UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Trabalho Prático 2
Meta-Heurística

Artur Papa - 3886 Luciano Belo - 3897
Vinícius de Oliveira Mendes - 3881

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

Florestal
Julho de 2023

CCF 480 - Meta-heuristíca

Trabalho Prático 2

Artur Papa - 3886

08 de Julho de 2023

Contents

1 Introdução

A programação genética é uma técnica de otimização que se baseia em algoritmos genéticos para resolver problemas de programação. Ela foi inspirada no processo de evolução biológica e utiliza uma abordagem de busca heurística para encontrar soluções aproximadas para problemas complexos.

A programação genética utiliza uma representação simbólica de soluções, em vez de uma representação numérica tradicional. Nessa abordagem, os indivíduos são representados por árvores de expressão, onde os nós internos representam operadores e os nós folha representam constantes ou variáveis. Essas árvores são evoluídas ao longo das gerações, combinando e modificando os indivíduos existentes por meio de operadores genéticos, como reprodução, cruzamento e mutação.

O processo de evolução ocorre em várias gerações, onde cada geração consiste em uma população de indivíduos. Os indivíduos são avaliados de acordo com uma função de aptidão que mede sua qualidade em relação ao problema em questão. Com base na aptidão, os indivíduos mais aptos são selecionados para reprodução, criando uma nova geração de indivíduos. Esse ciclo de seleção, reprodução e evolução continua até que uma condição de parada seja atingida, como um número máximo de gerações ou uma aptidão desejada.

A programação genética é amplamente aplicada em problemas de otimização, engenharia, aprendizado de máquina e outros campos onde é necessário encontrar soluções aproximadas para problemas complexos. Ela oferece flexibilidade na representação das soluções e é capaz de explorar espaços de busca amplos de forma eficiente.

Dessa maneira, foi usada programação genética para resolver dois problemas não lineares de matempatica com restrições especificadas previamente.

2 Problema proposto

Primeiramente, é válido ressaltar que foram apresentados dois problemas não lineares de minimização para serem resolvidos, o primeiro envolvia 13 variáveis de decisão e 9 restrições de desigualdade.

Minimize:

$$f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i - \sum_{i=5}^{13} x_i$$

sujeito a:

$$g_1(\vec{x}) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0$$

$$g_2(\vec{x}) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0$$

$$g_3(\vec{x}) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0$$

$$g_4(\vec{x}) = -8x_1 + x_{10} \leq 0$$

$$g_5(\vec{x}) = -8x_2 + x_{11} \leq 0$$

$$g_6(\vec{x}) = -8x_3 + x_{12} \leq 0$$

$$g_7(\vec{x}) = -2x_4 - x_5 + x_{10} \leq 0$$

$$g_8(\vec{x}) = -2x_6 - x_7 + x_{11} \leq 0$$

$$g_9(\vec{x}) = -2x_8 - x_9 + x_{12} \leq 0$$

com

$$0 \leq x \leq 1(i = 1, ..., 9), 0 \leq x_i \leq 100(i = 10, 11, 12) and 0 \leq x_{13} \leq 1$$

Já o segundo problema havia 2 variáveis de decisão e 5 restrições, sendo 2 de desigualdade e 3 de igualdade.

$Minimize$:

$$f(\vec{x}) = 3x_1 + 0.000001x_1 + 2x_2 + (0.000002/3)x_2$$

sujeito a:

$$g_1(\vec{x}) = -x_4 + x_3 - 0.55 \leq 0$$

$$g_2(\vec{x}) = -x_3 + x_4 - 0.55 \leq 0$$

$$h_3(\vec{x}) = 1000sin(-x_3 - 0.25) + 1000sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$

$$h_4(\vec{x}) = 1000sin(x_3 - 0.25) + 1000sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$

$$h_5(\vec{x}) = 1000sin(x_4 - 0.25) + 1000sin(x_4 - x_3 - 0.25) + 1294.8 = 0$$

com

$$0 \leq x_1 \leq 1200, 0 \leq x_2 \leq 1200, -0.55 \leq x_3 \leq 0.55 and - 0.55 \leq x_4 \leq 0.55$$

3 Implementação do algoritmo para o primeiro problema

O algoritmo implementado é uma versão simplificada de um algoritmo genético utilizado para resolver problemas de programação não linear com restrições. A implementação segue a estrutura típica de um algoritmo genético, com etapas de inicialização da população, avaliação da aptidão, seleção, cruzamento, mutação e aplicação de restrições.

O algoritmo é representado pela classe **GeneticProgramming**, que recebe parâmetros como o tamanho da população (**pop_size**), o número de gerações (**num_generations**), o tamanho do torneio para seleção (**tournament_size**), a taxa de mutação (**mutation_rate**) e o método de penalidade a ser utilizado para tratar as restrições (**penalty_method**).

A função objetivo do problema é definida no método **objective_function**, que recebe um indivíduo (solução_candidata) e calcula o valor da função objetivo para esse indivíduo. A função de avaliação **evaluate** calcula o valor da aptidão de um indivíduo considerando o método de penalidade escolhido. Se for utilizado o método de penalidade estática, a função **calculate_static_penalty** é chamada para calcular a penalidade das restrições para um indivíduo. Se

for utilizado o método de restrição de epsilon, a função **calculate_epsilon_constraint_method** é chamada para verificar as violações das restrições.

A geração inicial da população é realizada pelo método **generate_population**, que gera indivíduos aleatórios. A seleção de indivíduos para reprodução é feita pelo método **selection**, que utiliza o método do torneio para escolher os indivíduos de melhor aptidão. O cruzamento entre indivíduos é realizado pelo método **crossover**, que seleciona um ponto de corte e combina os genes dos pais para gerar os filhos. A mutação ocorre no método **mutate**, onde cada gene do indivíduo tem uma chance de ser mutado. A aplicação das restrições é feita no método **enforce_constraints**, que ajusta os genes do indivíduo para garantir que as restrições sejam respeitadas.

O algoritmo é executado no método **run**, onde a população é avaliada, e a cada geração, são selecionados os melhores indivíduos para formar a nova população. A melhor solução encontrada ao longo das gerações é armazenada e retornada ao final do algoritmo.

A implementação apresentada é uma versão básica e pode ser personalizada e aprimorada de acordo com as necessidades do problema específico a ser resolvido. É importante considerar que, dependendo da complexidade do problema e das restrições envolvidas, pode ser necessário ajustar os parâmetros do algoritmo, como o tamanho da população, a taxa de mutação e o método de penalidade, para obter resultados melhores.

4 Resultados para o primeiro problema

Após terem sidos calculadas as métricas foi pedido para gerar os boxplots para análise dos resultados para ambos os problemas, além disso, o grupo optou por gerar outros gráficos para ter uma análise mais precisa.

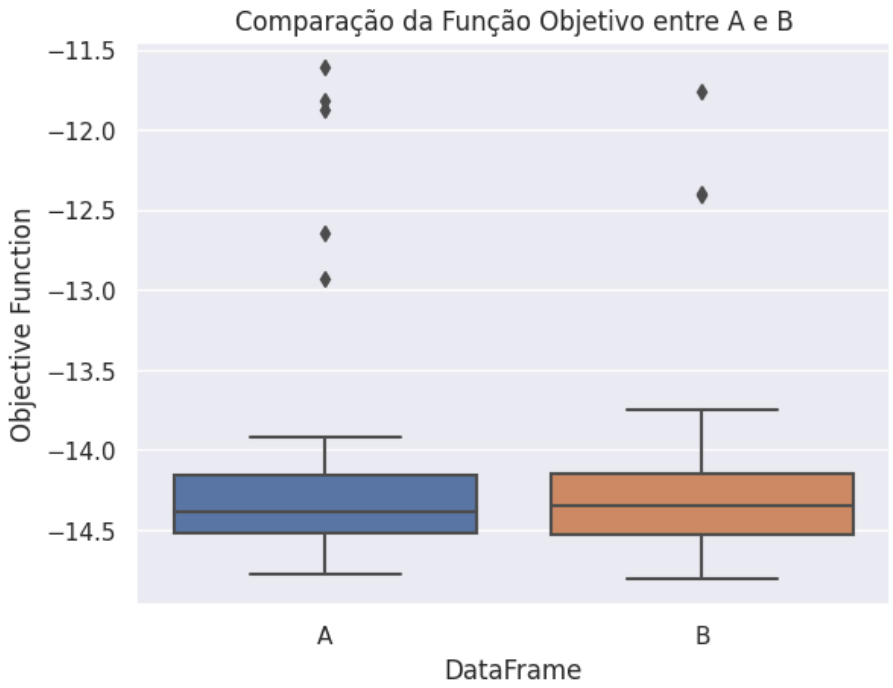

Figure 1: Comparação da Função objetivo entre A e B.

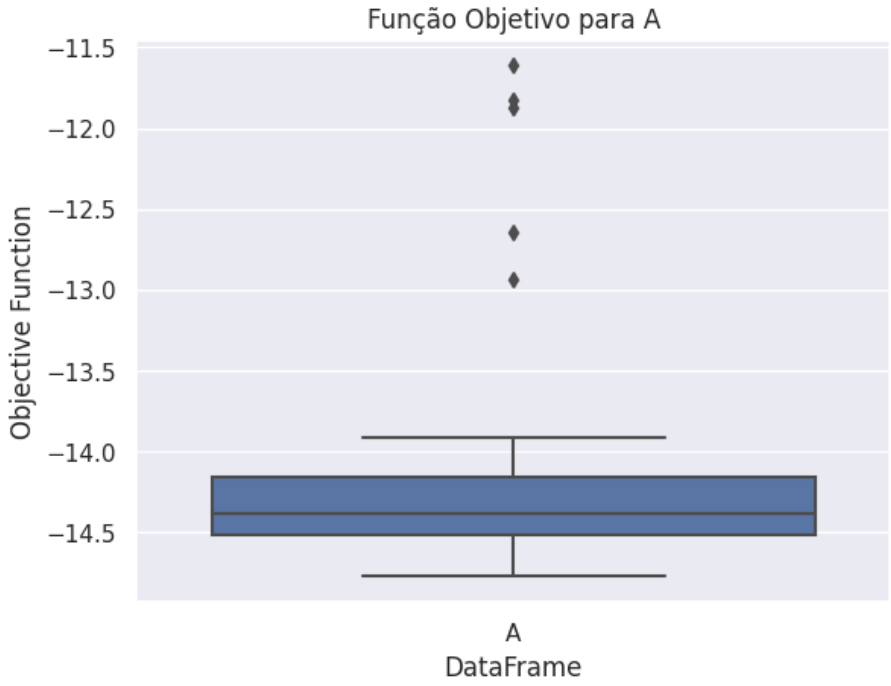

Figure 2: Função objetivo para A.

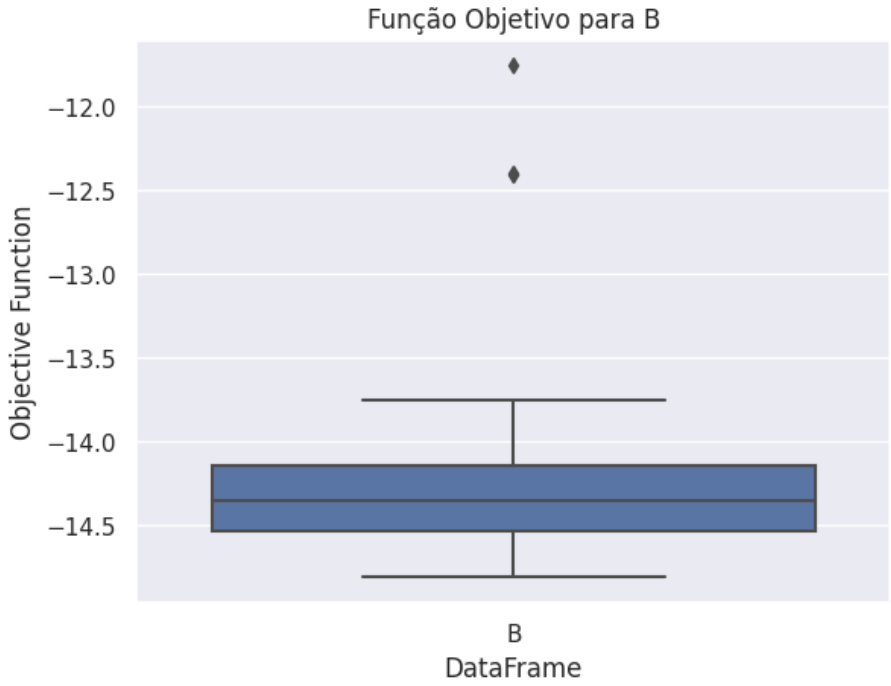

Figure 3: Função objetivo para B.

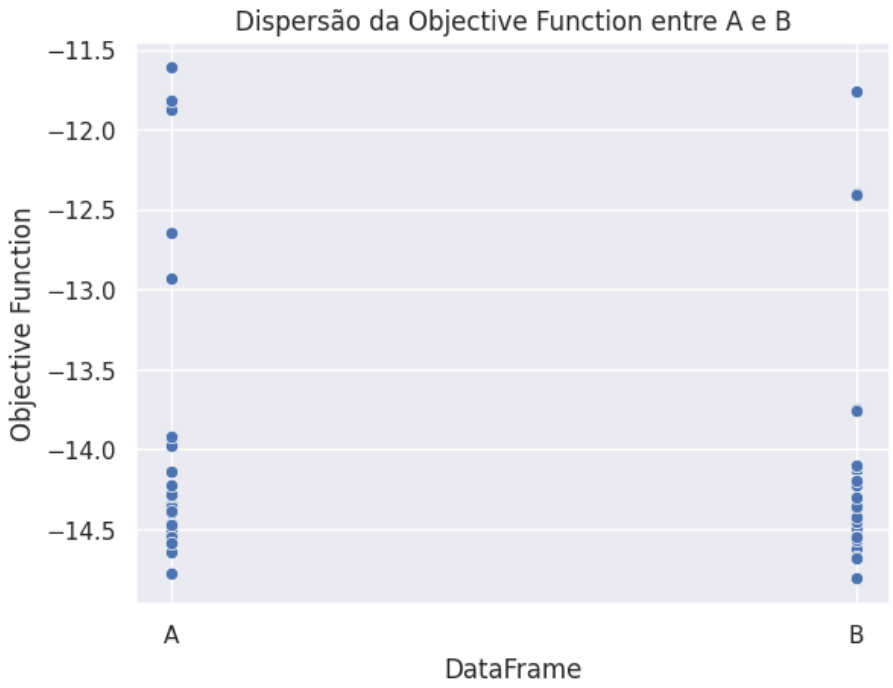

Figure 4: Dispersão da Objective Function entre A e B.

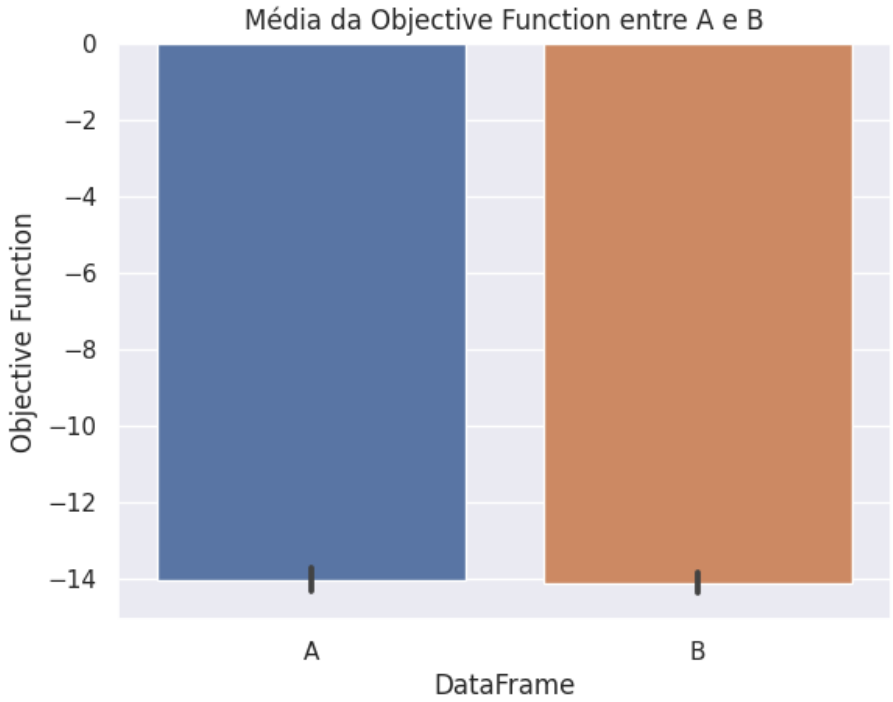

Figure 5: Média da Objective Function entre A e B.

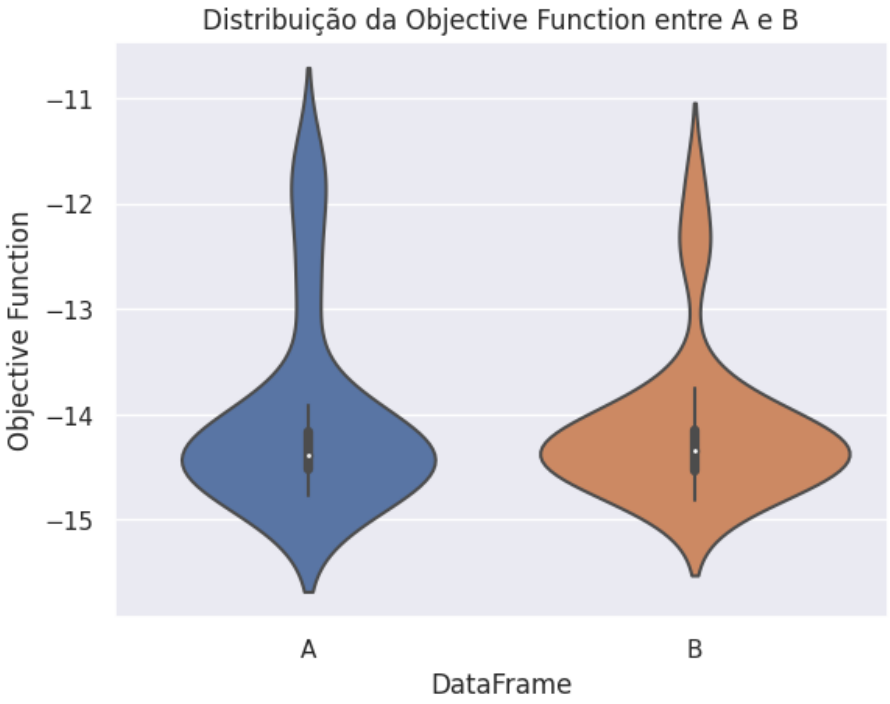

Figure 6: Distribuíção da Objective Function entre A e B.

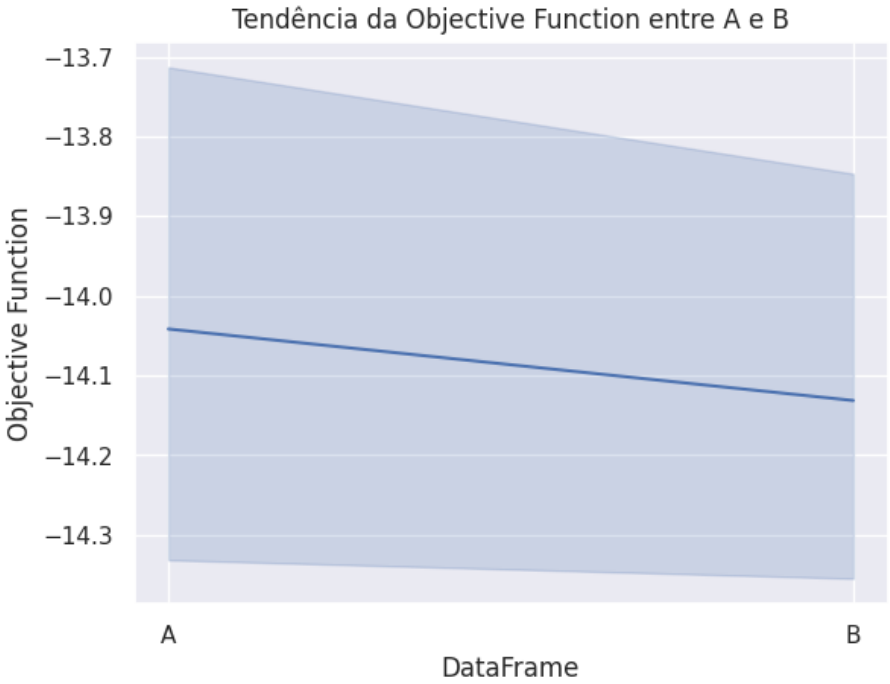

Figure 7: Tendência da Objective Function entre A e Bs.

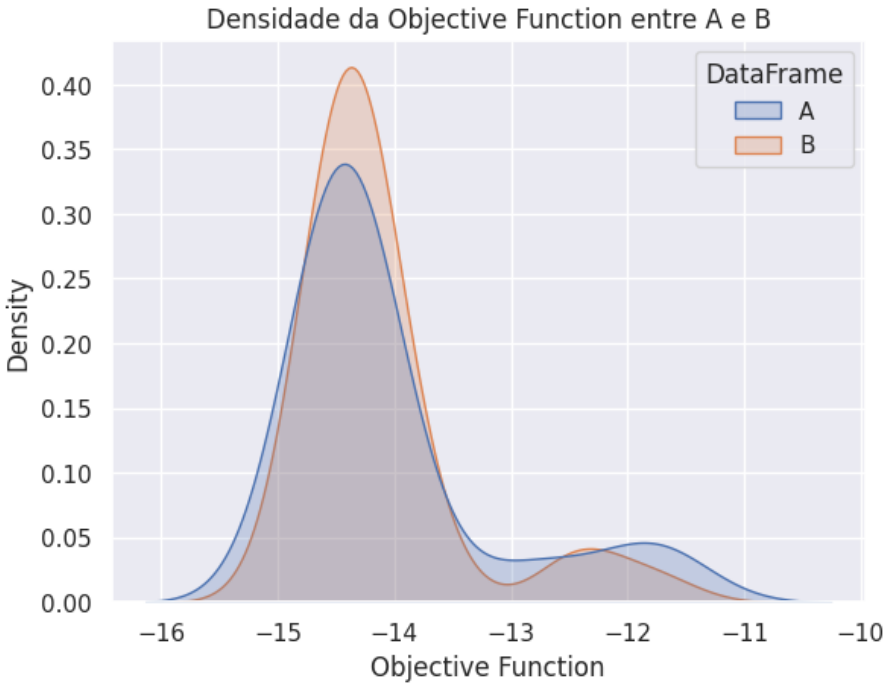

Figure 8: Densidade da Objective Function entre A e B.

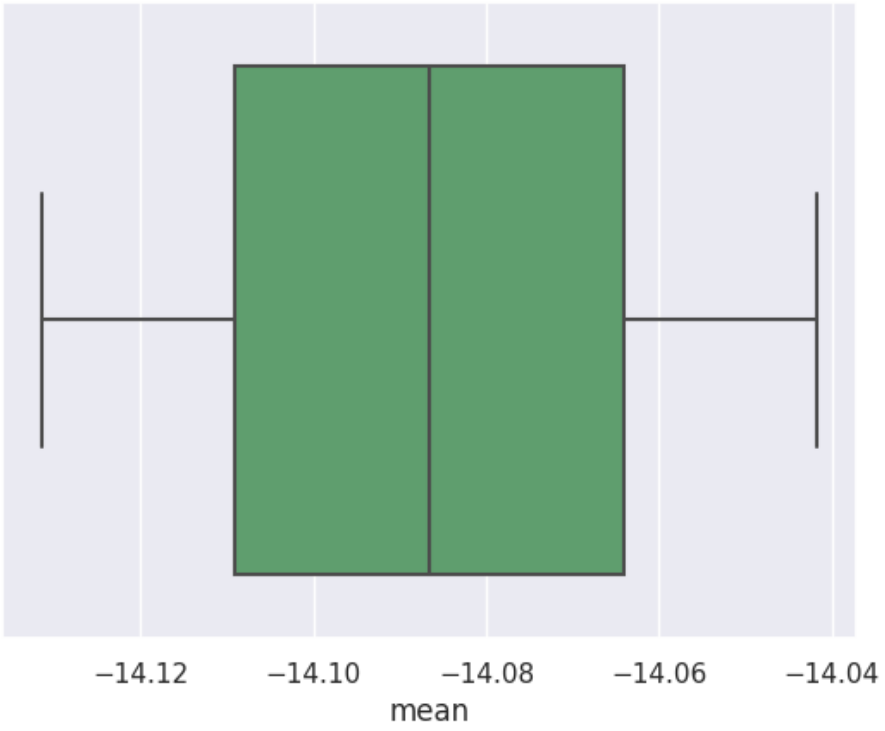

Figure 9: Box plot para a média entre as duas penalidades.

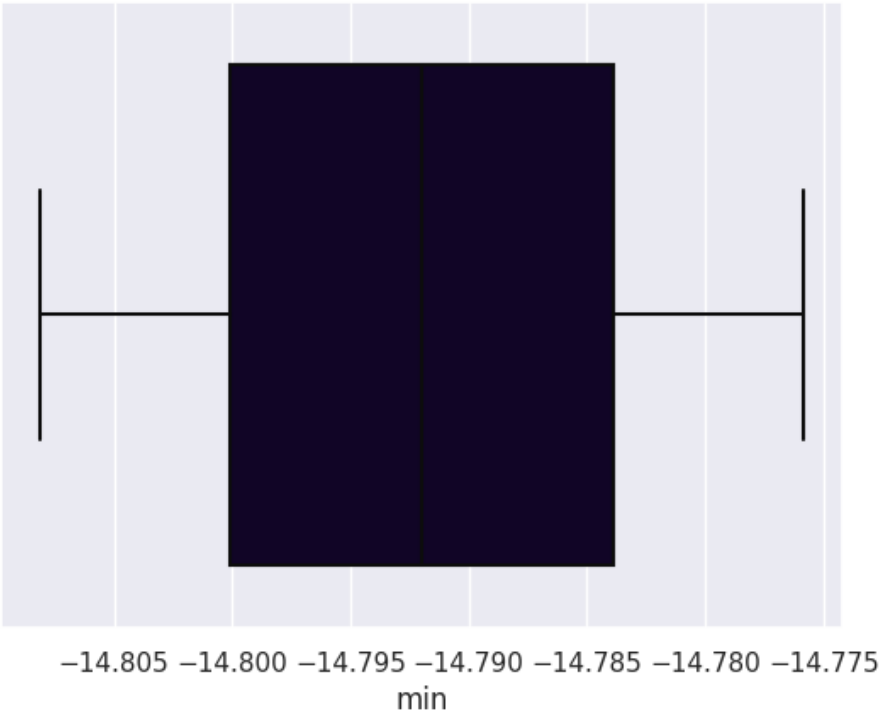

Figure 10: Box plot para o mínimo entre as duas penalidades.

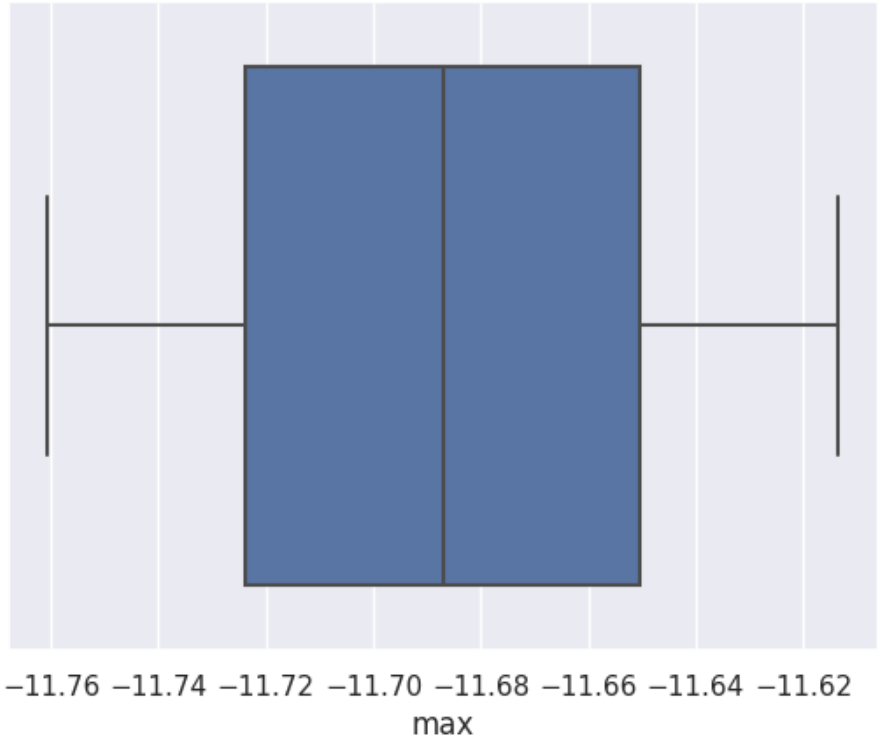

Figure 11: Box plot para o máximo entre as duas penalidades.

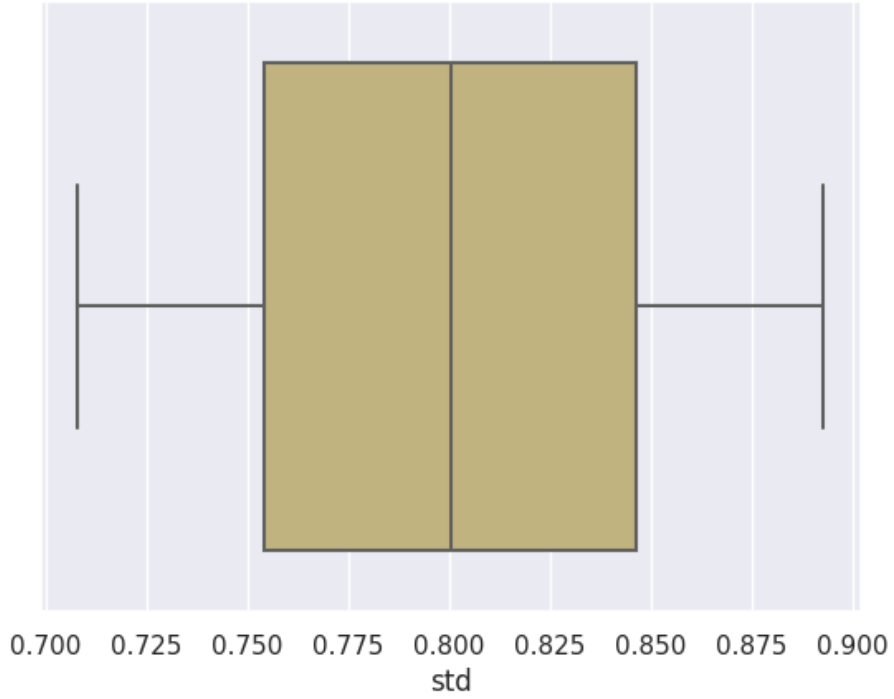

Figure 12: Box plot para o desvio-padrão entre as duas penalidades.

5 Implementação do algoritmo para o segundo problema

Para resolução proposta para esse problema, utilizamos as mesmas funções da atividade anterior realizando algumas modificações, entre elas:

- Mudança da função objetivo;
- Adaptação das restrições;
- Limitação do range das variáveis de decisão;

Ele possui inicialmente 2 variáveis de decisão (x1,x2) , entretanto, para delimitação das restrições, outras 2 são indicadas (x3, x4).

Optamos por manter os mesmos parâmetros do item anterior: população (**pop_size**), o número de gerações (**num_generations**), o tamanho do torneio para seleção (**tournament_size**), a taxa de mutação (**mutation_rate**) e o método de penalidade a ser utilizado para tratar as restrições (**penalty_method**); como forma de preestabelecer métricas de desempenho.

Um ponto que deve ser destacado, é que as mudanças relacionadas à adaptação das restrições impacta também na geração de penalidade para este caso, visto que optamos pode desenvolver uma forma de penalizar indivíduos de maneira diferente relacionado-a com as restrições impostas pelo problema. Dessa mesma maneira, as adaptações realizadas pela ,imitação do range das variáveis de decisão impactam nas funções de gerações de indivíduos e mutação, devido a adaptação das devidas inequações/equações.

Además, buscamos solucionar a segunda questão através do tipo de penalidade sorteado para nosso grupo (Epsilon Constraint), entretanto, nos deparamos com uma inconsistência de resultados e dessa maneira, identificamos alguns motivos para que a penalidade resultasse dessa maneira:

1. Singularidade e dominância fraca: Se houver singularidades nas regiões ótimas do problema ou se houver uma dominância fraca entre as soluções, o método Epsilon Constraint pode

não encontrar todas as soluções ótimas. Isso ocorre porque o método se concentra em uma única solução ótima para cada valor de épsilon, limitando a exploração de todo o espaço de solução.

2. Dificuldade em determinar o valor de épsilon: A escolha adequada do valor de épsilon pode ser desafiadora. O épsilon controla o compromisso entre as funções objetivas e as restrições. Um épsilon muito pequeno pode levar a soluções muito restritas, enquanto um épsilon muito grande pode resultar em soluções não factíveis. Encontrar o valor ideal de épsilon requer um bom entendimento do problema e pode exigir experimentação.

3. Dependência das restrições: O método Epsilon Constraint assume que as restrições do problema podem ser transformadas em objetivos usando variáveis auxiliares. No entanto, nem sempre é possível fazer essa transformação de forma direta e eficiente. Em alguns casos, as restrições podem ser complexas e não facilmente traduzidas em objetivos equivalentes.

4. Limitação de problemas multiobjetivo: O método Epsilon Constraint é mais adequado para problemas com um número moderado de objetivos. À medida que o número de objetivos aumenta, o método se torna menos eficiente e requer uma quantidade considerável de cálculos para avaliar todas as soluções possíveis.

5. Balanceamento entre objetivos: O método Epsilon Constraint não aborda diretamente o problema do balanceamento entre objetivos. Em muitos casos, existem trade-offs entre diferentes objetivos e encontrar um conjunto equilibrado de soluções ótimas pode ser desafiador. O método pode gerar soluções que se concentram em um único objetivo, sem considerar adequadamente os outros objetivos.

Posto isso, utilizamos a penalidade de Lagrange para que fosse possível observar as devidas soluções.

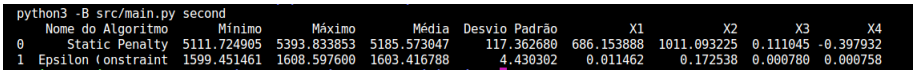

Figure 13: Diferença de resultados com o epsilon-constraint.

6 Resultados para o segundo problema

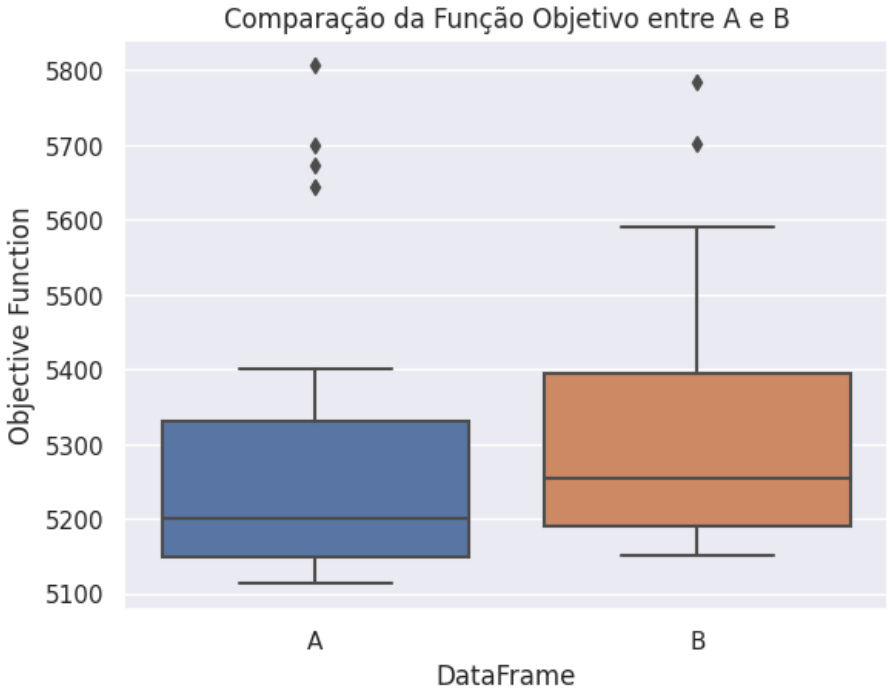

Figure 14: Comparação da Função Objetivo entre A e B.

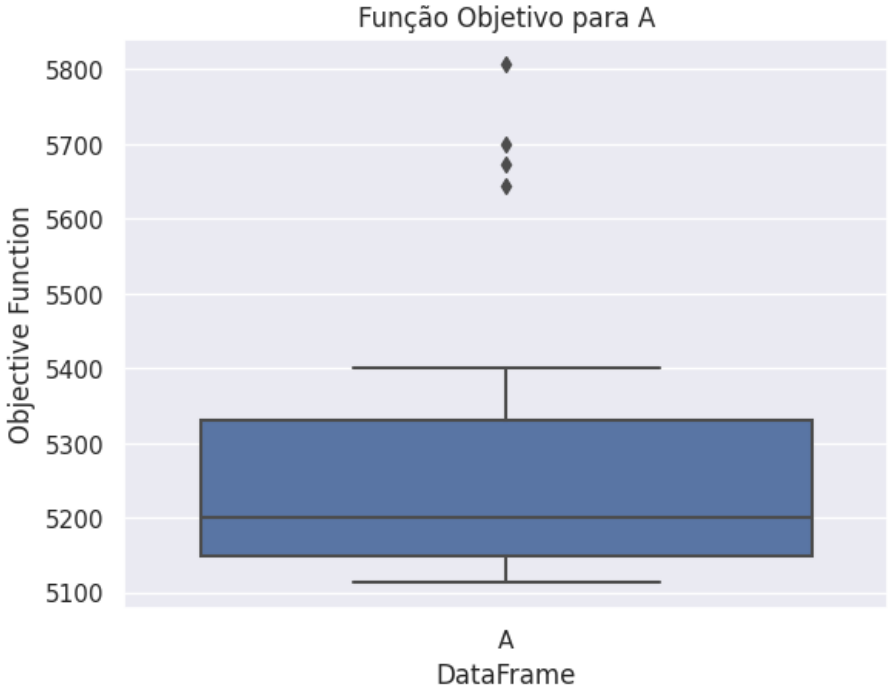

Figure 15: Função objetivo para A.

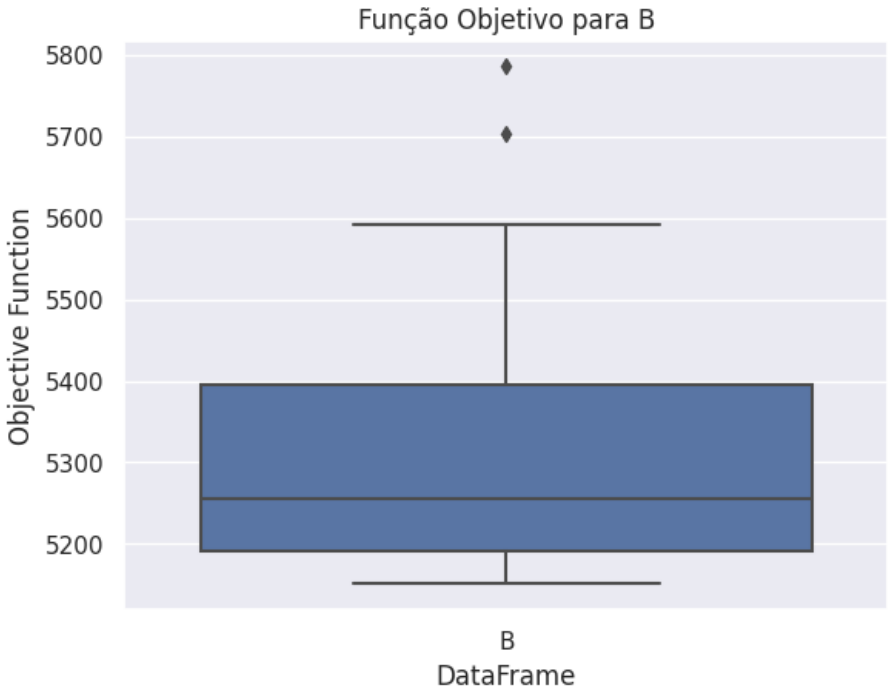

Figure 16: Função objetivo para B.

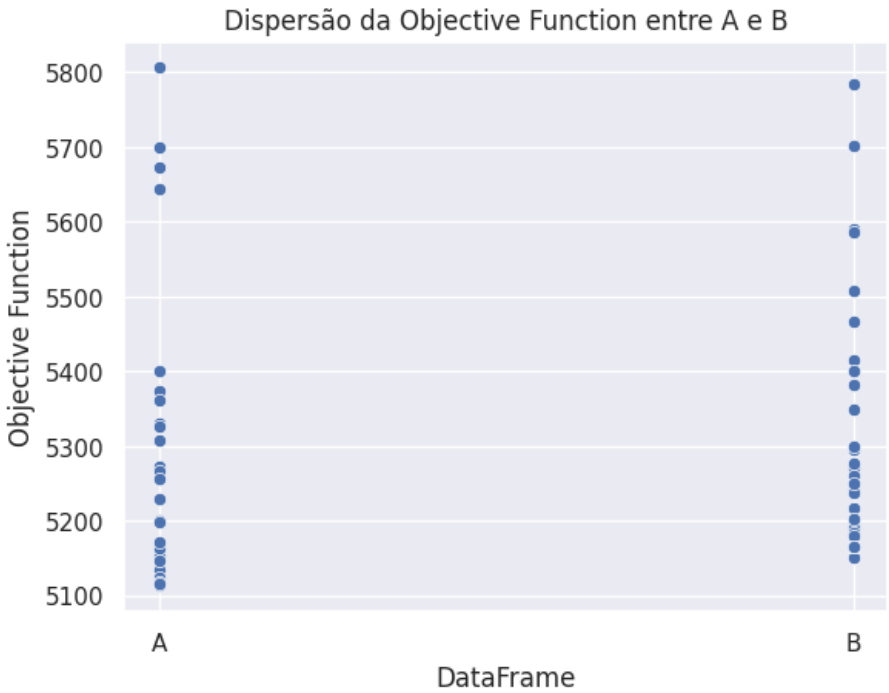

Figure 17: Dispersão da Objective Function entre A e B.

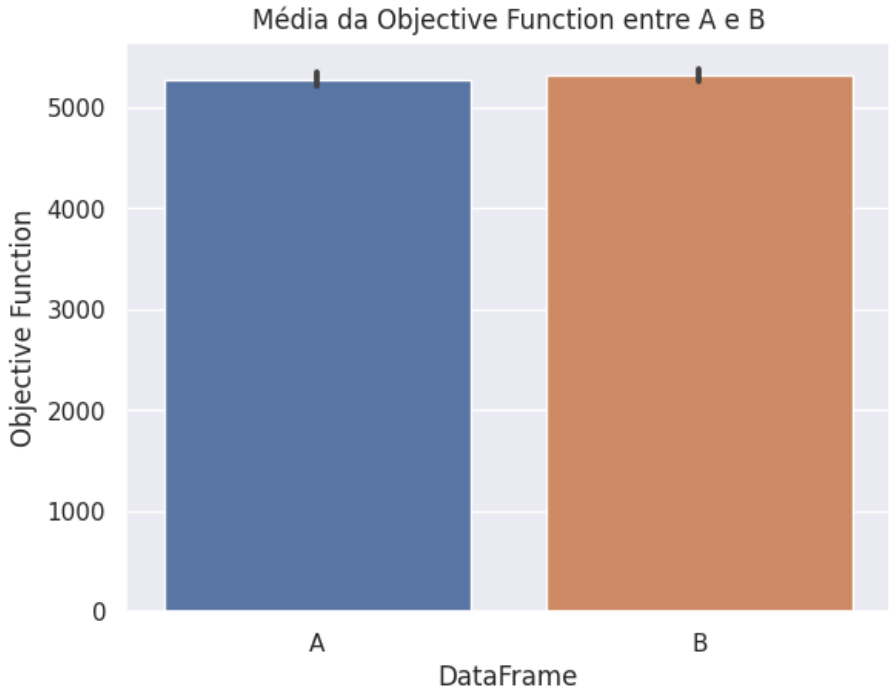

Figure 18: Média da Objective Function entre A e B.

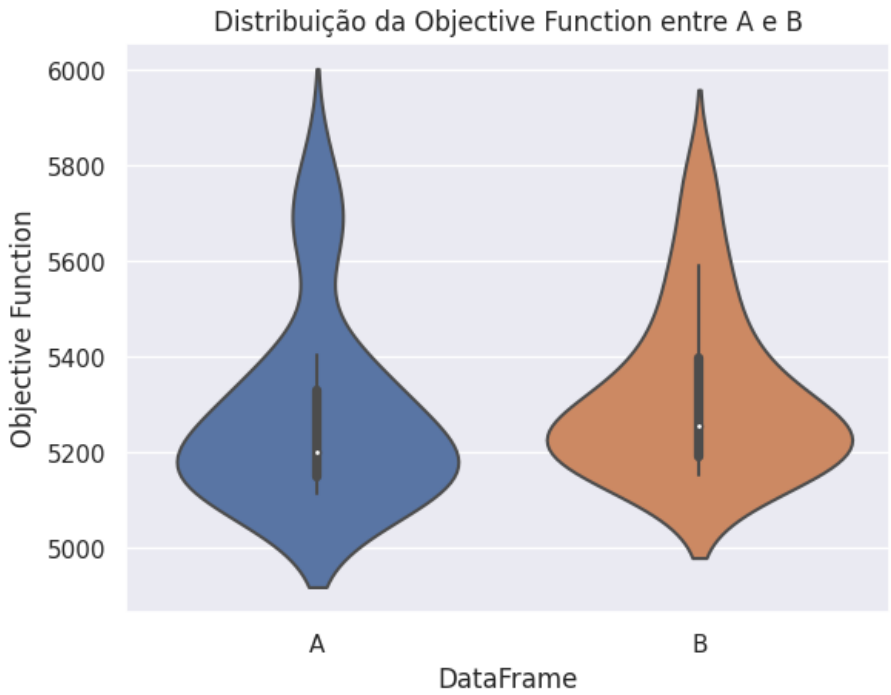

Figure 19: Distribuíção da Objective Function entre A e B.

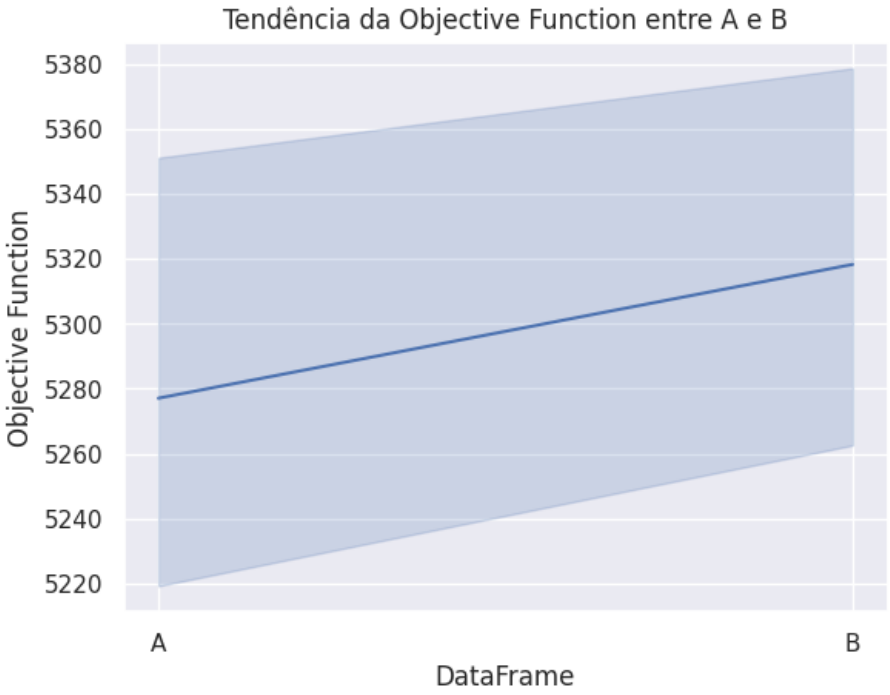

Figure 20: Tendência da Objective Function entre A e B.

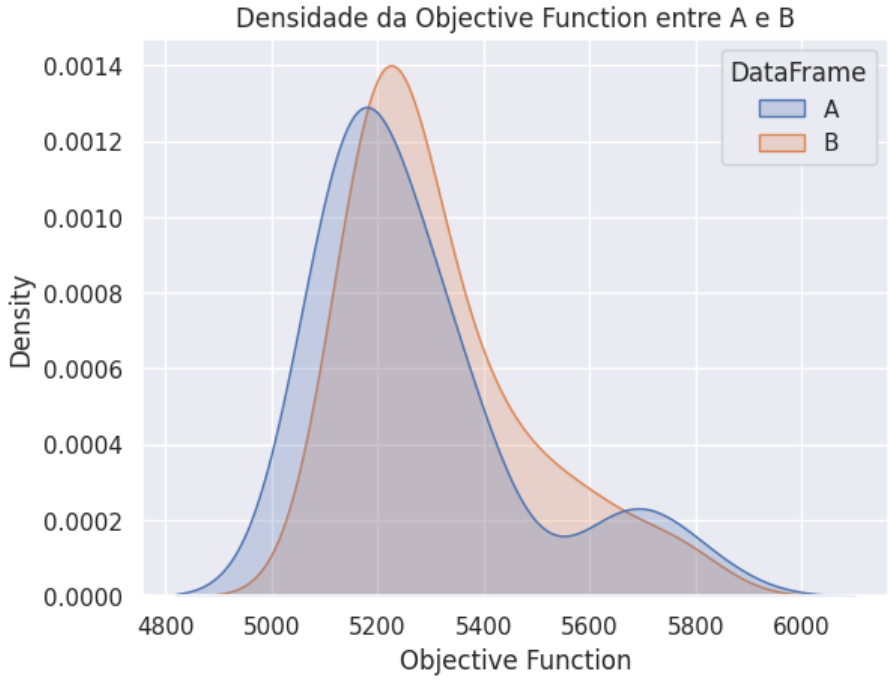

Figure 21: Densidade da Objective Function entre A e B.

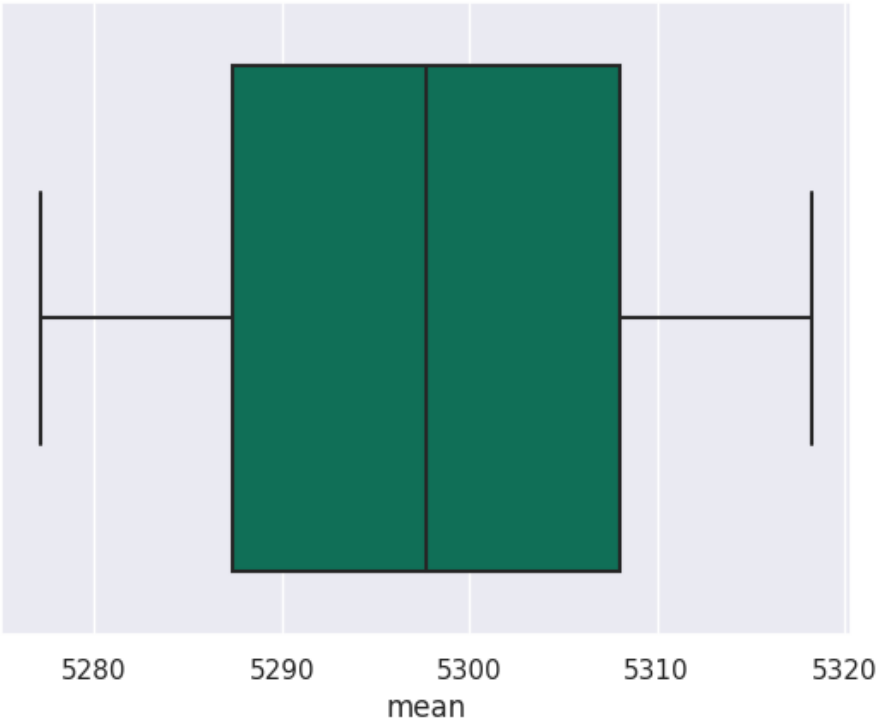

Figure 22: Box plot para a média entre as duas penalidades.

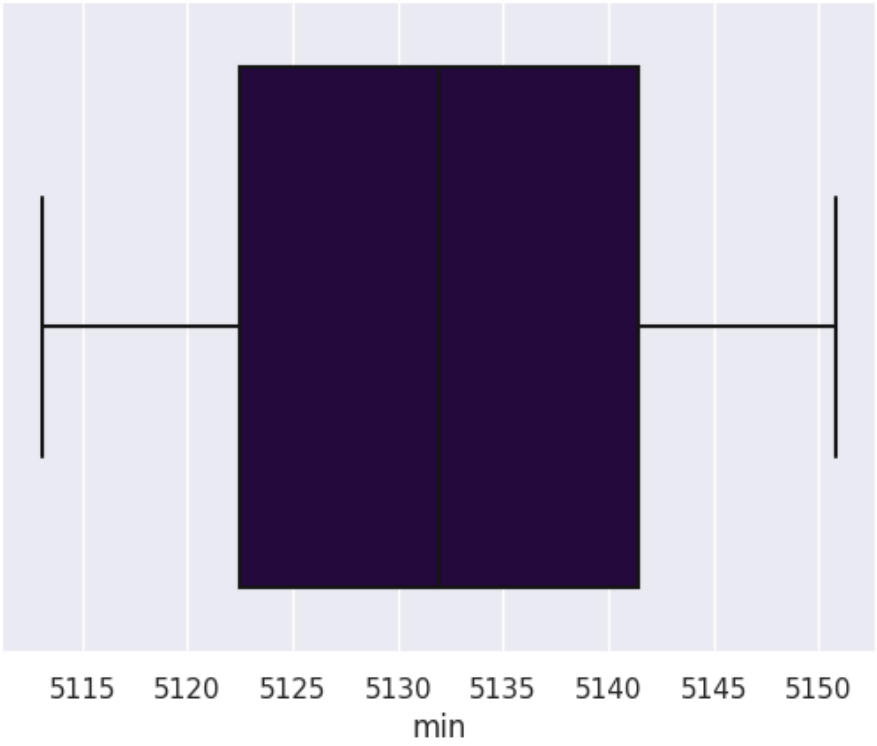

Figure 23: Box plot para o mínimo entre as duas penalidades.

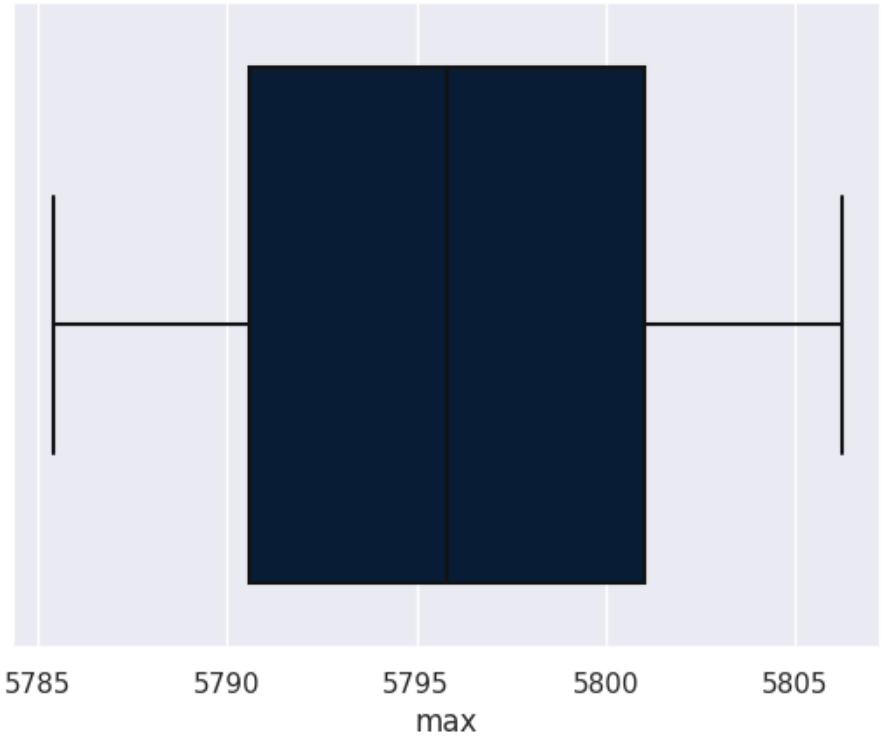

Figure 24: Box plot para o máximo entre as duas penalidades.

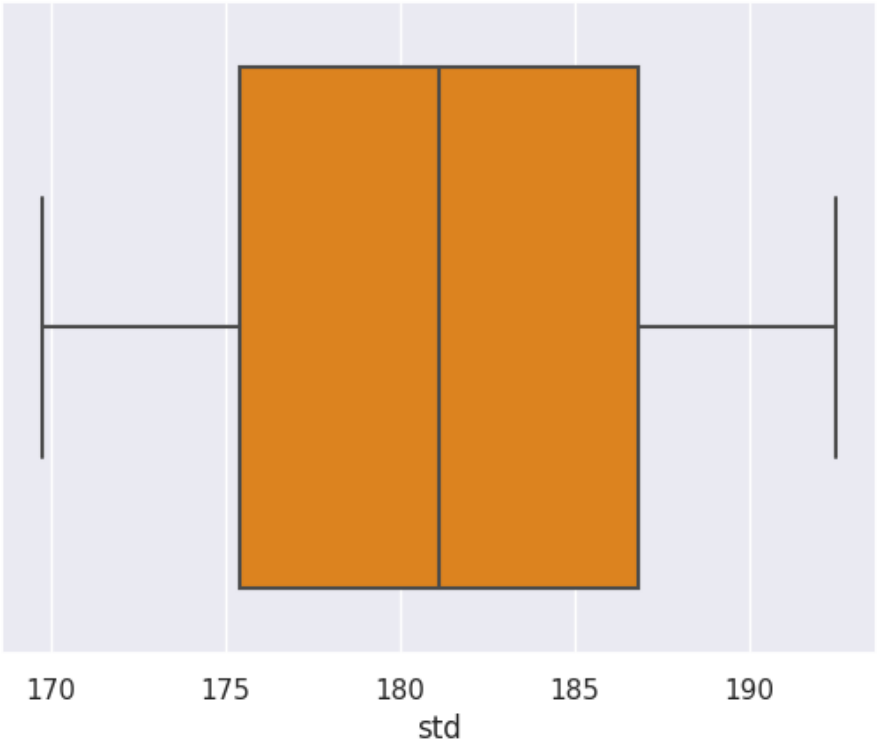

Figure 25: Box plot para o desvio-padrão entre as duas penalidades.

7 Como executar

O arquivo "Makefile" é um arquivo de script utilizado para automatizar tarefas de compilação e execução em sistemas Unix-like. No exemplo fornecido, o arquivo Makefile tem duas regras definidas: "first" e "second".

A regra "first" está associada ao comando 'python3 -B src/main.py first'. Isso significa que, ao executar o comando 'make first' no terminal, o Makefile executará o comando 'python3 -B src/main.py first'. O parâmetro "first" é passado para o script main.py, indicando que o problema 1 deve ser executado.

Da mesma forma, a regra "second" está associada ao comando 'python3 -B src/main.py second'. Ao executar o comando 'make second' no terminal, o Makefile executará o comando 'python3 -B src/main.py second'. O parâmetro "second" é passado para o script main.py, indicando que o problema 2 deve ser executado.

Em resumo, o Makefile fornecido simplifica a execução dos problemas 1 e 2, permitindo que você execute facilmente cada um deles executando os comandos 'make first' e 'make second', respectivamente. Isso ajuda a automatizar o processo de compilação e execução do código, tornando-o mais conveniente e eficiente.

```
# Executar com problema 1
first:
    python3 -B src/main.py first

# Executar com problema 2
second:
    python3 -B src/main.py second
```

8 Dashboard

Para a análise dos gráficos, foram utilizados dados gerados previamente em formato CSV. Esses dados foram processados e visualizados em um Jupyter Notebook, que está localizado na pasta "dashboard".

O Jupyter Notebook é uma aplicação web que permite a criação e compartilhamento de documentos interativos contendo código, visualizações e explicações em um ambiente integrado. Ele suporta várias linguagens de programação, incluindo Python, que foi usada para gerar os gráficos com base nos dados CSV.

Os arquivos de resultado CSV foram previamente gerados e estão localizados na pasta "out". Esses arquivos contêm os dados necessários para a criação dos gráficos e foram importados para o Jupyter Notebook para análise e visualização.

Dentro do Jupyter Notebook, foram utilizadas bibliotecas de plotagem de gráficos, como Matplotlib ou Seaborn, para criar visualizações a partir dos dados contidos nos arquivos CSV. Essas bibliotecas oferecem uma ampla gama de opções de plotagem, permitindo a criação de gráficos personalizados e informativos.

Assim, o Jupyter Notebook na pasta "dashboard" foi utilizado como uma ferramenta de análise e visualização dos dados contidos nos arquivos CSV, proporcionando insights e facilitando a interpretação dos resultados obtidos.

9 Conclusão

Neste relatório, foi aplicada a programação genética para resolver dois problemas de programação não linear. O objetivo foi encontrar soluções aproximadas para os problemas considerando as restrições envolvidas. Os resultados obtidos com a programação genética foram promissores, demonstrando a eficácia dessa abordagem para lidar com problemas complexos.

No problema 1, nosso objetivo era minimizar a função objetivo sujeita a restrições de desigualdade. Através da programação genética, conseguimos encontrar soluções que se aproximaram da solução ótima. A aplicação das restrições no algoritmo foi realizada por meio de penalidades estáticas, em que as violações das restrições eram penalizadas na função objetivo. Essa abordagem permitiu que o algoritmo buscasse soluções que respeitassem as limitações, mesmo que não fossem estritamente ótimas.

No problema 2, tínhamos a tarefa de minimizar a função objetivo considerando restrições de igualdade e desigualdade. Novamente, a programação genética mostrou-se eficaz na busca por soluções aproximadas. Nesse caso, as restrições foram tratadas por meio de uma abordagem de reparação, em que os indivíduos que violavam as restrições eram corrigidos durante o processo de evolução. Essa técnica permitiu que o algoritmo convergisse para soluções que satisfaziam todas as restrições impostas.

Em ambos os problemas, a programação genética demonstrou sua capacidade de encontrar soluções aproximadas em espaços de busca complexos. A flexibilidade da representação simbólica e a aplicação adequada das restrições contribuíram para os resultados positivos alcançados.

Em suma, a programação genética se mostrou uma abordagem eficaz e flexível para a resolução de problemas não lineares, permitindo a consideração das restrições envolvidas. Os resultados alcançados foram próximos considerando as duas restrições e não demonstraram estar distante do resultado óptimo do problema.