Otimização de Enxame de Partículas

Vítor Rezende Silva

Agosto 2024

1 Introdução

O PSO (particle swarm optimization) é um algoritmo de otimização inspirado pelo comportamento social e cooperativo apresentado por várias espécies de forma a realizar as suas necessidades no espaço de busca, se guiando pelas experiências pessoais e globais. Nele, é criado uma população de partículas, que são movidas em torno do espaço de busca de acordo com uma função objetivo. Esse movimento é influenciado pela sua melhor posição conhecida e a melhor posição conhecida por sua vizinhança.

2 Detalhes de implementação

O algoritmo foi implementado na linguagem Python usando programação orientada a objetos. Nele existem duas classes necessárias para sua execução, Particula e Enxame.

2.1 Partícula

A classe partícula é definida utilizando os seguintes parâmetros:

- Xmin e Xmax: limites para as posições de uma partícula (-2.0, 2.0).
- Vmin e Vmax: limites para a velocidade de uma partícula (-1.0, 1.0).
- c1: parâmetro definido pelo usuário, implica na intensificação de fatores cognitivos.
- c2: parâmetro definido pelo usuário, implica na intensificação de fatores sociais.
- n: número de dimensões da representação.
- w: parâmetro definido pelo usuário, representa um fator de diversificação.
- X: vetor que define as coordenadas da partículas, gerado aleatóriamente no intervalo [Xmin, Xmax].

- V: vetor que define a velocidade dessa partícula, gerado aleatóriamente no intervalo [Vmin, Vmax].
- Pbest: melhor posição encontrada pela partícula.
- f: fitness da melhor posição.

Além disso, a classe partícula é definida pelos seguintes métodos:

- Função objetivo:

$$f(\mathbf{x}) = -a \exp\left(-\frac{1}{b}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + \exp(1)$$

- atualiza_velocidade: atualiza a velocidade da particula, utilizando como parâmetro o valor global de melhor posição.

$$v_{k+1,ij} = wv_{k,ij} + c_1 r_{1j}(p_{k\text{best},ij} - x_{k,ij}) + c_2 r_{2j}(g_{k\text{best},j} - x_{k,ij})$$

- atuliza_posicao: atualiza a posição da partícula. Também verifica se o fitness da nova posição é melhor que Pbest e Gbest, os substituindo se for o caso.

$$x_{k+1,i} = x_{k,i} + v_{k+1,i}$$

2.2 Enxame

A classe Enxame representa um conjunto de partículas. Seus atributos são:

- Xmin e Xmax.
- Vmin e Vmax.
- c1 e c2.
- n.
- w.
- k: representa o número de iterações.
- m: representa o número de partículas.
- particulas: vetor de partículas de tamanho m.
- history: vetor que armazena todas as posições de todas as partículas em uma dada iteração.
- Gbest: melhor posição global.
- Gbestf: fitness de Gbest.

2.3 Topologia

A topologia empregada é tal que toda partícula tem conhecimento da melhor posição global.

2.4 Representação gráfica

Foram implementadas duas formas de representação gráfica, uma para problemas de dimensão n = 2 e outra para n = 3.

Essas representações foram feitas em forma de gifs que representam a posição de cada particula em cada iteração e como elas se movimentam.

3 Execução

Para executar o algoritmo basta instanciar um Enxame E e chamar o método PSO. Os atributos de E devem ser dados na seguinte ordem: Xmin, Xmax, Vmin, Vmax, c1, c2, k, m, n, 2

Por exemplo:

```
E = Enxame(-2.0, 2.0, -1.0, 1.0, 1.5, 2.0, 200, 100, 3, 0.7)
gbest, gbestf = E.PSO()
E.save_gif()
print(gbest, gbestf)
```

No código acima o enxame E é instanciado, logo em seguida chama o metodo PSO e o método save_gif para gerar a representação gráfica.

4 Testes

Os testes foram feitos utilizando o seguinte script:

```
from PSO import Enxame, Particula
import matplotlib.pyplot as plt

file = open("iteracoes.txt", 'w')

iteracoes = []
resultados = []
rotas = []
melhor = 1000000
cont = 0

for i in range(1000):
    print(i)
    E = Enxame(-2.0, 2.0, -1.0, 1.0, 1.5, 2.0, 100, 100, 3, 0.7)
    gbest, gbestf = E.PSO()
```

```
        file.write(str(gbestf) + " " + str(gbest) + "\n")
        iteracoes.append(i)
        resultados.append(gbestf)
        if gbestf < melhor:
            melhor = gbestf
            cont += 1

print(f"Melhor resultado encontrado: {melhor}\nVezes que um valor menor foi encontrado: {cor
plt.plot(iteracoes, resultados)
plt.xlabel('Iteração')
plt.ylabel('Fitness')
plt.show()
```

Ou seja, o algoritmo é executado 1000 vezes para um certo conjunto de parâmetros salvando os melhores resultados. Ao fim da execução o melhor valor de fitness é exibido, bem como um gráfico com a variação de resultados.

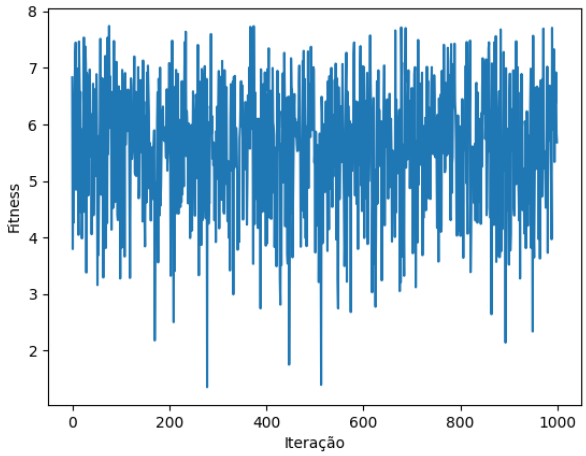
4.1 Teste 1

O primeiro teste foi realizado para o conjunto de parâmetros:

```
E = Enxame(-2.0, 2.0, -1.0, 1.0, 1.5, 2.0, 100, 100, 3, 0.7)
```

Ou seja, um problema com dimensão n = 3.

O menor valor encontrado foi 1.3469376330572023 para as coordenadas [0.11058796, 0.17945982, -0.11376732] e o gráfico gerado foi o seguite:



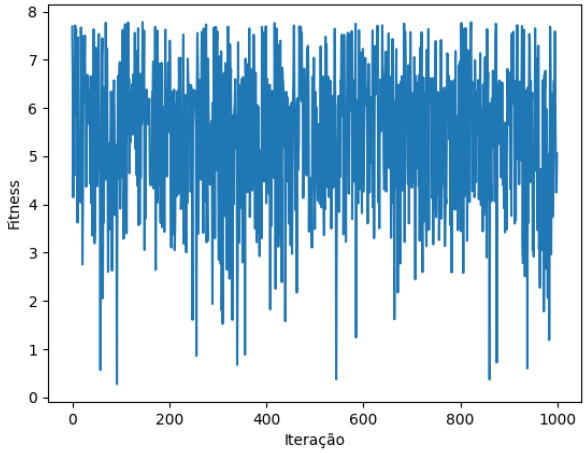
4.2 Teste 2

O segundo teste foi realizado para o conjunto de parâmetros:

```
E = Enxame(-2.0, 2.0, -1.0, 1.0, 1.5, 2.0, 100, 100, 2, 0.7)
```

Ou seja, os mesmos do primeiro teste, mas com o número de dimensões n = 2.

O menor valor encontrado foi 0.2764413658233811 para as coordenadas [0.03946369, -0.04810036] e o gráfico gerado foi o seguinte



É importante mencionar que em ambos os testes, o melhor valor foi encontrado apenas uma vez.