
بِسْمِ اللهِ الرَّحْمنِ الرَّحِيمِ
"In the name of Allah,
the entirely merciful, the especially merciful."
-Quran 1:1
HolyQuran.co

Genetic Algorithm to solve N-Queens Problem

Exercise 6

Genetic Algorithm to Solve N-Queens Problem

By: Mojtaba Zolfaghari

mowjix@gmail.com

Professor :Dr.Ali Shakiba

http://alishakiba.ir

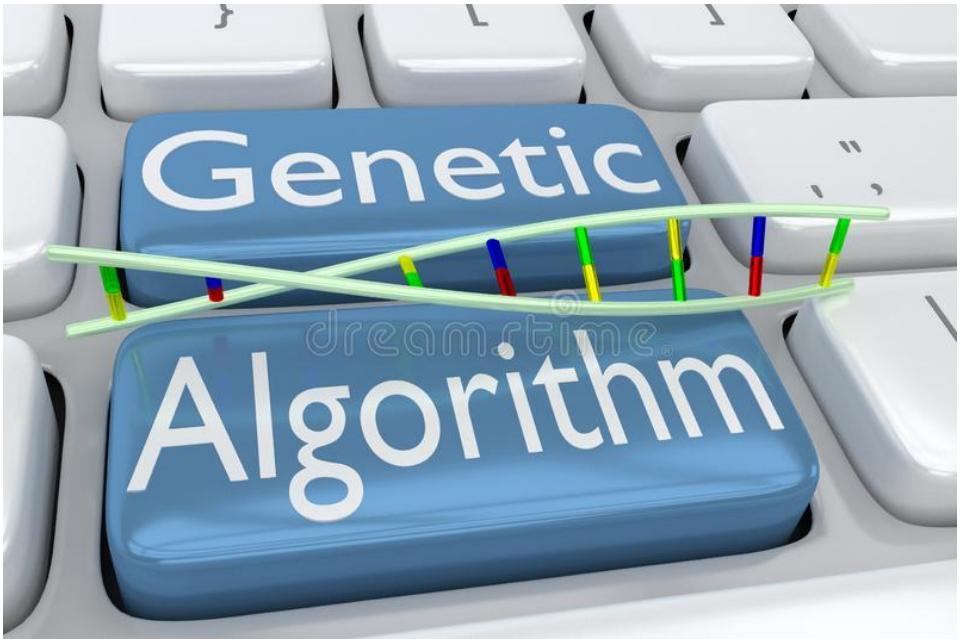
Preparedin the Faculty of Mathematics and Computer Science

Vali-e-Asr University of Rafsanjan

Apr. 2021

Genetic Algorithm to solve N-Queens in Python From Scratch

Last Update: 04/21/2021.



Genetic Algorithm to solve N-Queens Problem

Table of Contents

Title **Page**

Genetic Algorithm to solve N-Queens Problem

Introduction

In this PDF file we are going to have a brief discussion about how `Genetic Algorithm to Solve N-Queens Problem in Python From Scratch.py` does actually works! So let's go.

Importing libraries

In this case we just need to import random module in our project:

```
import random
```

Genetic Algorithm to solve N-Queens Problem

random_chromosome function

random_chromosome get size as a parameter then return a random integer list in range of [1,np]. Actually we can make random chromosomes with this function.

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

fitness function

fitness function takes chromosome as parameter computes pairs of non-attacking queens.

```
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))
```

Genetic Algorithm to solve N-Queens Problem

probability function

probability function takes chromosome and fitness as parameters and then returns probability of chromosome.

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

random_pick function

random_pick function takes population and probabilities as parameters.

```
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"
```

reproduce function

reproduce function takes two chromosomes and doing cross_over between two chromosomes

```
def reproduce(x, y):
    n = len(x)
    c = random.randint(0, n - 1)
    return x[0:c] + y[c:n]
```

mutate function

mutate function takes one parameter (chromosom) and then randomly changes the value of a random index of a chromosome

```
def mutate(x):
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(1, n)
    x[c] = m
    return x
```

print_chromosome function

print_chromosome function takes chrom as a parameter and then prints chrom and fitness(chrom)

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

genetic_queen function

genetic_queen takes population and fitness as parameters and then computes probabilities according to probability function. in the next step in a for loop according to random_pick function choose two the best chromosomes and according to reproduce function creates two new chromosomes from the best 2 chromosomes. in the next step probably if a random number was smaller than our mutation_probability = 0.03 then according to mutate function changes chromosome and finally by using print_chromosome function prints out than chromosome and finally append it in new_population list. This for loop continues until the chromosome fitness reaches its maximum fitness or traverses the entire population. At last this function returns new_population.

```
def genetic_queen(population, fitness):

    mutation_probability = 0.03

    new_population = []

    probabilities = [probability(n, fitness) for n in population]

    for i in range(len(population)):

        x = random_pick(population, probabilities)
        y = random_pick(population, probabilities)
        child = reproduce(x, y)

        if random.random() < mutation_probability:

            child = mutate(child)

        print_chromosome(child)
        new_population.append(child)

        if fitness(child) == maxFitness:
            break

    return new_population
```

print_board function

print_board takes one parameter (board) as a list and then prints it out as string with one space between each elements of list.

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

check_and_solve function

check_and_solve function takes maxFitness and population as parameters and then in the first during a while loop generates all of possible scenarios. In the second step during a for loop prints out all chromosomes with maximum fitness value and in the last step we are going to prints out last solution board.

```
def check_and_solve(maxFitness, population):
    generation = 1
    while not maxFitness in [fitness(chrom) for chrom in population]:
        print("=== Generation {} ===".format(generation))
        population = genetic_queen(population, fitness)
        print("\nMaximum Fitness = {}".format(max([fitness(n) for n in pop
ulation])))
        print("\n\n")
        generation += 1
    chrom_out = []
    print("Solved in Generation {}!".format(generation-1))
    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"

    print_board(board)
```

Genetic Algorithm to solve N-Queens Problem

Let's run this code!

Assume than we are going to solve 4-Queens problem.

```
nq = int(input("Enter Number of Queens: "))  # say N = 4
maxFitness = (nq*(nq-1))/2   # 4*3/2 = 6
population = [random_chromosome(nq) for _ in range(100)]

check_and_solve(maxFitness, population)
```

```
Enter Number of Queens: 4
```

```
Enter Number of Queens: 4
=== Generation 1 ===
Chromosome = [2, 3, 4, 2],  Fitness = 4
Chromosome = [1, 1, 3, 2],  Fitness = 4
Chromosome = [4, 3, 4, 2],  Fitness = 4
Chromosome = [1, 1, 2, 4],  Fitness = 4
Chromosome = [4, 2, 4, 4],  Fitness = 2
Chromosome = [2, 2, 4, 4],  Fitness = 3
Chromosome = [1, 3, 1, 1],  Fitness = 2
Chromosome = [2, 2, 4, 1],  Fitness = 4
Chromosome = [2, 1, 2, 3],  Fitness = 3
Chromosome = [4, 2, 4, 4],  Fitness = 2
Chromosome = [3, 1, 1, 2],  Fitness = 4
Chromosome = [2, 2, 3, 1],  Fitness = 4
Chromosome = [4, 3, 1, 2],  Fitness = 5
Chromosome = [4, 1, 3, 1],  Fitness = 4
Chromosome = [2, 1, 2, 4],  Fitness = 4
Chromosome = [3, 3, 1, 3],  Fitness = 2
Chromosome = [3, 2, 3, 2],  Fitness = 3
```

```
Maximum Fitness = 6


Solved in Generation 2!

One of the solutions:
Chromosome = [2, 4, 1, 3],  Fitness = 6
x Q x x
x x x Q
Q x x x
x x Q x
```

...The End...