

SOCIAL SPIDER OPTIMIZATION (SSO)

DAVI ISMAEL PALUCH

Introdução

Uma técnica de otimização inspirada no comportamento social das aranhas.

Aranhas interagem para encontrar soluções ótimas.

Modelagem da População de Aranhas

Cada solução é uma aranha com peso baseado em sua aptidão.

População (S): Todas as soluções possíveis.

Subconjuntos: Masculino (Ms) e Feminino (Fs).

Nf: 65-95% da população total.

$$N_m = S - N_f$$

Cálculo de Peso da Aranha

Segundo Luque-Chang et al. (2018), “cada aranha recebe um peso de acordo com o valor de aptidão da solução que ela simboliza.

- S : é o conjunto de todas as soluções candidatas.
- fit_i : é a aptidão da i -ésima solução candidata.
- wei : é o peso associado à i -ésima aranha.
- $best$: é o melhor valor de aptidão em “ S ”.
- $worst$: é o pior valor de aptidão em “ S ”.

$$we_i = \frac{fit_i - worst}{best - worst}$$

Cálculo de Vibrações

“a troca de informações é o principal mecanismo do SSO no processo de otimização”. A vibração que uma aranha percebe de outra é modelada como:

Vib $_{i,j}$: **vibração** que onde ocorre troca de informação entre i e j

w_j : é o peso da j -ésima aranha.

d : é a distância entre as aranhas i e j .

$-d^2_{i,j}$: é a “distância Euclidiana entre as aranhas i e j ”, como por exemplo:

$$d_{i,j} = \|\mathbf{s}_i - \mathbf{s}_j\|$$

$$Vib_{i,j} = w_j \cdot e^{-d^2_{i,j}}$$

Exemplificando Vibrações

Suponha que temos duas aranhas, i e j , e que a aranha j está mais próxima de i e tem um peso w_{ij} de 0.8. Além disso, a distância d_{ij} entre elas é de 5 unidades

$$Vib_{ij} = \frac{w_j}{d_{ij}^2} = \frac{0.8}{5^2} = \frac{0.8}{25} = 0.032$$

Vib_{ci}

São percebidas pelo indivíduo i como resultado das informações transmitidas pelo membro c que é um indivíduo que possui duas características importantes: é o membro mais próximo de i e possui um peso maior em comparação com i ($w_c > w_i$)

$$Vib_{ci} = \frac{w_c}{d_{ci}^2}$$

Vib_{bi}

São percebidas pelo indivíduo i como resultado das informações transmitidas pelo membro b , onde b é o indivíduo que possui o melhor peso (melhor valor de aptidão) de toda a população S , tal que (peso do indivíduo b) seja igual a .

$$w_b \max\{w_k \mid k \in N\}$$

$$Vib_{bi} = \frac{w_b}{d_{bi}^2}$$

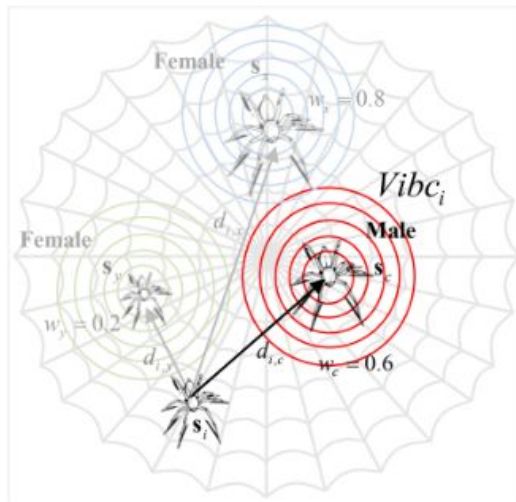
$$Vib_{fi}$$

São percebidas pelo indivíduo i como resultado das informações transmitidas pelo membro f , onde f é o indivíduo feminino mais próximo de i .

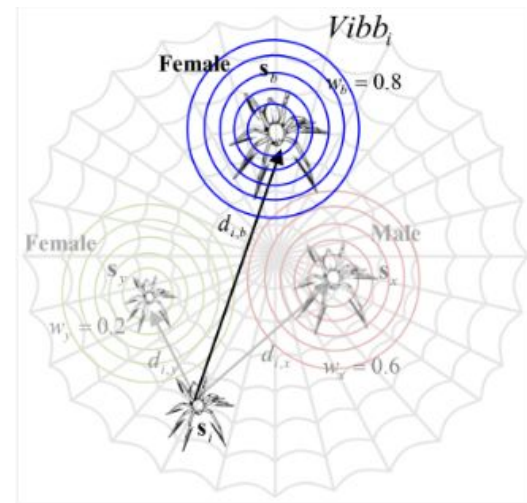
$$Vib_{fi} = \frac{w_f}{d_{fi}^2}$$

Visualização dos dados

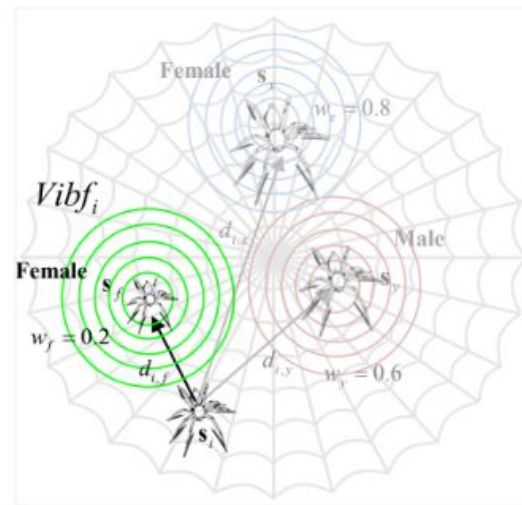
Na exemplificação a seguir podemos visualizar as 3 diferentes vibrações ou classificações para encontrar determinado indivíduo mais apto para solucionar um problema.



(a)



(b)



(c)

Estrutura dos Resultados

- Valor de fitness (fitness_value):


O valor de fitness indica a qualidade da solução. Em muitos algoritmos de otimização, o objetivo é minimizar ou maximizar essa função de fitness. No exemplo dado, os valores de fitness são negativos, o que sugere que a função objetivo é tal que valores menores (mais negativos) representam soluções melhores.

- Solução (solution):

A solução é uma lista de variáveis que foram ajustadas pelo algoritmo para tentar encontrar o valor ótimo da função objetivo. Neste caso específico, cada solução é composta por dois valores (porque $n = 2$).

QUALIDADE + [SOLUÇÕES]

python

 Copiar código

```
(-65.46893430797135, [2.69000086, 6.72054731])
```

- **Valor de fitness:** -65.46893430797135
- **Solução:** [2.69000086, 6.72054731]

Lista de Resultados

A lista `results` contém várias tuplas, cada uma representando um resultado de uma execução do algoritmo. Cada tupla inclui o valor de fitness e a solução correspondente. (51 results)

```
results = [  
    (-65.46893430797135, [2.69000086, 6.72054731]),  
    (-75758.21053478548, [10.09693194, 80.08705662]),  
    ...  
    (-30718.547768029537, [-3.85720852, -2.54384416])  
]
```

Função de Benchmark

```
def test_3():  
    global population, population_male, population_female, y, n, spiders, lim, pf,  
    rand = random.random() # random [0,1]  
    population = 100  
    population_female = int((0.9 - rand * 0.25) * population)  
    population_male = population - population_female  
    y = "-100*(z[1]-z[0]**2)**2 - (1 - z[0]**2)**2"  
    n = 2  
    bounds = np.array([[-100, 100], [-100, 100]])  
    lim = 200  
    pf = 0.7
```

Estatísticas para funções de Benchmark

```
=====
Estatísticas para a função de Rosenbrock
=====
Melhor Resultado: -6.385912689274824
Pior Resultado: -14678.091209495211
Média: -3609.3993451282013
Mediana: -2408.180197851583
Desvio Padrão: 3668.111434273947
```

```
=====
Estatísticas para a função de Rastrigin
=====
Melhor Resultado: -0.12258759839685496
Pior Resultado: -9.884372381859741
Média: -3.5334069430547226
Mediana: -3.076867985203741
Desvio Padrão: 1.9556075532072519
```

OBRIGADO!