```
# CAP 6635 Artificial Intelligence
# Local search (Genetic Algorithm to solve N-Queens problem)
# X. Zhu, June 20 2023
# Code adapted from: https://github.com/waqqasiq/n-queen-problem-using-genetic-algorithm/blob/master/N-Queen_GeneticAlgo.py

# Modified by Matthew Acs for HW 2


# Propose a solution to generate off-spring whose genetic code is a permutation [0.25]
# Implement your algorithm and compare its performance vs. the original genetic
# algorithm to solve N=16 queens problem (repeat 5 times and report average runtime).


# Code modified to include create only offsprings that are permutations
# Code modified to use swap mutations to preserve valid permutations
# Code modified to repeat original and genetic algorithms for 5 iterations
# Code modified to take average runtime of 5 iterations
```

Original Genetic Algorithm

```
import random
import time

#geneating random chromosomes
def random_chromosome(size):
    return [ random.randint(1, nq) for _ in range(nq) ]

#fitness function, calculating number of queen pairs not attacking each other.
def fitness(chromosome):
    horizontal_collisions = sum([chromosome.count(queen)-1 for queen in chromosome])/2
    diagonal_collisions = 0

    n = len(chromosome)
    left_diagonal = [0] * 2*n
    right_diagonal = [0] * 2*n
    for i in range(n):
        left_diagonal[i + chromosome[i] - 1] += 1
        right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

    diagonal_collisions = 0
    for i in range(2*n-1):
        counter = 0
        if left_diagonal[i] > 1:
            counter += left_diagonal[i]-1
        if right_diagonal[i] > 1:
            counter += right_diagonal[i]-1
        diagonal_collisions += counter / (n-abs(i-n+1))

    return int(maxFitness - (horizontal_collisions + diagonal_collisions)) #28-(2+3)=23

def probability(chromosome, fitness):
    return fitness(chromosome) / maxFitness

def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for c, w in populationWithProbabilty)
    r = random.uniform(0, total)
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
            return c
        upto += w
    assert False, "Shouldn't get here"

def reproduce(x, y): #doing cross_over between two chromosomes
    n = len(x)
    c = random.randint(0, n - 1)
    return x[0:c] + y[c:n]

def mutate(x):  #randomly changing the value of a random index of a chromosome
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(1, n)
```

```
        m = random.randint(1, n)
        x[c] = m
        return x

    def genetic_queen(population, fitness):
        mutation_probability = 0.05
        new_population = []
        probabilities = [probability(n, fitness) for n in population]
        for i in range(len(population)):
            x = random_pick(population, probabilities) #best chromosome 1
            y = random_pick(population, probabilities) #best chromosome 2
            child = reproduce(x, y) #creating two new chromosomes from the best 2 chromosomes
            if random.random() < mutation_probability:
                child = mutate(child)
            print_chromosome(child)
            new_population.append(child)
            if fitness(child) == maxFitness: break
        return new_population

    def print_chromosome(chrom):
        print("Chromosome = {},  Fitness = {}"
            .format(str(chrom), fitness(chrom)))

    average_runtime = []
    nq = int(input("Enter Number of Queens: ")) #say N = 8
    for i in range(5):
      if __name__ == "__main__":
          maxFitness = (nq*(nq-1))/2  # 8*7/2 = 28
          population = [random_chromosome(nq) for _ in range(100)]
          print("Maximum fitness = {}" .format(str(maxFitness)))

          generation = 1

          start = time.time()
          while not maxFitness in [fitness(chrom) for chrom in population]:
              print("=== Generation {} ===".format(generation))
              population = genetic_queen(population, fitness)
              print("")
              print("Maximum Fitness = {}".format(max([fitness(n) for n in population])))
              generation += 1
          chrom_out = []
          print("Solved in Generation {}!".format(generation-1))
          print("Runtime: " + str((time.time() - start)))
          average_runtime.append((time.time() - start))
          for chrom in population:
              if fitness(chrom) == maxFitness:
                  print("");
                  print("One of the solutions: ")
                  chrom_out = chrom
                  print_chromosome(chrom)

          board = []

          for x in range(nq):
              board.append(["x"] * nq)

          for i in range(nq):
              board[nq-chrom_out[i]][i]="Q"


          def print_board(board):
              for row in board:
                  print (" ".join(row))

          print()
          print_board(board)

    print()
    print("Average Runtime")
    print(sum(average_runtime) / len(average_runtime))
```

```
Chromosome = [1, 3, 7, 3, 2, 4, 6, 2],  Fitness = 25
Chromosome = [5, 6, 8, 3, 3, 1, 2, 4],  Fitness = 26
Chromosome = [5, 6, 7, 3, 5, 1, 2, 5],  Fitness = 23
Chromosome = [5, 6, 2, 3, 7, 1, 6, 2],  Fitness = 25
Chromosome = [1, 6, 7, 3, 5, 1, 2, 5],  Fitness = 24
Chromosome = [1, 4, 2, 3, 8, 4, 6, 2],  Fitness = 25
Chromosome = [5, 6, 2, 3, 2, 1, 1, 8],  Fitness = 25
Chromosome = [1, 6, 8, 3, 5, 1, 4, 8],  Fitness = 25
Chromosome = [1, 3, 7, 4, 7, 4, 2, 8],  Fitness = 25
Chromosome = [1, 4, 7, 7, 3, 1, 2, 8],  Fitness = 25
Chromosome = [1, 4, 7, 3, 5, 1, 2, 8],  Fitness = 26
Chromosome = [5, 6, 8, 3, 5, 4, 2, 8],  Fitness = 25
Chromosome = [1, 6, 2, 3, 8, 1, 2, 6],  Fitness = 24
Chromosome = [5, 6, 8, 3, 7, 1, 4, 8],  Fitness = 26
Chromosome = [5, 6, 8, 3, 7, 4, 4, 8],  Fitness = 25
Chromosome = [1, 6, 2, 3, 2, 4, 2, 8],  Fitness = 24
Chromosome = [5, 4, 7, 3, 2, 4, 2, 8],  Fitness = 25
Chromosome = [5, 6, 6, 3, 5, 1, 4, 8],  Fitness = 25
Chromosome = [1, 4, 8, 3, 3, 6, 2, 8],  Fitness = 25
Chromosome = [1, 6, 8, 3, 8, 4, 6, 2],  Fitness = 25
Chromosome = [1, 4, 7, 3, 7, 4, 2, 8],  Fitness = 25
Chromosome = [1, 6, 8, 3, 8, 1, 2, 8],  Fitness = 23
Chromosome = [1, 4, 6, 3, 2, 1, 1, 8],  Fitness = 24
Chromosome = [1, 3, 7, 3, 2, 1, 6, 5],  Fitness = 25
Chromosome = [5, 4, 7, 4, 2, 4, 2, 5],  Fitness = 22
Chromosome = [1, 6, 5, 3, 2, 6, 6, 2],  Fitness = 23
Chromosome = [5, 4, 6, 3, 8, 1, 6, 2],  Fitness = 26
Chromosome = [1, 4, 8, 3, 5, 4, 6, 8],  Fitness = 25
Chromosome = [5, 4, 8, 3, 5, 4, 6, 2],  Fitness = 25
Chromosome = [5, 4, 8, 3, 2, 6, 6, 2],  Fitness = 25
Chromosome = [5, 4, 8, 3, 2, 4, 6, 4],  Fitness = 24
Chromosome = [7, 4, 7, 4, 2, 6, 4, 7],  Fitness = 21
Chromosome = [1, 6, 8, 3, 7, 4, 2, 5],  Fitness = 28

Maximum Fitness = 28
Solved in Generation 98!
Runtime: 3.840564250946045

One of the solutions:
Chromosome = [1, 6, 8, 3, 7, 4, 2, 5],  Fitness = 28

x x Q x x x x x
x x x x Q x x x
x Q x x x x x x
x x x x x x x Q
x x x x x Q x x
x x x Q x x x x
x x x x x x Q x
Q x x x x x x x

Average Runtime
273.0827447891235
```

Permutation Genetic Algorithm

```
import random
import time

#geneating random chromosomes
def random_chromosome(size):
    return [ random.randint(1, nq) for _ in range(nq) ]

#fitness function, calculating number of queen pairs not attacking each other.
def fitness(chromosome):
    horizontal_collisions = sum([chromosome.count(queen)-1 for queen in chromosome])/2
    diagonal_collisions = 0

    n = len(chromosome)
    left_diagonal = [0] * 2*n
    right_diagonal = [0] * 2*n
    for i in range(n):
        left_diagonal[i + chromosome[i] - 1] += 1
        right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

    diagonal_collisions = 0
    for i in range(2*n-1):
        counter = 0
        if left_diagonal[i] > 1:
```

```
            counter += left_diagonal[i]-1
        if right_diagonal[i] > 1:
            counter += right_diagonal[i]-1
        diagonal_collisions += counter / (n-abs(i-n+1))

    return int(maxFitness - (horizontal_collisions + diagonal_collisions)) #28-(2+3)=23

def probability(chromosome, fitness):
    return fitness(chromosome) / maxFitness

def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for c, w in populationWithProbabilty)
    r = random.uniform(0, total)
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
            return c
        upto += w
    assert False, "Shouldn't get here"

def reproduce(x, y): #doing cross_over between two chromosomes, modified to only have valid permutations
    n = len(x)
    c = random.randint(0, n - 1)

    new = x[0:c] + y[c:n]
    pool = [i for i in range(1, n+1)]
    indices = []

    for i in range(n):
      if new[i] in pool:
        pool.remove(new[i])
      else:
        indices.append(i)

    for i in range(len(indices)):
      new[indices[i]] = pool.pop(0)

    return new

def mutate(x):  #swaps the values of two indices randomly, modified to preserve permutation integrity
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(0, n-1)
    temp = x[c]
    x[c] = x[m]
    x[m] = temp
    return x

def genetic_queen(population, fitness):
    mutation_probability = 0.05
    new_population = []
    probabilities = [probability(n, fitness) for n in population]
    for i in range(len(population)):
        x = random_pick(population, probabilities) #best chromosome 1
        y = random_pick(population, probabilities) #best chromosome 2
        child = reproduce(x, y) #creating two new chromosomes from the best 2 chromosomes
        if random.random() < mutation_probability:
            child = mutate(child)
        print_chromosome(child)
        new_population.append(child)
        if fitness(child) == maxFitness: break
    return new_population

def print_chromosome(chrom):
    print("Chromosome = {},  Fitness = {}"
        .format(str(chrom), fitness(chrom)))

average_runtime = []
nq = int(input("Enter Number of Queens: ")) #say N = 8
for i in range(5):
  if __name__ == "__main__":
      maxFitness = (nq*(nq-1))/2  # 8*7/2 = 28
      population = [random_chromosome(nq) for _ in range(100)]
      print("Maximum fitness = {}" .format(str(maxFitness)))

      generation = 1
```

```
        start = time.time()
        while not maxFitness in [fitness(chrom) for chrom in population]:
            print("=== Generation {} ===".format(generation))
            population = genetic_queen(population, fitness)
            print("")
            print("Maximum Fitness = {}".format(max([fitness(n) for n in population])))
            generation += 1
        chrom_out = []
        print("Solved in Generation {}!".format(generation-1))
        print("Runtime: " + str((time.time() - start)))
        average_runtime.append((time.time() - start))
        for chrom in population:
            if fitness(chrom) == maxFitness:
                print("");
                print("One of the solutions: ")
                chrom_out = chrom
                print_chromosome(chrom)

        board = []

        for x in range(nq):
            board.append(["x"] * nq)

        for i in range(nq):
            board[nq-chrom_out[i]][i]="Q"


        def print_board(board):
            for row in board:
                print (" ".join(row))

        print()
        print_board(board)

print()
print("Average Runtime")
print(sum(average_runtime) / len(average_runtime))
```

```
Maximum Fitness = 28
Solved in Generation 2!
Runtime: 0.08997535705566406

One of the solutions:
Chromosome = [5, 3, 1, 6, 8, 2, 4, 7],  Fitness = 28

x x x x Q x x x
x x x x x x x Q
x x x Q x x x x
Q x x x x x x x
x x x x x x Q x
x Q x x x x x x
x x x x x Q x x
x x Q x x x x x

Average Runtime
0.3648380756378174
```

Colab paid products - Cancel contracts here

✓ 4s completed at 2:31 PM