Trabalho Computacional 3: Resolução de problemas por GAs

Aluisio Gaspar - 2013836, Clysman Alves - 2220304

***Abstract*—This work focuses on solving various problems through meta-heuristic search and optimization methods. Meta-heuristic algorithms are powerful tools for addressing complex problems by utilizing heuristic strategies to explore the state space and find the best possible solutions. The approaches developed in this study are versatile and can be easily adapted to handle different types of objectives and constraints across various domains. The study is divided into three parts:**

- **The first part employs algorithms such as Hill Climbing, Local Random Search (LRS), and Global Random Search (GRS) to solve continuous problems with infinite state spaces.**
- **The second part explores the Simulated Annealing technique for solving a discrete problem.**
- **The third part involves the application of a genetic algorithm developed by our team to address a discrete domain problem, specifically focusing on the Traveling Salesman Problem (TSP).**

These methods demonstrate the effectiveness and adaptability of meta-heuristic algorithms in solving diverse and complex optimization problems.

***Index Terms*—Meta-heuristics, Optimization, Hill Climbing, Local Random Search, Global Random Search, Simulated Annealing, Genetic Algorithms, Traveling Salesman Problem (TSP), Continuous Problems, Discrete Problems.xx**

I. Introdução

NAs últimas décadas, a busca por soluções eficientes para problemas complexos de otimização tem sido um campo de intenso estudo e inovação, especialmente na área de Inteligência Artificial (IA). Problemas de otimização surgem em uma vasta gama de aplicações, desde o planejamento de rotas logísticas até o design de sistemas bioinformáticos, exigindo métodos robustos e adaptáveis para identificar soluções ótimas ou quase ótimas em um espaço de estados frequentemente vasto e intrincado.

Uma abordagem notavelmente eficaz para esses desafios é a utilização de algoritmos meta-heurísticos. Meta-heurísticas são estratégias de alto nível projetadas para guiar processos heurísticos de busca por soluções satisfatórias, explorando o espaço de estados de forma eficiente e eficaz. Diferentemente das técnicas tradicionais de otimização, as meta-heurísticas não dependem fortemente das características específicas do problema, permitindo sua aplicação em uma ampla variedade de contextos com diferentes restrições e objetivos.

Este trabalho focaliza a resolução de problemas complexos de busca e otimização através de diversas técnicas meta-heurísticas. Inicialmente, serão abordados problemas contínuos com espaço de estados infinito utilizando os algoritmos Hill Climbing, Local Random Search (LRS) e Global Random Search (GRS). Estes métodos exploram o espaço de soluções de maneiras distintas, oferecendo diferentes vantagens em termos de velocidade e eficiência na convergência para soluções ótimas.

A segunda parte do trabalho explora a técnica de Simulated Annealing aplicada a problemas discretos. O Simulated Annealing é conhecido por sua capacidade de evitar mínimos locais, permitindo a busca de soluções em um espaço de estados complexo através de um processo inspirado no recozimento de metais.

Finalmente, a terceira parte dedica-se à aplicação de algoritmos genéticos (AGs) no contexto de problemas discretos, especificamente adaptados para resolver o problema do caixeiro viajante (TSP). Os AGs, inspirados pelos princípios da evolução biológica, combinam mecanismos de seleção, crossover e mutação para evoluir uma população de soluções candidatas, buscando iterativamente melhorias que convergem para uma solução ótima ou quase ótima.

Cada uma dessas técnicas será detalhadamente explorada e aplicada em cenários específicos, demonstrando sua eficácia e adaptabilidade em enfrentar problemas de otimização complexos. Este trabalho, portanto, não só investiga a aplicabilidade dessas meta-heurísticas em diferentes domínios de problemas, mas também ilustra a flexibilidade e o potencial dos algoritmos genéticos em particular, na resolução de desafios de otimização em espaços discretos.

II. Parte 1 - Problema de Minimização/Maximização de função Custo/Objetivo

Para a implementação do projeto de algoritmos de otimização, foi necessário seguir uma abordagem estruturada, integrando teoria e prática. Abaixo está um resumo detalhado de como cada parte foi implementada, incluindo a configuração inicial, a execução dos algoritmos e a análise dos resultados.

A. Configuração Inicial

Primeiramente, foi essencial definir o problema de otimização de forma clara:

- **Função Objetivo** ($f(x)$): Representa a quantidade que desejamos minimizar ou maximizar. A função é definida matematicamente e avaliada com base nas variáveis de entrada.
- **Variáveis Independentes** (x): Conjunto de variáveis desconhecidas que afetam o valor da função objetivo.

Cada variável possui um domínio específico dentro do qual ela pode variar.
- **Restrições**: Limites impostos sobre as variáveis independentes. No caso de restrições do tipo caixa, cada variável x_j está restrita a um intervalo definido.

Para problemas sem restrições complexas, a fórmula geral utilizada foi:

$$\text{minimize } f(x),\ x = (x_1, x_2, \ldots, x_p)$$

$$\text{subject to } x_j \in \text{dom}(x_j)$$

B. Algoritmo de Hill Climbing

O algoritmo de Hill Climbing (Subida de Encosta) é uma técnica de otimização que busca encontrar a melhor solução para um problema ao fazer pequenas mudanças iterativas na solução atual e aceitar as mudanças que melhoram a solução. Ele é frequentemente utilizado em problemas de otimização onde o espaço de busca é grande e não há uma maneira direta de encontrar a solução ótima.

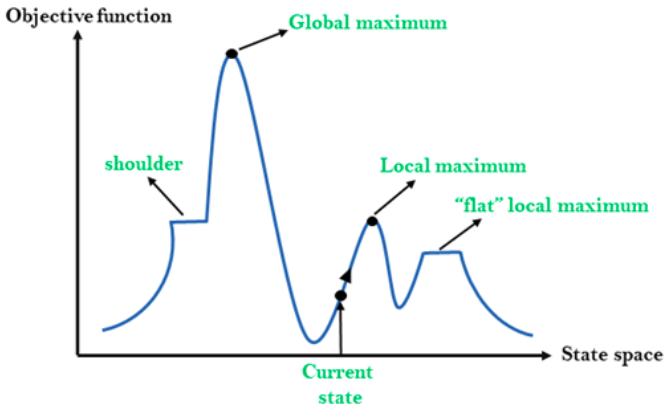

Fig. 1: Algoritmo da Subida de Encosta. Fonte: https://www.javatpoint.com/hill-climbing-algorithm-in-ai

Alguns passos para implementar o algoritmo são:

1) **Inicialização**: Começa com uma solução inicial, que pode ser escolhida aleatoriamente ou com base em algum critério específico.
2) **Avaliação**: Avalia a qualidade (fitness) da solução atual utilizando uma função de avaliação.
3) **Movimento**: Gera uma nova solução ao fazer uma pequena alteração na solução atual. Esta alteração é chamada de ”vizinho”.
4) **Decisão**: Compara a nova solução com a solução atual. Se a nova solução for melhor, ela substitui a solução atual. Caso contrário, a solução atual é mantida.
5) **Iteração**: Repete os passos de avaliação, movimento e decisão até que um critério de parada seja atingido, como um número máximo de iterações ou uma melhoria mínima aceitável na solução.

Algorithm 1 Pseudocódigo busca por subida de encosta

Inicializar o ponto inicial (zero ou limite de domínio) x_0
Definir o valor ϵ para candidato vizinho
Definir uma quantidade máxima de iterações $maxit$ e quantidade máxima de candidatos (possíveis vizinhos) $maxn$
Melhor valor $x_{best} \leftarrow x_0$ e melhor valor computado $f_{best} \leftarrow f(x_{best})$
$i \leftarrow 0$
while $i < maxit$ **and** houver melhoria **do**
  $j \leftarrow 0$
  melhoria $\leftarrow$ false
  while $j < maxn$ **do**
    $y \leftarrow$ candidato(x_{best})
    $F \leftarrow f(y)$
    if $F > f_{best}$ **then**
      $x_{best} \leftarrow y$
      $f_{best} \leftarrow F$
      melhoria $\leftarrow$ true
    end if
  end while
  $i \leftarrow i + 1$
end while=0

C. Algorito de Busca Aleatoria Local

O algoritmo de Busca Aleatória Local é uma técnica heurística para resolver problemas de otimização. Este algoritmo tenta encontrar uma solução aceitável, mas não necessariamente ótima, dentro de um espaço de busca grande.

Alguns passos para implementar o algoritmo são:

- **Inicialização**: Começa-se com uma solução inicial, que pode ser escolhida de forma aleatória ou baseada em algum critério específico.
- **Vizinhança**: Define-se um conjunto de soluções vizinhas à solução atual. As soluções vizinhas são aquelas que podem ser obtidas a partir da solução atual através de pequenas mudanças.
- **Seleção Aleatória**: Uma solução vizinha é selecionada aleatoriamente a partir do conjunto de vizinhança.
- **Avaliação**: A solução vizinha selecionada é avaliada de acordo com a função objetivo.
- **Atualização**: Se a solução vizinha for melhor que a solução atual (segundo a função objetivo), ela se torna a nova solução atual.
- **Iteração**: O processo é repetido até que um critério de parada seja atingido (por exemplo, um número máximo de iterações ou um tempo limite).

Algorithm 2 Busca Aleatória Local

Definir uma quantidade máxima de iterações N_{max}.
Definir x_l e x_u.
Definir valor de σ (perturbação aleatória).
$x_{\text{best}} \sim U(x_l, x_u)$
$f_{\text{best}} = f(x_{\text{best}})$
$i \leftarrow 0$
while $i < N_{\text{max}}$ **do**
  $n \sim N(0, \sigma)$
  $x_{\text{cand}} \leftarrow x_{\text{best}} + n$
  Verificar a violação da restrição em caixa.
  $f_{\text{cand}} = f(x_{\text{cand}})$
  if $f_{\text{cand}} > f_{\text{best}}$ **then**
    $x_{\text{best}} = x_{\text{cand}}$
    $f_{\text{best}} = f_{\text{cand}}$
  end if
  $i \leftarrow i + 1$
end while
FIM =0

D. Implementação dos Algoritmos de Otimização

Foram implementados os três algoritmos falados anteriomente de busca:

- Hill Climbing (Subida de Encosta)
  - **Ponto Inicial**: Definido como o valor do limite inferior do domínio de x.
  - **Geração de Candidatos**: Utilizou-se a vizinhança de x_{best} com $|x_{\text{best}} - y| \leq \epsilon$, onde y é um candidato vizinho e ϵ foi ajustado para um valor pequeno, como 0,1.
  - **Parada**: O algoritmo para quando não há melhoria após t iterações ou quando o número máximo de iterações (1000) é alcançado.
- Local Random Search (Busca Local Aleatória - LRS)
  - **Ponto Inicial**: Gerado aleatoriamente dentro dos limites definidos pelo domínio das variáveis independentes, utilizando uma distribuição uniforme.
  - **Desvio-Padrão** (σ): O valor σ foi ajustado para encontrar a solução ótima, variando dentro do intervalo (0, 1).
  - **Geração de Candidatos**: Novos candidatos foram gerados aleatoriamente usando σ.
  - **Parada**: Critérios semelhantes ao Hill Climbing, com possibilidade de parar antecipadamente se não houver melhorias.
- Global Random Search (Busca Aleatória Global - GRS)
  - **Ponto Inicial**: Assim como no LRS, foi gerado aleatoriamente dentro dos limites das variáveis.
  - **Geração de Candidatos**: Novos candidatos foram gerados aleatoriamente, sem a necessidade de considerar uma vizinhança específica.
  - **Desvio-Padrão** (σ): Ajustado para encontrar a solução ótima, semelhante ao LRS.
  - **Parada**: Usou os mesmos critérios dos algoritmos anteriores.

E. Execução dos Algoritmos e Coleta de Dados

Cada algoritmo foi executado por 100 rodadas, armazenando a solução obtida em cada uma delas. É importante notar que essas rodadas são distintas das iterações internas de cada algoritmo.

- **Rodadas**: Cada rodada consiste em uma execução completa do algoritmo, até atingir um critério de parada.
- **Iterações**: Referem-se aos passos internos que o algoritmo realiza para procurar a solução ótima em cada rodada.

F. Análise dos Resultados

Ao final das 100 rodadas, as soluções foram analisadas para determinar a moda das soluções de cada algoritmo. Esta moda representa a solução mais frequentemente obtida em todas as rodadas, dando uma medida de robustez do algoritmo.

- **Moda das Soluções**: Utilizada para identificar a solução mais comum em todas as execuções do algoritmo, o que pode indicar a sua eficácia em encontrar soluções ótimas ou próximas do ótimo.
- **Hiperparâmetros**: Foram ajustados para cada algoritmo visando a melhoria da performance, como o ϵ para Hill Climbing e σ para LRS e GRS.

G. Teste de Restrições e Limites

Foi fundamental garantir que cada candidato gerado estivesse dentro dos limites impostos pelas variáveis independentes. Esta verificação foi realizada antes de considerar uma nova solução como válida.

H. Ajuste dos Hiperparâmetros

Cada algoritmo foi ajustado para encontrar o valor ótimo de seus respectivos hiperparâmetros:

- ϵ para Hill Climbing.
- σ para LRS e GRS.

O processo de ajuste envolveu a execução repetida dos algoritmos com diferentes valores de hiperparâmetros para identificar aquele que gerava a melhor solução mais frequentemente.

III. Parte 2 - Problema das 8 rainhas

O problema das 8 rainhas é um clássico problema de xadrez e programação que envolve posicionar 8 rainhas em um tabuleiro de xadrez 8×8 de forma que nenhuma delas possa atacar as outras. Este problema foi proposto pela primeira vez por Max Bezzel em 1848 e sua primeira solução foi publicada por Franz Nack em 1850. Com o tempo, o problema foi estendido para a colocação de n rainhas em um tabuleiro $n \times n$.

A complexidade do problema pode ser entendida ao considerar que existem $4.426.165.368$ arranjos possíveis para posicionar as 8 rainhas em um tabuleiro 8×8. No entanto, uma abordagem inicial que reduz essa complexidade é posicionar uma rainha por coluna, o que diminui os arranjos possíveis para $16.777.216$. Apesar dessa redução, o problema ainda possui 92 soluções distintas.

Para resolver o problema das 8 rainhas, uma das técnicas que pode ser utilizada é a Têmpera Simulada (*Simulated Annealing*). A Têmpera Simulada é um algoritmo de otimização probabilística que se inspira no processo de recozimento metalúrgico. Este processo envolve aquecer e resfriar lentamente um material para reduzir defeitos, visando encontrar uma configuração de energia mínima.

A. Detalhes do Problema e Solução

O tabuleiro de xadrez é enumerado de forma que as colunas são numeradas crescentemente da esquerda para a direita (1 a 8), e as linhas são numeradas decrescentemente de cima para baixo (1 a 8). Para simplificar, cada rainha é posicionada em uma coluna única. Portanto, uma solução candidata pode ser representada por um vetor onde cada índice representa uma coluna e o valor no vetor representa a linha onde a rainha está posicionada.

Por exemplo, para o vetor de solução $x = [5, 1, 4, 2, 6, 1, 4, 7]$, a primeira rainha (coluna 1) está na linha 5 e a última rainha (coluna 8) está na linha 7.

A função de aptidão deve avaliar a qualidade de cada solução. Uma maneira de fazer isso é contando o número de pares de rainhas que se atacam. Como há 28 pares de rainhas possíveis, uma função de aptidão interessante poderia ser definida como:

$$f(x) = 28 - h(x)$$

onde $h(x)$ é o número de pares de rainhas que estão se atacando na solução x. Este problema pode ser visto como um problema de maximização da função $f(x)$, pois desejamos maximizar o número de pares de rainhas que não se atacam.

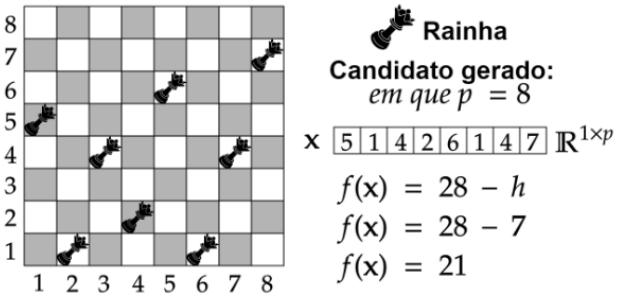

Fig. 2: Problema das 8-rainhas.

B. Têmpera Simulada

1) **Inicialização:** Começamos com uma solução inicial aleatória e uma temperatura inicial alta.
2) **Perturbação:** Geramos uma nova solução vizinha perturbando a solução atual (por exemplo, movendo uma rainha para uma linha diferente na mesma coluna).
3) **Avaliação:** Calculamos a função de aptidão da nova solução.
4) **Aceitação:** Decidimos se aceitamos a nova solução com base em um critério probabilístico que depende da diferença de aptidão entre as soluções e da temperatura atual. Soluções piores podem ser aceitas com uma certa probabilidade para evitar mínimos locais.
5) **Resfriamento:** Reduzimos a temperatura gradualmente.
6) **Parada:** O processo continua até que a temperatura atinja um valor mínimo ou um número máximo de iterações seja alcançado.

IV. Parte 3 - Problema do drone viajante

O problema do caixeiro viajante é um clássico da otimização combinatória que envolve encontrar a rota mais curta para visitar uma lista de cidades e retornar ao ponto de origem. Este problema torna-se exponencialmente complexo à medida que o número de cidades aumenta, tornando inviável a busca por força bruta. Para solucionar este problema, aplicamos um algoritmo genético (AG), uma técnica inspirada nos princípios da evolução natural, que se mostra eficaz na busca de soluções aproximadas para problemas complexos.

A. Detalhes do Problema e Solução

A implementação do algoritmo genético para resolver o problema do caixeiro viajante foi dividida em várias etapas principais:

1) **Inicialização da População**:
   - Cada indivíduo na população representa uma rota possível que o drone pode seguir.
   - A população inicial é gerada de forma aleatória, garantindo a diversidade genética necessária para a evolução.
2) **Representação Cromossômica**:
   - Cada indivíduo (rota) é representado por um cromossomo, que é uma sequência de genes.
   - No contexto deste trabalho, cada gene corresponde a um ponto a ser visitado, totalizando 100 genes por cromossomo (já que o ponto de origem é fixo).
3) **Função de Aptidão**:
   - A função de aptidão avalia a qualidade de cada rota.
   - Calcula-se a distância total percorrida pelo drone ao seguir a sequência de pontos definida pelo cromossomo, incluindo o retorno ao ponto de origem.
   - A rota com menor distância total possui a melhor aptidão.
4) **Operadores Genéticos**:
   - **Seleção**: Utilizamos a seleção por torneio, onde um conjunto de indivíduos é escolhido aleatoriamente e o mais apto dentro deste grupo é selecionado para a reprodução.
   - **Cruzamento (Crossover)**: Implementamos o crossover de ordem (Order Crossover - OX), que preserva a ordem relativa dos genes, essencial para manter a viabilidade das rotas.
   - **Mutação**: Utilizamos a mutação por inversão de subsequência, onde um segmento da rota é invertido, introduzindo variações que ajudam a explorar o espaço de busca de maneira mais eficaz.
5) **Critério de Parada**:

- O algoritmo é executado até alcançar um número máximo de gerações ou até que a variação na aptidão da população atinja um limite estabelecido, indicando convergência.

V. Resultados - Parte 1

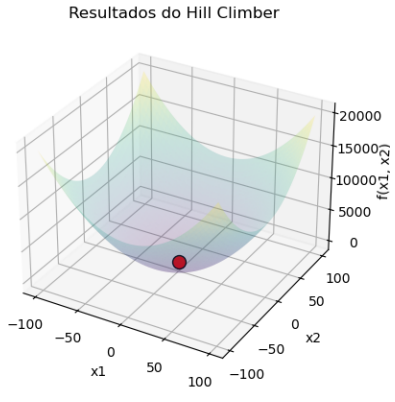

Fig. 3: Função 1 - Hill Climbing

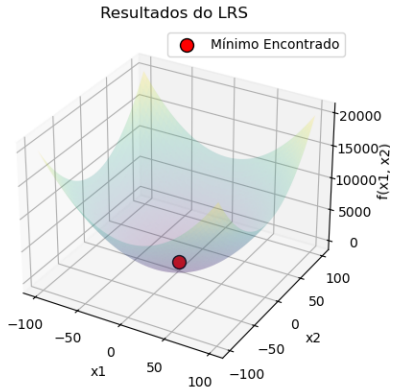

Fig. 4: Função 1 - LRS

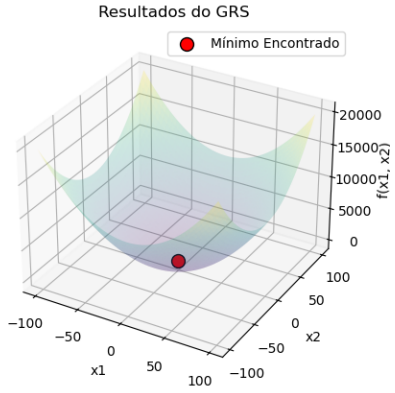

Fig. 5: Função 1 - GRS

	HC	LRS	GRS
count	100.000000	100.000000	100.000000
mean	-1.712088	-1.139863	0.073757
std	34.405488	32.001673	2.408639
min	-66.890277	-62.746126	-5.531387
25%	-25.910543	-22.513112	-1.589050
50%	-0.007173	-0.006645	-0.123972
75%	20.481084	18.737764	1.484487
max	65.279317	70.504034	6.664482

TABLE I: Resumo estatístico da função 1

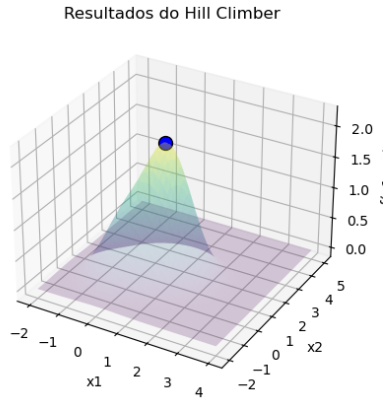

Fig. 6: Função 2 - Hill Climbing

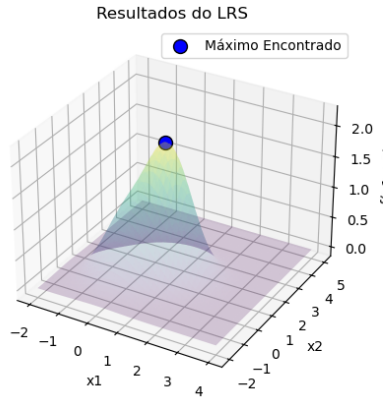

Fig. 7: Função 2 - LRS

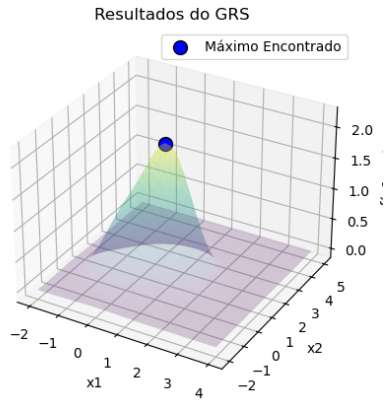

Fig. 8: Função 2 - GRS

	HC	LRS	GRS
count	100.000000	100.000000	100.000000
mean	0.912315	0.911890	0.916269
std	0.003499	0.003706	0.073459
min	0.903640	0.902769	0.714903
25%	0.909777	0.909541	0.870185
50%	0.912667	0.911457	0.909555
75%	0.915059	0.914368	0.966293
max	0.922459	0.921474	1.120682

TABLE II: Resumo estatístico da função 2

	HC	LRS	GRS
count	100.000000	100.000000	100.000000
mean	0.242067	-0.249347	-0.011577
std	4.836996	4.671267	0.294835
min	-7.996026	-7.991233	-1.127832
25%	-3.972978	-4.229907	-0.140514
50%	-0.001037	-0.477327	-0.026685
75%	4.986182	3.970339	0.122792
max	7.986774	8.000000	1.043196

TABLE III: Resumo estatístico da função 3

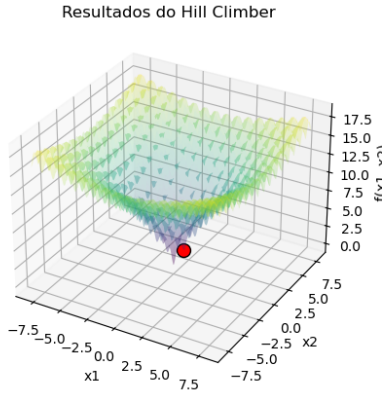

Fig. 9: Função 3 - Hill Climbing

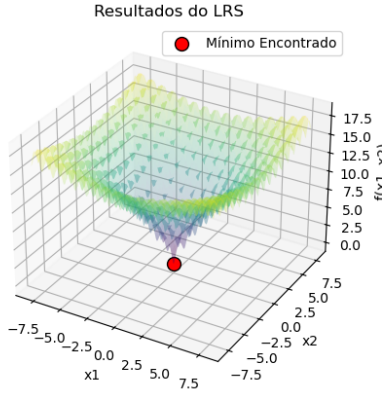

Fig. 10: Função 3 - LRS

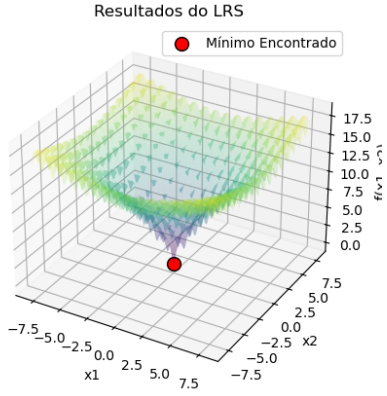

Fig. 11: Função 3 - GRS

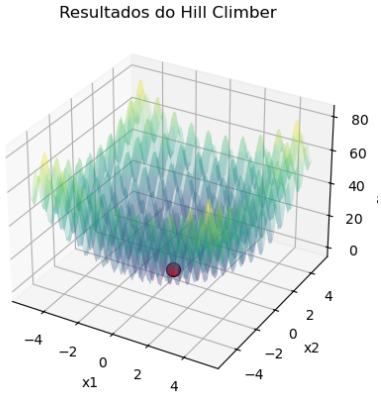

Fig. 12: Função 4 - Hill Climbing

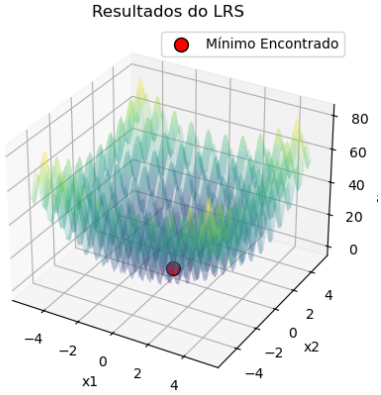

Fig. 13: Funçõa 4 - LRS

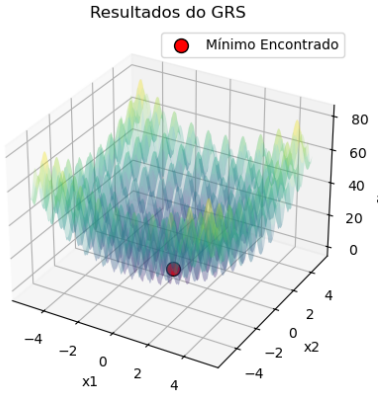

Fig. 14: Função 4 - GRS

VI. Resultados - Parte 2

O algoritmo para o problema das 8-Damas acha as 92 soluções possiveis, gerando o tabuleiro e a temperatura.

	HC	LRS	GRS
count	100.000000	100.000000	100.000000
mean	0.875087	-0.567006	0.110222
std	2.661805	2.997430	0.735101
min	-4.979786	-4.980233	-1.075841
25%	-0.994278	-2.985884	-0.058988
50%	0.993033	-0.994338	0.022988
75%	2.985138	1.991398	0.942751
max	4.978446	4.980057	1.984750

TABLE IV: Resumo estatístico da função 4

	HC	LRS	GRS
count	100.000000	100.000000	100.000000
mean	-0.278365	-1.036821	-0.240886
std	6.437989	6.297326	1.688002
min	-9.734761	-9.734235	-9.879996
25%	-3.910597	-3.911033	-0.089795
50%	-3.909389	-3.910061	0.032321
75%	6.728888	6.728952	0.190950
max	10.000000	10.000000	0.535522

TABLE V: Resumo estatístico da função 5

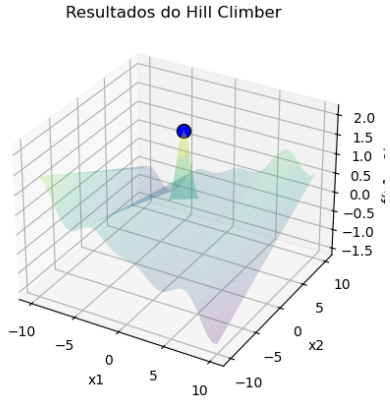

Fig. 15: Função 5 - Hill Climbing

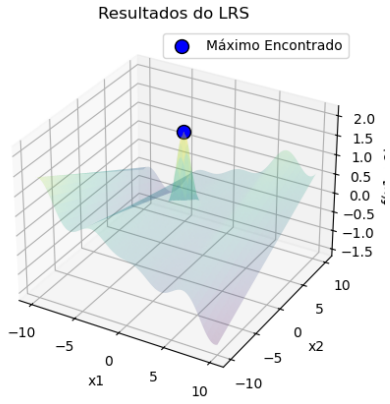

Fig. 16: Função 5 - LRS

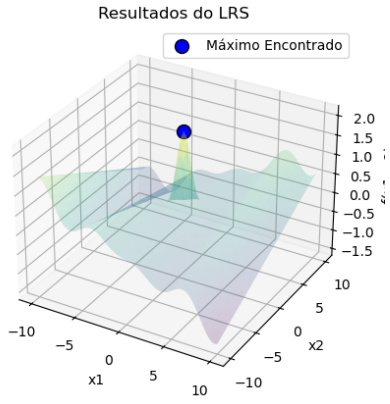

Fig. 17: Função 5 - GRS

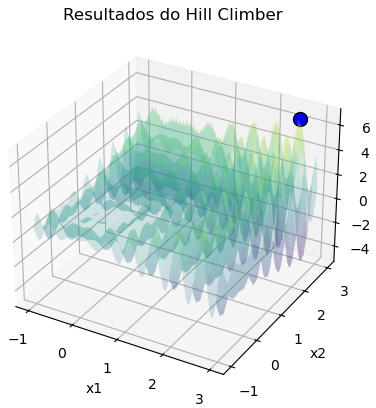

Fig. 18: Função 6 - Hill Climbing

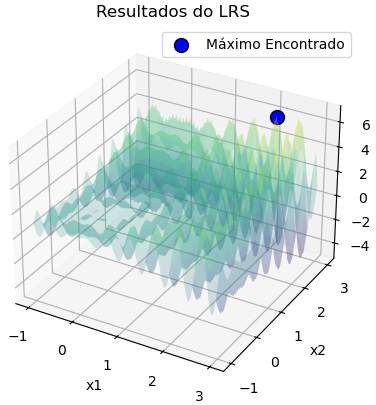

Fig. 19: Função 6 - LRS

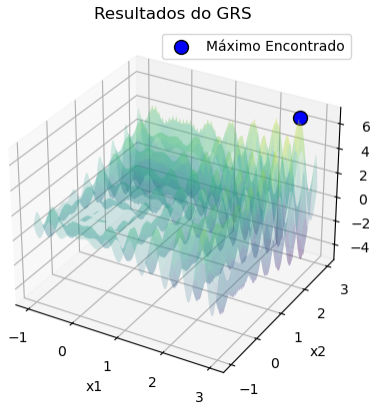

Fig. 20: Função 6 - GRS

	HC	LRS	GRS
count	100.000000	100.000000	100.000000
mean	1.164270	1.064394	2.536763
std	1.192536	1.174089	0.209265
min	-0.640290	-0.639269	1.592508
25%	0.156942	-0.156655	2.592441
50%	1.135866	1.130019	2.616248
75%	2.129966	2.127738	2.644000
max	2.631931	2.634599	2.702955

TABLE VI: Resumo estatístico da função 6

	HC	LRS	GRS
count	100.000000	100.000000	100.000000
mean	1.661706	1.671624	2.198871
std	0.811101	0.841812	0.054616
min	0.000000	0.000000	2.021887
25%	0.821031	0.752026	2.157495
50%	2.200964	2.201390	2.198890
75%	2.203007	2.204947	2.233210
max	3.085556	3.086996	2.331871

TABLE VII: Resumo estatístico da função 7

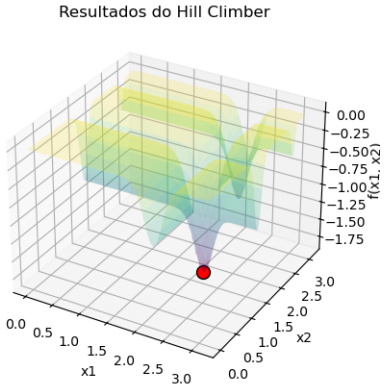

Fig. 21: Função 7 - Hill Climbing

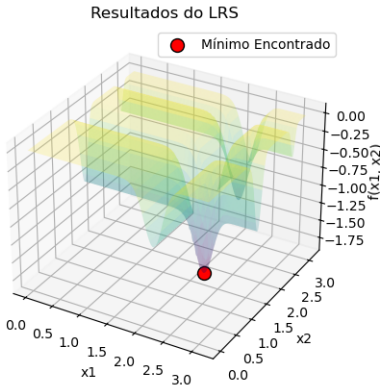

Fig. 22: Função 7 - LRS

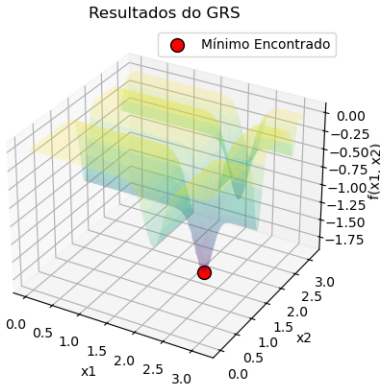

Fig. 23: Função 7 - GRS

VII. Resultados - Parte 3

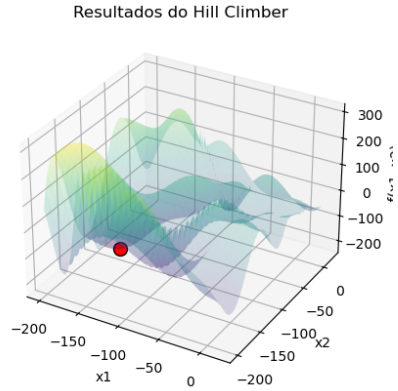

Fig. 24: Função 8 - Hill Climbing

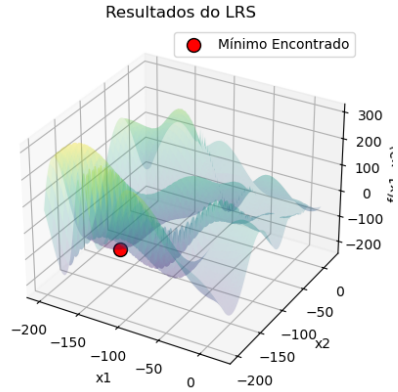

Fig. 25: Função 8 - LRS

VIII. Implementações

Disponivel em: otimization-and-genetical-algorithms

References

[1] M. C. Goldbarg and E. F. G. Goldbarg, *Otimização Combinatória e Metaheurísticas: Algoritmos e Aplicações*, 1st ed. Campus/Elsevier, 2012.

[2] D. Kopec, *Classic Computer Science Problems in Python*. Manning Publications, 2019.

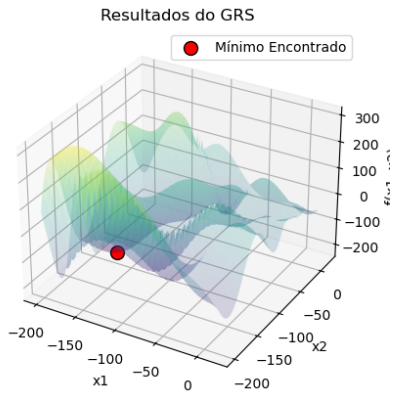

Fig. 26: Função 8 - GRS

	HC	LRS	GRS
count	100.000000	100.000000	100.000000
mean	-97.036336	-97.251770	-170.588269
std	61.530315	66.783901	3.260976
min	-200.000000	-200.000000	-179.680100
25%	-147.178202	-165.112658	-172.835994
50%	-101.291945	-96.842542	-170.469952
75%	-48.930597	-33.379658	-168.037874
max	20.000000	8.467740	-163.502106

TABLE VIII: Resumo estatístico da função 8

```
Nova solução encontrada: [3, 6, 8, 2, 4, 1, 7, 5]
```

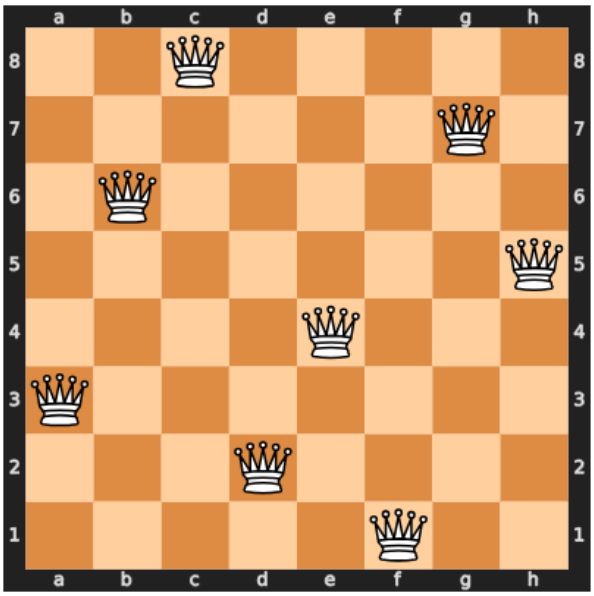

Fig. 27: Exemplo de resultado encontrado para o problema das 8-Damas

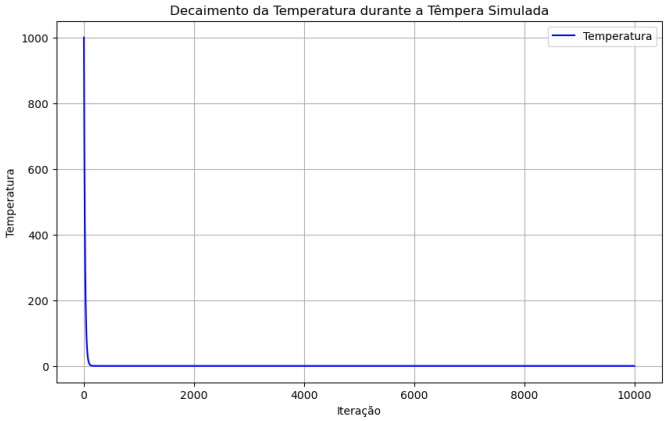

Fig. 28: Resultados tempera simulada

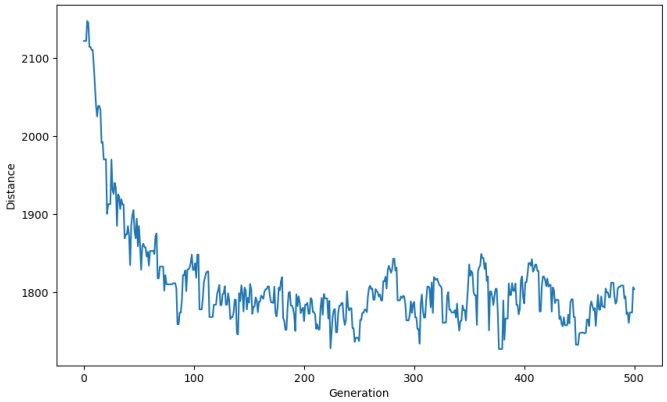

Fig. 29: Relação entre a distância e a geração

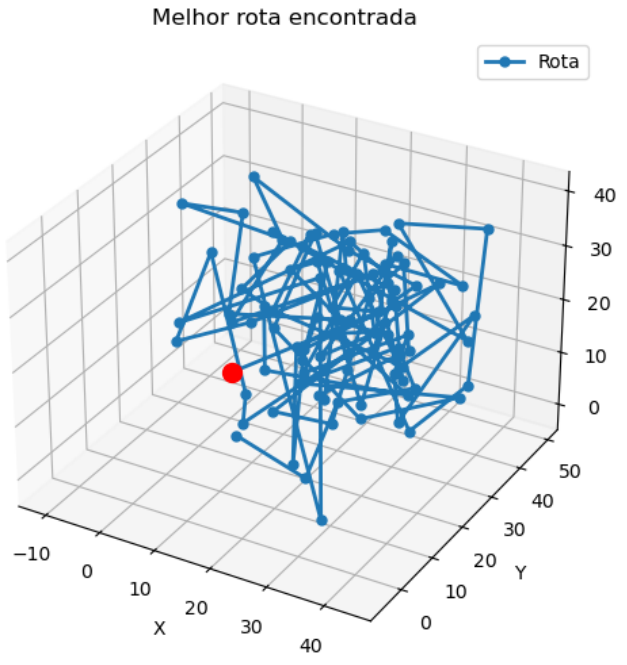

Fig. 30: Melhor rota encontrada