Trabalho Final Otimização em Meta-Heurística

Daniel Levacov

7 de abril de 2025

Resumo

Este é um estudo detalhando as diferenças entre os algoritmos de otimização por enxame de partículas DEEPSO e C-DEEPSO, utilizando as funções de benchmark de Rosenbrock e Rastrigin em 10, 30 e 50 dimensões para testá-las.

1. Introdução

A otimização é uma área essencial da ciência da computação e da pesquisa operacional, aplicável em diversos âmbitos, desde a engenharia até a economia. Dentro desse campo, algoritmos bioinspirados têm se mostrado particularmente eficazes na resolução de problemas complexos e multidimensionais. Entre esses algoritmos, destacam-se o DEEPSO (Particle Swarm Optimization Differential Evolution Enhanced) e sua variação, o C-DEEPSO (Combinatorial Differential Evolutionary Particle Swarm Optimization).

O DEEPSO é uma técnica híbrida que combina os princípios do Particle Swarm Optimization (PSO) e do Differential Evolution (DE). O PSO, inspirado no comportamento social de animais como pássaros e peixes, utiliza partículas que se movem pelo espaço de busca influenciadas tanto pela sua própria experiência quanto pela experiência coletiva do grupo. Já o DE, inspirado na evolução genética, usa operadores como mutação, recombinação e seleção para explorar o espaço de soluções. A combinação dessas duas abordagens no DEEPSO resulta em um algoritmo robusto que aproveita a exploração global do PSO e a exploração local eficiente do DE.

Por outro lado, o C-DEEPSO é uma extensão do DEEPSO que incorpora técnicas adicionais para aprimorar o desempenho em problemas de otimização combinatória. Ele se destaca por utilizar memórias distintas, como a "b-memory"(Best Memory), que armazena as melhores soluções encontradas pelas partículas, e a "c-memory"(Current Memory), que retém as posições atuais das partículas. Essas memórias permitem ao C-DEEPSO lidar de maneira mais eficaz com problemas complexos e de alta dimensionalidade.

Neste artigo, comparamos os desempenhos dos algoritmos DEEPSO e C-DEEPSO utilizando duas funções de benchmark conhecidas: a função de Rosenbrock e a função de Rastrigin. Estas funções são amplamente utilizadas para avaliar a eficácia de algoritmos de

otimização devido às suas características desafiadoras, como a presença de múltiplos mínimos locais (no caso da Rastrigin) e um vale estreito que se estende em direção ao mínimo global (no caso da Rosenbrock). Os testes foram realizados em três diferentes dimensões: 10, 30 e 50, proporcionando uma análise abrangente do comportamento dos algoritmos em diferentes cenários de complexidade.

Ao longo deste estudo, investigamos como a estrutura e as características específicas do DEEPSO e do C-DEEPSO influenciam seu desempenho.

2. O cálculo

2.1. Funções de Rosenbrock e Rastrigin

As funções de Rosenbrock e Rastrigin são amplamente utilizadas como benchmarks para testar a performance de algoritmos de otimização devido às suas propriedades distintas que desafiam as técnicas de busca de ótimos globais.

2.1.1. Função de Rosenbrock

A função de Rosenbrock, também conhecida como função do vale ou banana, é uma função não convexa utilizada como benchmark para testar o desempenho de algoritmos de otimização. A função é definida pela fórmula:

$$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right]$$

Onde $x = (x_1, x_2, \ldots, x_n)$ é um vetor de variáveis de decisão e n é a dimensionalidade do problema.

Características principais da função de Rosenbrock:

- Possui um mínimo global em $f(1, 1, \ldots, 1) = 0$.
- A função cria um vale estreito e curvo que contém o mínimo global, tornando a busca por este mínimo um desafio para muitos algoritmos de otimização.

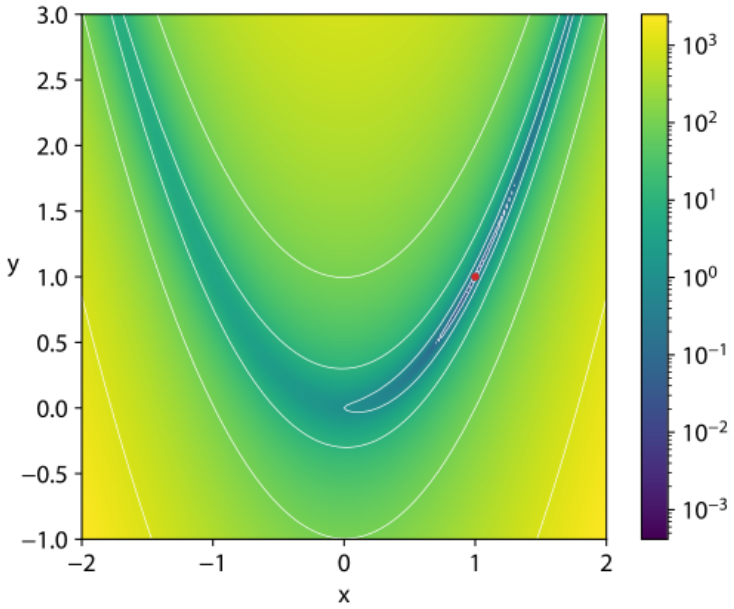

Figura 2.1: Contorno da função de Rosenbrock 2D

2.1.2. Função de Rastrigin

A função de Rastrigin é uma função multimodal usada para testar a capacidade de algoritmos de otimização de escapar de ótimos locais. A função é definida pela fórmula:

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) \right]$$

Onde $x = (x_1, x_2, \ldots, x_n)$ é um vetor de variáveis de decisão e n é a dimensionalidade do problema.

Características principais da função de Rastrigin:

- Possui um mínimo global em $f(0, 0, \ldots, 0) = 0$.
- A função apresenta uma grande quantidade de mínimos locais, o que desafia a habilidade dos algoritmos de otimização de encontrar o mínimo global.

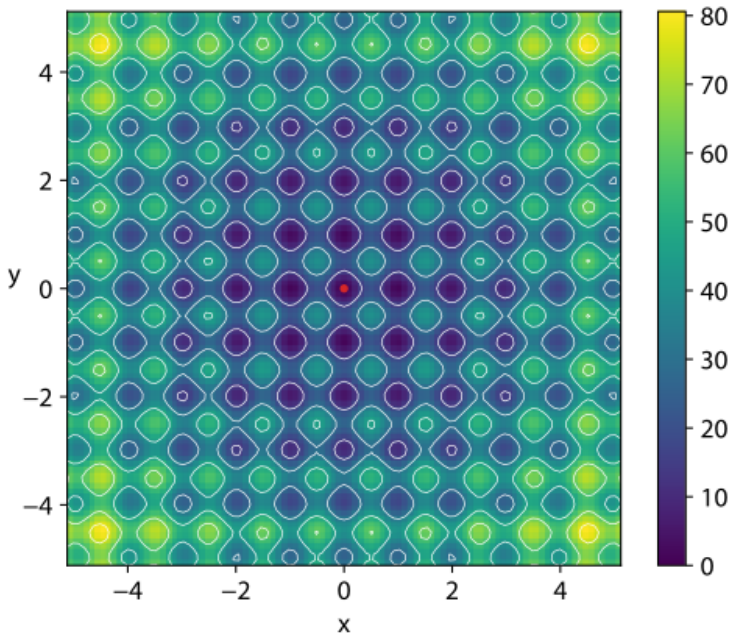

Figura 2.2: Contorno da função de Rastrigin 2D

2.2. Cálculo do DEEPSO

O DEEPSO (Particle Swarm Optimization Differential Evolution Enhanced) é um algoritmo híbrido que combina elementos do Particle Swarm Optimization (PSO) e do Differential Evolution (DE). O objetivo do DEEPSO é aproveitar as vantagens de ambas as técnicas para resolver problemas de otimização de maneira eficiente e robusta.

2.2.1. Componentes e Operações do DEEPSO

1. **Inicialização das Partículas:**
   - Cada partícula representa uma solução candidata no espaço de busca.
   - As partículas são inicializadas com posições aleatórias e velocidades iniciais iguais a 0.
2. **Atualização de Velocidade e Posição:**
   - As partículas são atualizadas com base em suas velocidades e posições, seguindo a fórmula do PSO modificada pelo operador de mutação do DE.
   - A velocidade de uma partícula i é atualizada de acordo com a equação:

$$V_t = w_i^* V_{t-1} + w_p^*(X_r - X_{t-1}) + w_c^* C(X_{gb}^* - x_{t-1})$$

   Onde:

  - w_i é o fator de inércia, controlando a influência da velocidade anterior
  - w_p e w_c são os coeficientes cognitivo e social respectivamente
  - X_r é um vetor, que aqui é definido como o melhor valor já atingido pelo fitness da partícula i (seu "best")
  - X_{gb} é a melhor posição já encontrada por todo o enxame
  - w_i^*, w_p^*, w_c^* e X_{gb}^* são versões mutadas das respectivas variáveis
- A nova posição da partícula i é então calculada como:

$$X_t = X_{t-1} + V_t$$

3. **Avaliação das Partículas:**
   - Cada nova posição é avaliada usando a função objetivo do problema.
   - Se a nova posição de uma partícula é melhor que sua posição de melhor histórico ($pBest_i$), então $pBest_i$ é atualizado.
   - Se a nova posição de uma partícula é melhor que a posição de melhor histórico global ($gBest$), então $gBest$ é atualizado.
4. **Memória de Partículas:**
   - **b-memory (Best Memory):** Armazena as melhores soluções encontradas pelas partículas.
   - **c-memory (Current Memory):** Armazena as posições atuais das partículas.
   - Estas memórias ajudam a guiar as partículas na exploração do espaço de busca e na retenção de boas soluções.
5. **Convergência:**
   - O processo de atualização de velocidades, posições e memórias continua iterativamente até que um critério de parada seja atendido, como um número máximo de iterações ou uma convergência para uma solução satisfatória. No nosso caso estamos utilizando o critério de parada de 100 iterações.

2.3. Cálculo do C-DEEPSO

O C-DEEPSO (Competitive Differential Evolution Particle Swarm Optimization) é uma variante do DEEPSO que introduz uma abordagem competitiva para melhorar o desempenho do algoritmo em termos de exploração do espaço de busca. Ele combina o PSO, DE e um mecanismo de competição para selecionar as melhores soluções.

2.3.1. Componentes e Operações do C-DEEPSO

1. **Inicialização das Partículas:**

   - Cada partícula representa uma solução candidata no espaço de busca.
   - As partículas são inicializadas com posições aleatórias e velocidades iniciais iguais a 0.

2. **Atualização de Velocidade e Posição:**

   - As partículas são atualizadas com base em suas velocidades e posições, seguindo a fórmula do PSO modificada pelo operador de mutação do DE.
   - A velocidade de uma partícula i é atualizada de acordo com a equação:

   $$V_t = w_i^* V_{t-1} + w_A^*(X_{st} - X_{t-1}) + w_c^* C(X_{gb}^* - x_{t-1})$$

   Onde:

   - w_i é o fator de inércia, controlando a influência da velocidade anterior
   - w_A e w_c são os coeficientes cognitivo e social respectivamente
   - X_st é um vetor, cuja definição será explicada posteriormente
   - X_{gb} é a melhor posição já encontrada por todo o enxame
   - w_i^*, w_p^*, w_c^* e X_{gb}^* são versões mutadas das respectivas variáveis

   - A nova posição da partícula i é então calculada como:

   $$X_t = X_{t-1} + V_t$$

   - O cálculo de X_{st} é dado da seguinte forma:

   $$X_{st} = X_r + F(X_{best} - X_r) + F(X_{r1} - X_{r2})$$

   Onde:

   - X_{best} é a posição com melhor fitness já encontrada pela partícula
   - X_r é um vetor que combina as posições de uma partícula do enxame atual, e uma da memória b
   - X_{r1} e X_{r2} são vetores com a posição de uma partícula da memória b da partícula (caso sua memória b tenha pelo menos 3 membros) ou de um membro aleatório do enxame atual (caso contrário)
   - F é um fator específico do C-DEEPSO

3. **Avaliação das Partículas:**

   - Cada nova posição é avaliada usando a função objetivo do problema.

- Se a nova posição de uma partícula é melhor que sua posição de melhor histórico ($pBest_i$), então $pBest_i$ é atualizado.
- Se a nova posição de uma partícula é melhor que a posição de melhor histórico global ($gBest$), então $gBest$ é atualizado.

4. **Memória de Partículas:**
    - **b-memory (Best Memory):** Armazena as melhores soluções encontradas pelas partículas.
    - **c-memory (Current Memory):** Armazena as posições atuais das partículas.
    - Estas memórias ajudam a guiar as partículas na exploração do espaço de busca e na retenção de boas soluções.
5. **Convergência:**
    - O processo de atualização de velocidades, posições, memórias e competição continua iterativamente até que um critério de parada seja atendido, como um número máximo de iterações ou uma convergência para uma solução satisfatória. No nosso caso estamos utilizando o critério de parada de 100 iterações.

2.4. Hill Climbing

Hill Climbing é uma técnica de otimização que busca encontrar a solução mínima (ou máxima) para um problema começando com uma solução arbitrária e, iterativamente, fazendo pequenos ajustes para melhorar essa solução. Este método é baseado em uma busca local, onde em cada iteração a solução atual é substituída por uma vizinha melhor, até que não seja possível encontrar uma melhoria.

O algoritmo de Hill Climbing pode ser descrito da seguinte forma:

1. **Inicialização**: Comece com uma solução inicial aleatória.
2. **Avaliação**: Avalie a solução atual.
3. **Movimento**: Realize um pequeno ajuste na solução atual para gerar uma nova solução vizinha.
4. **Substituição**: Se a nova solução vizinha for melhor do que a solução atual, substitua a solução atual pela nova solução.
5. **Parada**: Repita os passos 2 a 4 até que um critério de parada seja atendido (por exemplo, número máximo de iterações ou não haver melhoria após um certo número de tentativas).

O Hill Climbing é um método simples e intuitivo, mas possui algumas limitações, como a tendência a ficar preso em mínimos (ou máximos) locais.

No contexto dos algoritmos DEEPSO e C-DEEPSO, o Hill Climbing pode ser utilizado para melhorar a qualidade das soluções encontradas, explorando as vizinhanças das partículas (ou soluções) de forma mais eficiente. Essa integração pode potencialmente acelerar a convergência e melhorar a precisão dos resultados obtidos pelos algoritmos.

3. Teste t (t-test)

O teste t, ou t-test, é uma ferramenta estatística utilizada para determinar se há uma diferença significativa entre as médias de dois grupos independentes. Ele assume que os dados seguem uma distribuição normal e que as variâncias dos dois grupos são aproximadamente iguais.

Existem dois tipos principais de teste t:

3.1. Teste t de Student para amostras independentes

O teste t de Student para amostras independentes compara as médias de duas amostras independentes para determinar se elas são estatisticamente diferentes uma da outra.

3.2. Teste t pareado (ou emparelhado)

O teste t pareado, também conhecido como t-test de amostras pareadas, é usado quando as observações em cada amostra são emparelhadas ou relacionadas de alguma forma. Ele compara as médias das diferenças entre as observações emparelhadas.

Em ambos os casos, o teste t produz um valor t e um p-value associado. O valor t é calculado como a diferença entre as médias das duas amostras, dividida pela variabilidade das amostras. O p-value indica a probabilidade de obter um valor t igual ou mais extremo se a hipótese nula (geralmente de igualdade das médias) fosse verdadeira.

Neste estudo iremos utilizar o teste t de student para amostras independentes.

4. Implementação

Todo o código do projeto foi implementado em Python, e está disponível no link do Google Colaboratory.

4.1. Implementação do DEEPSO

A fórmula de atualização de velocidade e posição:

$$V_t = w_i^* V_{t-1} + w_A^*(X_{st} - X_{t-1}) + w_c^* C(X_{gb}^* - x_{t-1})$$

foi implementada da seguinte forma:

```
178
179             # Atualiza a velocidade
180             parte1 = w_i_mut * velocidades[p]
181             parte2 = w_m_mut * (x_bests[p] - posicoes[p])
182
183             pre1_parte3 = (x_gb_mut - posicoes[p])
184             pre2_parte3 = transpoe(pre1_parte3)
185
186             pre3_parte3 = C @ pre2_parte3
187             pre4_parte3 = transpoe(pre3_parte3)
188
189             parte3 = w_s_mut * pre4_parte3
190             velocidades[p] = parte1 + parte2 + parte3
191
192             # Atualiza a posição
193             posicoes[p] += velocidades[p]
194
```

Figura 4.1: Código da fórmula de atualização de velocidade e posição de partículas do DEEPSO

4.2. Implementação do C-DEEPSO

A fórmula de atualização de velocidade e posição:

$$V_t = w_i^* V_{t-1} + w_A^*(X_{st} - X_{t-1}) + w_c^* C(X_{gb}^* - x_{t-1})$$

foi implementada da seguinte forma:

```
306            # Atualiza a velocidade
307            parte1 = w_i_mut * velocidades[p]
308            parte2 = w_m_mut * (x_st - posicoes[p])
309
310            pre1_parte3 = (x_gb_mut - posicoes[p])
311            pre2_parte3 = transpoe(pre1_parte3)
312
313            pre3_parte3 = C @ pre2_parte3
314            pre4_parte3 = transpoe(pre3_parte3)
315
316            parte3 = w_s_mut * pre4_parte3
317            velocidades[p] = parte1 + parte2 + parte3
318
319            # Atualiza a posição
320            posicoes[p] += velocidades[p]
321
```

Figura 4.2: Código da fórmula de atualização de velocidade e posição de partículas do C-DEEPSO

Já a fórmula de criação do vetor X_{st} foi implementada da seguinte forma:

```
276            # Cria o Xst
277            # Xst = Xr + F(Xbest - Xr) + F(Xr1 - Xr2)
278            pos_random = np.copy(random.choice(posicoes))
279            pos_random_b = np.copy(random.choice(np.array(memoria_b[p])))
280            x_r = (pos_random + pos_random_b) / 2
281            if len(memoria_b[p]) >= 3:
282                x_r1 = np.copy(random.choice(memoria_b[p]))
283                x_r2 = np.copy(random.choice(memoria_b[p]))
284            else:
285                x_r1 = np.copy(random.choice(posicoes))
286                x_r2 = np.copy(random.choice(posicoes))
287            x_st = np.copy(x_r) + F * (x_bests[p] - x_r) + F * (x_r1 - x_r2)
```

Figura 4.3: Código da criação do vetor X_{st} no C-DEEPSO

4.3. Implementação do Hill Climbing

O algoritom de Hill Climbing (ou busca local) foi ativado entre as gerações 20 e 80 dos algoritmos DEEPSO e C-DEEPSO.

```python
def hill_climb(funcao, dim, posicao, fitness, fator_max):
    pos = np.copy(posicao)

    cand = pos + np.random.uniform(-fator_max, fator_max, dim)
    cand = np.clip(cand, -5, 5)
    fitness_cand = funcao(cand, dim)

    if fitness_cand < fitness:
        return cand, fitness_cand, True
    else:
        return posicao, fitness, False
```

Figura 4.4: Definição da função de Hill Climbing

```python
log_hc = ""
if iter >= 20 and iter < 80 and com_hill_climb == True:
    fator_max = x_gb_fitness
    pos_hc, fitness_hc, funcionou = hill_climb(funcao, dim, x_gb, x_gb_fitness, fator_max)
    log_hc = x_gb_fitness - fitness_hc
    if funcionou == True:
        x_gb = np.copy(pos_hc)
        x_gb_fitness = fitness_hc
```

Figura 4.5: Chamada da função de Hill Climbing no DEEPSO e C-DEEPSO

4.4. Definição dos Hiperparâmetros

A ferramenta utilizada para definir os hiperparâmetros dos algoritmos a fim de atingir a melhor solução possível foi o Optuna.

Optuna é uma biblioteca Python de otimização de hiperparâmetros automatizada e de código aberto, projetada para otimizar parâmetros de modelos de machine learning de forma eficiente. Ela utiliza técnicas de busca automatizada, como otimização bayesiana e busca em grade, para encontrar os melhores conjuntos de parâmetros de forma automatizada, economizando tempo e recursos computacionais.

O seguinte código consiste da função objetivo criada para achar os hiperparâmetros ideais:

```python
# Define the objective function for Optuna
def objective(trial, algoritmo, funcao, dim):
    w_i = trial.suggest_float('w_i', 0.01, 0.9)
    w_m = trial.suggest_float('w_m', 0.01, 0.9)
    w_s = trial.suggest_float('w_s', 0.01, 0.9)
    t_mut = trial.suggest_float('t_mut', 0.01, 1)
    t_com = trial.suggest_float('t_com', 0.01, 1)
    if algoritmo == meu_cdeepso: F = trial.suggest_float('F', 0.01, 2)
    num_particulas = trial.suggest_int('num_particulas', 10, 100)
    # max_iter = trial.suggest_int('max_iter', 50, 500)

    best_fitness = float('inf')
    if algoritmo == meu_cdeepso: cdeepso_gen = meu_cdeepso(funcao, w_i, w_m, w_s, t_mut, t_com, F, com_hill_climb=True,
                                                           dim=dim, num_particulas=num_particulas, max_iter=100, opcao=1)
    else: cdeepso_gen = meu_deepso(funcao, w_i, w_m, w_s, t_mut, t_com, com_hill_climb=True, dim=dim, num_particulas=num_particulas, max_iter=100)

    for x_gb, x_gb_fitness, _ in cdeepso_gen:
        if x_gb_fitness < best_fitness:
            best_fitness = x_gb_fitness

    return best_fitness
```

Figura 4.6: Definição da função objetivo do Optuna

Segue um exemplo da chamada da função objetivo do Optuna para o C-DEEPSO com a função Rastrigin de 10 dimensões. Para cada conjunto de algoritmo, função e número de dimensões foram gerados 50 trials. Em cada trial eram gerados hiperparâmetros subsequentemente melhores, até o último nos dar os hiperparâmetros que utilizaríamos nos algoritmos.

```python
# Run Optuna to find the best hyperparameters
study = optuna.create_study(direction='minimize')
study.optimize(lambda trial: objective(trial, meu_cdeepso, rastrigin_n, dim=10), n_trials=50)

print("Best hyperparameters: ", ajeita_params(study.best_params))
print("Best score: ", study.best_value)
```

Figura 4.7: Chamada da função objetivo do Optuna

5. Resultados

Para analisar o desempenho dos algoritmos DEEPSO e C-DEEPSO, os dois foram executados 30 vezes para cada conjunto de função e número de dimensões.

5.1. Resultados Gerais

A tabela a seguir demonstra os resultados médios entre as 30 execuções de média, mediana, e desvio padrão (DP) de cada conjunto de algoritmo, função e número de dimensões. Além disso, foi executado o teste t de Student para determinar se os conjuntos dos melhores indivíduos das últimas gerações do DEEPSO e do C-DEEPSO são significativamente diferentes.

Tabela 5.1: Tabela de Resultados

Função/Dimensões	Algoritmo	Média	Mediana	DP	p-value	Hipótese Nula
Rosenbrock 10D	DEEPSO	20.01532	7.31312	24.96455	0.034485	Rejeitada
	C-DEEPSO	242.65338	156.38998	209.15622		
Rosenbrock 30D	DEEPSO	34.41522	32.24281	10.46327	0.17942	Aceita
	C-DEEPSO	732.74280	675.25349	333.72296		
Rosenbrock 50D	DEEPSO	55.53060	55.43040	3.89658	0.19968	Aceita
	C-DEEPSO	9126.07728	8300.55878	3145.23736		
Rastrigin 10D	DEEPSO	11.52192	10.80394	4.20980	0.51743	Aceita
	C-DEEPSO	25.59473	25.35759	9.16370		
Rastrigin 30D	DEEPSO	164.94551	169.65125	54.60030	0.84223	Aceita
	C-DEEPSO	163.44266	160.21353	26.80722		
Rastrigin 50D	DEEPSO	249.25797	244.73574	34.60257	0.85458	Aceita
	C-DEEPSO	345.68101	344.29313	28.74530		

Podemos então perceber que, em geral, o algoritmo DEEPSO encontra soluções melhores que o algoritmo C-DEEPSO. O único caso em que isso não é verdade é na função de Rastrigin de 30 dimensões, onde o C-DEEPSO supera ligeiramente o DEEPSO.

Além disso, vemos que a hipótese nula só é rejeitada para o Rosenbrock de 10 dimensões. Isso significa que, para o Rosenbrock de 10 dimensões, os resultados da análise estatística fornecem evidências suficientes para afirmar que há uma diferença significativa entre os conjuntos dos melhores indíviduos das 30 execuções do DEEPSO e do C-DEEPSO.

6. Conclusão

Este estudo comparou os desempenhos dos algoritmos DEEPSO e C-DEEPSO, nas funções de teste de Rosenbrock e Rastrigin, em dimensões variando de 10 a 50. Os resultados indicam que, **com a implementação específica utilizada**, o DEEPSO demonstrou consistentemente melhores resultados em termos de média, mediana e desvio padrão em comparação com o C-DEEPSO, em quase todos os cenários avaliados. Especificamente, o DEEPSO mostrou superioridade significativa na função Rosenbrock em todas as dimensões estudadas e na função Rastrigin em 10 e 50 dimensões.

Uma exceção notável foi observada na função Rastrigin em 30 dimensões, onde o C-DEEPSO apresentou desempenho ligeiramente superior em comparação com o DEEPSO, conforme evidenciado pela média, medaina e desvio padrão das últimas gerações das 30 execuções. Este resultado sugere que a escolha do algoritmo pode depender da função específica e do número de dimensões envolvidas.

Em resumo, os resultados deste estudo reforçam a eficácia do DEEPSO como uma escolha preferencial para otimização em problemas complexos de múltiplas dimensões, embora com a ressalva de que o desempenho ótimo pode variar dependendo das características específicas da função objetivo e da implementação do algoritmo.

Muito obrigado!