CÓDIGO PYTHON PARA ALGORITMO GENÉTICO (GA)

Editado por: Dr. Arnaldo de Carvalho Junior.

1. Introdução:

O Algoritmo Genético (AG) foi proposto por John Holland em 1975. Desde a sua origem, encontrou muitas aplicações interessantes em vários ramos da ciência e da engenharia.

Algoritmos Genéticos são rápidos, fáceis de implementar e altamente personalizáveis. Algoritmos Genéticos são amplamente utilizados em vários campos, como engenharia, finanças, inteligência artificial e problemas de otimização, onde os métodos tradicionais podem ser impraticáveis ou computacionalmente caros. Sua capacidade de explorar vastos espaços de soluções e encontrar soluções quase ideais os torna ferramentas valiosas para enfrentar desafios complexos de otimização.

O algoritmo começa com uma população inicial e, por meio de uma série de operações genéticas, como seleção, cruzamento (recombinação) e mutação, novas gerações são criadas. Os indivíduos mais aptos, aqueles com melhores soluções, têm maior chance de serem selecionados para produzir descendentes, que herdam características de seus pais. À medida que as gerações avançam, a população tende a evoluir em direção a melhores soluções, à medida que os indivíduos mais aptos dominam. Este processo continua até que um critério de terminação, como atingir um número máximo de gerações ou alcançar uma solução satisfatória, seja atendido.

Um código Python para algoritmos genéticos é implementado neste documento. Neste código Python para algoritmos genéticos, você pode implementar o algoritmo genético para seus requisitos específicos com pequenas modificações.

2. GA em Python

Os Anexos I e II trazem 2 exemplos de GA em Python.

Referências

EVOLUCIONARY GENIOS, Python code for Genetic Algorithm. Disponível em: https://evolutionarygenius.com/python-code-for-genetic-algorithms/. Acessado em Maio 13, 2024.

GEEKXFORGEEKS, Genetic Algorithms, GeeksforGeeks.org, 2024. Disponível em: <https://www.geeksforgeeks.org/genetic-algorithms/>. Acessado em Junho 04, 2025.

Anexo I

Segue o código Python simples e pronto para implementar para algoritmos genéticos.

Google Colab:

https://colab.research.google.com/drive/1rQPq-jEf8FaQb_IeY87i274Ti_0Wreug

```
# Ref: https://evolutionarygenius.com/python-code-for-genetic-algorithms/
# Adaptado por: Dr. Arnaldo de Carvalho Junior – maio 2025

import random

# Parâmetros do Algoritmo Genético
POPULATION_SIZE = 100
GENERATION_COUNT = 50
CROSSOVER_RATE = 0.8
MUTATION_RATE = 0.1
CHROMOSOME_LENGTH = 4
LOWER_BOUND = -5.12
UPPER_BOUND = 5.12

# Gera um cronossomo randômico
def generate_chromosome():
  return [random.randint(0, 1) for _ in range(CHROMOSOME_LENGTH)]

# Evalia a aptidão de um indivíduo
def evaluate_fitness(chromosome):
  x = decode_chromosome(chromosome)
  #Here you can change the objective function value
  fitness_value = sum([gene**2 for gene in x])
  return fitness_value

# Decodificar cromossomo binário à representação de valor real
def decode_chromosome(chromosome):
  x = []
  for gene in chromosome:
    value = LOWER_BOUND + (UPPER_BOUND - LOWER_BOUND) * int("".join(map(str,
chromosome)), 2) / (2 ** CHROMOSOME_LENGTH - 1)
    x.append(value)
  return x

# Realizar seleção usando a seleção de torneios
def selection(population):
  tournament_size = 5
  selected_parents = []
  for _ in range(len(population)):
    tournament = random.sample(population, tournament_size)
    winner = min(tournament, key=lambda x: x[1])
    selected_parents.append(winner[0])
```

```
  return selected_parents

# Faça um crossover entre dois pais
def crossover(parent1, parent2):
  if random.random() < CROSSOVER_RATE:
    crossover_point = random.randint(1, CHROMOSOME_LENGTH - 1)
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2
  else:
    return parent1, parent2

# Faça a mutação em um individuo
def mutate(individual):
  mutated_individual = individual.copy()
  for i in range(CHROMOSOME_LENGTH):
    if random.random() < MUTATION_RATE:
      mutated_individual[i] = 1 - mutated_individual[i]
  return mutated_individual

# Gera uma população inicial
population = [(generate_chromosome(), 0) for _ in range(POPULATION_SIZE)]

# Laço principal do algorítmo genético
for _ in range(GENERATION_COUNT):
  # Avalie a aptidão de cada indivíduo na população
  population = [(chromosome, evaluate_fitness(chromosome)) for chromosome, _ in
population]

  # Seleciona pais para reprodução
  parents = selection(population)

  # Criar filhos através do crossover e mutação
  offspring = []
  for i in range(0, POPULATION_SIZE, 2):
    parent1 = parents[i]
    parent2 = parents[i + 1]
    child1, child2 = crossover(parent1, parent2)
    child1 = mutate(child1)
    child2 = mutate(child2)
    offspring.extend([child1, child2])

  # Substitua a população velha com os filhos
  population = [(chromosome, 0) for chromosome in offspring]

# Selecione o melhor indivídulo como a solução
best_individual = min(population, key=lambda x: x[1])[0]
decoded_solution = decode_chromosome(best_individual)
fitness_value = evaluate_fitness(best_individual)
# Imprimir a solução
```

```
print("Melhor solução:", decoded_solution)
print("Melhor aptidão:", fitness_value)
```

```
Melhor solução: [-3.072, -3.072, -3.072, -3.072]
Melhor aptidão: 37.748736
```

Anexo II

Algoritmo para encontrar a frase objetivo.

Google Colab:

https://colab.research.google.com/drive/1mCTqkW46_vltVcTUX8_xQ19QOOgiHj8w

```
# Programa em Python3 para criar uma string alvo, iniciando de uma string
# aleatória com algoritmo genético (GA)
# Ref: https://www.geeksforgeeks.org/genetic-algorithms/
# Adaptado por: Dr. Arnaldo de Carvalho Junior - Junho 2025

import random

# Número de Indivíduos em cada geração
POPULATION_SIZE = 100

# Genes válidos
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''

# Frase (String) Objetivo a ser gerada
TARGET = "Eu gosto de pesquisar no EAILAB!"

class Individual(object):
  '''
  Classe representando indivíduos na população
  '''
  def __init__(self, chromosome):
    self.chromosome = chromosome
    self.fitness = self.cal_fitness()

  @classmethod
  def mutated_genes(self):
    '''
    cria genes aleatórios para mutação
    '''
    global GENES
    gene = random.choice(GENES)
    return gene

  @classmethod
  def create_gnome(self):
    '''
    cria cromossos ou string de genes
    '''
    global TARGET
    gnome_len = len(TARGET)
```

```
    return [self.mutated_genes() for _ in range(gnome_len)]

  def mate(self, par2):
    '''
    Realiza acassalamento e gera novos descendentes
    '''

    # Cromosso para descendentes
    child_chromosome = []
    for gp1, gp2 in zip(self.chromosome, par2.chromosome):

      # Probabilidade Aleatória
      prob = random.random()

      # Se a probabilidade < 0.45, inserir gene
      # do parente 1
      if prob < 0.45:
        child_chromosome.append(gp1)

      # Se a probabilidade < 0.45, inserir gene
      # gene do parente 2
      elif prob < 0.90:
        child_chromosome.append(gp2)

      # caso contrário, inserir gene (mutante) aleatório,
      # para manter diversidade
      else:
        child_chromosome.append(self.mutated_genes())

    # cria novo indivíduo (descendente) usando
    # cromossomo gerado para descendente
    return Individual(child_chromosome)

  def cal_fitness(self):
    '''
    Calcula o fitness score, ele é o número de caracteres
    na frase que diferem da frase objetivo.
    '''
    global TARGET
    fitness = 0
    for gs, gt in zip(self.chromosome, TARGET):
      if gs != gt: fitness+= 1
    return fitness

# Código driver
def main():
  global POPULATION_SIZE

  #geração atual
  generation = 1
```

```
  found = False
  population = []

  # Cria população inicial
  for _ in range(POPULATION_SIZE):
        gnome = Individual.create_gnome()
        population.append(Individual(gnome))

  while not found:

    # Organiza a população em ordem crecente de pontuação de aptidão
     # (fitness score)
    population = sorted(population, key = lambda x:x.fitness)

    # Se o indivíduo possui menor pontuação de aptidão, por exemplo:
    # 0 então o objetivo foi alcançado e o loop é interrompido
    if population[0].fitness <= 0:
      found = True
      break

    # Caso contrário, gerar novo descendente para nova geração
    new_generation = []

    # Realizar elitismo, que significa 10@ da população mais apta
    # segue para próxima geração
    s = int((10*POPULATION_SIZE)/100)
    new_generation.extend(population[:s])

    # De 50% da população mais apta, indivíduos From 50% of fittest population,
Individuals
    # irão acasalar para produzir descendentes
    s = int((90*POPULATION_SIZE)/100)
    for _ in range(s):
      parent1 = random.choice(population[:50])
      parent2 = random.choice(population[:50])
      child = parent1.mate(parent2)
      new_generation.append(child)

    population = new_generation

    print("Geração: {}\tFrase: {}\tAptidão: {}".\
       format(generation,
       "".join(population[0].chromosome),
       population[0].fitness))

    generation += 1


  print("Geração: {}\tFrase: {}\tAptidão: {}".\
     format(generation,
```

```
    "".join(population[0].chromosome),
    population[0].fitness))

if __name__ == '__main__':
  main()
```

```
Geração: 1      Frase: EM-w.RZ(#Gk "IC48)%/!z=oP{A F4"_ Aptidão: 28
Geração: 2      Frase: EM-w.RZ(#Gk "IC48)%/!z=oP{A F4"_ Aptidão: 28
Geração: 3      Frase: EM-w.RZ(#Gk "IC48)%/!z=oP{A F4"_ Aptidão: 28
Geração: 4      Frase: zuF7:zui
$e[ge%U&-ooKrnRK7,SAA%! Aptidão: 26
Geração: 5      Frase: zuF7:zui
$e[ge%U&-ooKrnRK7,SAA%! Aptidão: 26
Geração: 6      Frase: E_x1%ySf";e cuOCussR}dooA{u;LA"e Aptidão: 24
Geração: 7      Frase: E_x1%ySf";e cuOCussR}dooA{u;LA"e Aptidão: 24
Geração: 8      Frase: E_x1qyt(";? cu}CuisR%rnoA{9&LAWC Aptidão: 22
Geração: 9      Frase: E_x1qyt(";? cu}CuisR%rnoA{9&LAWC Aptidão: 22
Geração: 10     Frase: 8bI1%zt(#;Q ge[Cu$%gr now{AHLA"! Aptidão: 20
Geração: 11     Frase: EbMY%)t]#;e ge[TuisoK now{ZHLAkO Aptidão: 19
Geração: 12     Frase: EbMY%)t]#;e ge[TuisoK now{ZHLAkO Aptidão: 19
```

```
Geração: 938    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 939    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 940    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 941    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 942    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 943    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 944    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 945    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 946    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 947    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 948    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 949    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 950    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 951    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 952    Frase: Eu gosto de pesquisar no QAILAB! Aptidão: 1
Geração: 953    Frase: Eu gosto de pesquisar no EAILAB! Aptidão: 0
```