(population_next)
        has_goal_state, goal_state = scan_goal_state(population)

    return goal_state, num_generations
```

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

```
from GeneticAlgorithm import search as GA_search
from StateFormulation import display_state

population_size = 8
print("Running Genetic Algorithm...")
goal_state, num_generations =
GA_search(population_size=population_size)
print("\nGOAL STATE REACHED:", goal_state)
print()
display_state(goal_state)
print("\nPopulation Size:", population_size)
print("Number of Generations:", num_generations)
```

Sample Output



```
Running Genetic Algorithm...

GOAL STATE REACHED: [7, 2, 0, 5, 1, 4, 6, 3]

['-', '-', '-', '-', '-', '-', '-', 'Q']
['-', '-', 'Q', '-', '-', '-', '-', '-']
['Q', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'Q', '-', '-']
['-', 'Q', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', 'Q', '-', '-', '-']
['-', '-', '-', '-', '-', '-', 'Q', '-']
['-', '-', '-', 'Q', '-', '-', '-', '-']

Population Size: 8
Number of Generations: 4449
```