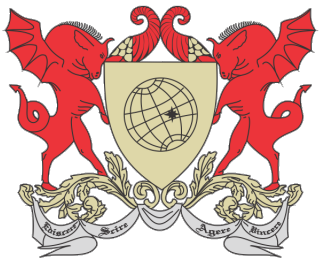

UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Trabalho I
META-HEURÍSTICAS

Luciano Belo de Alcântara Júnior - 3897

Trabalho I apresentado à disciplina de Meta-Heurísticas do curso de Ciência da Computação da Universidade Federal de Viçosa.

Florestal
17 de Maio de 2023

CCF 480 - META-HEURÍSTICAS

Trabalho I

Luciano Belo de Alcântara Júnior - 3897

17 de Maio de 2023

Contents

1 Introdução

O presente trabalho tem como objetivo o entendimento e aplicação de meta-heurísticas vistas na disciplina de CCF 480. O foco primário é a implementação do algoritmo baseado em **ILS** (*Iterated Local Search*) e outro de escolha, neste cenário o **Hill Climbing** para minimizar as seguintes funções:

1. $f(x, y) = x^2 + y^2 + 25(\sin^2(x) + \sin^2(y))$
2. $f(x, y) = -(y + 47) \sin\left(\sqrt{|y + 0.5y + 47|}\right) - x \sin\left(\sqrt{|x - (y + 47)|}\right)$

2 Algoritmos

2.1 Iterated Local Search

A meta-heurística Iterated Local Search (ILS) [1] é um algoritmo de busca heurística que combina a busca local com uma perturbação aleatória para encontrar soluções melhores em problemas de otimização. Ela é especialmente útil em problemas onde a busca local tradicional pode ficar presa em ótimos locais subótimos.

Para aplicar um algoritmo ILS, temos quatro etapas a serem especicadas: (a)Procedimento para gerar solução inicial, que gera uma solução inicial s0 para o problema; (b) Procedimento BuscaLocal, que retorna uma solução possivelmente melhoradas; (c)Procedimento Perturbacao, que modifica a solução corrente guiando a uma solução intermediárias e (d) Procedimento CriterioAceitacao, que decide de qual solução a próxima perturbação será aplicada.

2.2 Hill Climbing

A metaheurística Hill Climbing [2], é um algoritmo de otimização que tenta encontrar a melhor solução em um espaço de busca. Esta heurística assenta num princípio simples: "subir a colina", ou seja, fazer movimentos que levem a uma melhoria gradual da solução atual.

O funcionamento básico do Hill Climbing é o seguinte:

- Inicialização: Começamos com uma solução inicial que pode ser gerada aleatoriamente ou definida de outra forma.
- Avaliação: Calculamos a medida de qualidade da solução atual. Essa métrica é chamada de função objetivo ou função de avaliação. Isso determina o quão boa é a solução em termos de otimização.

- Vizinhança: Criamos soluções de vizinhança para a solução atual. Uma solução secundária é obtida com uma pequena modificação na solução atual. Essas alterações podem ser feitas de algumas maneiras diferentes, dependendo do problema.

- Movimento: Selecionamos a melhor solução das soluções vizinhas com base na função objetivo. Se uma solução vizinha for melhor que a atual, passamos para essa solução e a tornamos a nova solução atual. Caso contrário, encerramos o algoritmo porque pensamos ter encontrado um ótimo local.

- Iteração: as etapas 2 a 4 são repetidas até que uma condição de término seja alcançada. Essa condição pode ser um número máximo de iterações, uma melhoria mínima da solução ou algum outro critério especificado.

Hill Climbing é um algoritmo de busca local, o que significa que ele pode ficar preso em ótimos locais e não encontrar a melhor solução global. Isso ocorre porque o algoritmo só faz movimentos com base em melhorias locais imediatas, ignorando movimentos que podem levar a melhorias de longo prazo.

3 Implementação

O trabalho foi implementado utilizando a linguagem de programação golang [3]. No geral, o Go é uma escolha atraente para o desenvolvimento de software devido à sua simplicidade, eficiência, concorrência embutida e suporte multiplataforma. É uma linguagem versátil que pode ser usada em uma ampla variedade de aplicativos, desde sistemas de baixo nível até serviços web de alto desempenho.

o Go assim como a linguagem C compila todos os arquivos-fonte do projeto e suas dependências, gerando um arquivo executável específico para o sistema operacional e arquitetura em que o comando é executado. O arquivo executável resultante é geralmente chamado de "binário" e pode ser executado sem a necessidade de um ambiente de desenvolvimento ou a instalação do Go.

Sendo assim para executar o projeto temos duas formas principais: makefile e imagem docker. Pelo makefile conseguir rodar o projeto de forma local e também gerar a build tendo como resultado o executável.

Abaixo, temos os comandos disponíveis pelo Makefile:

```
build:
        go build -o dist/meta-heuristic-go
dev:
        go run main.go ./data/b.csv
```

```
run:
        ./dist/meta-heuristic-go  ./data/b.csv
```

Utilizando o *docker* é possível criar uma imagem do projeto utilizando o comando abaixo:

```
docker build -t meta-heuristic-go .
```

Para executar o contêiner Docker a partir da imagem criada pelo comando anterior execute o comando abaixo, sendo que */caminho/do/arquivo* deve ser substituido pelo arquivo .csv de entrada:

```
docker run meta-heuristic-go /caminho/do/arquivo
```

Vale ressaltar que para o projeto foram criados quatro arquivos *csv* seguindo o que foi especificado na documentação do trabalho que serão os arquivos de entrada. Sendo que evaluate faz refêrencia a função a ser minimizada poder ser **"first" (1)** ou **"second" (2)**

	Padrão	Padrão	Padrão	Padrão	Padrão
1	evaluate	x_low	x_high	y_low	y_high
2	first	-5	5	-5	5

Figure 1: Example CSV

Ademais ao final da execução das 30 iterações é gerada uma tabela no terminal e como saída um gráfixo boxplot (ficará na pasta out na raiz do projeto). Caso esteja executando via *docker* é possível copiar a imagem gerada para um diretório qualquer,seguindo o comando abaixo (a imagem "image.png" é meramente um exemplo, fique atendo ao nome do arquivo mostrado no terminal para copiar corretamente o gráfico):

```
docker cp <container-id>:/go/src/app/out/image.png .
```

Por fim, ainda sobre como executar o projeto, é possível executar um arquivo *sh* que já executa os quatro arquivos de testes especificados:

```
sh run.sh
```

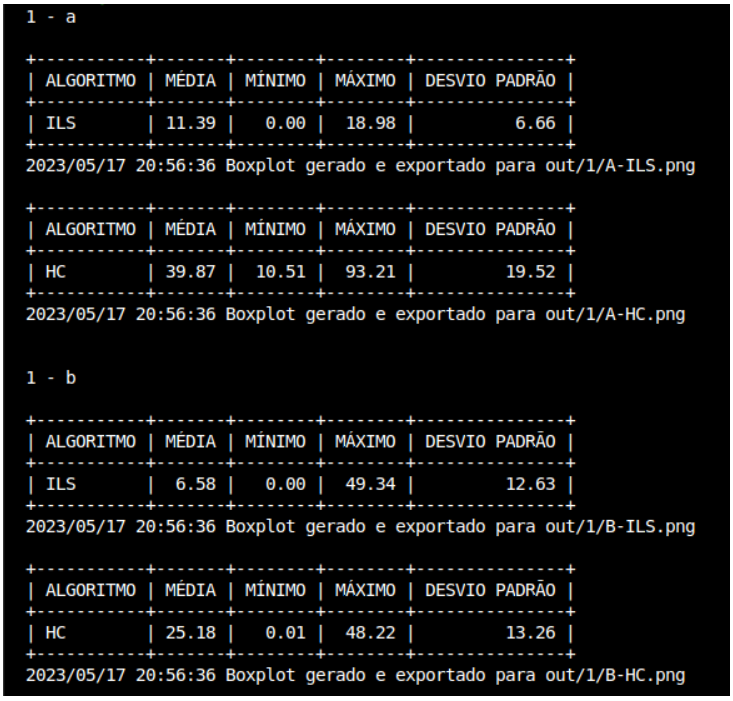

Figure 2: Prévia dos resultados

3.1 Código

app/
- base/
- constants/
- core/
  - math/
  - packages/
- domain/
  - csv/
  - model/
  - reader/
- export/
- use-cases/
- utils/

A estrutura de pastas mencionada segue uma organização comum encontrada em

muitos projetos de software. Sendo que a função de cada estrutura de pastas é:

- base/: Essa pasta contém os arquivos e configurações básicas do projeto. Inclui configurações de inicialização, arquivos de configuração global, utilitários comuns e outros recursos essenciais para o funcionamento do projeto. Neste caso, é encontrada a função principal **Bootstrap** que é executava no arquivo *main.go*

- constants/: Essa pasta armazena as constantes do projeto. São valores fixos que são usados em várias partes do código e não mudam ao longo da execução do programa.

- core/: Essa pasta contém a lógica principal do projeto, responsável por implementar as funcionalidades principais. Ela é dividida em duas subpastas:

  - math/: Essa subpasta contém pacotes ou módulos relacionados a funcionalidades matemáticas. Neste caso, é criado uma função objetivo que calcula o resultado com base na escolha entre as funções (1) e (2).
  - packages/: Essa subpasta contém pacotes utilitários compartilhados que são as duas meta-heurísticas.

- domain/: Essa pasta representa a camada de domínio do projeto. Ela encapsula as regras de negócio, os modelos de dados e a lógica específica do domínio em que o projeto está inserido. Ela é dividida em três subpastas:

  - csv/: Essa subpasta contém pacotes ou módulos relacionados à manipulação de arquivos CSV. Inclui funcionalidades de leitura, escrita, validação e processamento de arquivos CSV.
  - model/: Essa subpasta contém os modelos e entidades do domínio. Ela representa as estruturas de dados que são usadas para representar os objetos do mundo real com os quais o projeto está lidando.
  - reader/: Essa subpasta contém pacotes ou módulos relacionados à leitura de dados. Inclui funcionalidades de leitura dos dados CSV validados pelo domínio csv.

- export/: Essa pasta é responsável pela exportação de dados. Ela contém funcionalidades para gerar a tabela com os resultados e o gráfixo boxplot

- use-cases/: Essa pasta contém os casos de uso ou as interações do sistema.

- utils/: Essa pasta contém utilitários e funções auxiliares que podem ser usados em várias partes do projeto. Ela inclui funcionalidades genéricas que não estão diretamente relacionadas ao domínio específico do projeto, mas são úteis para tarefas comuns de programação. Por exemplo, geração de valores randômicos

3.1.1 Hill Climbing

O algoritmo utilizado para o Hill Climbing segue os seguintes passos:

1. Configuração dos parâmetros do algoritmo:
    - maxIterations: O número máximo de iterações que o algoritmo irá executar.
    - stepSize: O tamanho do passo utilizado para gerar os próximos estados durante a busca.
2. Inicialização do estado da solução:
    - initialState: É criado um estado inicial da solução com coordenadas X e Y aleatórias dentro dos limites especificados pelos parâmetros data.XLow, data.XHigh, data.YLow e data.YHigh.
3. Criação de instâncias de serviços e variáveis de controle:
    - evaluateService: É criada uma instância do serviço de avaliação (EvaluateService) que será usado para calcular o objetivo da função de avaliação nos diferentes estados da solução.
    - currentState: É atribuído o estado inicial à variável currentState.
    - bestObjective: É calculado o valor objetivo da função de avaliação para o estado inicial e atribuído à variável bestObjective.
4. Loop principal do algoritmo:
    - O loop será executado maxIterations vezes.
    - Em cada iteração, um próximo estado é gerado adicionando um valor aleatório ao estado atual nas coordenadas X e Y.
    - O valor objetivo da função de avaliação é calculado para o próximo estado.
    - É verificado se o valor objetivo do próximo estado é melhor do que o valor objetivo atual (bestObjective). Se for, o bestObjective é atualizado.
    - O estado atual é atualizado para o próximo estado.
5. Retorno do melhor valor objetivo encontrado durante as iterações do algoritmo.

3.1.2 Iterable Local Search

A heurística foi implementada utilizando quatro funções principais, sendo que elas tem as seguintes finalidades:

1. IteratedLocalSearch(data *reader.DataCSV) float64: Essa função representa o algoritmo ILS completo. Recebe um objeto data contendo os dados necessários para o cálculo da função objetivo. A função inicia gerando uma solução inicial aleatória utilizando a função GenerateRandomSolution. Em seguida, são configurados os parâmetros do algoritmo, como o número máximo de iterações e o fator de perturbação. A função EvaluateService é utilizada para calcular a função objetivo da melhor solução inicial. Em um loop principal, o algoritmo executa a busca local na solução atual, faz a perturbação dessa solução, executa outra busca local na solução perturbada e atualiza a melhor solução caso a perturbada seja melhor. O processo é repetido pelo número máximo de iterações e a função retorna o valor da melhor solução encontrada.

2. localSearch(solution model.SolutionState, data *reader.DataCSV) model.SolutionState: Essa função realiza a busca local em uma solução. Recebe uma solução inicial solution e o objeto data contendo os dados necessários para o cálculo da função objetivo. É definido o número máximo de iterações para a busca local. Utilizando a função EvaluateService, a função itera um número máximo de vezes, perturbando a solução atual e verificando se a solução perturbada é melhor que a solução atual. Caso seja, a solução atual é atualizada. A função retorna a melhor solução encontrada.

3. perturb(solution model.SolutionState, perturbationFactor float64, data *reader.DataCSV) model.SolutionState: Essa função realiza a perturbação em uma solução. Recebe uma solução solution, o fator de perturbação perturbationFactor e o objeto data com os limites dos dados. A função utiliza a função perturbCoordinate para perturbar as coordenadas da solução, gerando uma nova solução perturbada. A função retorna a nova solução perturbada.

4. perturbCoordinate(coordinate, perturbationFactor, low, high float64) float64: Essa função perturba uma coordenada de acordo com um fator de perturbação. Recebe a coordenada atual coordinate, o fator de perturbação perturbationFactor e os limites low e high. A função gera uma perturbação aleatória utilizando a função RandomStep do pacote utils. Em seguida, a coordenada é atualizada somando a perturbação. É feita uma verificação para garantir que a coordenada perturbada esteja dentro dos limites especificados. A função retorna a coordenada perturbada.

Explicando melhor a função IteratedLocalSearch, ela tem os seguintes passos:

1. Geração da solução inicial:

    - Utiliza a função `GenerateRandomSolution` do pacote `utils` para gerar uma solução inicial aleatória com base nos limites `data.XLow`, `data.XHigh`, `data.YLow` e `data.YHigh`.

- A solução inicial é atribuída à variável `initialSolution`.
- A melhor solução é inicializada com a solução inicial, ou seja, `bestSolution` recebe `initialSolution`.

2. Configuração dos parâmetros do algoritmo:
   - Define o número máximo de iterações como 100 e o fator de perturbação como 0.1.
   - Esses valores são atribuídos às variáveis `maxIterations` e `perturbationFactor`, respectivamente.
3. Cálculo do objetivo da melhor solução inicial:
   - Cria uma instância de `EvaluateService` do pacote `math`.
   - Utiliza o método `CalculateObjective` de `evaluateService` para calcular o valor da função objetivo da melhor solução inicial, com base nos dados de avaliação `data.Evaluate`, coordenadas `bestSolution.X` e `bestSolution.Y`.
   - O resultado é atribuído à variável `bestObjective`.
4. Loop principal do algoritmo:
   - Executa um loop `for` que itera de 0 até `maxIterations - 1`.
   - A cada iteração:
     - Realiza uma busca local na melhor solução atual (`bestSolution`) chamando a função `localSearch`.
     - Realiza uma perturbação na solução retornada pela busca local utilizando a função `perturb`.
     - Realiza outra busca local na solução perturbada chamando novamente a função `localSearch`.
     - Calcula o valor da função objetivo para a solução perturbada utilizando o método `CalculateObjective` de `evaluateService`.
     - Verifica se o valor da função objetivo da solução perturbada (`perturbedObjective`) é menor que o da melhor solução atual (`bestObjective`).
       * Caso seja, atualiza a melhor solução (`bestSolution`) para a solução perturbada e atualiza o valor da função objetivo (`bestObjective`) para o valor da solução perturbada.
5. Retorna o valor da melhor solução encontrada (`bestObjective`).

4 Resultados

Para cada algoritmo, foi executado 30 vezes de modo independente para cada função objetivo e foram obtidos os seguintes resultados

4.1 Tabelas

4.1.1 1

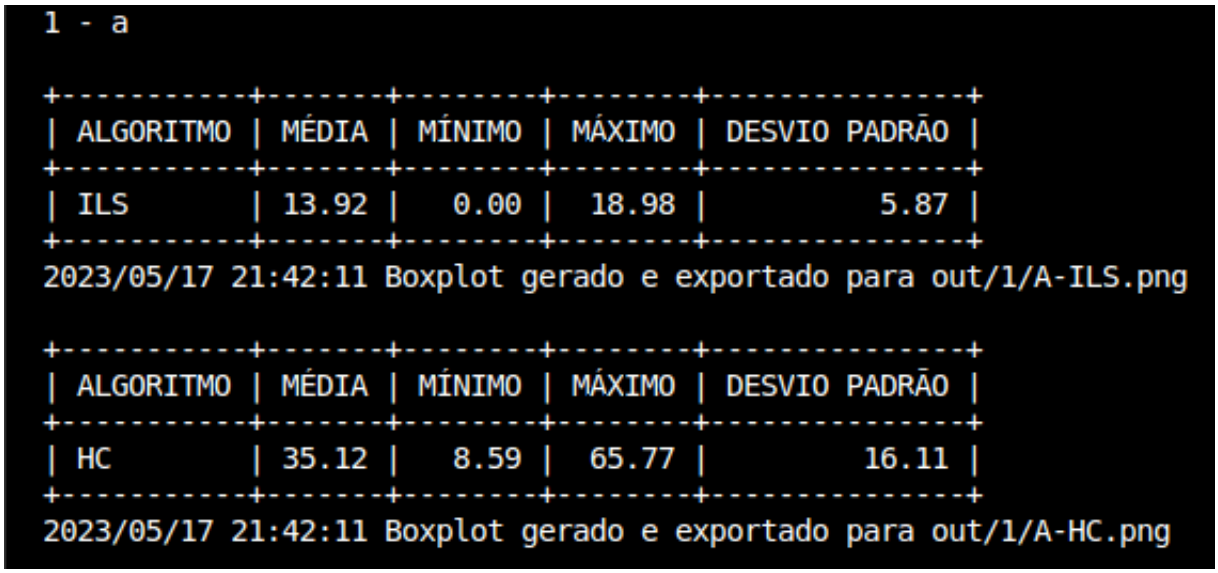

Figure 3: 1 - A

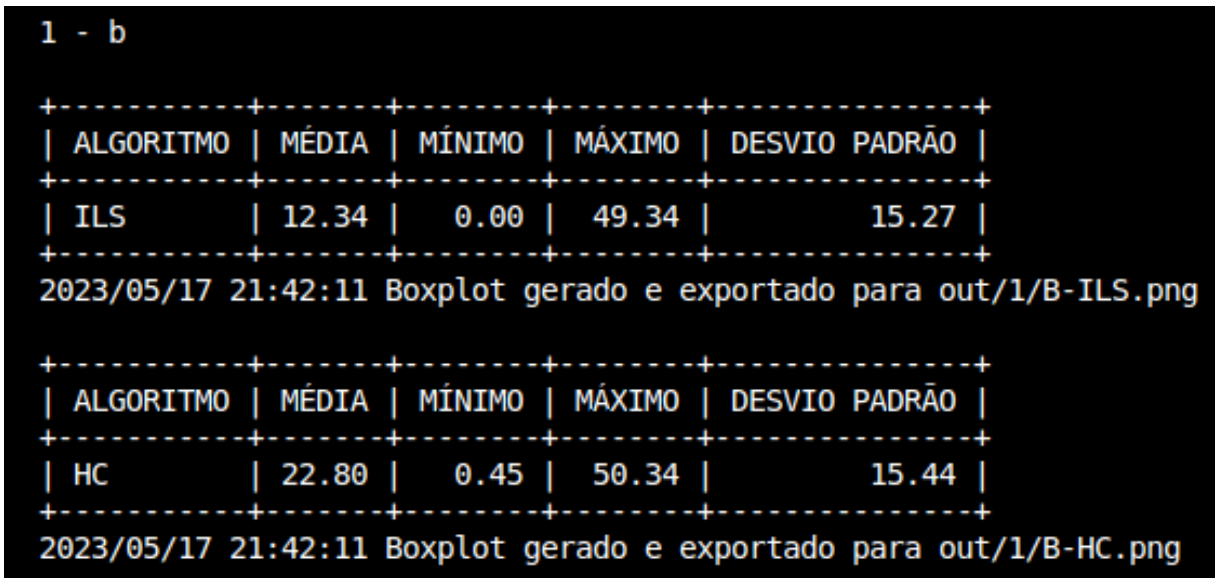

Figure 4: 1 - B

4.1.2 2

```
2 - c

+-----------+---------+---------+--------+---------------+
| ALGORITMO |  MÉDIA  | MÍNIMO  | MÁXIMO | DESVIO PADRÃO |
+-----------+---------+---------+--------+---------------+
| ILS       | -217.56 | -716.46 | 166.14 |        215.17 |
+-----------+---------+---------+--------+---------------+
2023/05/17 21:42:11 Boxplot gerado e exportado para out/2/C-ILS.png

+-----------+-------+---------+--------+---------------+
| ALGORITMO | MÉDIA | MÍNIMO  | MÁXIMO | DESVIO PADRÃO |
+-----------+-------+---------+--------+---------------+
| HC        |  1.94 | -585.96 | 687.86 |        327.43 |
+-----------+-------+---------+--------+---------------+
2023/05/17 21:42:11 Boxplot gerado e exportado para out/2/C-HC.png
```

Figure 5: 2 - C

```
2 - d

+-----------+---------+---------+---------+---------------+
| ALGORITMO |  MÉDIA  | MÍNIMO  | MÁXIMO  | DESVIO PADRÃO |
+-----------+---------+---------+---------+---------------+
| ILS       | -876.25 | -959.64 | -502.60 |        110.74 |
+-----------+---------+---------+---------+---------------+
2023/05/17 21:42:11 Boxplot gerado e exportado para out/2/D-ILS.png

+-----------+---------+---------+--------+---------------+
| ALGORITMO |  MÉDIA  | MÍNIMO  | MÁXIMO | DESVIO PADRÃO |
+-----------+---------+---------+--------+---------------+
| HC        | -486.07 | -938.48 | 141.22 |        326.18 |
+-----------+---------+---------+--------+---------------+
2023/05/17 21:42:11 Boxplot gerado e exportado para out/2/D-HC.png
```

Figure 6: 2 - D

4.2 Boxplot

4.2.1 1 - A

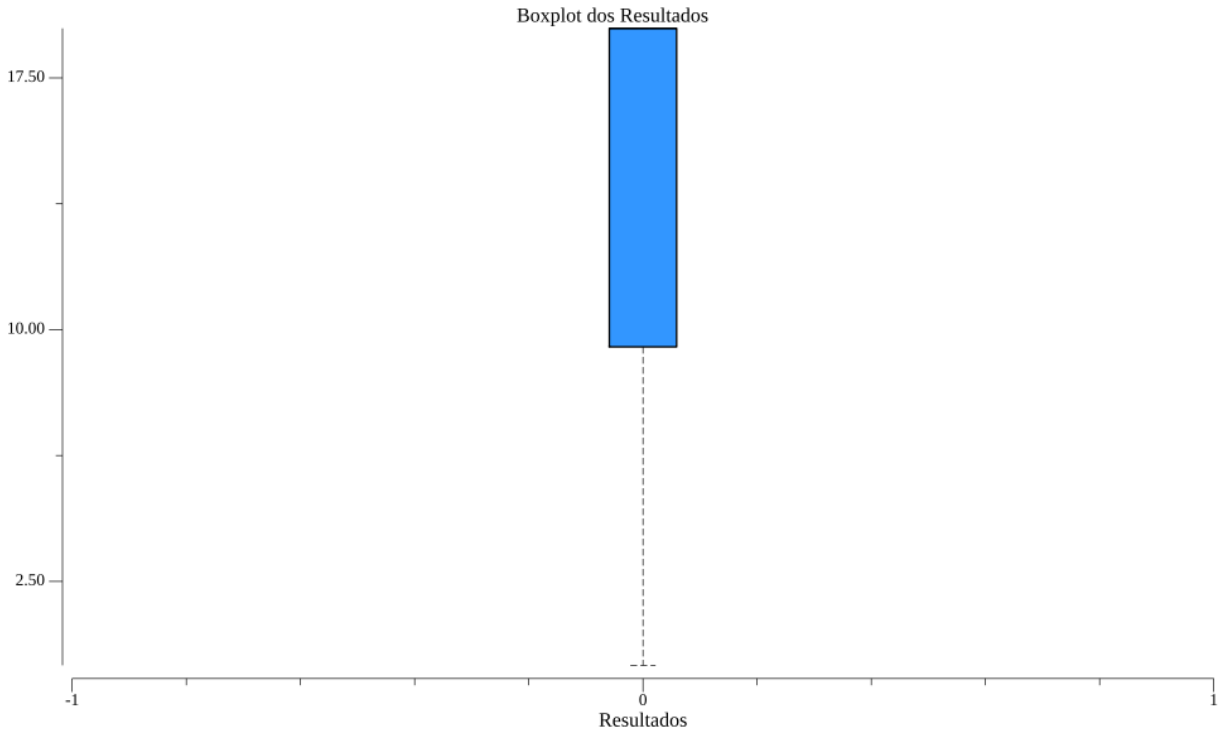

Figure 7: 1 - A - ILS

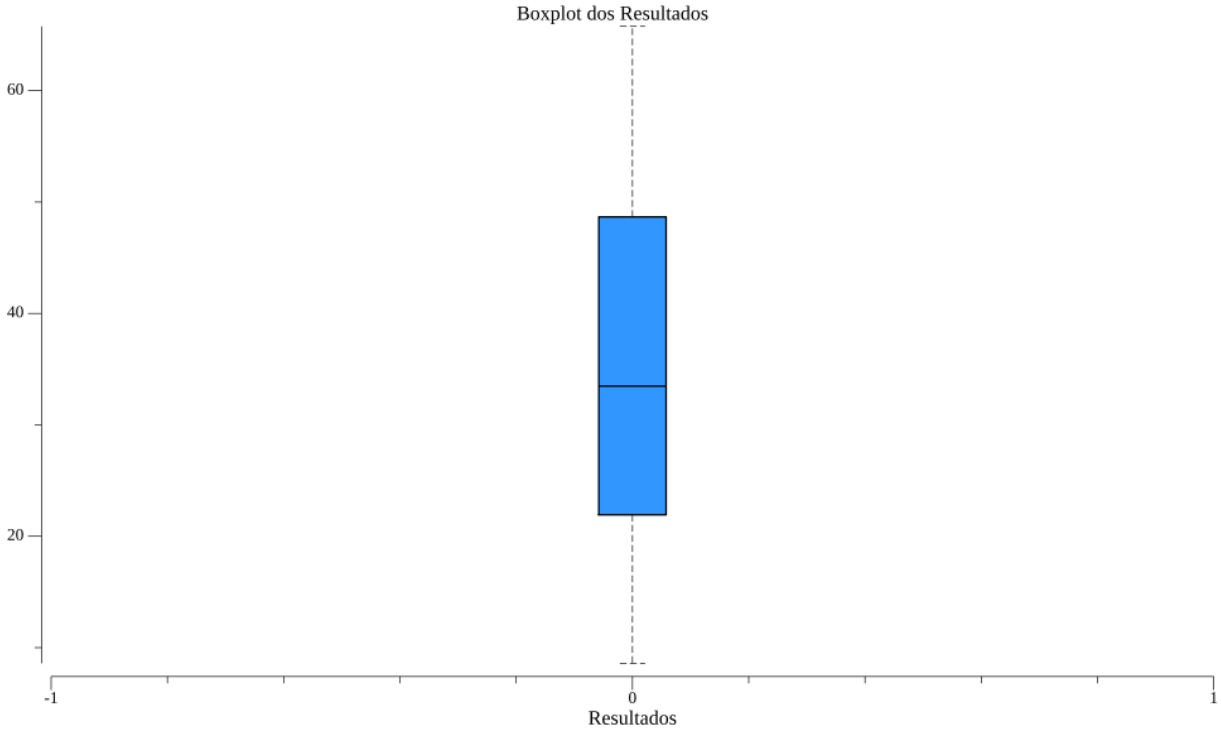

Figure 8: 1 - A - HC

4.2.2 1 - B

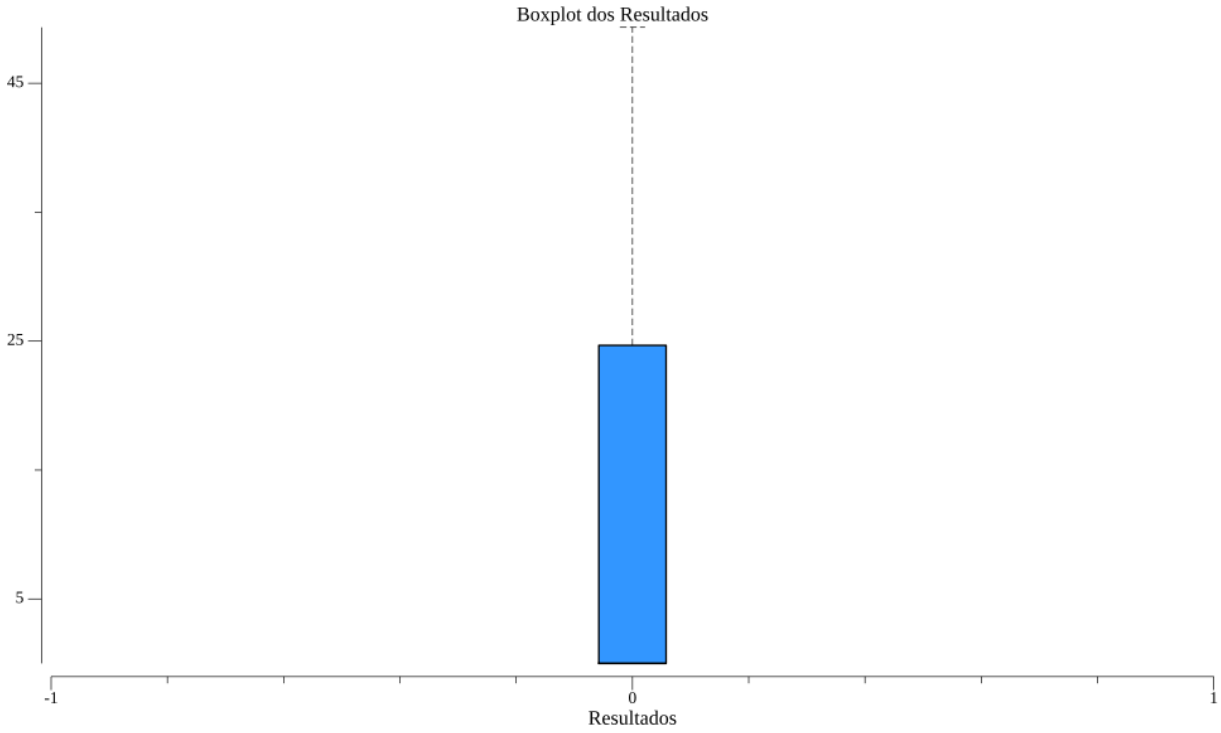

Figure 9: 1 - B - ILS

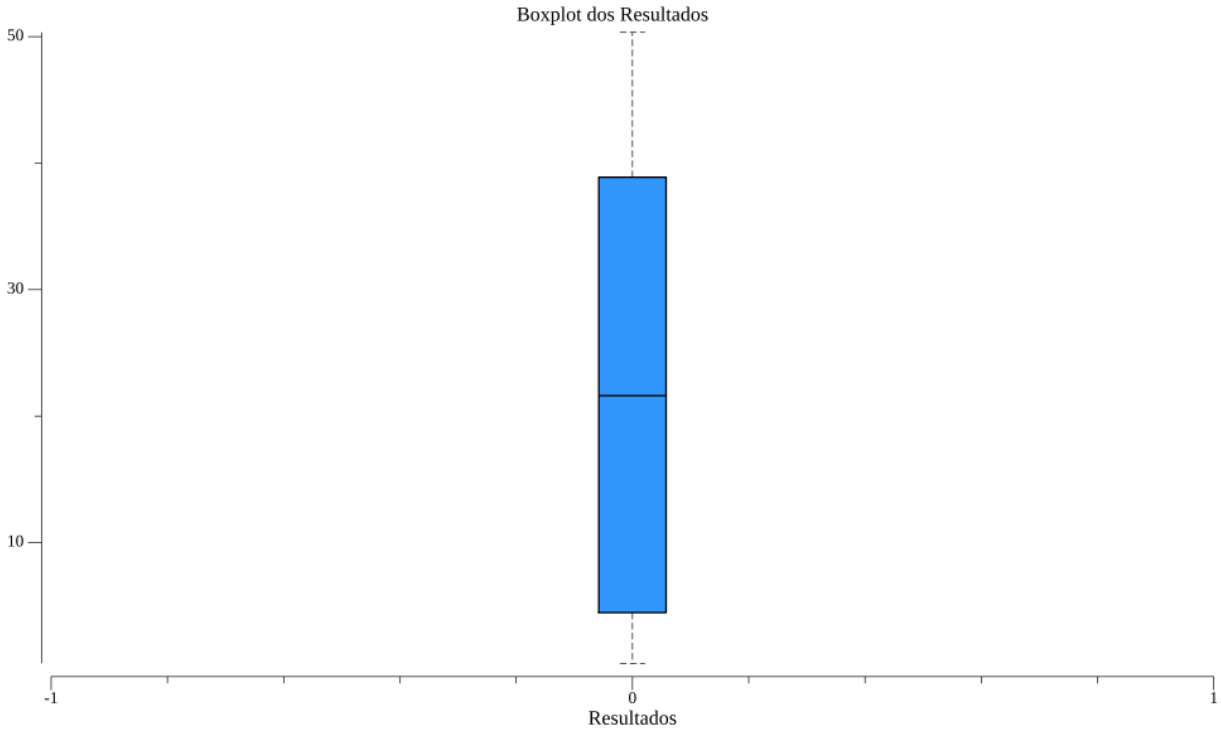

Figure 10: 1 - B - HC

4.2.3 2 - C

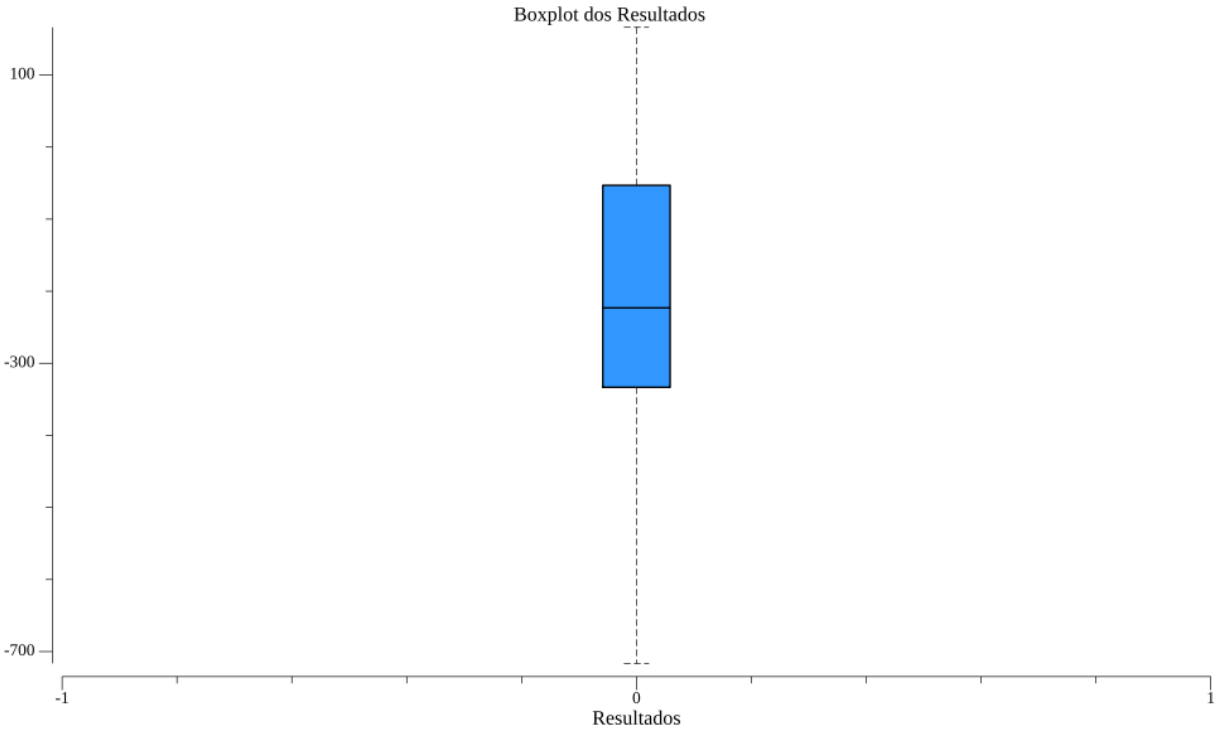

Figure 11: 2 - C - ILS

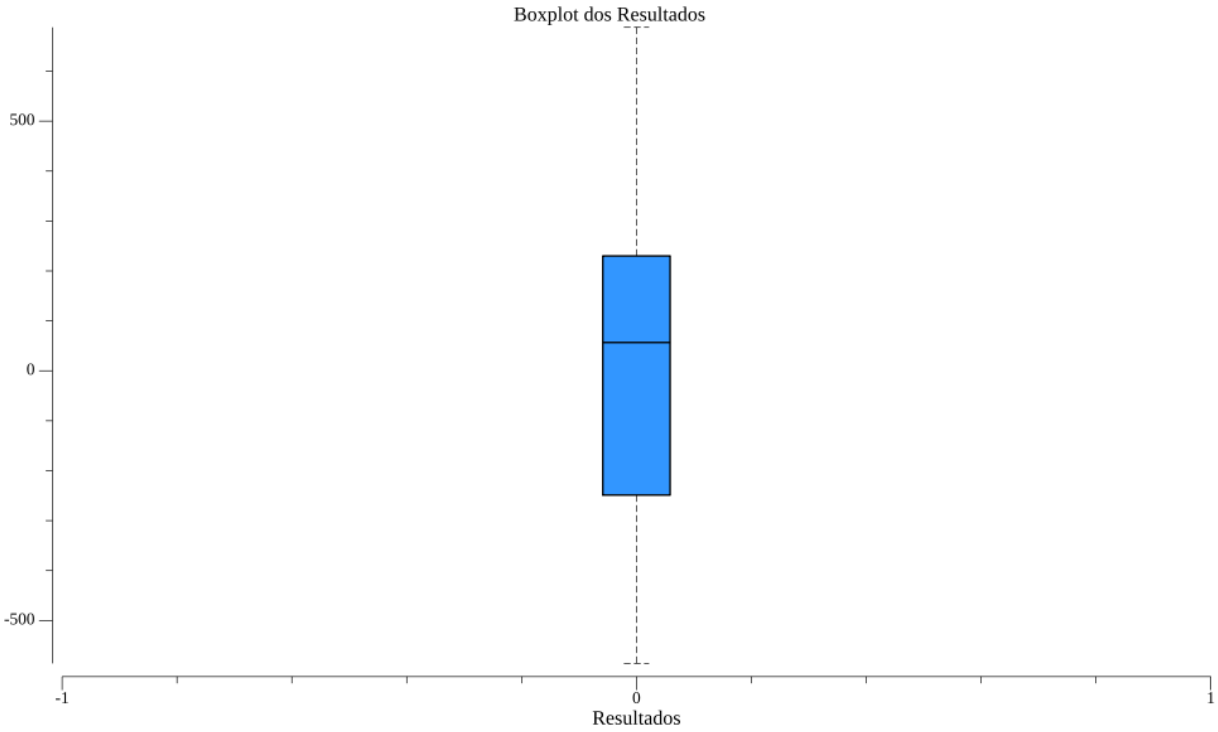

Figure 12: 2 - C - HC

4.2.4 2 - D

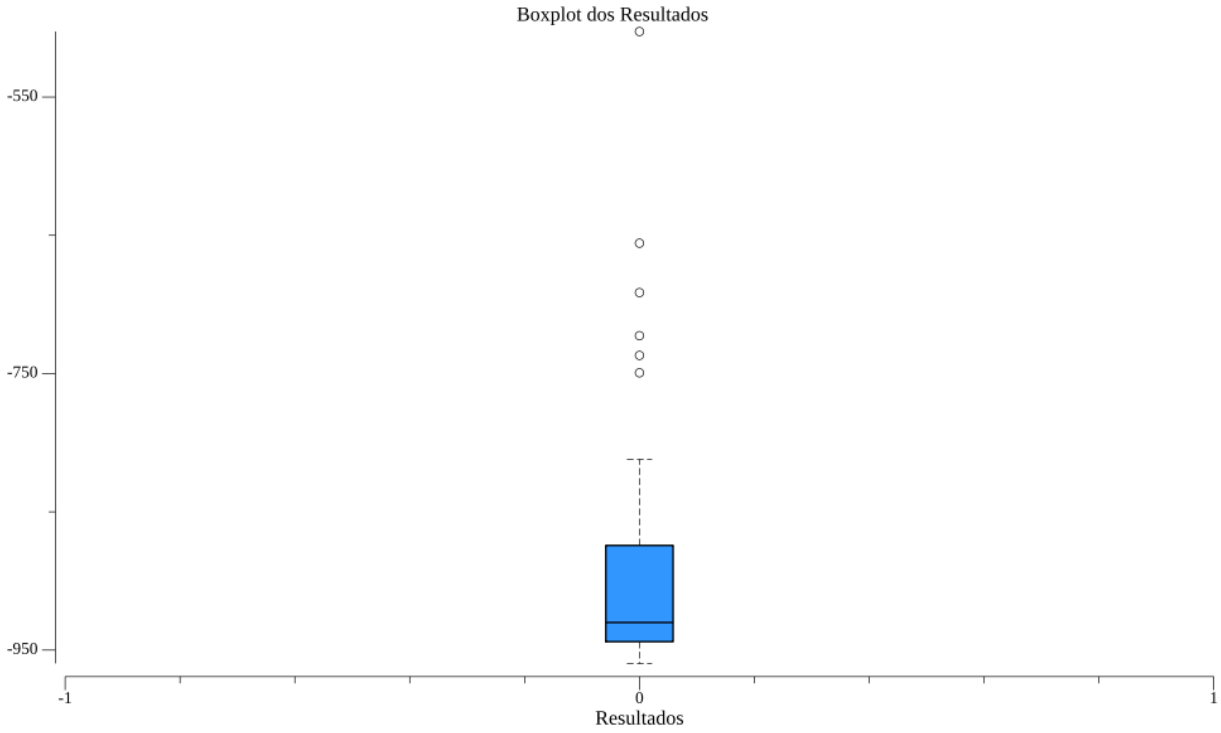

Figure 13: 2 - D - ILS

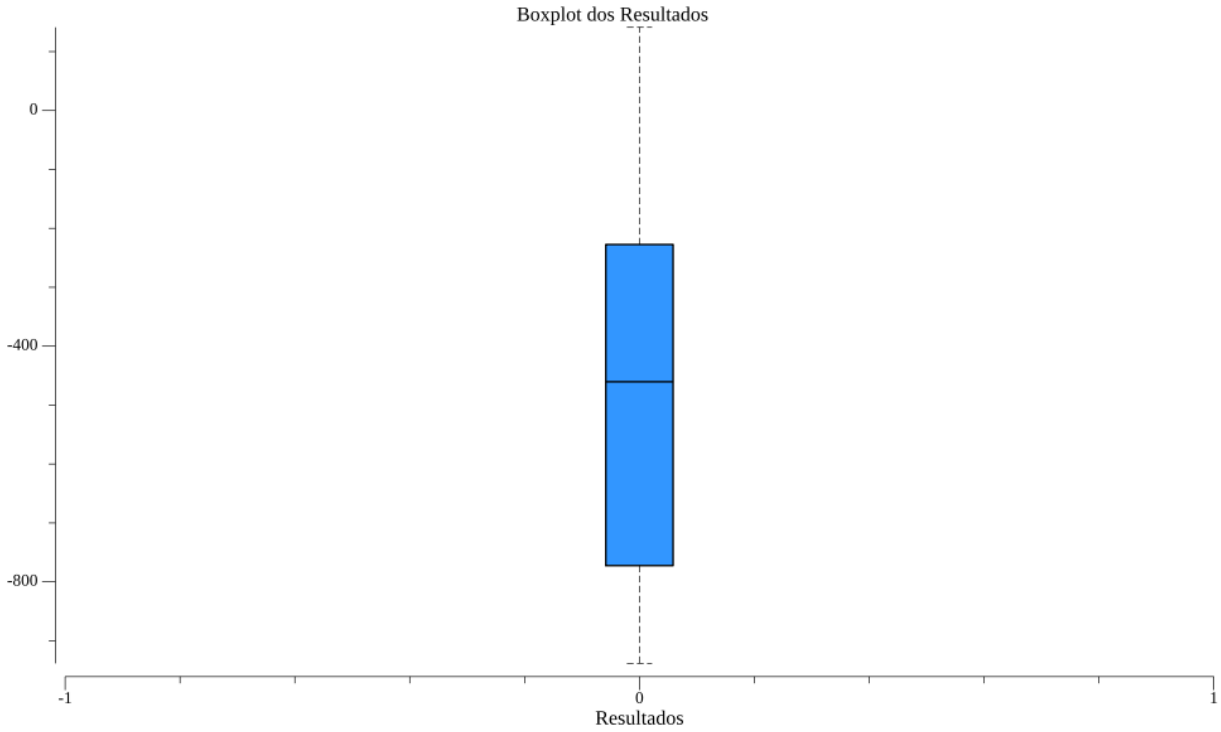

Figure 14: 2 - D - HC

5 Conclusão

Com base no desenvolvimento e análise dos algoritmos de Hill Climbing e ILS, podemos concluir que essas meta-heurísticas são ferramentas poderosas para resolver problemas de otimização.

O Hill Climbing é uma heurística simples, mas eficaz, baseada no princípio de "subir colinas" para encontrar melhores soluções. Trabalhe de forma iterativa e tome medidas que levem à melhoria local imediata. No entanto, essa abordagem local pode fazer com que a escalada fique atolada em ótimos locais e não alcance os ótimos globais.

ILS é uma metaheurística que combina busca local e perturbação aleatória para evitar otimização local subótima. Isso consiste em gerar a solução inicial, executar uma pesquisa local nessa solução, suspender a solução e executar novamente a pesquisa local. Este processo é repetido para o número de iterações especificado. O ILS pode explorar diferentes regiões do espaço de busca, o que o torna mais robusto do que o montanhismo.

Ambos os algoritmos dependem da definição correta de parâmetros como número de iterações, tamanho do passo e coeficientes de perturbação para obter melhores resultados. Além disso, a seleção de funções de pontuação apropriadas, destruição e estratégias de busca local também desempenham papéis importantes no desempenho e eficácia dessas meta-heurísticas. Para o propósito deste relatório, implementamos e usamos essas meta-heurísticas em Golang para resolver um problema de otimização específico. Ao analisar o código e os resultados obtidos, pudemos observar o comportamento e a eficiência desses algoritmos em encontrar melhores soluções.

Em resumo, Hill Climbing e ILS são abordagens valiosas para resolver problemas de otimização, mas cada uma tem suas próprias características e limitações. A escolha da meta-heurística mais adequada depende da natureza do problema e das limitações que ele impõe. Com implementação adequada e ajuste de parâmetros, essas técnicas podem fornecer soluções satisfatórias para uma ampla variedade de problemas do mundo real.

References

[1] Ils. http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/ILS.pdf. (Accessed on 17/05/2023).

[2] hc. https://iao.hfuu.edu.cn/images/teaching/lectures/metaheuristic_optimization/05_hill_climbing.pdf. (Accessed on 17/05/2023).

[3] Ils. https://go.dev/. (Accessed on 17/05/2023).