UNIVERSIDADE FEDERAL DE MINAS GERAIS

Escola de Engenharia

Engenharia Elétrica

COMPUTAÇÃO EVOLUCIONÁRIA

Relatório 1 – N Rainhas

Aluno:

Gabriel Saraiva Espeschit - 2015065541

19 de setembro de 2019

1. Introdução

O objetivo do trabalho é escrever um algoritmo genético capaz de determinar uma combinação que satisfaça o problema das N-Rainhas. O trabalho foi desenvolvido em Python. O problema das N-Rainhas foi expresso da seguinte maneira:

"Dado um tabuleiro de xadrez regular (NxN) e N rainhas, posicione as N -Rainhas no tabuleiro de forma que elas não se coloquem em xeque"

2. Metodologia

Para resolver esse problema utilizou-se de 2 funções dadas (as quais foram traduzidas para Python): "*fitness*" e "*cutandcrossfill_crossover*". A primeira é para avaliar o fitness do membro atual, ou seja, quantos cheques estão em jogo naquele momento. A segunda função é para gerar herdeiros a partir dos pais.

São passados os seguintes parâmetros para função "*NRainhas*" desenvolvida: *tam_pop, N, gen, num_pais* e *prob_mut*. Ela segue a seguinte lógica:

1) Gera-se uma população com "*tam_pop*" indivíduos. Cada indivíduo é representado por um vetor de tamanho "*N*" (o valor padrão de "*N*" é 8) em o índice do vetor representa a coluna e o valor representa a fileira, de tal modo que cada vetor é uma permutação de um vetor 1, 2, 3, ..., N.
2) Entra em uma iteração que roda "*gen*" vezes (o valor padrão de "*gen*" é 1000):
    a. Selecionamos os pais, dentre "*num_pais*" (o valor padrão de "*num_pais*" é 5) randômicos, escolhemos os dois melhores para alimentar na função "*cutandcrossfill_crossover*" e gerar 2 filhos.
    b. Cada filho gerado tem uma probabilidade "*prob_mut*" (o valor padrão de "*prob_mut*" é 0.8) de sofre mutações, isto é, o valor contido em um índice trocar de lugar com o valor contido em um outro (ou no mesmo) índice.
    c. Os filhos são inseridos na população e os 2 piores indivíduos são eliminados da população.
    d. A função roda "*gen*" vezes. É possível parar as iterações assim que atingirmos o de fitness desejado.
3) Os resultados dos melhores indivíduos de cada geração e o fitness médio de cada individuo são plotados.

3. Resultados

Os resultados para N = 8 foram conforme esperados. O algoritmo convergiu por volta da centésima geração, sem precisar fazer alterações nos parâmetros da função.

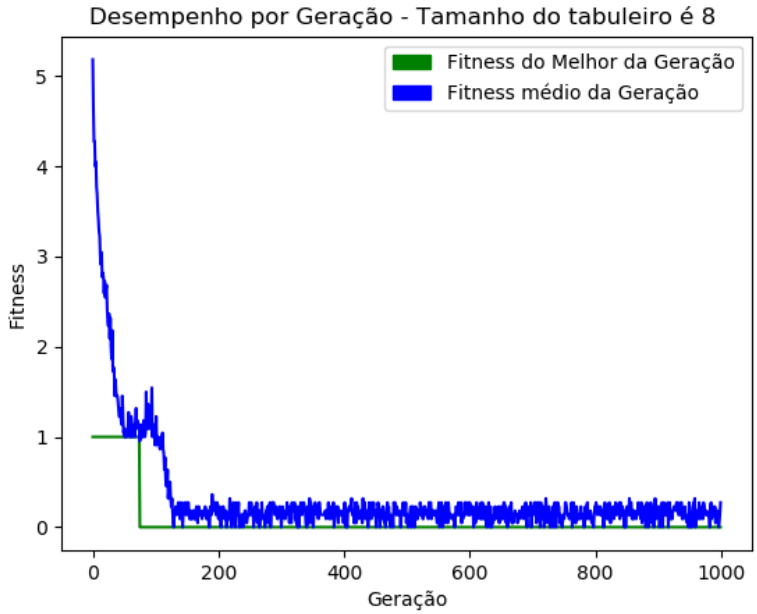

Figura 1 - Fitness médio e melhor pra N=8

Para N=20, o resultado também convergiu, porém mais lentamente. Isso se deve à maior complexidade associada ao aumento do tabuleiro.

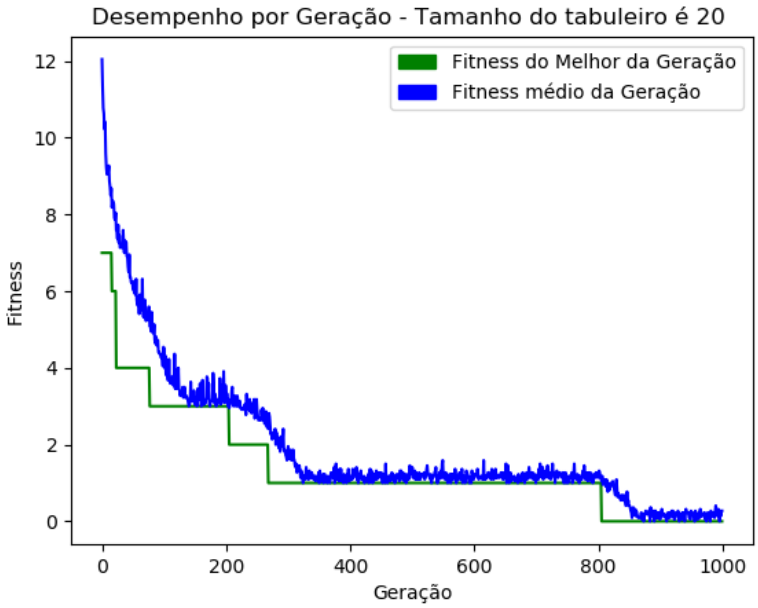

Figura 2 - Fitness médio e melhor pra N=20

Para N=50 essa complexidade já aparente na convergência do algoritmo, sendo que sem alterar os parâmetros de entrada, não foi possível chegar ao resultado desejado.

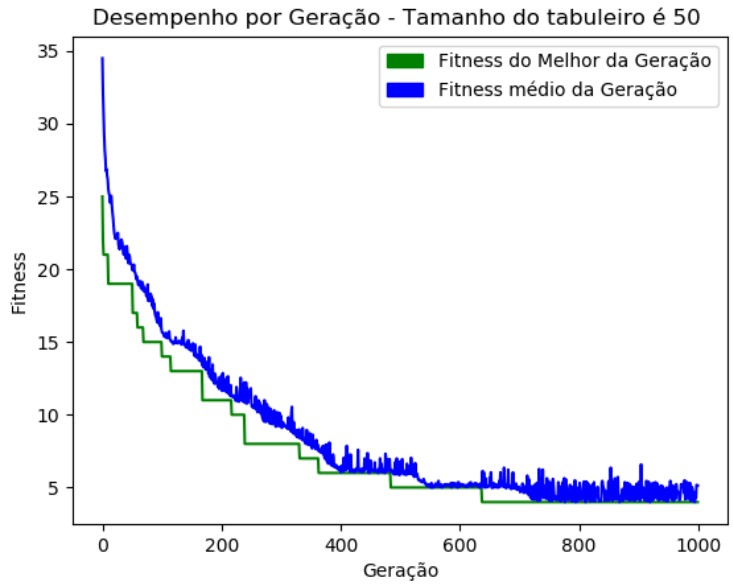

Figura 3 - Fitness médio e melhor pra N=50 e 1000 iterações

Para convergência, foi necessário alterar a quantidade de iterações. Ao investigar o efeito de aumentar o num_pais ou o tam_pop, que são variáveis que interferem diretamente na convergência do algoritmo, verificou-se que é comum a ocorrência de mínimos locais. Para convergência desejada, aumentar o número de iterações foi a melhor alternativa verificada.

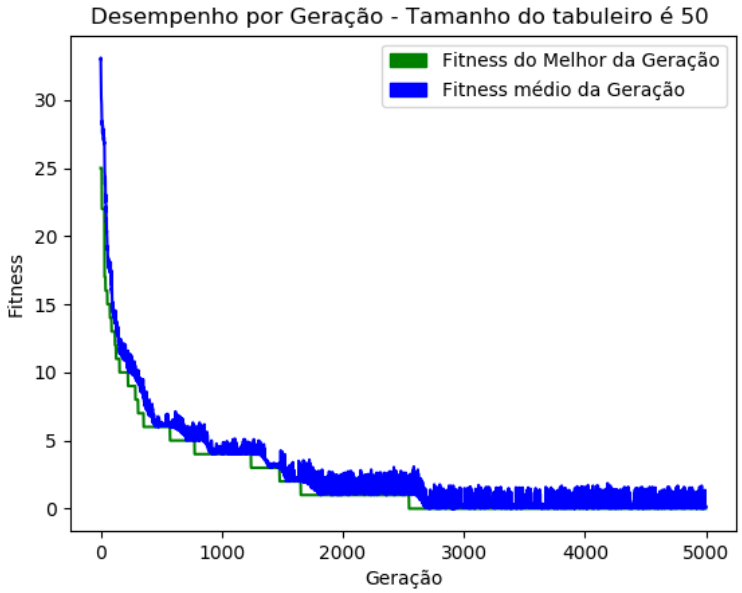

Figura 4 - Fitness médio e melhor pra N=50 e 5000 iterações

4. Código

Segue abaixo o código desenvolvido:

```
import numpy as np
from random import sample, random, randint
from statistics import mean
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

def fitness(sol):
    f = 0
    n = len(sol)
    for i in range(n):
        for j in range(n):
            if abs(i-j) == abs(sol[i]-sol[j]) and j != i:
                f += 1

    return f/2

def cutandcrossfill_crossover(p1, p2):
    N = len(p1)
    pos = np.random.randint(1, N)
    f1 = p1[:pos]
    f2 = p2[:pos]
    for i in range(N):
        check1 = 0
        check2 = 0
        for j in range(len(f1)):
            if p2[i] == f1[j]:
                check1 = 1
        for j in range(len(f2)):
            if p1[i] == f2[j]:
                check2 = 1
        if check1 == 0:
            f1.append(p2[i])
        if check2 == 0:
            f2.append(p1[i])
    return f1, f2

def NRainhas(tam_pop=20, N=8, gen=1000, num_pais=5, prob_mut=0.8):
    # tam_pop: Tamanho da População
    # N: Tamanho do Tabuleiro
    # gen: Quantas Gerações?
    # num_pais: Numero de pais para olhar cruzamento
    # prob_mut: Probabilidade de mutação
    # it: Controle de Iterações
    it = 0
    melhor = []
    pior = []
    medio = []

    # Inicializando a População
    M = []
    for i in range(tam_pop):
        M.append(sample(range(1, N+1), N))

    fit = []
```

```
    while it < gen:

        # Seleção de Pais
        id_pais = sample(range(tam_pop), num_pais)
        fit_pais = []

        for i in range(len(id_pais)):
            fit_pais.append(fitness(M[id_pais[i]]))

        # Ordenar e alocar pais
        id_aux = np.argsort(fit_pais)
        p1 = M[id_pais[id_aux[0]]]
        p2 = M[id_pais[id_aux[1]]]
        # Filhos
        f1, f2 = cutandcrossfill_crossover(p1, p2)

        # Mutações
        if random() < prob_mut:
            aux1 = randint(0, N-1)
            aux2 = randint(0, N-1)
            aux = f1[aux1]
            f1[aux1] = f1[aux2]
            f1[aux2] = aux

        if random() < prob_mut:
            aux1 = randint(0, N-1)
            aux2 = randint(0, N-1)
            aux = f2[aux1]
            f2[aux1] = f2[aux2]
            f2[aux2] = aux
        # Colocando os Filhos na População
        M.append(f1)
        M.append(f2)

        # Selecionando os Mais Aptos
        for i in range(len(M)):
            fit.append(fitness(M[i]))
            # if fit[i] == 0:
                # it = gen            # Opcional: se quiser parar assim que
chegar no resultado desejado
        id_fit = np.argsort(fit)
        piores = id_fit[-2:]
        for index in sorted(piores, reverse=True):
            del M[index]

        melhor.append(min(fit))
        pior.append(max(fit))
        medio.append(mean(fit))

        fit = []
        if it % 100 == 0:
            print(f'Estamos na iteração: {it}')
        it += 1

    plt.plot(melhor, c='g')
    plt.plot(medio, c='b')
    green_patch = mpatches.Patch(color='g', label='Fitness do Melhor da
Geração')
    blue_patch = mpatches.Patch(color='b', label='Fitness médio da Geração')
```

```
    plt.legend(handles=[green_patch, blue_patch])
    # plt.plot(pior, c='r')
    plt.title(f'Desempenho por Geração - Tamanho do tabuleiro é {N} ')
    plt.xlabel('Geração')
    plt.ylabel('Fitness')
    plt.show()

if __name__ == '__main__':
    NRainhas(tam_pop=20, N=8)
    NRainhas(tam_pop=20, N=20)
    NRainhas(tam_pop=20, N=50, num_pais=5, gen=5000)
```