Algoritmo Genético e PSO. Na Prática

PROBLEMA 1 — Algoritmo Genético (AG)

Uma escola irá organizar um evento de extensão e possui várias atividades possíveis para incluir na programação.

Cada atividade possui um benefício (pontuação) e o evento possui um limite máximo de atividades que podem ser escolhidas.

O objetivo é selecionar o conjunto de atividades que maximize o benefício total, respeitando o limite estabelecido.

PROBLEMA 1 — Algoritmo Genético (AG)

Dados do problema

Considere as seguintes atividades:

Atividade	Benefício
A1	10
A2	5
A3	8
A4	6
A5	7

Cada atividade pode ser selecionada (1) ou não selecionada (0).

O evento pode conter no máximo 3 atividades.

O que é um Algoritmo Genético?

- Algoritmo Genético (AG) é uma técnica de otimização inspirada na evolução natural.
  - Ideia central:
  - Soluções competem entre si
  - As melhores sobrevivem
- Novas soluções surgem por cruzamento e mutação
  - Muito usado quando:
  - o espaço de busca é grande
  - não há solução exata simples
  - o problema é combinatório

Inspiração biológica

Biologia	Algoritmo Genético
Indivíduo	Solução candidata
Gene	Parte da solução
População	Conjunto de soluções
Seleção natural	Escolha das melhores
Mutação	Pequena alteração
Evolução	Melhoria ao longo do tempo

Quando usar o algoritmo genético?

- Use AG quando o problema:
  - tem muitas combinações possíveis
  - não é facilmente resolvido por fórmulas
  - envolve decisões discretas (sim/não)
- Exemplos:
  - seleção de atividades
  - mochila
  - escalonamento
  - seleção de atributos
  - planejamento

Representação da solução

No AG, precisamos representar uma solução.

Exemplo (vetor binário):

[1, 0, 1, 0, 1]

Interpretação:

1 → item selecionado

0 → item não selecionado

Função Fitness

A função de fitness mede o quão boa é uma solução.

- Quanto maior o fitness, melhor o indivíduo
- O AG tenta maximizar (ou minimizar) essa função

Exemplo:

- fitness = soma dos benefícios
- penalidade se violar restrições

Por que usar a biblioteca DEAP?

- DEAP (Distributed Evolutionary Algorithms in Python) é uma biblioteca popular para AG.
- Vantagens:
  - não implementamos o algoritmo do zero
  - estrutura clara e modular
  - usada em ensino e pesquisa
  - fácil de adaptar

Estrutura básica do DEAP

No DEAP, configuramos:

- 1 Tipo de fitness (max ou min)
- 2 Tipo de indivíduo
- 3 População
- 4 Operadores genéticos:
  - seleção
  - cruzamento
  - mutação
- 5 Avaliação (fitness)

```python
from deap import base, creator, tools
import random

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat,
                 creator.Individual, toolbox.attr_bool, n=5)
```

Aqui definimos:

solução = lista binária

objetivo = maximizar

Função Fitness

- Recebe um indivíduo
- Retorna uma tupla
- Quanto maior, melhor

```python
def fitness(individuo):
    beneficios = [10, 5, 8, 6, 7]
    return (sum(b * i for b, i in zip(beneficios, individuo)),)
```

Operadores Genéticos

- Cruzamento: mistura soluções
- Mutação: evita estagnação
- Seleção: escolhe os melhores

```
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=3)
```

Evolução da População

```
from deap import algorithms

pop = toolbox.population(n=20)

algorithms.eaSimple(
    pop, toolbox,
    cxpb=0.7,
    mutpb=0.2,
    ngen=30,
    verbose=False
)
```

ngen → número de gerações

cxpb → chance de cruzamento

mutpb → chance de mutação

Resultado Final

```
melhor = tools.selBest(pop, 1)[0]
print(melhor, melhor.fitness.values)
```

Pontos Importantes

- AG não é determinístico
- Resultado pode mudar a cada execução
- Parâmetros influenciam muito
- Boa modelagem é mais importante que o algoritmo

Como impor restrições em Algoritmo Genético?

Algoritmos Genéticos não lidam naturalmente com restrições.

A técnica mais comum é:

Penalizar soluções inválidas na função de fitness

Ou seja:

soluções válidas → fitness alto

soluções inválidas → fitness baixo

Restrições do problema

Temos 5 atividades

Cada indivíduo é um vetor binário

Restrição: MÁXIMO 3 ATIVIDADES

Nova função fitness

```python
def fitness(individuo):
    beneficios = [10, 5, 8, 6, 7]

    # Soma dos benefícios das atividades escolhidas
    valor = sum(b * i for b, i in zip(beneficios, individuo))

    # Número de atividades selecionadas
    total_atividades = sum(individuo)

    # Penalização se ultrapassar 3 atividades
    if total_atividades > 3:
        penalidade = (total_atividades - 3) * 10
        valor -= penalidade

    return (valor,)
```