# Experimento de resolução do problema das n rainhas por algoritmo genético

**Eduardo Marsola do Nascimento**

Escola de Artes, Ciências e Humanidades – Unive0072sidade de São Paulo (USP)
Rua Arlindo Bettio, 1000 – 03828-000 – São Paulo – SP – Brasil

edunasci@usp.br

***Abstract.*** *This paper describes an experiment that uses a genetic algorithm to solve the n queens problem. The experiment is divided into two parts, the first part solved the problem up to 12 queens using brute force, the second part solved the problem up to 50 queens using a simple genetic algorithm with elitism. The following parameters were used: population size =100 individuals, 1% mutação and 5% elitism. The genetic algorithm could find at least one best solution for the n queens problem, with n=5, 6, 7, ...,49, 50, in less than 40 minutes, using monotask implementation on a Intel i5 10$^{th}$ generation (i5-1035G1 @1.00GHz max frequency 3.6GHz) notebook. Only to compare, the brute force implementation on the same machine took 30 minutes to find one best solutions for 13 queens problem.*

***Resumo.*** *Este artigo descreve um experimento que uso um algoritmo genético para resolver o problema das n rainhas. O experimento foi dividido em duas partes, a primeira parte resolveu o problema com até 12 rainhas utilizando força bruta, a segunda parte resolveu o problema até 50 rainhas utilizando um algoritmo genético simples com elitismo. Os seguintes parâmetros foram utilizados: tamanho da população = 100 indivíduos, 1% mutação e 5% de elitismo. O algoritmo genético pode encontrar pelo menos uma melhor solução para o problema das n rainhas, com n = 5, 6, ..., 49, 50, in menos de 40 minutos, utilizando uma implementação monotarefa em um notebook Intel i5 10$^{a}$ geração (i5-1035G1 @1.00GHz max frequency 3.6GHz). Apenas para comparativo, uma implementação de força bruta na mesma máquina levou 30 minutos para encontrar uma melhor solução para o problema de 13 rainhas.*

## 1. Introdução

Os algoritmos genéticos são uma classe de algoritmos de busca que foram criados tentando copiar o comportamento da reprodução de seres vivos na natureza.

A cada geração, a população se modifica, mantendo de forma probabilística as características dos indivíduos que estão melhor adaptados ao ambiente.

A avaliação de melhor adaptação é feita por uma função fitness que avalia cada indivíduo da população.

Este experimento conclui que o algoritmo genético é mais eficiente para encontrar a solução de busca quando o espaço de busca é grande.

Ele conseguiu resolver o problema de 13 rainhas, encontrando uma melhor solução em 9 segundos. Um algoritmo de força bruta levou 30 minutos para encontrar uma solução do mesmo problema. Também foram feitos testes para descobrir todas as melhores soluções por força bruta. Neste caso foram enc

Os experimentos foram realizados em um notebook Intel i5 10ª geração (i5-1035G1 @1.00GHz max frequency 3.6GHz).

## 2. Criação dos algoritmos utilizados

O primeiro ponto foi definir um alfabeto a ser utilizado. Conforme visto em aula o alfabeto mais eficiente é aquele que representa o tabuleiro de xadrez por uma sequência de n números, com cada número podendo variar de 1 a n. Por este alfabeto, o tabuleiro de 8 rainhas da Figura 1 é representado pela sequência 7,2,6,3,1,4,8,5. Cada número representa a casa de uma coluna, contadas de cima para baixo, o número 7 representa a 7ª casa de baixo para cima da primeira coluna, o número 2 representa a segunda casa de baixo para cima da segunda coluna e assim sucessivamente.

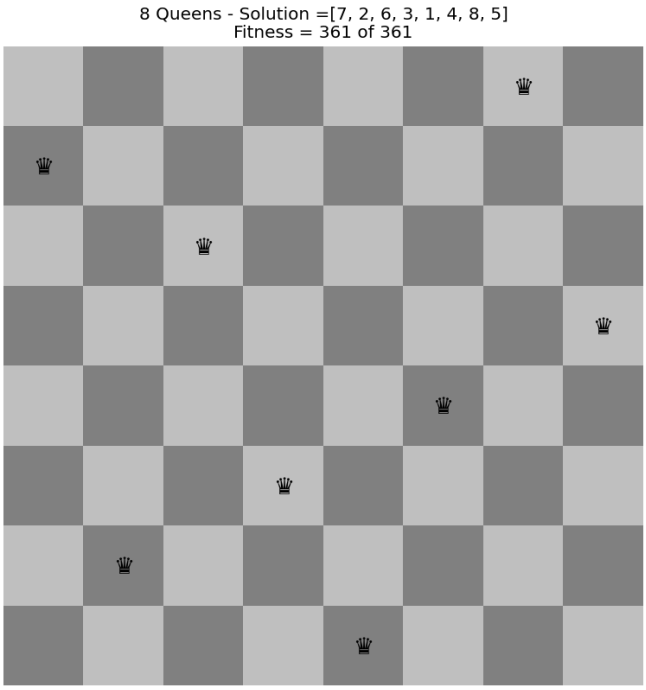



**Figura 1 - representação do tabuleiro com tamanho 8x8 e 8 rainhas.**

Se este alfabeto for considerado como sendo uma permutação n números, ele diminui sensivelmente o espaço de busca. Por exemplo uma permutação dos números de 1 a 8 representa um espaço de busca de 40.320 soluções, uma sequência de 8 números de 1 à 8 possui um espaço de busca de 16.777.216 soluções.

Depois de escolhido o alfabeto, foi criada a função de fitness. Para isso observou-se que num problema com n rainhas, caminhando as colunas, da esquerda para a direita, e contando o número de rainhas atacadas à direita, temos:

$nRainhasAtacadas = \frac{(1+n)\times n}{2}$, para o pior caso.

Então convencionou que o fitness seria:

$$fitness = \left(\left(\frac{(1+n)\times n}{2}\right) \times 10 + 1\right) - nRainhasAtacadas \times 10.$$

Desta maneira o fitness sempre será um número positivo maior que zero. A multiplicação por 10 é para impor uma probabilidade 10 vezes menor para os piores casos.

O alfabeto e a medida de fitness foram os mesmos utilizados tanto no algoritmo por força bruta quanto no algoritmo genético.

O algoritmo de força bruta consistia apenas em varrer todas as permutações de n números de 1 à n. Cada permutação foi considerada um indivíduo e seu fitness foi calculado. Abaixo o gráfico do fitness calculado para todas as permutações possíveis do problema de 5 rainhas.

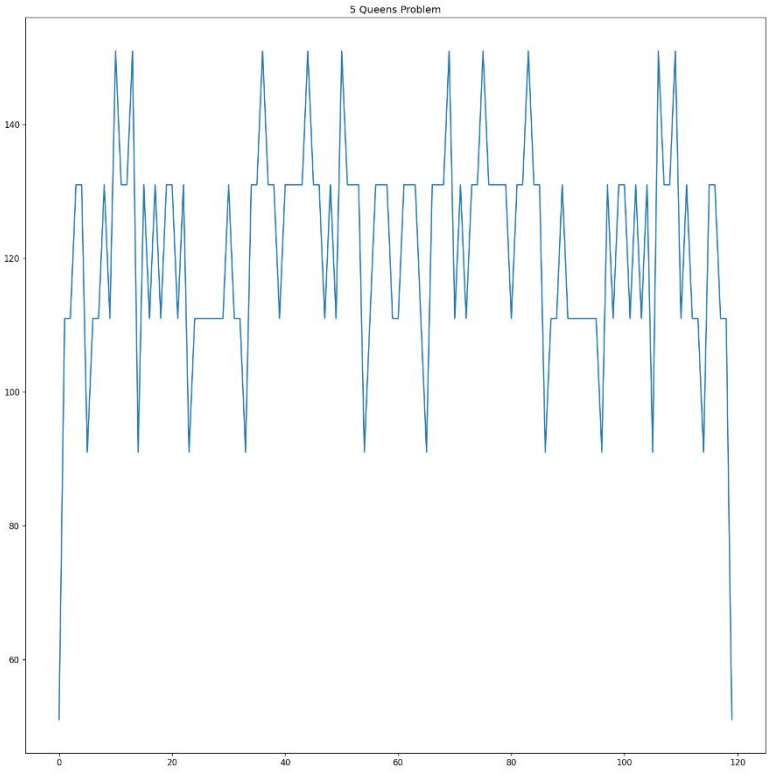



**Figura 2 - fitness de todas as permutações do problema de 5 rainhas**

Para o algoritmo genético foram criados 3 operados: mutação, crossover e overlap.

O operador de mutação é o operador *SWAP* descrito por [Back et al., 2000, p. 245-246].

O operador de *crossover* foi construído com base no operador David's order crossover também descrito por [Back et al., 2000, p. 274-276].

O operador *overlap* simplesmente clona um indivíduo da geração mãe na geração filha.

A escolha do operador que será aplicado, é feita por uma seleção por roleta, considerando os pesos de cada operador passados como parâmetros.

A seleção dos indivíduos para reprodução também é feita por roleta, considerando o fitness de cada indivíduo como peso.

Após a criação da geração filha utilizando os operadores de mutação, crossover e overlap. Aplica-se o elitismo, trocando os indivíduos com menor fitness da geração filha pelos indivíduos com maior fitness da geração mãe.

O pseudocódigo simplificado do algoritmo criado é o seguinte:

**Algoritmo Genético para solução do problema de n rainhas**

Entrada: n=quantidade de rainhas, t= tamanho da população, pesos (overlap, mutação e crossover) – pesos dos operadores genéticos, e – quantidade de indivíduos para serem utilizados como elite, parâmetros de parada – maxGeração, maxFitness

$P_0 \leftarrow inicializa\ a\ população\ com\ t\ indivíduos\ de\ tamanho\ n$

$g = 0$

$Enquanto\ g <$ maxGeração

$calcula\ fitness\ P_0$

$se\ fitness(individuo_i) == maxFitness,\ retorna\ individuo_i$

$F_0 \leftarrow \emptyset$

$Enquanto\ tamanho\ F_0 < t$

$optipo \leftarrow escolhe\ por\ roleta\ entre\ overlap, mutação\ ou\ crossover$

$i_1[, i_2] \leftarrow escolhe\ por\ roleta\ individuo(s) para\ operador$

$f_1 = optipo(i_1[, i_2])$

$F_0 = F_0 + f_1$

$elite \leftarrow seleciona\ e\ individuos\ com\ melhor\ fitness\ de\ P_0$

$P_0 \leftarrow F_0 - (e\ individuos\ com\ menor\ fitness\ de\ F_0) + elite$

$Retorna\ individual\ com\ maior\ fitness\ de\ P_0$

## 3. Resultados

As tabelas abaixo apresentam os tempos tanto da busca por força bruta quanto da busca por algoritmo genético.

**Tabela I - tempos e quantidades de solução busca por força bruta**

| Solução por Força Bruta | | | | |
|---|---|---|---|---|
| Quantidade de Rainhas | Tempo todas soluções | Tempo primeira melhor solução | Quantidade de soluções possíveis | Quantidade de Melhores Soluções |
| 1 | < 1s | < 1s | 1 | 1 |
| 2 | < 1s | < 1s | 2 | 0 |
| 3 | < 1s | < 1s | 6 | 0 |
| 4 | < 1s | < 1s | 24 | 2 |
| 5 | < 1s | < 1s | 120 | 10 |
| 6 | < 1s | < 1s | 720 | 4 |
| 7 | < 1s | < 1s | 5040 | 40 |
| 8 | 12s | < 1s | 40320 | 92 |
| 9 | 75s | < 1s | 362880 | 352 |
| 10 | 75s | < 1s | 3628800 | 724 |
| 11 | 25m | <10s | 3,99E+07 | 2680 |
| 12 | 4h | 137s | 4,79E+08 | 14200 |
| 13 | Não Calculado | 30m | 6,23E+09 | Não Calculado |

**Tabela II – resultados do algoritmo genético**

| Solução por Algoritmo Genético | | | | | |
|---|---|---|---|---|---|
| Quantidade de Rainhas | Tempo | Quantidade de soluções possíveis | Gerações | Fitness Encontrado/Máximo | Melhor Solução |
| 5 | < 1s | 120 | 0 | 151/151 | Sim |
| 6 | < 1s | 720 | 1 | 211/211 | Sim |
| 7 | < 1s | 5040 | 1 | 281/281 | Sim |
| 8 | 1s | 40320 | 13 | 361/361 | Sim |
| 9 | < 1s | 362880 | 5 | 451/451 | Sim |
| 10 | 1s | 3628800 | 17 | 551/551 | Sim |
| 11 | 4s | 3,99E+07 | 49 | 661/661 | Sim |
| 12 | 5s | 4,79E+08 | 57 | 781/781 | Sim |
| 13 | 9s | 6,23E+09 | 114 | 911/911 | Sim |
| 14 | 24s | 8,72E+10 | 318 | 1051/1051 | Sim |
| 15 | 1s | 1,31E+12 | 11 | 1201/1201 | Sim |
| 16 | 31s | 2,09E+13 | 396 | 1361/1361 | Sim |
| 17 | 25s | 3,56E+14 | 315 | 1531/1531 | Sim |
| 18 | 12s | 6,40E+15 | 150 | 1711/1711 | Sim |
| 19 | 68s | 1,22E+17 | 826 | 1901/1901 | Sim |
| 20 | 57s | 2,43E+18 | 672 | 2101/2101 | Sim |
| 21 | 181s | 5,11E+19 | 2000 | 2241/2311 | Não |
| 22 | 140s | 1,12E+21 | 1536 | 2531/2531 | Sim |
| 23 | 186s | 2,59E+22 | 2000 | 2731/2761 | Não |
| 24 | 189s | 6,20E+23 | 2000 | 2941/3001 | Não |
| 25 | 191s | 1,55E+25 | 2000 | 3221/3251 | Não |
| 26 | 195s | 4,03E+26 | 2000 | 3441/3511 | Não |
| 27 | 198s | 1,09E+28 | 2000 | 3701/3781 | Não |
| 28 | 202s | 3,05E+29 | 2000 | 3971/4061 | Não |
| 29 | 207s | 8,84E+30 | 2000 | 4261/4351 | Não |
| 30 | 213s | 2,65E+32 | 2000 | 4591/4651 | Não |

Para os testes do algoritmo de força bruta, foram encontradas uma melhor solução para os problemas de tamanho n=1 até n=13, sendo n o número de rainhas. O motivo de parar em n=13 foi a necessidade de mais memória RAM para geração da permutação utilizando as bibliotecas padrão da linguagem Python. Uma alternativa seria gerar uma função própria para a permutação, porém estaria além do propósito deste exercício.

Além disso, nos testes do algoritmo de força bruta, foram geradas todas as melhores soluções possíveis para os problemas de tamanho n=1 até n=12 gerando, sendo n o número de rainhas. O motivo de parar em n=12 foi o tempo necessário para processamento. Em n=12, o tempo estava em 4 horas, para n=13 foi estimado que o processamento levaria mais de 50 horas.

No processamento de força bruta, foram gerados gráficos da função fitness para cada indivíduo até n=9. Por causa da quantidade de dados, não foi possível gerar gráficos diretamente para problemas com n >= 10.

Abaixo temos os gráficos da função fitness para n=7 e n=9

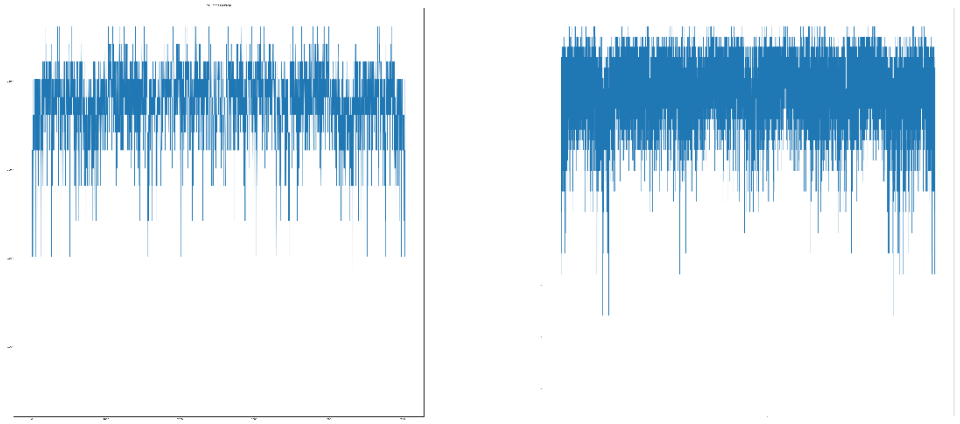


**Figura 3 - gráfico de fitness para n=7 e n=9**

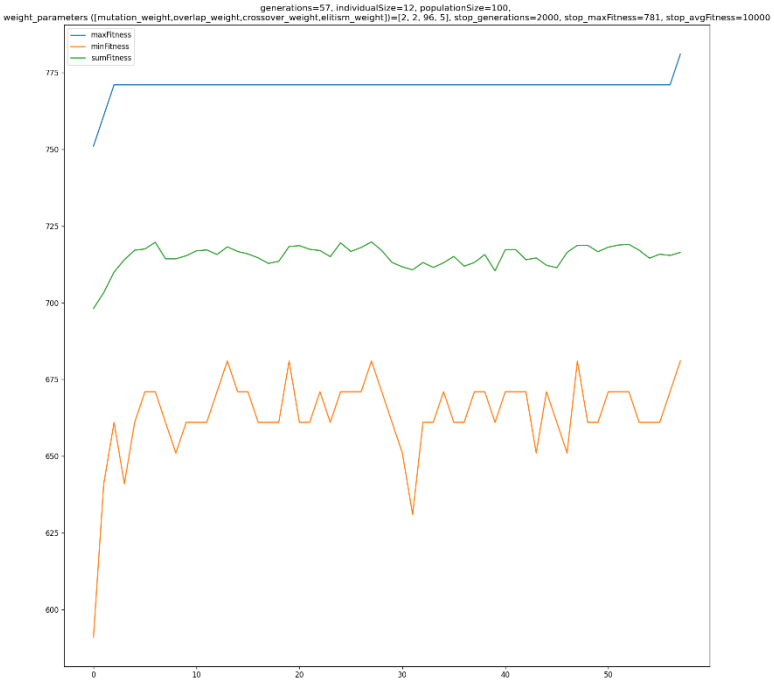
Para os testes do algoritmo genético, trabalhou-se com os seguintes parâmetros, tamanho da população = 100, peso do operador de mutação = 1, peso do operador de crossover = 99, quantidade de indevidos para elitismo = 5 e quantidade máxima de gerações = 2000. Além disso foi considerado como critério de parada, o fitness da melhor solução possível para aquele problema.

No caso do algoritmo genético, os problemas foram solucionados de n=5 até n=30, onde n é o número de rainhas. Problemas menores que n=5 teriam um número de soluções possíveis, menos que a população escolhida como parâmetro. Para n = 4, existem apenas 24 soluções, sendo 2 as melhores possíveis.

O algoritmo genético é probabilístico, então, cada vez que ele for executado, o tempo necessário para chegar a melhor solução irá variar.

Durante a execução do algoritmo genético foram gerados gráficos com o fitness máximo, médio e mínimo verificado a cada geração. Também foram geradas representações gráficas do tabuleiro com as soluções encontradas.

A seguir os gráficos gerados para n=12, n=20 e n=30.



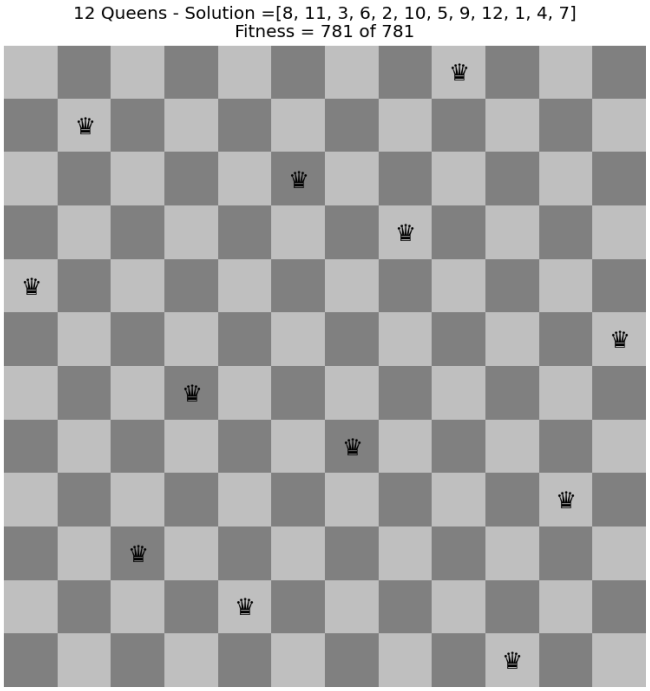
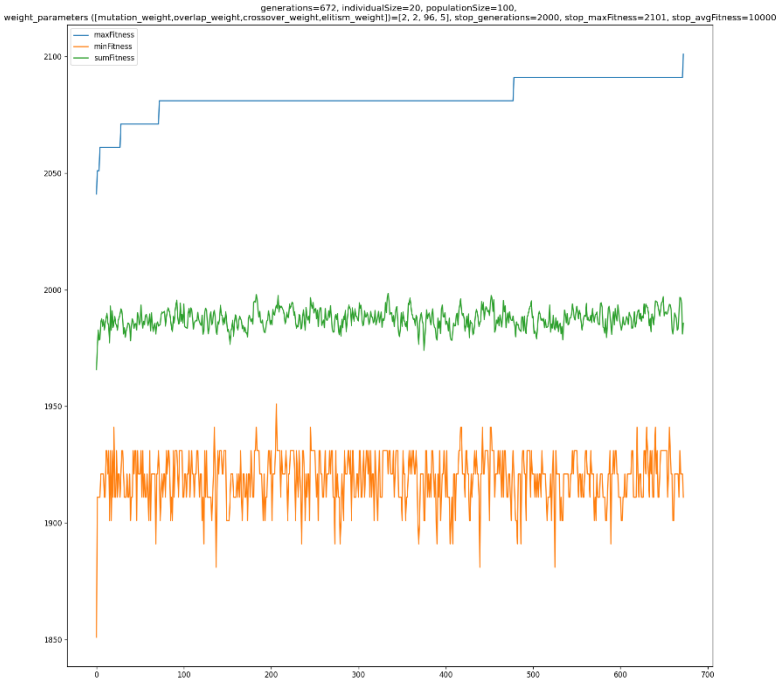




**Figura 4 - gráficos de fitness e representação gráfica da solução para n=12**



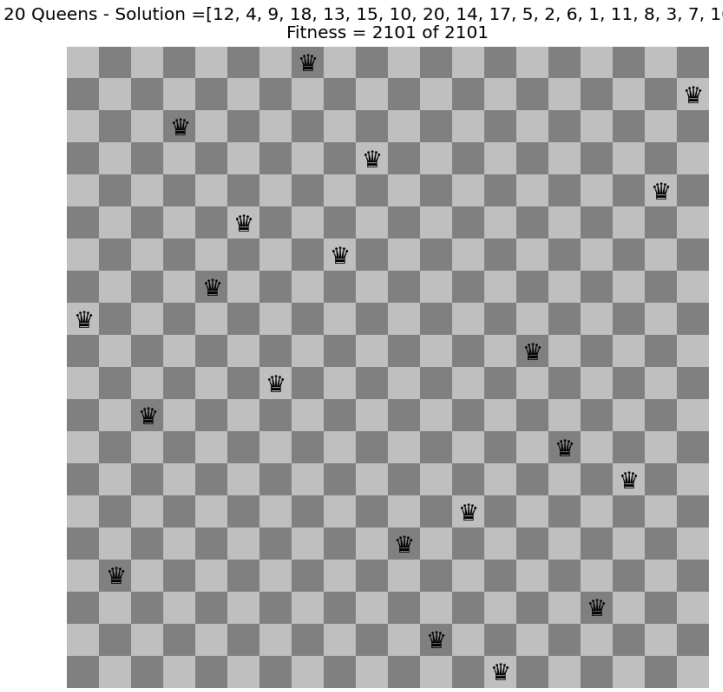
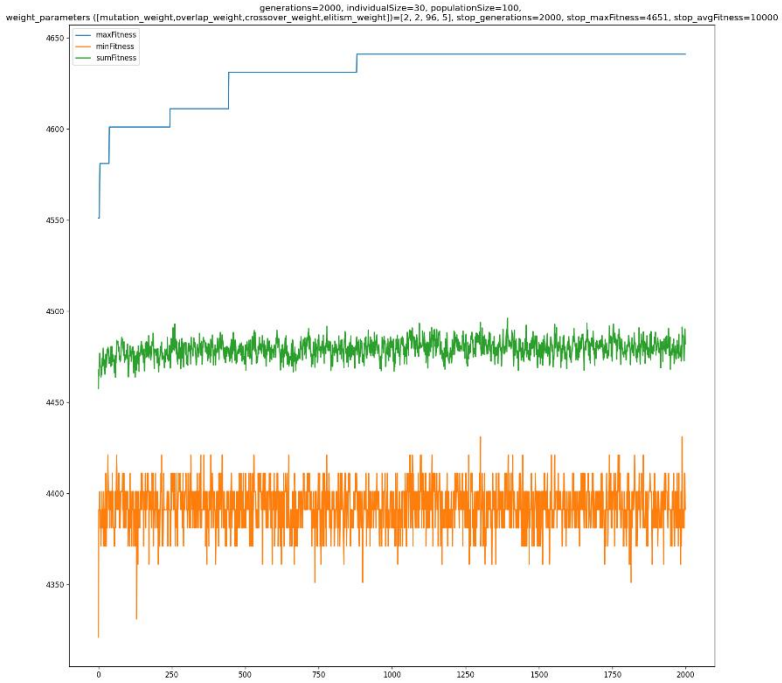
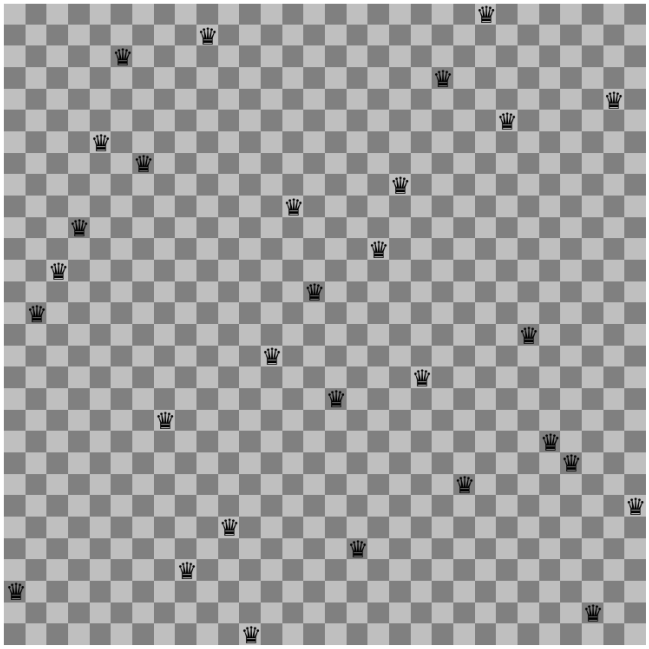




**Figura 5- gráficos de fitness e representação gráfica da solução para n=20**




olution =[3, 16, 18, 20, 24, 28, 23, 11, 4, 29, 6, 1, 14, 21, 17, 12, 5, 19, 22, 13, 27, 8, 30, 25, 15

Fitness = 4591 of 4651



## Conclusão

Conforme esperado, o experimento demonstrou que um algoritmo genético é eficiente para resolver o problema das n rainhas de forma eficiente, porém nem sempre trará a melhor solução.

O código fonte e os gráficos gerados pelo experimento estão disponíveis no github https://github.com/edunasci/SIN5006 [Nascimento, 2023]

## Referências


Bäck, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (2018). Evolutionary computation 1: Basic algorithms and operators. CRC press. Disponível em: <https://digi.lib.ttu.ee/services/copycat-pdf.php?ID=248>. Acessado em: 3/9/2023.

NASCIMENTO, Eduardo Marsola do. Repositório exemplos SIN5006 - Inteligência Computacional 2023.2, 2023. URL: < https://github.com/edunasci/SIN5006 >. Acessado em: 8/9/2023.

Peres, Sarajane Marques. SIN5006 – Inteligência Computacional. Notas de aula.