UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Trabalho Prático 1
Meta-Heurística

Artur Papa - 3886

Trabalho prático individual apresentado à disciplina de CCF 480 - Meta-heuristíca do curso de Ciência da Computação da Universidade Federal de Viçosa.

Florestal
Maio de 2023

CCF 480 - Meta-heuristíca

Trabalho Prático 0

Artur Papa - 3886

17 de Maio de 2023

Contents

1 Introdução

A meta-heurística é um campo da ciência da computação e da otimização que visa resolver problemas complexos e de difícil resolução através do desenvolvimento de algoritmos e estratégias genéricas. Diferentemente dos métodos exatos, que buscam encontrar a solução ótima para um problema, as meta-heurísticas são técnicas aproximadas que permitem encontrar soluções de boa qualidade em um tempo razoável, mesmo para problemas de grande escala.

O objetivo das meta-heurísticas é explorar eficientemente o espaço de soluções de um problema, utilizando mecanismos de busca, otimização e aprendizado. Elas são baseadas em princípios inspirados na natureza, no comportamento de organismos vivos, em algoritmos evolutivos, na física, entre outros.

Dessa maneira, foi proposto para este trabalho o uso da meta-heurística ***Iterated Local Search (ILS)*** para resolver duas funções objetivo, além dessa, foi escolhido pelo aluno o ***Hill Climbing*** como segunda opção para fazer comparação com os resultados para o primeiro algoritmo.

Além disso, foi feito um *benchmark* com 30 iterações para averiguar os resultados de cada meta-heurística e, por fim, foi plotado um boxplot para cada métrica para a análise dos resultados

2 Desenvolvimento

A meta-heurística ***ILS (Iterated Local Search)*** é uma abordagem popular e eficiente para resolver problemas de otimização. Ela pertence à classe de algoritmos de busca local e é conhecida por combinar de forma inteligente a exploração local e a diversificação em busca de soluções de alta qualidade.

O ILS começa com uma solução inicial e, em seguida, itera entre duas fases principais: busca local e perturbação. Na fase de busca local, a solução corrente é explorada em seu entorno para encontrar uma solução vizinha melhor. Isso envolve fazer pequenas alterações na solução e avaliar se essas alterações levam a uma melhoria. A busca local visa melhorar gradualmente a solução inicial.

Após um número definido de iterações de busca local, entra em ação a fase de perturbação. Nessa fase, a solução corrente é modificada de forma mais radical, introduzindo perturbações que alteram a estrutura da solução. Essa perturbação tem o propósito de escapar de mínimos locais e explorar regiões diferentes do espaço de soluções.

O processo de busca local e perturbação é repetido por um número pré-definido de iterações ou até que um critério de parada seja atingido. Ao final, o ILS retorna a melhor solução encontrada durante as iterações.

3 Código

Outrossim, vale ressaltar que foi usada a linguagem **python** juntamente com a ferramenta **jupyter notebooks** para fazer os algoritmos e plotar os gráficos posteriormente.

Assim, para este trabalho, foi pedido para resolver as seguintes funções:

$$f(x, y) = x^2 + y^2 + 25(sin^2(x) + sin^2(y))$$

Figura 1: Função 1

$$f(x,y) = -(y+47)sin\left(\sqrt{|y+0.5y+47|}\right) - xsin\left(\sqrt{|x-(y+47)|}\right)$$

Figura 2: Função 2

Sendo que para a primeira teríamos os intervalos de -5 $\leq$ x, y $\leq$ 5 para o primeiro caso e o segundo intervalo de -2 $\leq$ x, y $\leq$ 2 para o segundo caso. Já para a segunda função teríamos os intervalos de -512 $\leq$ x, y $\leq$ 512 e 400 $\leq$ x, y $\leq$ 512.

3.1 Iterated Local Search (ILS)

Para o ***Iterated Local Search (ILS)*** foi feito o mesmo algoritmo conforme mostrado nos slides e, para a função de busca local foi usado o mecanismo de *swap* para calcular a nova solução local.

```python
def local_search(solution, num_func):
    best_solution = solution.copy()  # Crie uma cópia da solução atual como a melhor solução
    best_cost = calculate_cost(solution, num_func)  # Calcule o custo da melhor solução

    improved = True
    while improved:
        improved = False

        for i in range(len(solution)):
            for j in range(i + 1, len(solution)):
                # Troque os elementos i e j na solução
                new_solution = solution.copy()
                new_solution[i], new_solution[j] = new_solution[j], new_solution[i]

                # Calcule o custo da nova solução
                new_cost = calculate_cost(new_solution, num_func)

                # Se a nova solução for melhor, atualize a melhor solução
                if new_cost < best_cost:
                    best_solution = new_solution
                    best_cost = new_cost
                    improved = True

        # Atualize a solução atual para a melhor solução encontrada nesta iteração
        solution = best_solution

    return [best_cost, best_solution]
```

Figura 3: Função de busca local

```python
def ILS(num_iter, lb, ub, num_func):
    initial_best = initial_solution(lb, ub)
    best_cost, best_solution = local_search(initial_best, num_func)
    i = 0
    while (i < num_iter):
        perturbed = perturbation(best_solution, lb, ub)
        local_cost, local_solution = local_search(perturbed, num_func)
        if local_cost < best_cost:
            best_cost, best_solution = local_cost, local_solution
        i+=1
    return best_cost
```

Figura 4: Algoritmo ILS

3.2 Hill Climbing

Já para o ***Hill Climbing***, além de ter feito o algoritmo padrão foi usado a variável *step_size* para que o algoritmo não fique preso em um mínimo local e, assim como no ILS, a meta-heurística usa o critério de aceitação ao final de cada iteração.

```python
def hillclimbing(objf, bounds, num_iter, step_size):
    solution = bounds[:, 0] + np.random.rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
    solution_eval = objf(solution[0], solution[1])

    for i in range(num_iter):
        candidate = solution + np.random.rand(len(bounds)) * step_size
        candidate_eval = objf(candidate[0], candidate[1])

        if candidate_eval <= solution_eval:
            solution, solution_eval = candidate, candidate_eval
        #print(">%d f(%s) = %.5f" %(i, solution, solution_eval))

    return solution_eval
```

Figura 5: Algoritmo Hill Climbing

3.3 Resultados

Para o benchmark, foi pedido para rodar cada algoritmo 30 vezes e salvar os resultados em uma tabela, calculando métricas como, média, mínimo, máximo e desvio padrão. Vale ressaltar que cada algoritmo possui um número máximo de 100 iterações, dessa forma, a cada uma iteração do benchmark, o algoritmo irá rodar 100 vezes e retornar o melhor resultado. Posto isso, após termos executado todas as 30 iterações, os resultados geram um arquivo **CSV** com todas as métricas.

Assim, seguem as tabelas com os resultados para cada função e intervalo requisitados no trabalho.

	Algoritmo	Mínimo	Máximo	Média	Desvio-padrão
0	ILS	0.974163	11.984885	7.006049	3.470723
1	Hill Climbing	5.503417	71.431203	27.820400	15.324648

Figura 6: Resultados primeira função primeiro intervalo

	Algoritmo	Mínimo	Máximo	Média	Desvio-padrão
0	ILS	0.175303	19.140065	6.080278	5.299212
1	Hill Climbing	0.020258	34.590015	13.061027	11.111981

Figura 7: Resultados primeira função segundo intervalo

	Algoritmo	Mínimo	Máximo	Média	Desvio-padrão
0	ILS	-5355.502300	-2312.109639	-3875.322265	751.724998
1	Hill Climbing	-759.719439	621.453789	-18.616268	310.908905

Figura 8: Resultados segunda função primeiro intervalo

	Algoritmo	Mínimo	Máximo	Média	Desvio-padrão
0	ILS	-6430.695249	-4358.288408	-5278.360207	617.200073
1	Hill Climbing	-981.905080	431.781303	-334.308930	430.011009

Figura 9: Resultados segunda função segundo intervalo

3.4 Box plot

Após terem sido calculadas as métricas, foram feitos *box plots* para validar os resultados de cada algoritmo, é importante dizer que foi gerado um *box plot* para cada métrica de cada função, comparando as duas meta-heurísticas, ou seja, como temos quatro métricas e quatro intervalos, foram *plotados* dezesseis gráficos.

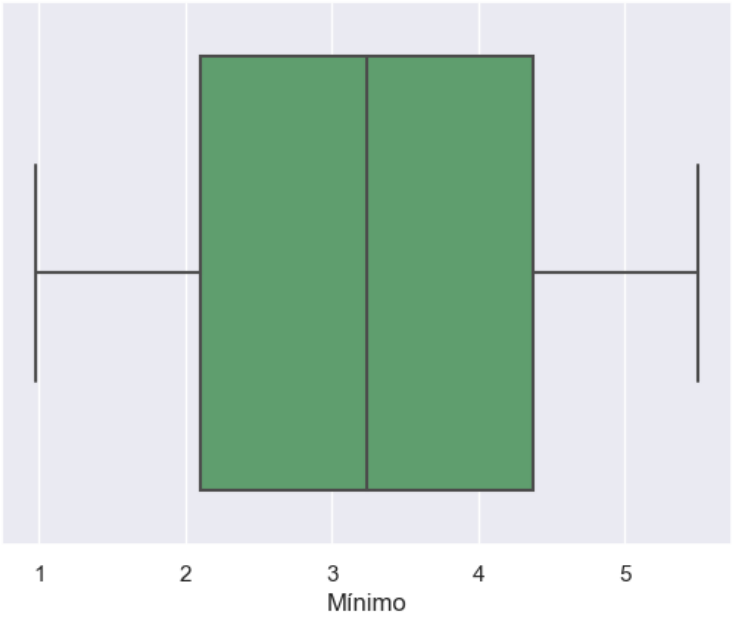

Figura 10: Mínimo da primeira função e primeiro intervalo

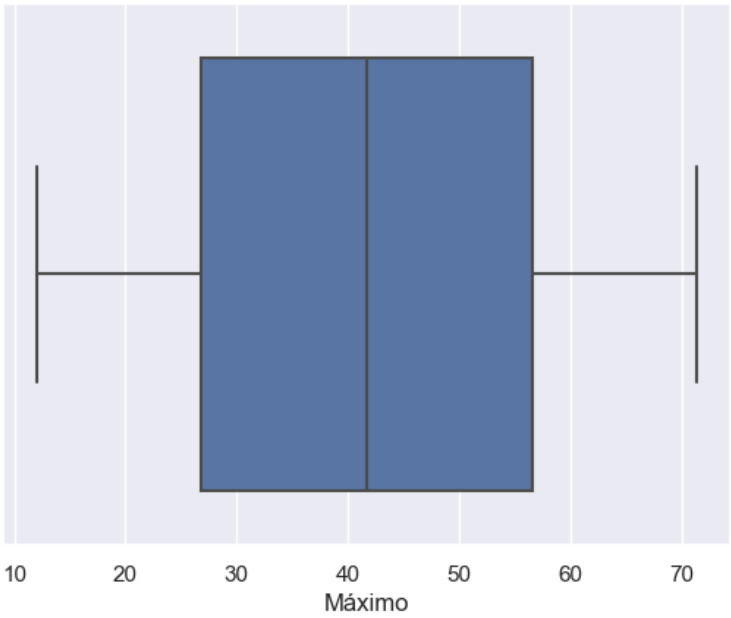

Figura 11: Máximo da primeira função e primeiro intervalo

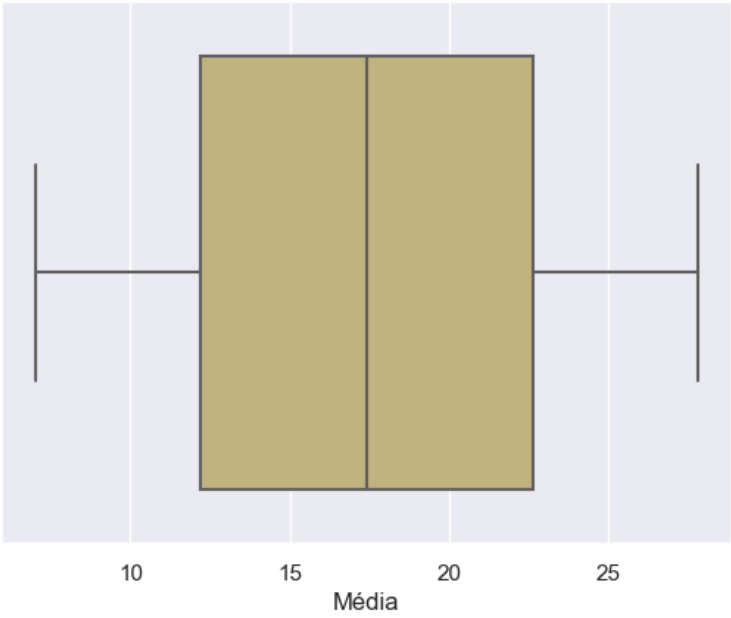

Figura 12: Média da primeira função e primeiro intervalo

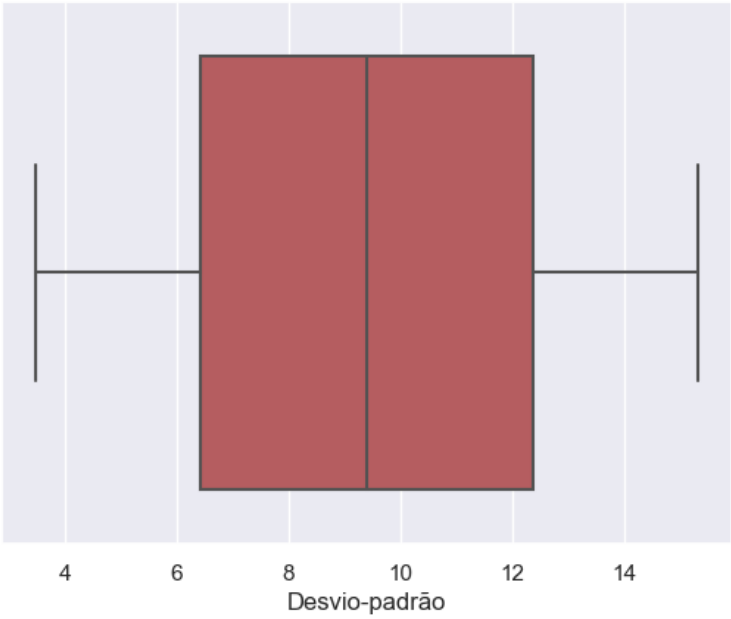

Figura 13: Desvio padrão da primeira função e primeiro intervalo

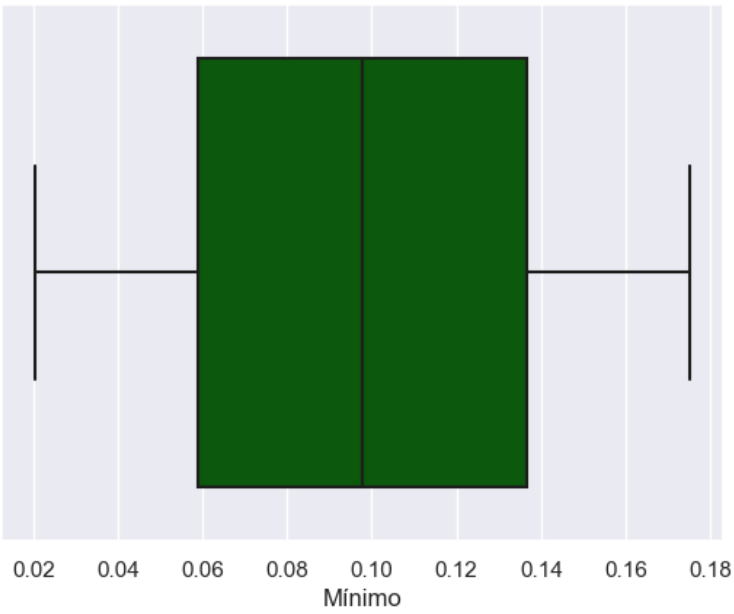

Figura 14: Mínimo da primeira função e segundo intervalo

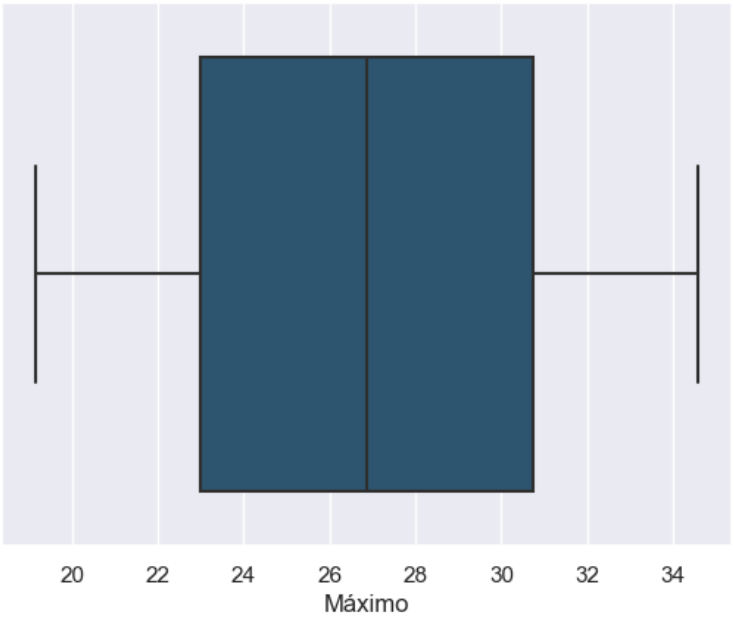

Figura 15: Máximo da primeira função e segundo intervalo

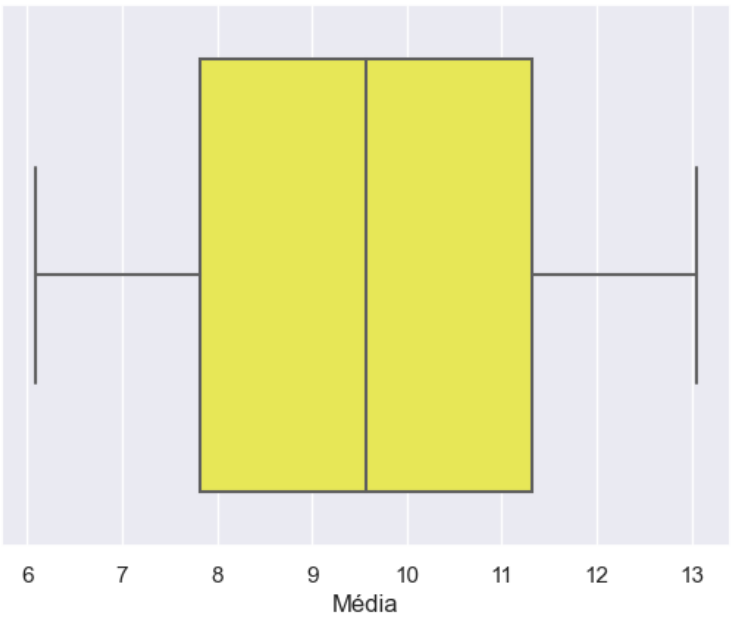

Figura 16: Média da primeira função e segundo intervalo

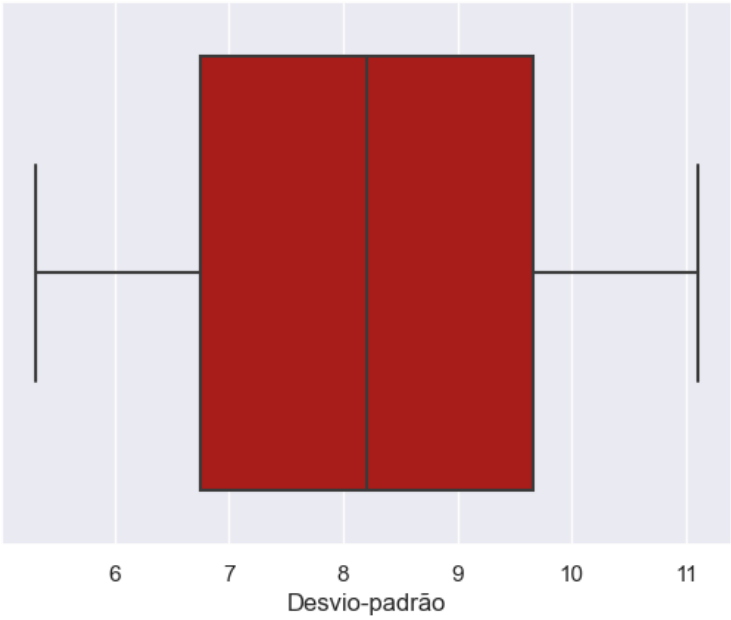

Figura 17: Desvio padrão da primeira função e segundo intervalo

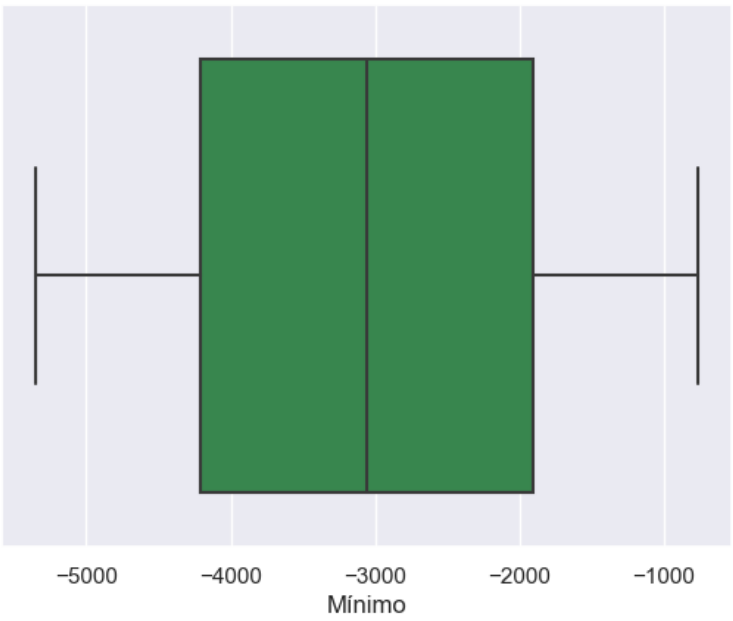

Figura 18: Mínimo da segunda função e primeiro intervalo

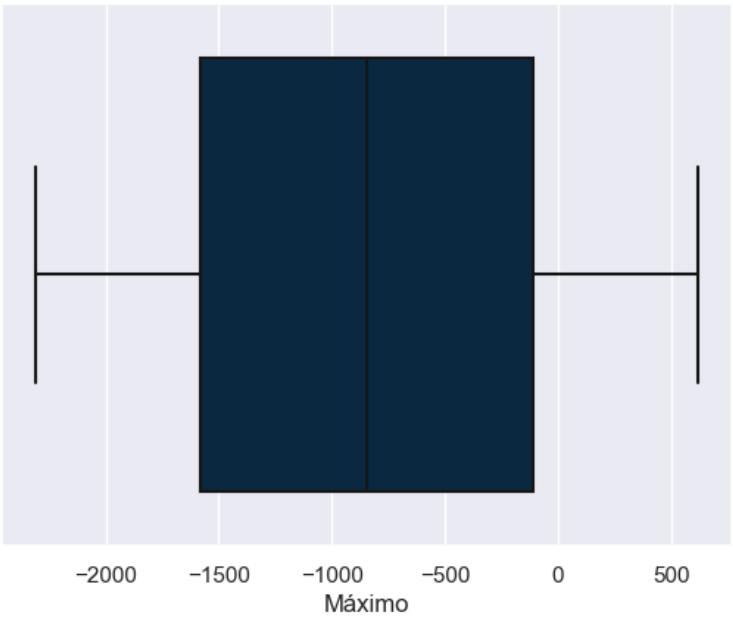

Figura 19: Máximo da segunda função e primeiro intervalo

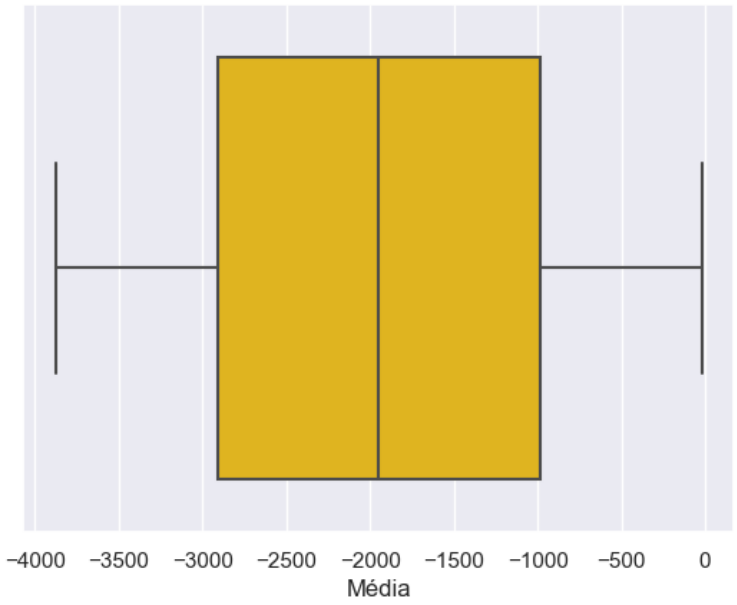

Figura 20: Média da segunda função e primeiro intervalo

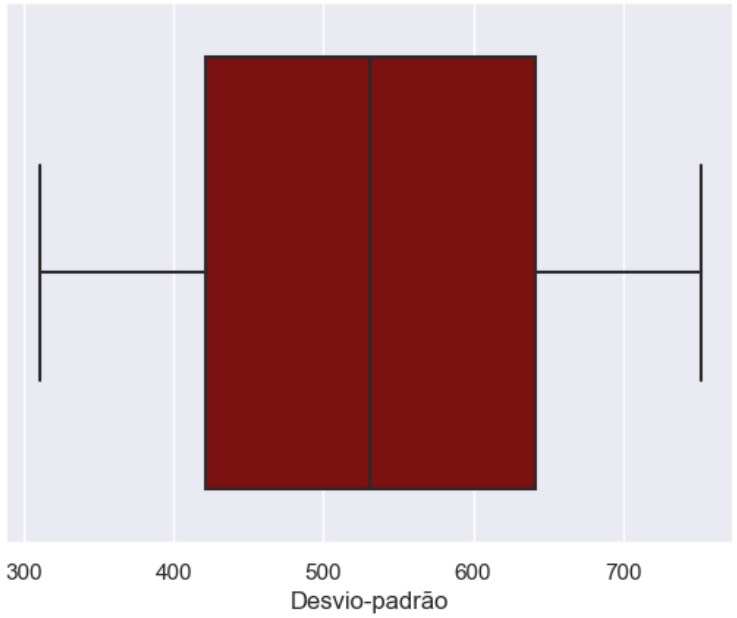

Figura 21: Desvio padrão da segunda função e primeiro intervalo

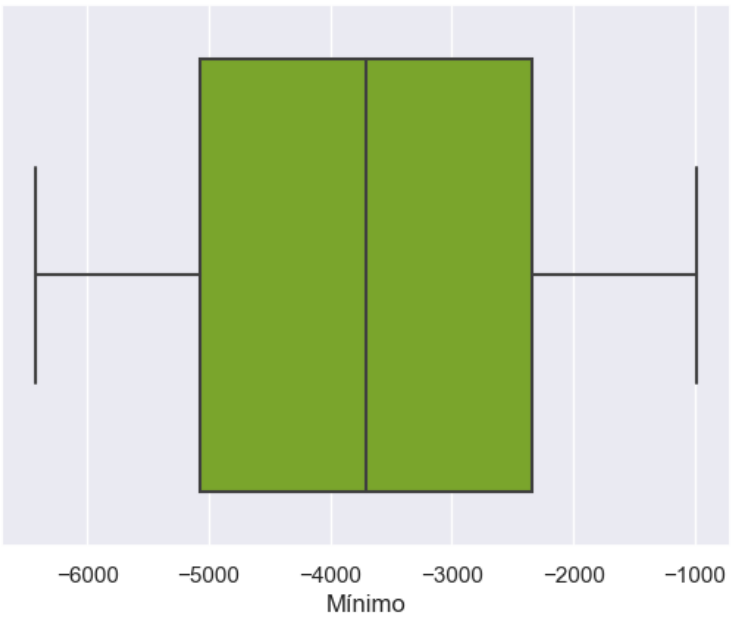

Figura 22: Mínimo da segunda função e segundo intervalo

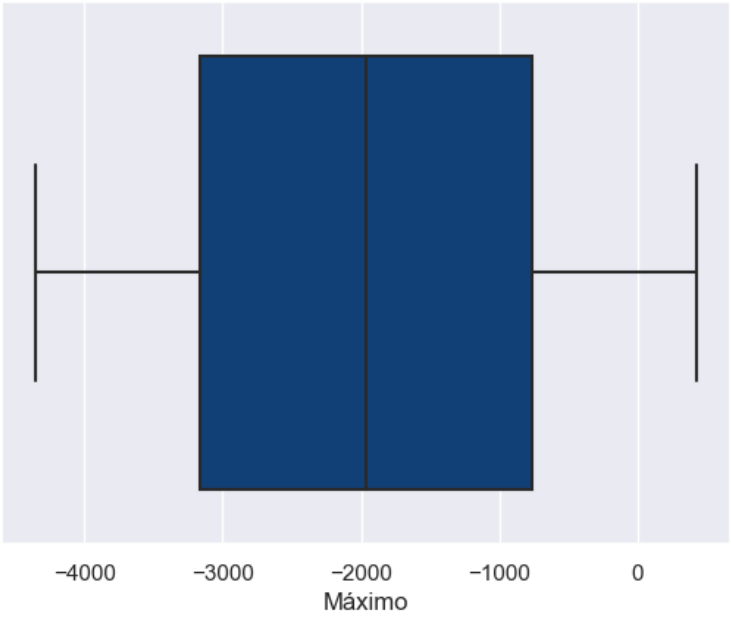

Figura 23: Máximo da segunda função e segundo intervalo

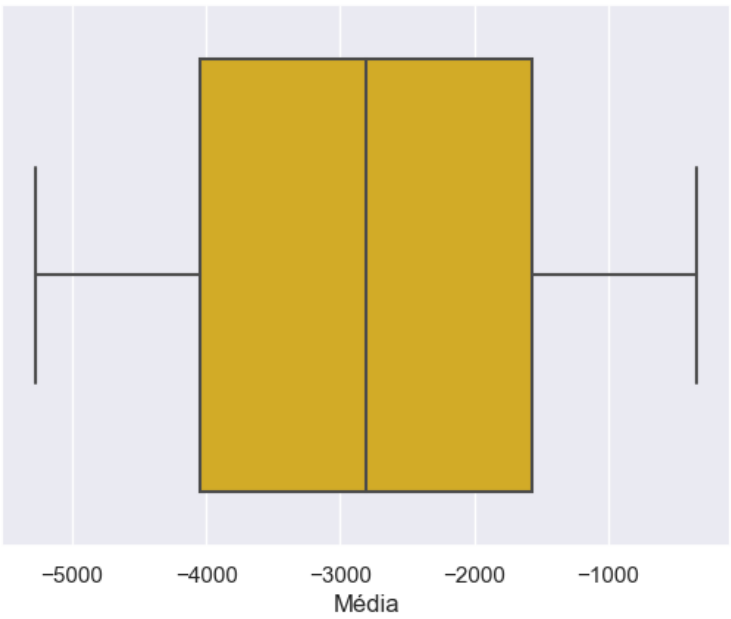

Figura 24: Média da segunda função e segundo intervalo

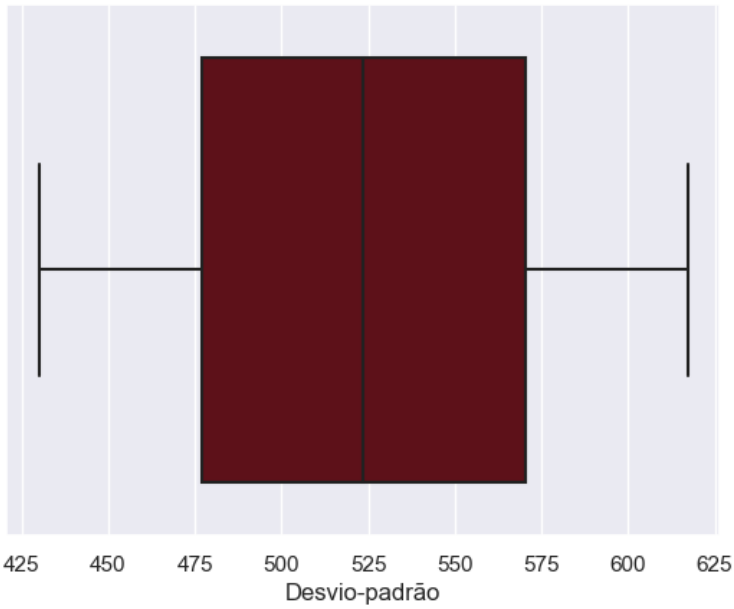

Figura 25: Desvio padrão da segunda função e segundo intervalo

4 Conclusão

Posto isso, neste trabalho prático foi explorado duas meta-heurísticas amplamente utilizadas na área de otimização: Hill Climbing e ILS (Iterated Local Search).

O Hill Climbing é um algoritmo de busca local que busca melhorar iterativamente uma solução, fazendo pequenas alterações e escolhendo aquela que resulta em uma melhoria. Embora seja simples e fácil de implementar, o Hill Climbing pode ficar preso em ótimos locais e não explorar todo o espaço de soluções, o que limita sua capacidade de encontrar a solução global ótima.

Por outro lado, o ILS é uma meta-heurística que combina a busca local com a perturbação aleatória. Ele realiza iterações de busca local em soluções perturbadas, permitindo escapar de ótimos locais e explorar diferentes partes do espaço de soluções. Isso aumenta a probabilidade de encontrar soluções melhores e mais diversificadas. O ILS mostrou-se eficaz em muitos problemas de otimização e tem sido amplamente aplicado em diversas áreas.

Ao analisar os resultados obtidos com essas duas meta-heurísticas, observamos que o Hill Climbing é mais rápido na convergência e pode encontrar soluções de boa qualidade para problemas relativamente simples. No entanto, ele é menos eficaz em problemas mais complexos, onde a presença de ótimos locais pode afetar seu desempenho.

Por outro lado, o ILS demonstrou uma capacidade maior de explorar e encontrar soluções de alta qualidade, mesmo em problemas complexos com espaços de solução extensos. Sua combinação de busca local e perturbação aleatória permite escapar de ótimos locais e explorar diferentes partes do espaço de soluções, levando a melhores resultados.

Em resumo, tanto o Hill Climbing quanto o ILS são técnicas valiosas no campo da otimização, cada uma com suas próprias vantagens e limitações. Desse modo, foi visto que o ILS se saiu melhor que o Hill Climbing em todos os casos, o que demonstra a eficácia e a eficiência dessa meta-heurística para problemas contínuos e complexos.