UNIVERSIDADE
FEDERAL DO
RIO DE JANEIRO
UFRJ

Inteligência Artificial

Algoritmo Genético aplicado ao problema das N-Rainhas

Renan Mendanha
Jonathan Suhett

Junho
2022

Sumário

Introdução

Neste relatório abordaremos as escolhas de modelagem e implementação que foram feitas a fim de resolver o problema das N-Rainhas, utilizando o Algoritmo Genético para achar uma arrumação do tabuleiro próxima à solução ótima.

Além disso, comentaremos sobre os resultados obtidos ao rodar esse algoritmo em tabuleiros com $N = \{4,\ 8,\ 16,\ 32\}$ rainhas e dimensão $N \times N$.

Modelagem

a) Representação de um tabuleiro $N \times N$ com N rainhas:

O tabuleiro será representado por uma lista l de N elementos, onde cada elemento l_i representará uma coluna contendo uma rainha (gene), e o valor presente neste elemento de l representará a fileira na qual essa rainha está localizada.

Portanto, com essa modelagem, haverá exatamente uma rainha em cada coluna do tabuleiro, eliminando assim, a necessidade de verificar ataques na vertical, além disso, como estaremos levando em conta somente um ataque entre um par de rainhas, não contaremos encontros de movimento à esquerda da coluna analisada.

As fileiras e colunas serão indexadas de 0 a $N - 1$, portanto, é possível montar um alfabeto $\Sigma = \{0,\ 1,\ \dots\ ,\ N - 1,\ N\}$ que consegue representar um gene para a construção da palavra de um indivíduo.

Com estas definições, podemos montar o seguinte exemplo: Um tabuleiro $4x4$ no qual $l = [0, 3, 1, 1]$ representa as rainhas na primeira fileira e primeira coluna, na quarta fileira segunda coluna, na segunda fileira terceira coluna e segunda fileira quarta coluna.

Implementação base

a) Gerando os tabuleiros:

```
# Criando q tabuleiros aleatórios de dimensão n
# Retorna a lista dos tabuleiros criados
def tabuleiros(n,q):
  tabList = []
  for i in range(q):
    tab = []
    for j in range(n):
      tab.append(int(random.random()*n)) #define aleatoriamente a posição da rainha
    tabList.append(tab)           #adiciona à lista de tabuleiros
  return tabList
```

b) Os estados vizinhos:

```python
# Recebe um tabuleiro
# Retorna todos os estados vizinhos
def todosVizinhos(tab):
  vizinhos = []
  for i in range(len(tab)): # iterador da coluna
    for j in range(len(tab)): # iterador da linha
      temp = tab.copy()
      if tab[i] != j :  #quando é igual, trata-se do próprio tabuleiro, e não de um vizinho
        temp[i] = j     #altera a posição da rainha na coluna i
        vizinhos.append(temp)
  return vizinhos
```

c) Retornando um estado vizinho aleatório:

```python
# Recebe um tabuleiro
# Retorna um estado vizinho deste tabuleiro
def umVizinho(tab):
  vizinho = tab.copy()
  coluna = int(random.random()*len(tab))
  linha = int(random.random()*len(tab))
  while linha == tab[coluna]:  #testa se a função não sorteou o próprio tabuleiro
    linha = int(random.random()*len(tab))
  vizinho[coluna] = linha
  return vizinho

#A função todosVizinhos poderia ter sido usada, mas não pareceu muito eficiente
# gerar todos os vizinhos para depois selecionar apenas um.
```

d) Calculando o número de ataques:

```python
# Recebe um tabuleiro
# Retorna o numero de ataques entre as rainhas
def numeroAtaques(tab):
  num = 0
  for i in range(len(tab)):
    for j in range(i+1,len(tab)): # Só avalia colunas à direita
      k = j-i  #diferença entre as colunas i e j usada para avaliar ataque diagonal
      if tab[i]==tab[j] or tab[i]+k==tab[j] or  tab[i]-k==tab[j]:
        num += 1
  return num
```

Algoritmo Genético

a) O algoritmo Genético será implementado com base nas funções descritas no tópico anterior e respeitando a modelagem proposta, incluindo a palavra l que codifica o conjunto de parâmetros (genes) de um indivíduo e o alfabeto.

b) Os indivíduos de uma população serão gerados através da função $tabuleiros$, passando como parâmetro o tamanho do tabuleiro e o tamanho da população.

c) Os operadores:

- Função de adaptação para avaliar um tabuleiro:

  A função de adaptação leva em conta o número de ataques, mas como o objetivo é minimizar os mesmos e AGs têm caráter de maximização do valor de adaptação, subtraímos o número de ataques do tabuleiro

avaliado do número máximo de ataques possíveis em um tabuleiro de tamanho N.

```
# Função que avalia os tabuleiros e
# retorna lista com o valor de adaptação dos indivíduos em uma população

def avalia(tabs):
  adapt = []
  maxAtk = 0
  for i in range(TAMTAB): #faz o somatório para determinar ataques máximos
    maxAtk += i
  for i in range(len(tabs)):
    adapt.append(maxAtk - numeroAtaques(tabs[i])) # se não ocorrer ataques,
    # o valor é ótimo (maxAtk), valor minimo = 0 -> todas as rainhas se atacam
  return adapt
```

- Construção da roleta viciada:

```
# Função que constrói a roleta
  # Recebe a lista da população, suas avaliações(que funcionam como pesos) e seu tamanho,
  # retornando uma lista com os indivíduos escolhidos pela roleta,
  # respeitando os pesos das avaliações,
  # de forma que quem tiver uma avaliação menor, tem menos chances de ser escolhido

def roleta(pop, pesos, tampop):
  return random.choices(pop, weights=pesos, k=tampop)
```

- Construção da população intermediária(seleção):

  A seleção foi feita com base nas operações de elitismo e execução da roleta.

- Função de crossover:

```
# Função que faz o crossover entre os indivíduos da população intermediária

def crossover(pop,pcros):
  # a lista é percorrida aos pares, então i pode ser a metade do comprimento da lista
  for i in range(int(len(pop)/2)):
    if random.random() < pcros:    #define se será feito crossover
      ia = 2*i
      ib = (2*i)+1
      temp1 = pop[ia].copy()
      temp2 = pop[ib].copy()
      corte = int(random.random()*TAMTAB)  #define o ponto de corte
      for j in range(corte,TAMTAB):
        temp1[j] = pop[ib][j]
        temp2[j] = pop[ia][j]
      pop[ia] = temp1
      pop[ib] = temp2
  return pop
```

- Função de mutação:

```
# Função que faz a mutação dos indivíduos da população inermediária

def mutacao(pop,pmut):
  for i in range(len(pop)):
    if random.random() < pmut:
      # a mutação é um movimento de alguma rainha (estado vizinho)
      pop[i] = umVizinho(pop[i])
  return pop
```

d) O Algoritmo Genético Básico:

O algoritmo pode gerar uma população inicial aleatória ou aplicá-la em uma já existente, assim somos capazes de guardar o progresso de cada geração.

A função “genético” é responsável por gerenciar o funcionamento do algoritmo, criando a população (se necessário), chamando as funções que vão modificar a população a cada geração, e por fim, chamando a função que plota os gráficos.

```python
# Implementação do algoritmo genético
  #Os operadores estão definidos abaixo

TAMTAB = 8 # tamanho do tabuleiro


#probabilidades são passadas em intervalo de 0 a 1
def genetico(tampop,ngen,pcros,pmut,popIni=None,elitismo=False):

  #gera população inicial se ela não for passada como parâmetro
  if popIni == None or len(popIni) == 0:
    popIni = tabuleiros(TAMTAB,tampop)

  # avalia população inicial
  ava = avalia(popIni)
  melhor = 0
  sum = 0
  for i in range(tampop):
    if ava[i]>melhor:
      melhor = ava[i]
    sum += ava[i]

  # cria lista de melhores e médias para geração dos gráficos
  adapt = [melhor]
  med = [sum/tampop]

  # população intermediária
  popInt = popIni.copy()

  #para cada geração
  for i in range(ngen):
    #avalia tabuleiro
    ava = avalia(popInt)
    #elitismo
    index = 0
    elit = popInt[index]
    if elitismo:
      for i in range(tampop):
        if ava[i]>ava[index]:
          elit = popInt[i]
    #constrói população intermediária usando a roleta
    popInt = roleta(popInt, ava, tampop)
    #faz crossover
    popInt = crossover(popInt,pcros)
    #faz mutação
    popInt = mutacao(popInt,pmut)
    #continuação elitismo
    if elitismo:
      popInt[0] = elit
    #adiciona melhor indivíduo e média às listas
    melhor = 0
    sum = 0
    for i in range(tampop):
      if ava[i]>melhor:
        melhor = ava[i]
      sum += ava[i]
    adapt.append(melhor)
    med.append(sum/tampop)

  #plota os gráficos
  graphGenAdapt(ngen,adapt,med)

  # printando o melhor indivíduo da última geração
  index = 0
  avaUlt = avalia(popInt)
  melhorUlt = avaUlt[index]
  for i in range(tampop):
    if avaUlt[i]>melhorUlt:
      melhorUlt = avaUlt[i]
      index = i
  print("Melhor indivíduo da última geração: ", popInt[index], "Com adaptação = ", melhorUlt)
```

- Saída e plot

```
# ------- gerando os gráficos -------

# Geração X Função de ataptação do melhor indivíduo e Adaptação média
def graphGenAdapt(ngen,adapt,med):

  # plota a linha de avaliação ótima
  maxAtk = 0
  for i in range(TAMTAB): #faz o somatório para determinar ataques máximos
    maxAtk += i
  plt.axhline(y=maxAtk, linestyle='--', color="red", label="Avaliação ótima")

  # plota os resultados obtidos do grafico Geração X Melhor individuo
  plt.plot(range(ngen+1),adapt,label="Melhor indivíduo de cada geração")
  #plt.xscale("log")
  plt.legend()
  plt.xlabel("Geração")
  plt.ylabel('Valor de ataptação')
  plt.title('Geração X Função de ataptação do melhor indivíduo')
  plt.show()

  # avaliação ótima
  plt.axhline(y=maxAtk, linestyle='--', color="red", label="Avaliação ótima")

  # plota os resultados obtidos do gráfico Geração X Média
  plt.plot(range(ngen+1),med,label="Média de cada geração")
  #plt.xscale("log")
  plt.legend()
  plt.xlabel("Geração")
  plt.ylabel('Média do valor de ataptação')
  plt.title('Geração X Adaptação média da geração')
  plt.show()
```

Análise dos resultados

a) Considerando o problema das 4-Rainhas:
   - Obtendo resultados com elitismo e sem elitismo para os parâmetros tamanho da população = 10, número de gerações = 10, probabilidade de crossover = 75% e probabilidade de mutação = 1%:

Caso sem elitismo - genetico(10,10,0.75,0.01)

```
1 Melhor indivíduo da última geração:  [1, 3, 1, 0] Com adaptação =  4
2 Melhor indivíduo da última geração:  [2, 0, 1, 3] Com adaptação =  5
3 Melhor indivíduo da última geração:  [0, 3, 1, 2] Com adaptação =  5
4 Melhor indivíduo da última geração:  [2, 1, 3, 0] Com adaptação =  5
5 Melhor indivíduo da última geração:  [3, 3, 0, 2] Com adaptação =  5
6 Melhor indivíduo da última geração:  [2, 3, 1, 0] Com adaptação =  4
7 Melhor indivíduo da última geração:  [2, 3, 1, 3] Com adaptação =  4
8 Melhor indivíduo da última geração:  [2, 0, 3, 0] Com adaptação =  5
9 Melhor indivíduo da última geração:  [1, 3, 0, 0] Com adaptação =  5
10 Melhor indivíduo da última geração:  [3, 3, 0, 2] Com adaptação =  5
```

Caso com elitismo - genetico(10,10,0.75,0.01,elitismo=True)

```
1 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
2 Melhor indivíduo da última geração:  [1, 0, 2, 0] Com adaptação =  4
3 Melhor indivíduo da última geração:  [0, 3, 1, 2] Com adaptação =  5
4 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
5 Melhor indivíduo da última geração:  [2, 0, 3, 0] Com adaptação =  5
6 Melhor indivíduo da última geração:  [1, 0, 2, 0] Com adaptação =  4
7 Melhor indivíduo da última geração:  [3, 1, 2, 2] Com adaptação =  4
8 Melhor indivíduo da última geração:  [2, 0, 3, 1] Com adaptação =  6
9 Melhor indivíduo da última geração:  [2, 0, 3, 3] Com adaptação =  5
10 Melhor indivíduo da última geração:  [2, 0, 3, 0] Com adaptação =  5
```

- Alterando os parâmetros de entrada para o caso com elitismo:

Aumentando o tamanho da população para 50 indivíduos:
genetico(50,10,0.75,0.01,elitismo=True)

```
1 Melhor indivíduo da última geração:  [2, 0, 3, 1] Com adaptação =  6
2 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
3 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
4 Melhor indivíduo da última geração:  [2, 1, 3, 0] Com adaptação =  5
5 Melhor indivíduo da última geração:  [2, 0, 3, 1] Com adaptação =  6
6 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
7 Melhor indivíduo da última geração:  [2, 0, 3, 1] Com adaptação =  6
8 Melhor indivíduo da última geração:  [2, 0, 3, 1] Com adaptação =  6
9 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
10 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
```

Aumentando o número de gerações para 50:
genetico(10,50,0.75,0.01,elitismo=True)

```
1 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
2 Melhor indivíduo da última geração:  [2, 1, 3, 0] Com adaptação =  5
3 Melhor indivíduo da última geração:  [3, 0, 2, 3] Com adaptação =  4
4 Melhor indivíduo da última geração:  [2, 0, 3, 1] Com adaptação =  6
5 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
6 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
7 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
8 Melhor indivíduo da última geração:  [3, 1, 0, 2] Com adaptação =  5
9 Melhor indivíduo da última geração:  [2, 1, 3, 0] Com adaptação =  5
10 Melhor indivíduo da última geração:  [2, 0, 1, 3] Com adaptação =  5
```

Aumentando a probabilidade de crossover para 85%:
genetico(10,10,0.85,0.01,elitismo=True)

```
1 Melhor indivíduo da última geração:  [1, 3, 2, 2] Com adaptação =  4
2 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
3 Melhor indivíduo da última geração:  [3, 0, 0, 1] Com adaptação =  4
4 Melhor indivíduo da última geração:  [1, 3, 0, 3] Com adaptação =  5
5 Melhor indivíduo da última geração:  [1, 3, 2, 0] Com adaptação =  5
6 Melhor indivíduo da última geração:  [1, 1, 2, 0] Com adaptação =  4
7 Melhor indivíduo da última geração:  [2, 0, 3, 1] Com adaptação =  6
```

```
8 Melhor indivíduo da última geração:  [0, 2, 3, 1] Com adaptação =  5
9 Melhor indivíduo da última geração:  [0, 3, 0, 2] Com adaptação =  5
10 Melhor indivíduo da última geração:  [3, 0, 3, 1] Com adaptação =  5
```

Aumentando a probabilidade de mutação para 5%:
genetico(10,10,0.75,0.05,elitismo=True)

```
1 Melhor indivíduo da última geração:  [2, 0, 1, 3] Com adaptação =  5
2 Melhor indivíduo da última geração:  [3, 1, 0, 2] Com adaptação =  5
3 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
4 Melhor indivíduo da última geração:  [3, 1, 2, 0] Com adaptação =  4
5 Melhor indivíduo da última geração:  [2, 0, 3, 1] Com adaptação =  6
6 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
7 Melhor indivíduo da última geração:  [2, 0, 3, 3] Com adaptação =  5
8 Melhor indivíduo da última geração:  [2, 0, 3, 3] Com adaptação =  5
9 Melhor indivíduo da última geração:  [1, 3, 0, 3] Com adaptação =  5
10 Melhor indivíduo da última geração:  [1, 3, 0, 2] Com adaptação =  6
```

Podemos observar que com uma população maior, o avanço da adaptação entre as gerações garantiu um melhor resultado, e este se manteve de maneira mais estável (não perdendo o melhor resultado para mutações maléficas) para a maioria das gerações futuras.

Com um maior número de gerações, conseguimos observar claramente a convergência do algoritmo, levando em consideração a média registrada da população.

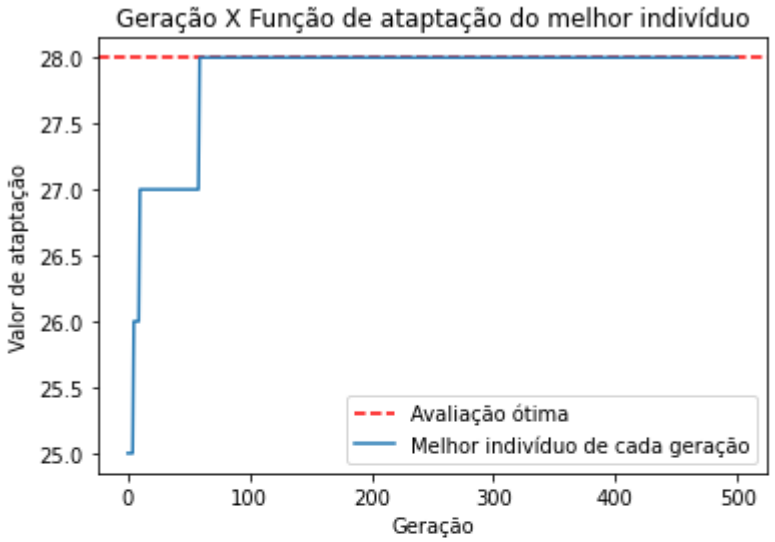
Ao aumentar o crossover com uma população pequena, observamos que esta se tornou homogênea rápido demais e chegou a um estado de estagnação precoce, esse problema se resolveria aumentando a probabilidade de mutação e o número de gerações.

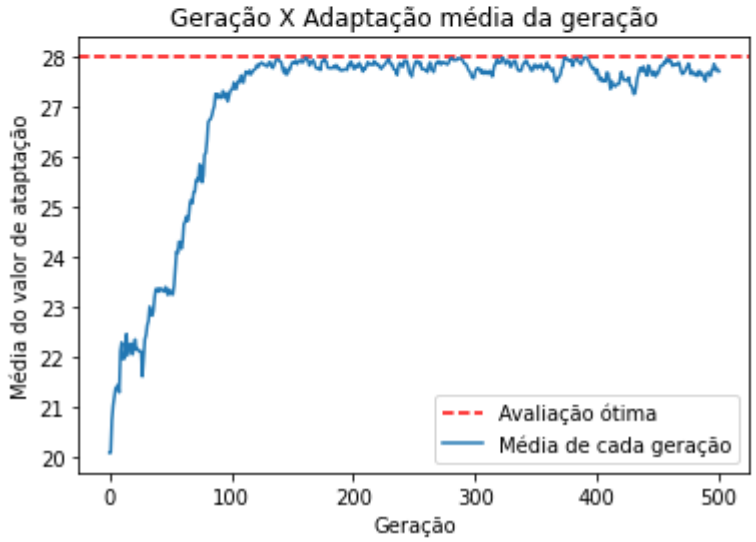
Quando aumentamos a mutação, em uma população pequena, como foi o caso, o resultado se torna muito mais aleatório, com a população sendo mais fortemente impactada tanto por mudanças boas quanto por mudanças ruins.

b) Analisando os tabuleiros com 8, 16 e 32 rainhas:
   - Quanto maior o tamanho do tabuleiro, maior a complexidade e tempo gasto para chegar a uma solução ótima, já que o número de rainhas que serão dispostas aumenta, portanto, seria necessário mais tempo computacional e mais gerações a fim de se alcançar um resultado ótimo.

A análise será feita com uma população de 100 indivíduos em um período de 500 gerações. Com probabilidade de crossover de 75% e de mutação 1%.
Será mostrado um gráfico referente à primeira vez que o algoritmo foi rodado, para ilustrar o comportamento do algoritmo.

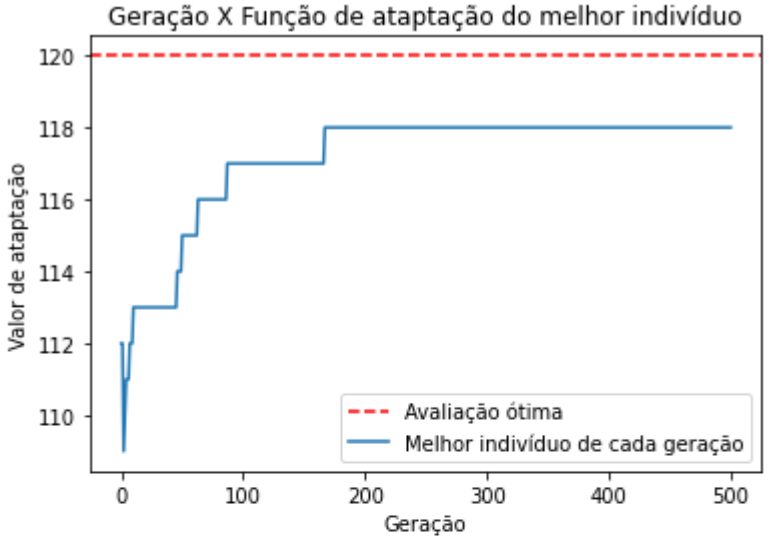
Tabuleiro com 8 rainhas

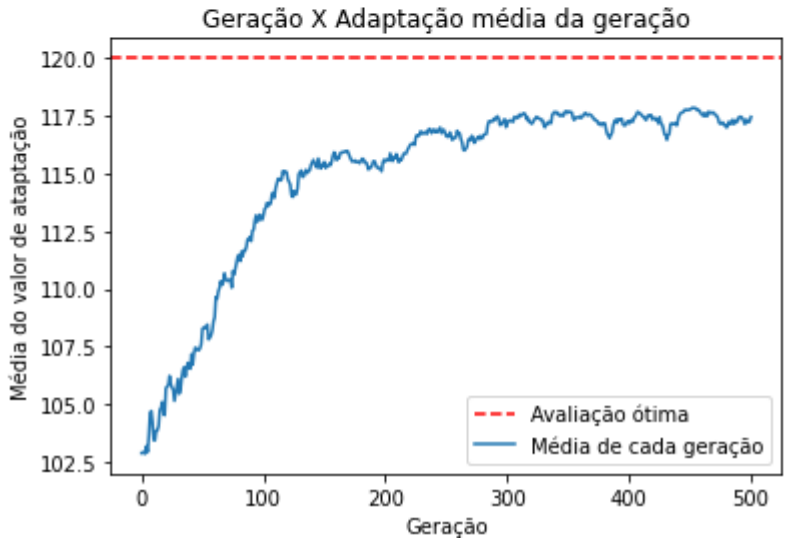





```
1 Melhor:  [5, 2, 0, 7, 3, 1, 6, 4] Adaptação =  28
2 Melhor:  [1, 3, 6, 0, 7, 5, 0, 2] Adaptação =  27
3 Melhor:  [3, 0, 4, 7, 1, 6, 2, 5] Adaptação =  28
4 Melhor:  [2, 4, 2, 7, 5, 3, 1, 6] Adaptação =  27
5 Melhor:  [1, 6, 2, 7, 7, 0, 3, 5] Adaptação =  27
6 Melhor:  [1, 4, 7, 0, 6, 3, 5, 7] Adaptação =  27
7 Melhor:  [0, 3, 4, 7, 1, 6, 2, 5] Adaptação =  27
8 Melhor:  [0, 5, 3, 6, 3, 7, 4, 1] Adaptação =  27
9 Melhor:  [0, 4, 7, 1, 3, 6, 2, 5] Adaptação =  27
10 Melhor:  [7, 4, 1, 5, 2, 6, 3, 7] Adaptação =  27
```

Tabuleiro com 16 rainhas

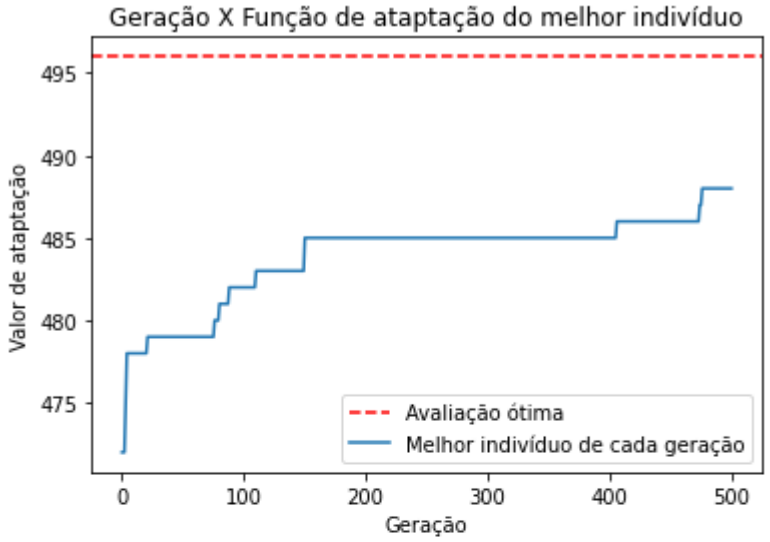


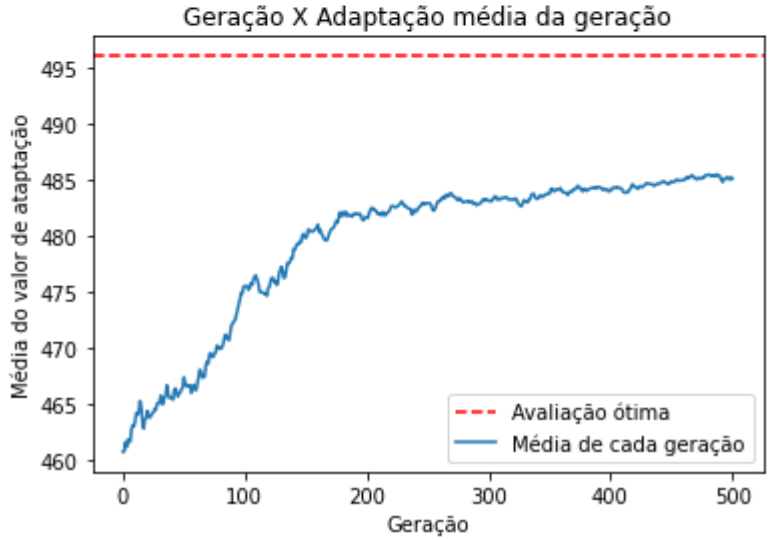



```
1 Melhor:  [6, 4, 7, 14, 11, 2, 15, 1, 9, 13, 10, 0, 7, 5, 12, 8]
Adaptação =  118
2 Melhor:  [1, 4, 11, 7, 12, 5, 0, 14, 3, 8, 15, 9, 0, 2, 10, 12]
Adaptação =  117
3 Melhor:  [10, 5, 14, 6, 0, 12, 7, 15, 13, 3, 1, 8, 11, 2, 4, 15]
Adaptação =  119
4 Melhor:  [6, 13, 15, 4, 1, 3, 5, 12, 8, 0, 2, 7, 14, 11, 9, 0]
Adaptação =  118
5 Melhor:  [15, 3, 6, 14, 11, 2, 7, 5, 8, 1, 13, 7, 9, 12, 0, 4]
Adaptação =  117
6 Melhor:  [7, 4, 11, 14, 8, 15, 0, 2, 0, 5, 1, 13, 9, 12, 12, 3]
Adaptação =  118
7 Melhor:  [14, 12, 11, 4, 1, 10, 3, 9, 2, 2, 13, 7, 0, 8, 6, 15]
Adaptação =  117
8 Melhor:  [9, 5, 9, 0, 14, 10, 2, 15, 7, 11, 4, 1, 12, 8, 3, 13]
Adaptação =  118
9 Melhor:  [8, 5, 3, 15, 7, 4, 8, 0, 14, 1, 10, 6, 1, 11, 2, 12]
Adaptação =  118
```

```
10 Melhor:  [4, 6, 1, 5, 14, 8, 13, 3, 7, 15, 11, 3, 5, 10, 12, 9]
Adaptação =  116
```

Tabuleiro com 32 rainhas





1 Melhor: [15, 10, 12, 29, 8, 5, 11, 23, 0, 26, 30, 14, 6, 31, 7, 16, 3, 1, 20, 1, 27, 2, 24, 19, 21, 3, 0, 26, 9, 4, 25, 29]
Adaptação = 488

2 Melhor: [20, 18, 2, 22, 5, 23, 17, 1, 13, 0, 14, 26, 14, 21, 28, 3, 19, 26, 30, 25, 9, 6, 12, 15, 31, 11, 24, 6, 23, 16, 8, 27]
Adaptação = 489

3 Melhor: [25, 14, 3, 13, 10, 23, 2, 30, 27, 20, 17, 15, 21, 30, 26, 29, 5, 4, 23, 29, 19, 1, 16, 8, 28, 11, 19, 24, 18, 31, 6, 1]
Adaptação = 487

```
4 Melhor:  [31, 0, 22, 5, 17, 26, 3, 12, 18, 21, 28, 25, 4, 6, 23, 18,
7, 11, 14, 20, 9, 2, 29, 1, 28, 10, 8, 13, 15, 24, 30, 19]
Adaptação =  490

5 Melhor:  [10, 30, 16, 4, 25, 3, 28, 18, 12, 2, 29, 10, 21, 28, 17, 9,
24, 0, 8, 4, 25, 7, 31, 14, 26, 13, 1, 11, 15, 26, 0, 20]
Adaptação =  487

6 Melhor:  [15, 21, 8, 0, 13, 30, 5, 7, 18, 28, 2, 22, 4, 21, 5, 27,
11, 1, 23, 25, 1, 6, 9, 13, 17, 29, 2, 16, 24, 20, 31, 19]
Adaptação =  486

7 Melhor:  [14, 4, 8, 26, 15, 11, 7, 1, 28, 24, 5, 7, 20, 29, 31, 22,
25, 30, 28, 6, 18, 3, 22, 28, 21, 9, 30, 1, 27, 10, 19, 0]
Adaptação =  487

8 Melhor:  [24, 5, 5, 20, 27, 4, 13, 15, 18, 2, 24, 7, 21, 28, 16, 0,
22, 30, 26, 10, 1, 4, 23, 12, 29, 7, 2, 19, 14, 23, 25, 8]
Adaptação =  489

9 Melhor:  [16, 22, 14, 9, 17, 6, 29, 2, 25, 31, 17, 15, 5, 11, 0, 23,
13, 28, 27, 24, 30, 8, 19, 2, 12, 21, 4, 1, 11, 18, 20, 23]
Adaptação =  487

10 Melhor:  [2, 10, 24, 31, 8, 1, 27, 29, 16, 8, 17, 30, 22, 24, 7, 28,
0, 12, 0, 19, 10, 7, 9, 6, 3, 26, 23, 30, 4, 18, 15, 19]
Adaptação =  485
```

Analisando os resultados, podemos ver que, para todos os casos, os resultados se aproximam do valor ótimo, apesar de não terem sido obtidos nenhuma vez.
É interessante notar que os três gráficos apresentam comportamentos bastante similares entre si. A maior mudança é que aparentemente, quanto menor o número de rainhas, mais rapidamente o resultado converge para o valor ótimo.