

Trabalho Computacional AV3: Busca/Otimização Meta-heurística

Marcos Antonio Felix - 1810449

Gil Melo Bandeira Torres - 1720537

Disciplina: Inteligência Artificial
Computacional

Professor: Paulo Cirillo Souza Barbosa

Resumo: Este relatório aborda a aplicação de técnicas de busca e otimização meta-heurística, que consistem em métodos computacionais inspirados em fenômenos naturais, físicos ou sociais, capazes de encontrar soluções aproximadas para problemas complexos onde métodos exatos são inviáveis. A Etapa 1 do trabalho concentrou-se na resolução de problemas contínuos envolvendo oito funções matemáticas bidimensionais, utilizando os algoritmos Hill Climbing (HC), Local Random Search (LRS) e Global Random Search (GRS), com execução de 100 rodadas por função, análise da moda das soluções e visualização gráfica em 3D. A Etapa 2 abordou um problema discreto clássico — o Problema das 8 Rainhas — utilizando o algoritmo de Têmpera Simulada (Simulated Annealing), com o objetivo de encontrar todas as 92 soluções únicas possíveis, representando visualmente os tabuleiros e o decaimento da temperatura ao longo das iterações. Este estudo reforça a eficácia das meta-heurísticas na resolução de problemas tanto contínuos quanto combinatórios.

Palavras-chave: Meta-heurística, Têmpera Simulada, Hill Climbing, Problema das 8 Rainhas, Busca Local Aleatória, Busca Aleatória Global.

I. INTRODUÇÃO

Com o avanço da Ciência da Computação e a crescente complexidade dos problemas enfrentados por sistemas inteligentes, tornou-se essencial o desenvolvimento de métodos capazes de encontrar boas soluções sem a necessidade de explorar exaustivamente todos os caminhos possíveis. Foi nesse cenário que surgiram os algoritmos de busca e otimização **meta-heurística** — estratégias inspiradas em processos naturais, físicos ou sociais, como a evolução biológica, o comportamento de enxames e o resfriamento térmico de materiais. Essas técnicas se tornaram fundamentais para resolver problemas onde métodos exatos são inviáveis devido ao custo computacional ou à própria estrutura da função objetivo.

Problemas de otimização podem ser classificados de diversas formas: quanto ao tipo de variável (contínuos ou

discretos), à quantidade de variáveis (univariados ou multivariados) e à topologia da função (unimodais ou multimodais). No contexto deste trabalho, abordamos dois grandes grupos: os problemas contínuos, nos quais as variáveis assumem valores reais dentro de um domínio bidimensional, e os problemas discretos, onde as soluções envolvem combinações inteiras em um espaço finito.

Na **Etapa 1**, foram analisadas oito funções matemáticas clássicas de otimização contínua, representando diferentes desafios — desde superfícies simples e convexas até funções altamente multimodais com muitos ótimos locais. Para cada função, aplicamos os algoritmos **Hill Climbing (HC)**, **Busca Aleatória Local (LRS)** e **Busca Aleatória Global (GRS)**, realizando 100 rodadas independentes por técnica e coletando métricas como a moda das soluções, além de visualizações tridimensionais da paisagem de busca com os pontos encontrados.

Na **Etapa 2**, o foco se voltou para um problema clássico da Inteligência Artificial: o **Problema das 8 Rainhas**, um desafio discreto de maximização em que o objetivo é posicionar 8 rainhas em um tabuleiro de xadrez de forma que nenhuma se ataque mutuamente. Para resolvê-lo, utilizamos o algoritmo de **Têmpera Simulada (Simulated Annealing)**, que incorpora o conceito de aceitação probabilística de soluções piores para escapar de ótimos locais, inspirado no processo de resfriamento de metais. Além de encontrar uma solução válida, buscamos identificar todas as 92 configurações únicas possíveis, analisando também o número de execuções necessárias e o comportamento da temperatura ao longo das iterações..

II. ETAPA I: PROBLEMA DE MINIMIZAÇÃO/MAXIMIZAÇÃO DE FUNÇÃO CUSTO/OBJETIVO

A. Análise da problemática

Antes de qualquer abordagem algorítmica, é fundamental compreender a formulação matemática do problema de otimização. Nesta etapa, o primeiro passo consistiu na definição de uma função objetivo — uma função matemática que representa a quantidade a ser otimizada

(minimizada ou maximizada), estando diretamente ligada ao desempenho do sistema modelado. Essa função depende de um conjunto de variáveis de decisão e pode estar sujeita a um conjunto de restrições que delimitam a região viável do espaço de busca.

A formulação geral adotada segue a estrutura:

$$f(x) = f(x_1, x_2)$$

onde $x = (x_1, x_2)$, $\in R^2$, sendo x_1 e x_2 variáveis contínuas definidas dentro de um domínio específico, como por exemplo $[-5, 5]$ ou $[0, \pi]$, dependendo da função utilizada. A função $f(x)$ pode representar diferentes tipos de superfícies, como parábolas convexas, funções oscilatórias, ou expressões compostas por exponenciais, produtos e cossenos. Cada função representa um modelo de cenário real ou abstrato onde há um valor ótimo (mínimo ou máximo) a ser encontrado.

Durante o desenvolvimento do trabalho, foram escolhidas oito funções com características distintas, contemplando casos unimodais (com único ótimo global), multimodais (com vários ótimos locais), além de diferentes perfis topológicos, variando de suaves a altamente irregulares. Essa diversidade é crucial para analisar como os algoritmos se comportam em diferentes contextos de complexidade.

A principal dificuldade associada à resolução analítica de tais problemas é a inexistência de derivadas conhecidas, presença de muitos ótimos locais ou regiões planas. Por isso, optamos pelo uso de algoritmos de busca baseados em meta-heurísticas, como HC, LRS e GRS, que não requerem conhecimento analítico da função e são mais robustos frente à irregularidade do espaço de busca.

B. Algoritmos Implementados

Hill Climbing (subida de encosta): É uma técnica de busca local iterativa que visa otimizar uma função objetivo $f(x)$ a partir de pequenas perturbações sucessivas em uma solução inicial. Ele assume que uma boa solução pode ser encontrada caminhando na direção de melhoria local, aceitando apenas vizinhos que proporcionem um valor melhor da função. A estratégia é eficaz quando aplicada a funções unimodais, mas apresenta limitações consideráveis em ambientes multimodais.

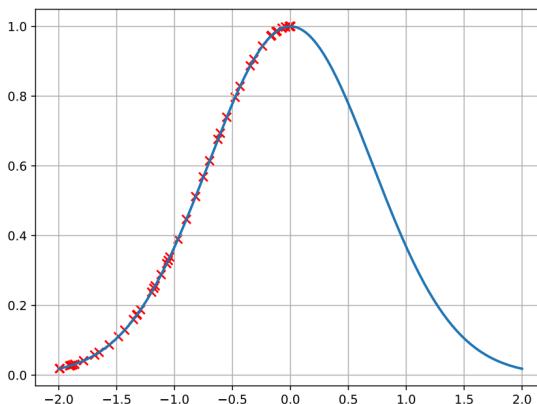


Imagen 1: O gráfico da função com caminho de execução.

Na Imagem 1, vemos a trajetória do algoritmo (pontos vermelhos) ao longo da função $f(x) = \sin(3x) + 0.5\sin(5x)$. O algoritmo inicia em um ponto arbitrário e progride até atingir um ótimo local, onde permanece por não encontrar vizinhos melhores — evidenciando sua incapacidade de escapar de picos secundários.

Esse comportamento está diretamente relacionado à forma como o algoritmo é estruturado, conforme descrito no pseudocódigo apresentado a seguir:

Algorithm 1: Pseudocódigo busca por subida de encosta.

```

1: Inicializar o ponto inicial (zero ou limite de domínio)  $x_0$ 
2: definir o valor  $\varepsilon$  para candidato vizinho.
3: Definir uma quantidade máxima de iterações  $max_{it}$  e quantidade máxima de candidatos (possíveis vizinhos)  $max_n$ .
4: Melhor valor  $x_{best} \leftarrow x_0$  e melhor valor computado  $f_{best} \leftarrow f(x_{best})$ .
5:  $i \leftarrow 0$ 
6: while  $i < max_{it}$  E houver melhoria do
7:    $j \leftarrow 0$ 
8:   melhoria  $\leftarrow$  false
9:   while  $j < max_n$  do
10:     $j \leftarrow j + 1$ 
11:     $y \leftarrow candidato(x_{best})$ .
12:     $F \leftarrow f(y)$ 
13:    if  $F > f_{best}$  then
14:       $x_{best} \leftarrow y$ 
15:       $f_{best} \leftarrow F$ .
16:      melhoria  $\leftarrow$  true
17:    break
18:  end if
19:   $j \leftarrow j + 1$ 
20: end while
21:  $i \leftarrow i + 1$ 
22: end while
23: fim

```

Algoritmo 1: Pseudocódigo busca por subida de encosta.

O algoritmo Hill Climbing realiza uma busca local iterativa, partindo de um ponto inicial x_0 , definido no limite inferior do domínio. A cada iteração, são gerados até max_n vizinhos dentro de uma vizinhança de raio ε . O primeiro vizinho que apresentar melhora na função objetivo é aceito. O processo continua até que não haja mais melhorias ou até atingir o número máximo de iterações max_{it} . O algoritmo é rápido e eficiente para funções unimodais, mas pode ficar preso em ótimos locais quando aplicado a funções multimodais, pois só aceita soluções que melhoram a atual.

Algorithm 2: Pseudocódigo candidato.

```

1: Definir o valor  $\varepsilon$ .
2: A partir de  $x$  Gerar vizinho-candidato aleatório  $y$  que respeite:  $|x - y| \leq \varepsilon$ 
3: FIM.

```

Algoritmo 2: Pseudocódigo geração de candidatos.

Este algoritmo gera candidatos y ao redor de x , respeitando a condição $|x-y| \leq \varepsilon$. Ele garante que o movimento no espaço de busca seja local e controlado, essencial para o funcionamento do Hill Climbing.

Local Random Search (LRS - Busca Aleatória Local): É um algoritmo de otimização que opera testando soluções candidatas dentro de uma vizinhança próxima à melhor solução encontrada até o momento. Classificado como uma heurística estocástica, o LRS incorpora elementos de aleatoriedade em seu processo de busca, particularmente na geração de números aleatórios para explorar essa vizinhança.

Diferentemente de algoritmos que dependem do cálculo de gradientes para determinar a direção de busca, o LRS é um

método livre de gradiente. Essa característica o torna particularmente útil para funções descontínuas ou problemas onde a derivada não é bem definida, inviável ou computacionalmente cara de se calcular. O LRS trabalha com uma única solução a cada iteração, ajustando-a gradualmente com base na avaliação de candidatos vizinhos.

A solução inicial x_{best} para o LRS é gerada aleatoriamente a partir de uma distribuição uniforme dentro dos limites impostos pelas variáveis independentes do problema (x^l, x^u). Para gerar um novo candidato (x_{cand}), é adicionado um ruído estocástico à solução x_{best} . Este ruído é tipicamente amostrado de uma distribuição normal multivariada com vetor médio nulo e matriz de covariância $\sigma^2 I_d$, onde σ (desvio-padrão) é um hiperparâmetro crucial, geralmente especificado entre 0 e 1. A adequação desse σ é fundamental para balancear a exploração e a exploração na vizinhança local. Além disso, o algoritmo verifica e trata qualquer violação de restrições de caixa para garantir que os candidatos permaneçam dentro do domínio válido.

A seguir, o pseudocódigo da Busca Aleatória Local detalha o fluxo de execução do algoritmo:

```
Algorithm 4: Pseudocódigo busca aleatória global.
1: Definir uma quantidade máxima de iterações  $N_{max}$ .
2: Definir  $x^l$  e  $x^u$ .
3:  $x_{best} \sim U(x^l, x^u)$ 
4:  $f_{best} = f(x_{best})$ 
5:  $i \leftarrow 0$ 
6: while  $i < N_{max}$  do
7:    $x_{cand} = \leftarrow \sim U(x^l, x^u)$ 
8:    $f_{cand} = f(x_{cand})$ 
9:   if  $f_{cand} > f_{best}$  then
10:     $x_{best} = x_{cand}$ 
11:     $f_{best} = f_{cand}$ 
12:   end if
13: end while
14: FIM.
```

Algoritmo 3: Pseudocódigo busca aleatória local.

As Imagens 2 e 3 ilustram a trajetória de execução do LRS em uma função multimodal, destacando como o algoritmo busca por melhorias em sua vizinhança. Embora as Imagens 2 e 3 e a descrição do LRS demonstrem sua capacidade de explorar múltiplos picos e, potencialmente, encontrar um ótimo global, sua eficácia depende do ajuste do σ e da natureza da função objetivo. Em alguns casos, o LRS ainda pode se prender em ótimos locais se o σ for muito pequeno, não permitindo saltos maiores no espaço de busca.

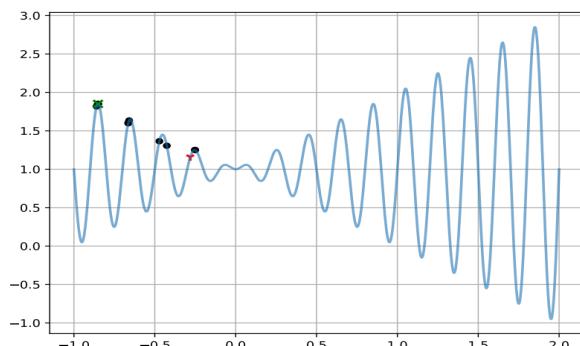


Imagen 2: Trajetórias de busca do algoritmo Local Random Search (LRS)

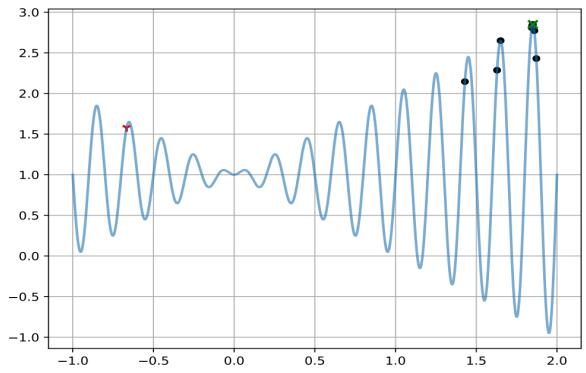


Imagen 3: Ilustração de uma trajetória mais exploratória do LRS

Global Random Search (GRS - Busca Aleatória Global): É um algoritmo de otimização estocástico projetado para encontrar a solução ótima para uma função, especialmente útil em cenários onde os pontos de mínimo e máximo da função custo/objetivo não são previamente conhecidos. Sua natureza heurística reside no fato de que não é derivado de princípios matemáticos formais, mas sim de um conhecimento intuitivo sobre o domínio do problema.

A principal característica do GRS é que ele gera soluções candidatas aleatoriamente dentro de todo o domínio da função a ser otimizada. Por ser um método estocástico, sua execução inicia com uma solução inicial aleatória, e a geração de cada candidato subsequente também depende de rotinas que produzem números aleatórios. É um método de "solução única", o que significa que apenas uma solução candidata é gerada a cada iteração.

Uma vantagem significativa do GRS é que ele é um método livre de gradiente, ou seja, não requer o cálculo de gradientes para determinar as direções de atualização das soluções. Isso o torna particularmente adequado para funções descontínuas (discretas), onde o cálculo de gradientes seria problemático ou impossível.

Apesar de sua capacidade de explorar amplamente o espaço de busca, o GRS não oferece garantia de encontrar a solução ótima em uma única execução. Para contornar essa limitação e obter uma solução subótima mais robusta, recomenda-se realizar várias execuções do algoritmo. Nesses casos, a identificação da "melhor" solução subótima é dada pela solução que aparece com maior frequência entre as execuções (a moda das soluções).

A seguir, o pseudocódigo da Busca Aleatória Global detalha o fluxo de execução do algoritmo:

Algoritmo 4: Pseudocódigo busca aleatória global.

```

1: Definir uma quantidade máxima de iterações  $N_{max}$ .
2: Definir  $x^l$  e  $x^u$ .
3:  $\mathbf{x}_{best} \sim U(x^l, x^u)$ 
4:  $f_{best} = f(\mathbf{x}_{best})$ 
5:  $i \leftarrow 0$ 
6: while  $i < N_{max}$  do
7:    $\mathbf{x}_{rand} = \sim U(x^l, x^u)$ 
8:    $f_{rand} = f(\mathbf{x}_{rand})$ 
9:   if  $f_{rand} > f_{best}$  then
10:     $\mathbf{x}_{best} = \mathbf{x}_{rand}$ 
11:     $f_{best} = f_{rand}$ 
12:   end if
13: end while
14: FIM.

```

Algoritmo 4: Pseudocódigo busca aleatória global.

Exemplos de trajetórias de busca do algoritmo Global Random Search (GRS) em uma função objetivo multimodal. A aleatoriedade na geração de candidatos permite uma exploração ampla do espaço de busca, o que aumenta a probabilidade de encontrar o ótimo global em comparação com métodos de busca local.

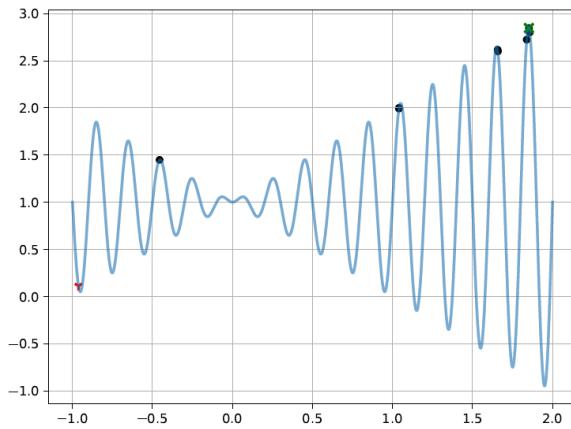


Imagem 4: Trajetórias de busca do algoritmo Global Random Search (GRS)

C. Resultados Obtidos

Os algoritmos HC, LRS e GRS foram executados 100 vezes em cada uma das 8 funções contínuas. Para cada função, registramos a melhor solução, a moda das soluções e sua frequência, além de gráficos com a distribuição dos resultados.

Parâmetros utilizados:

- Rodadas: 100
- Número de iterações: 1000
- Vizinhança(ϵ): 0.1
- Desvio padrão (σ): 0.5
- Máx. vizinhos por iteração: 100

Função 1: $f(x_1, x_2) = x_1^2 + x_2^2$ com $x_1, x_2 \in [-100, 100]$

.

Hill Climbing - Caminhos (100 execuções)

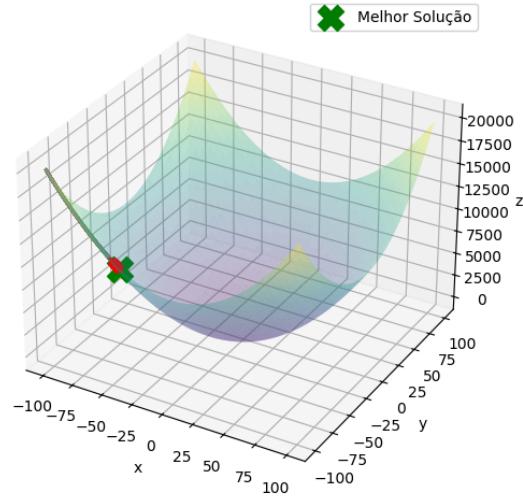


Imagem 5: Função I - HC

Local Random Search - Caminhos (100 execuções)

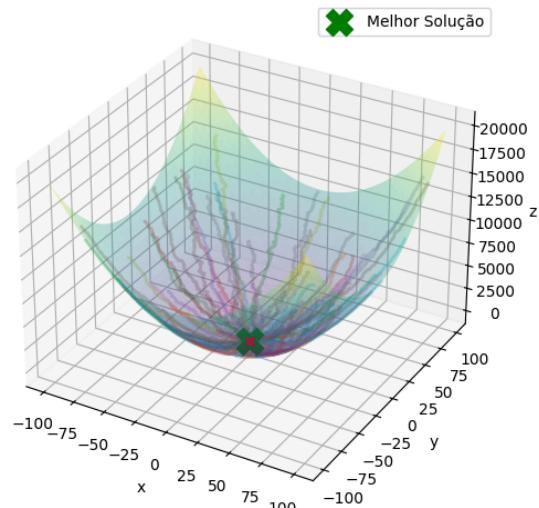


Imagem 6: Função I - LRS

Global Random Search - Pontos Visitados (X)

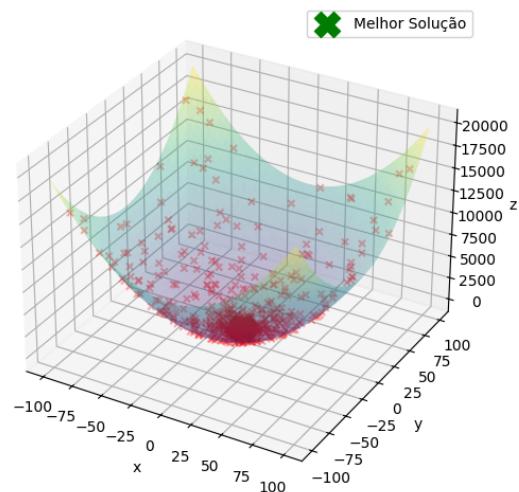


Imagem 7: Função I - GRS

Algoritmo	Moda (x, y)	f(modal)	Frequência
Hill Climbing	[-66.26, -65.96]	8741.109200	1/100
LRS	[0.003, 0.014]	0.000205	2/100
GRS	[-2.915, -2.222]	13.434509	1/100

Tabela 1: Moda das soluções de f1.

Função 2: $f(x_1, x_2) = e^{-(x_1^2+x_2^2)} + 2e^{-(x_1-1.7)^2+(x_2-1.7)^2}$
com $x_1 \in [2, 4]$ e $x_2 \in [2, 5]$.

Hill Climbing - Caminhos (F2)

 Melhor Solução

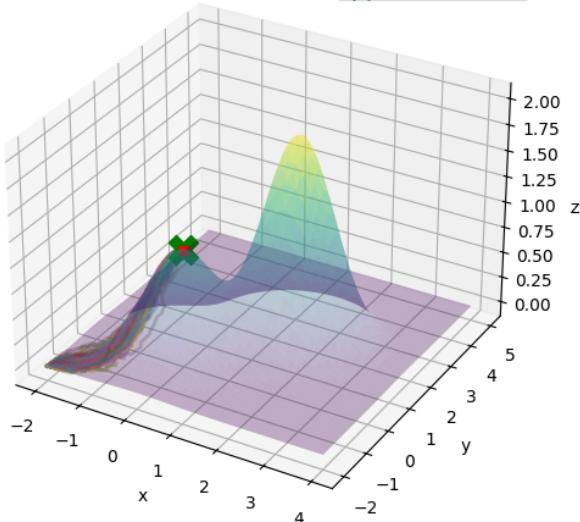


Imagen 8: Função II - HC

Local Random Search - Caminhos (F2)

 Melhor Solução

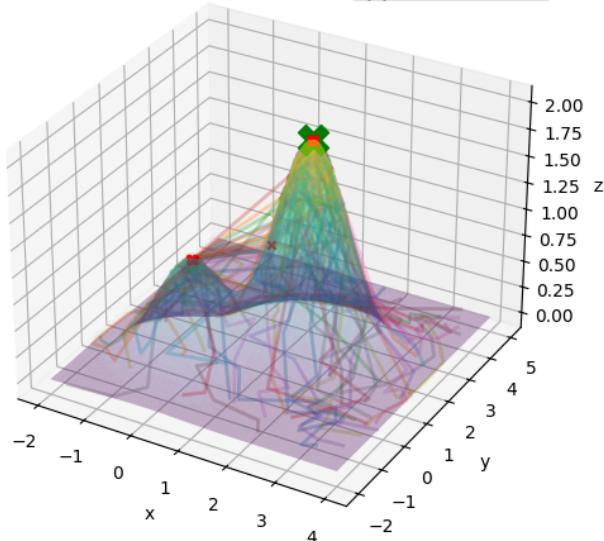


Imagen 9: Função II - LRS

Global Random Search - Pontos Visitados (F2)

 Melhor Solução

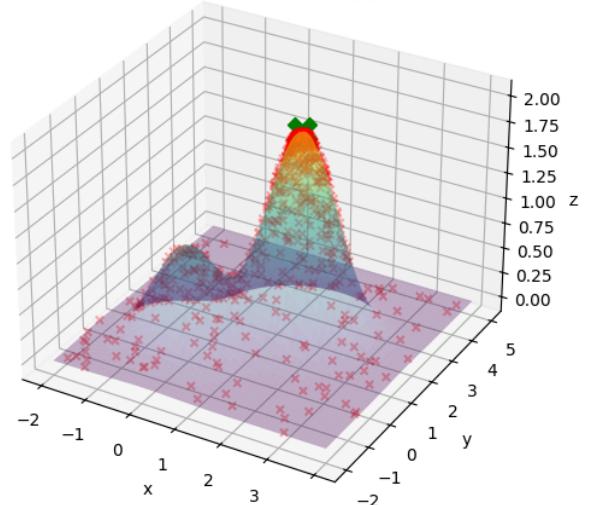


Imagen 10: Função II - GRS

Algoritmo	Moda (x, y)	f(modal)	Frequência
Hill Climbing	[0.012, 0.009]	1.006408	3/100
LRS	[1.696, 1.698]	2.003112	2/100
GRS	[1.758, 1.663]	1.993419	1/100

Tabela 2: Moda das soluções de f2.

Função 3:

$f(x_1, x_2) = -20e^{-0.2\sqrt{0.5(x_1^2+x_2^2)}} - e^{0.5\cos(2\pi x_1)+\cos(2\pi x_2)} + 20 + e^1$
com $x_1, x_2 \in [-8, 8]$.

Hill Climbing - Caminhos (F3)

 Melhor Solução

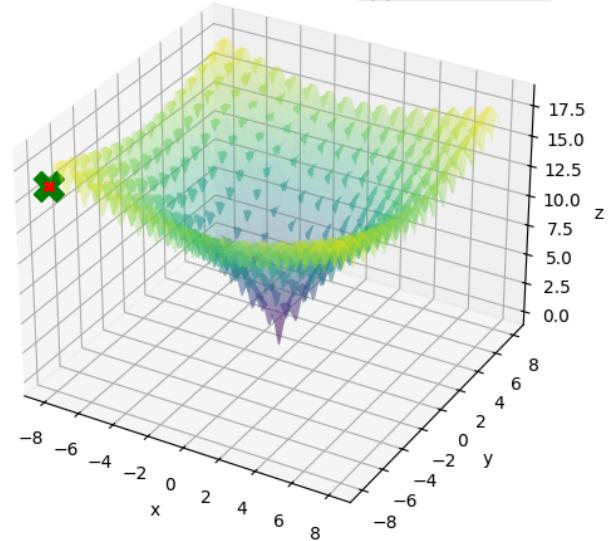


Imagen 11: Função III - HC

Local Random Search - Caminhos (F3)

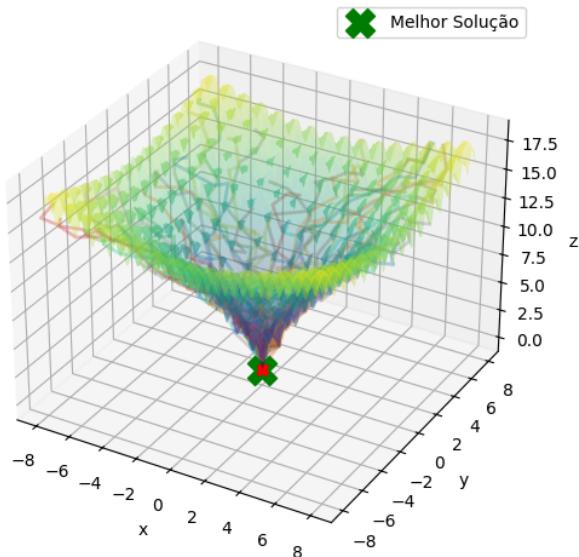


Imagen 12: Função III - LRS

Global Random Search - Pontos Visitados (F3)

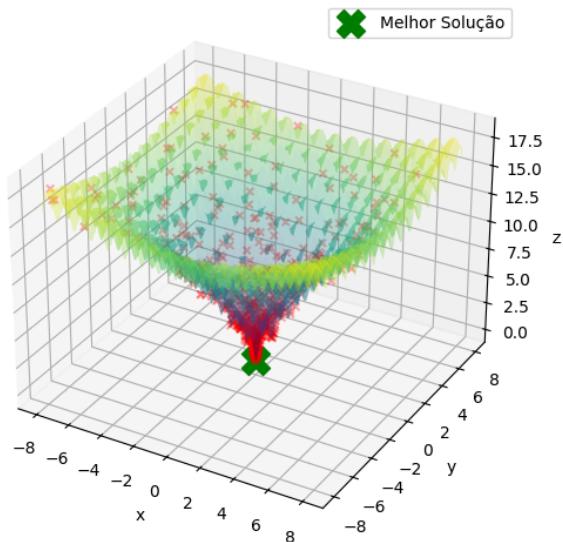


Imagen 13: Função III - GRS

Algoritmo	Moda (x, y)	$f(\text{moda})$	Frequência
Hill Climbing	[-8.0, -7.992]	15.960555	12/100
LRS	[-0.019, -0.003]	0.064229	1/100
GRS	[-0.144, 0.166]	1.578311	1/100

Tabela 3: Moda das soluções de f3.

Função 4:

$$f(x_1, x_2) = x_1^2 - 10\cos(2\pi x_1) + x_2^2 - 10\cos(2\pi x_2) + 20$$

com $x_1, x_2 \in [-5.12, 5.12]$.

Hill Climbing - Caminhos (F4)

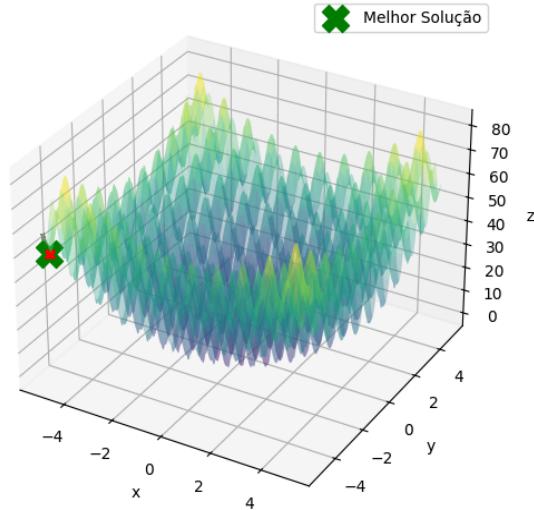


Imagen 14: Função IV - HC

Local Random Search - Caminhos (F4)

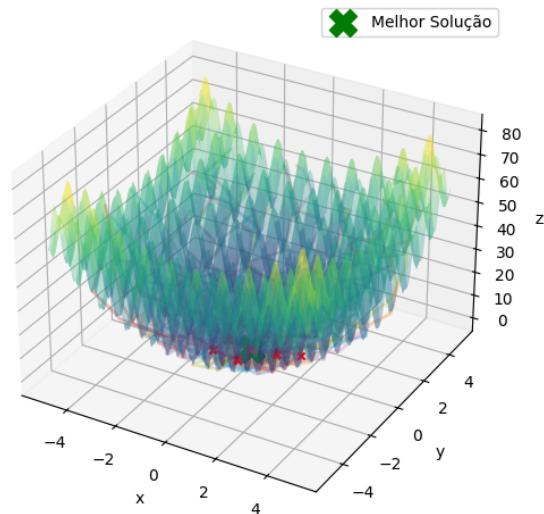


Imagen 15: Função IV - HC

Global Random Search - Pontos Visitados (F4)

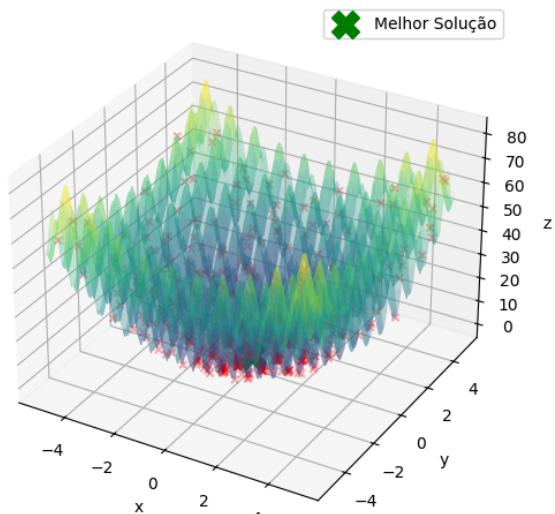


Imagen 16: Função IV - GRS

Algoritmo	Moda (x, y)	f(modal)	Frequência
Hill Climbing	[-4.978, -4.977]	49.750638	4/100
LRS	[0.015, -0.004]	0.047779	2/100
GRS	[-0.032, -0.061]	0.931743	1/100

Tabela 4: Moda das soluções de f4.

Função 5: $f(x_1, x_2) = \frac{x_1 \cos(x_1)}{20} + 2e^{-(x_1)^2 - (x_2 - 1)^2 + 0.01 \cdot x_1 \cdot x_2}$
 $x_1 \in [-10, 10]$ e $x_2 \in [-10, 10]$.

Hill Climbing - Caminhos (F5)

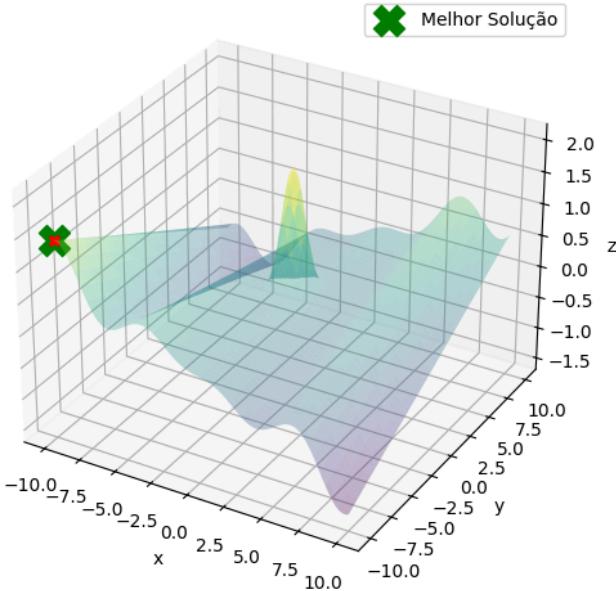


Imagen 17: Função V - HC

Global Random Search - Pontos Visitados (F5)

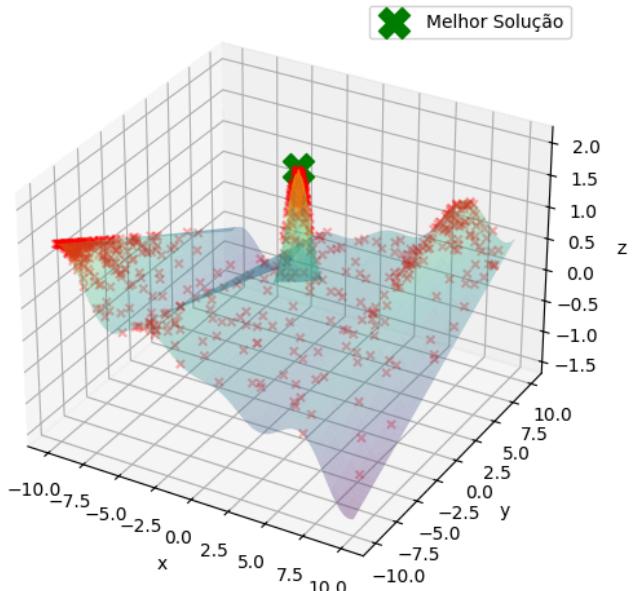


Imagen 19: Função V - GRS

Algoritmo	Moda (x, y)	f(modal)	Frequência
Hill Climbing	[-9.733, -10.0]	1.437016	38/100
LRS	[6.729, 10.0]	0.976465	13/100
GRS	[0.127, 1.265]	1.842444	1/100

Tabela 5: Moda das soluções de f5.

Função 6:

$f(x_1, x_2) = x_1 \cdot \sin(4\pi x_1) - x_2 \cdot \sin(4\pi x_2) + 1$ com
 $x_1 \in [-1, 3]$ e $x_2 \in [-1, 3]$.

Hill Climbing - Caminhos (F6)

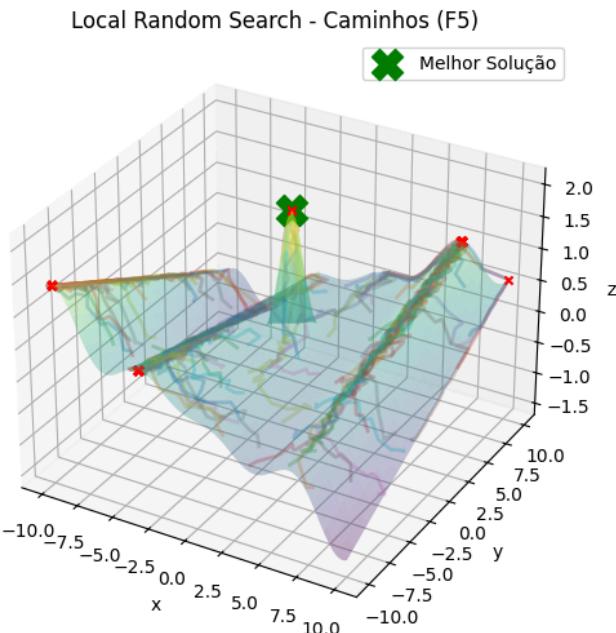


Imagen 18: Função V - LRS

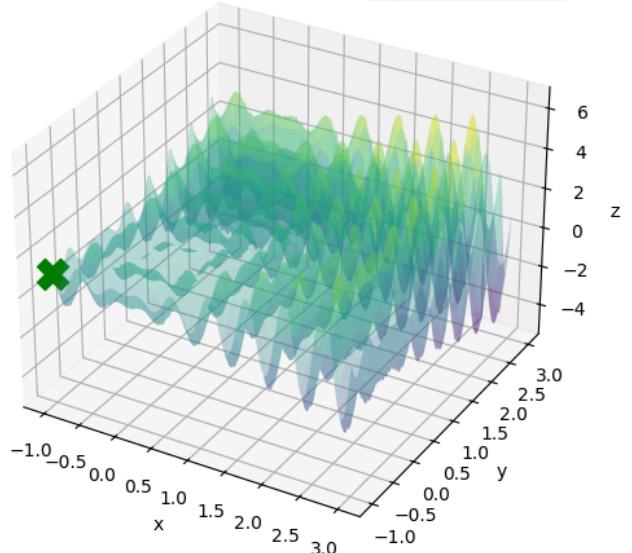


Imagen 20: Função VI - HC

Local Random Search - Caminhos (F6)

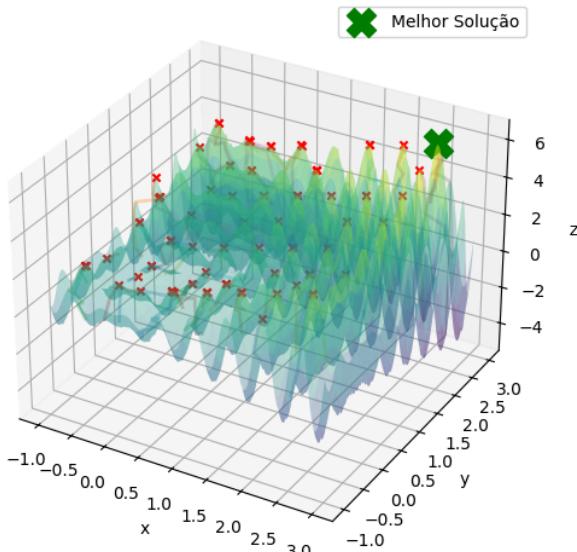


Imagen 21: Função VI - LRS

Global Random Search - Pontos Visitados (F6)

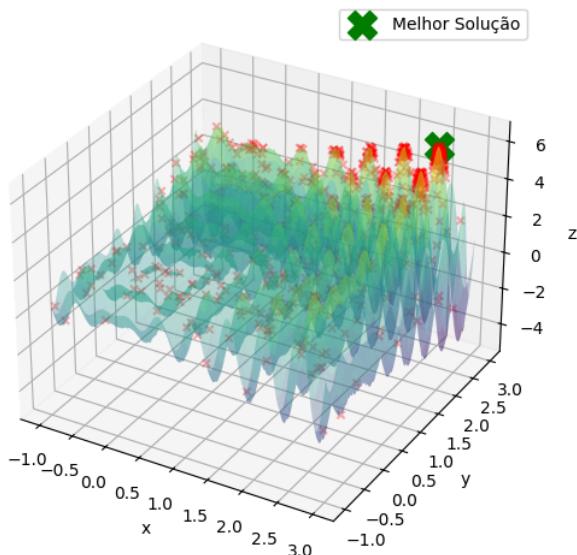


Imagen 22: Função VI - GRS

Algoritmo	Moda (x, y)	$f(\text{moda})$	Frequência
Hill Climbing	$[-1, -1]$	1.000000	100/100
LRS	$[-0.159, -0.156]$	1.289019	1/100
GRS	$[2.624, 2.601]$	6.107396	1/100

Tabela 6: Moda das soluções de f6.

Função 7:

$$f(x_1, x_2) = -\sin(x_1) \cdot \sin\left(\frac{x_1^2}{\pi}\right)^{20} - \sin(x_2) \cdot \sin\left(\frac{x_2^2}{\pi}\right)^{20}$$

com $x_1 \in [0, \pi]$ e $x_2 \in [0, \pi]$.

Hill Climbing - Caminhos (F7)

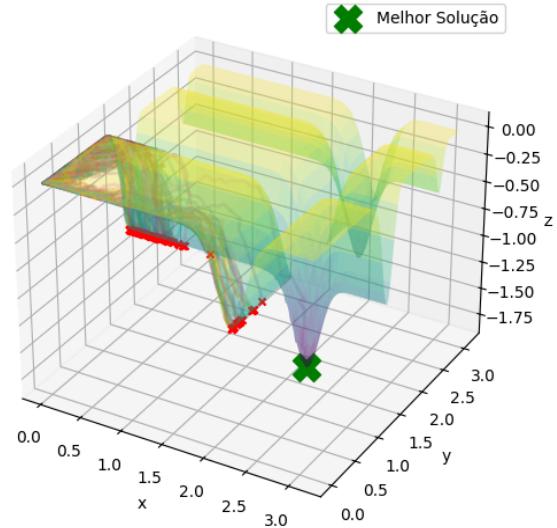


Imagen 23: Função VII - HC

Local Random Search - Caminhos (F7)

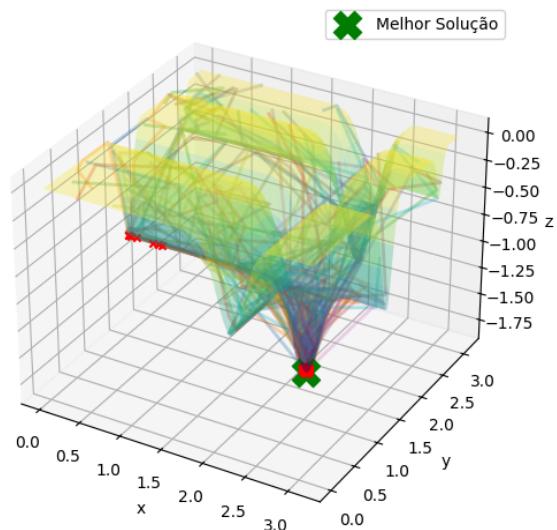


Imagen 24: Função VII - LRS

Global Random Search - Pontos Visitados (F7)

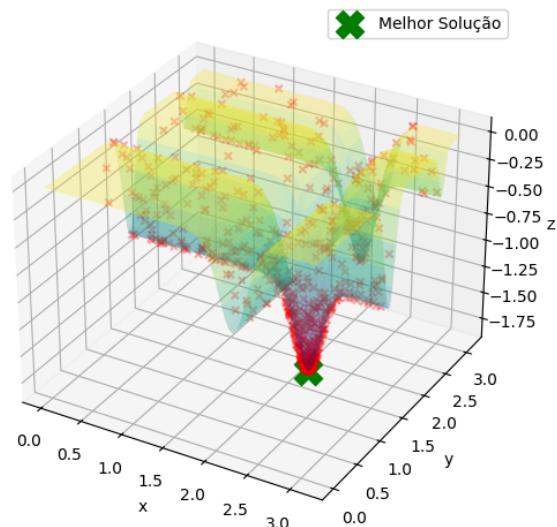


Imagen 25: Função VII - GRS

Algoritmo	Moda (x, y)	f(modas)	Frequência
Hill Climbing	[0.0, 1.571]	-0.999998	7/100
LRS	[2.186, 1.556]	-1.787958	2/100
GRS	[2.234, 1.578]	-1.783419	1/100

Tabela 7: Moda das soluções de f7.

Função 8:

$$f(x_1, x_2) = -(x_2 + 47) \sin(\sqrt{|\frac{x_1}{2} + (x_2 + 47)|}) - x_1 \sin(\sqrt{|x_1 - (x_2 + 47)|}) \text{ com } x_1 \in [-200, 20] \text{ e } x_2 \in [-200, 20].$$

Hill Climbing - Caminhos (F8)

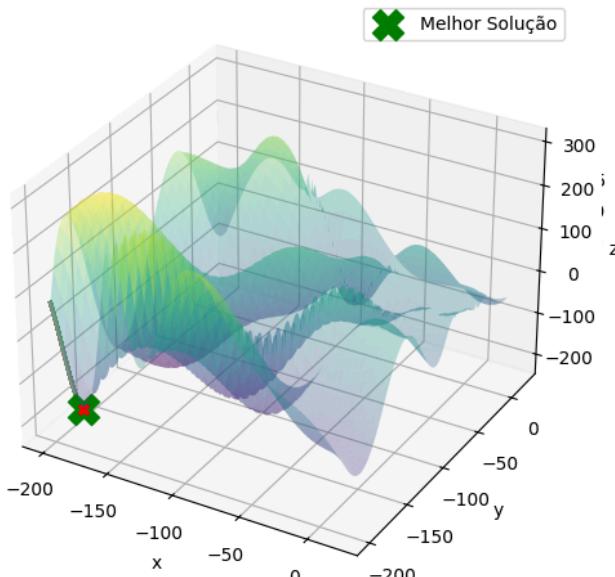


Imagen 26: Função VIII - HC

Local Random Search - Caminhos (F8)

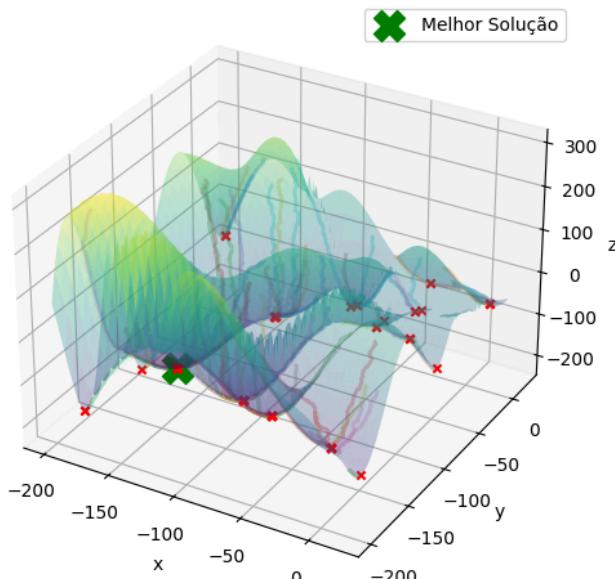


Imagen 27: Função VIII - LRS

Global Random Search - Pontos Visitados (F8)

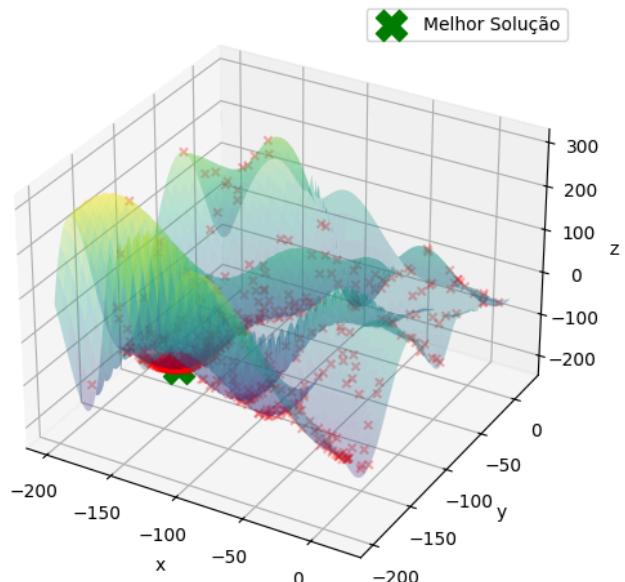


Imagen 28: Função VIII - GRS

Algoritmo	Moda (x, y)	f(modas)	Frequência
Hill Climbing	[-177.033, -200.0]	-148.391719	33/100
LRS	[-177.033, -200.0]	-148.391719	2/100
GRS	[-180.982, -99.974]	-201.295093	1/100

Tabela 8: Moda das soluções de f8.

D. Análise

O algoritmo Hill Climbing (HC) apresentou desempenho inferior na maioria das funções avaliadas, devido à sua limitação inerente de exploração. Por realizar uma busca local estrita e aceitar apenas soluções melhores a cada iteração, ele frequentemente se prende a ótimos locais, especialmente quando o ponto de partida já se encontra em uma região de alto valor local da função objetivo. Isso o torna pouco eficaz em cenários multimodais com múltiplos picos ou vales. O Local Random Search (LRS) demonstrou um desempenho mais robusto, explorando melhor o espaço de busca por meio de perturbações aleatórias em torno da solução atual. Essa característica permitiu que o LRS escapasse de ótimos locais em várias execuções, sendo eficaz tanto em funções unimodais quanto multimodais. O Global Random Search (GRS) foi o algoritmo que obteve os melhores resultados em nosso experimento. Sua estratégia de amostragem uniforme e global do espaço de busca permitiu alcançar ótimos globais de forma consistente. No entanto, seu desempenho está associado ao fator aleatoriedade, o que implica em dependência de sorte e alto custo computacional, dado que muitas amostras podem ser necessárias até encontrar soluções próximas do ótimo. Apesar disso, o GRS se mostrou uma abordagem eficaz para problemas onde a exploração global é essencial.

III. ETAPA 2: PROBLEMA DE DOMÍNIO DISCRETO

A. Análise da problemática

A segunda etapa deste trabalho direcionou-se à resolução de um desafio fundamental na otimização combinatória: o

Problema das 8 Rainhas. Este problema clássico da Inteligência Artificial consiste em posicionar oito rainhas em um tabuleiro de xadrez 8×8 de tal forma que nenhuma rainha possa atacar outra. Para que nenhuma rainha se ataque, é necessário que não estejam na mesma linha, na mesma coluna ou na mesma diagonal.

Diferentemente dos problemas de otimização contínua abordados na Etapa I, onde as variáveis assumem valores reais em um domínio bidimensional, o Problema das 8 Rainhas possui um espaço de busca discreto e finito. Embora o número total de arranjos possíveis das 8 rainhas em um tabuleiro 8×8 seja vasto, a complexidade pode ser reduzida ao se adotar a regra de posicionar uma rainha por coluna, limitando o espaço de busca a 8^8 arranjos. Apesar disso, o problema possui 92 soluções distintas e válidas, que são os ótimos globais.

A representação de cada solução candidata é crucial. Neste trabalho, uma solução é modelada como um vetor de 8 componentes, onde cada índice do vetor representa uma coluna do tabuleiro, e o valor correspondente indica a linha em que a rainha está posicionada naquela coluna. Por exemplo, o candidato $x=[5,1,4,2,6,1,4,7]$ significa que a rainha na coluna 1 está na linha 5, a rainha na coluna 2 está na linha 1, e assim por diante.

B. Algoritmo Implementado

Simulated annealing (Têmpera Simulada): O algoritmo de Têmpera Simulada é uma poderosa técnica de otimização probabilística inspirada no processo físico da têmpera de metais. Na metalurgia, a têmpera envolve aquecer um metal a uma alta temperatura inicial e, em seguida, resfriá-lo gradualmente para aumentar sua dureza e estabilidade estrutural. Essa analogia é transposta para a computação para encontrar o ótimo global de um problema de otimização.

A essência da Têmpera Simulada reside em sua capacidade de escapar de ótimos locais, um desafio comum para algoritmos de busca puramente gulosos como o Hill Climbing. Imagine uma bola de pingue-pongue em uma superfície com múltiplos vales (mínimos locais) e picos (máximos locais). Se a bola for simplesmente solta, ela provavelmente cairá em um mínimo local próximo ao ponto de partida. No entanto, se a superfície for agitada (simulando "alta temperatura"), a bola pode ser deslocada de um mínimo local e ter a chance de encontrar um mínimo global. O "truque" é aplicar agitação suficiente para permitir que a bola escape de ótimos locais, mas reduzir essa agitação gradualmente para que ela não seja desalojada do ótimo global uma vez que o encontre.

A principal diferença em relação ao algoritmo Hill Climbing é a estratégia de aceitação de movimentos. Enquanto o Hill Climbing aceita apenas movimentos que melhoram a situação, a Têmpera Simulada pode aceitar movimentos aleatórios que pioram a solução (para maximização, uma

queda no valor da função objetivo; para minimização, um aumento no valor) com uma certa probabilidade menor que 1. Essa probabilidade de aceitação diminui exponencialmente tanto com a "má qualidade" do vizinho (a diferença na função objetivo) quanto com a redução da "temperatura" (T) ao longo das iterações. Quanto maior a temperatura, maior a chance de aceitar uma solução pior, permitindo uma exploração mais ampla do espaço de busca. À medida que a temperatura diminui, o algoritmo torna-se mais "guloso", favorecendo melhorias e buscando convergir para um ótimo.

A aceitação probabilística é comumente baseada na distribuição de Boltzmann-Gibbs, dada por

$$P_{ij} = e^{-\frac{f(x_j) - f(x_i)}{T}},$$

onde x_j é o candidato gerado, x_i é a melhor solução corrente, e T é a temperatura no instante atual. O algoritmo emprega uma abordagem de escalonamento (ou resfriamento) da temperatura ao longo das iterações, com diferentes métodos de decaimento, como decaimento multiplicativo ($T_{i+1} = (\text{decaimento}) \cdot T_i$), decaimento

baseado em raiz quadrada ($T_{i+1} = \frac{T_i}{1 + (\text{decaimento}) \cdot \sqrt{i}}$), ou decaimento linear ($T_{i+1} = T_i - \Delta T$, onde $\Delta T = \frac{T_0 - T_{nt}}{nt}$). A escolha do escalonamento de temperatura é crucial para a convergência e eficácia do algoritmo.

A seguir, o pseudocódigo da Têmpera Simulada detalha o fluxo de execução do algoritmo:

Algoritmo 5: Pseudocódigo Têmpera Simulada.

```

1: Definir uma quantidade máxima de iterações  $N_{max}$  e  $T$ .
2: Definir  $x$  e  $x^0$ .
3: Definir valor de  $\sigma$  (perturbação aleatória).
4:  $x_{best} \sim U(x^0, x^0)$ 
5:  $f_{best} = f(x_{best})$ 
6:  $i \leftarrow 0$ 
7: while  $i < N_{max}$  do
8:    $n \sim N(0, \sigma)$ 
9:    $x_{rand} \leftarrow x_{best} + n$ 
10:  Verificar a violação da restrição em caixa.
11:   $f_{rand} = f(x_{rand})$ 
12:   $P_{ij} \leftarrow e^{-(f_{rand} - f_{opt})/T}$ 
13:  if  $f_{rand} < f_{best}$  ou  $P_{ij} \geq U(0, 1)$  then
14:     $x_{best} = x_{rand}$ 
15:     $f_{best} = f_{rand}$ 
16:  end if
17:   $i \leftarrow i + 1$ 
18:  escalonar( $T$ )
19: end while
20: FIM.

```

Algoritmo 5: Pseudocódigo tempéra simulada.

C. Resultados Obtidos

Para avaliar a qualidade de uma solução, a função objetivo foi projetada para quantificar o número de pares de rainhas que estão se atacando. Uma função de aptidão interessante é $f(x) = 28 - h(x)$, onde $h(x)$ representa o número de pares de rainhas atacantes para a solução x . Com um total máximo de 28 pares de rainhas que podem se atacar no tabuleiro, o objetivo do problema se torna maximizar a função $f(x)$, o que equivale a minimizar o número de ataques entre as rainhas

A natureza combinatória e a presença de múltiplos ótimos (as 92 soluções válidas) tornam este problema inadequado para métodos de busca baseados em gradiente ou algoritmos puramente gulosos, que podem facilmente se prender em ótimos locais. Por isso, optou-se pela utilização do algoritmo de Têmpera Simulada justamente por sua capacidade de aceitar transições para soluções inferiores, o que permite escapar de ótimos locais e alcançar soluções mais próximas do ótimo global.

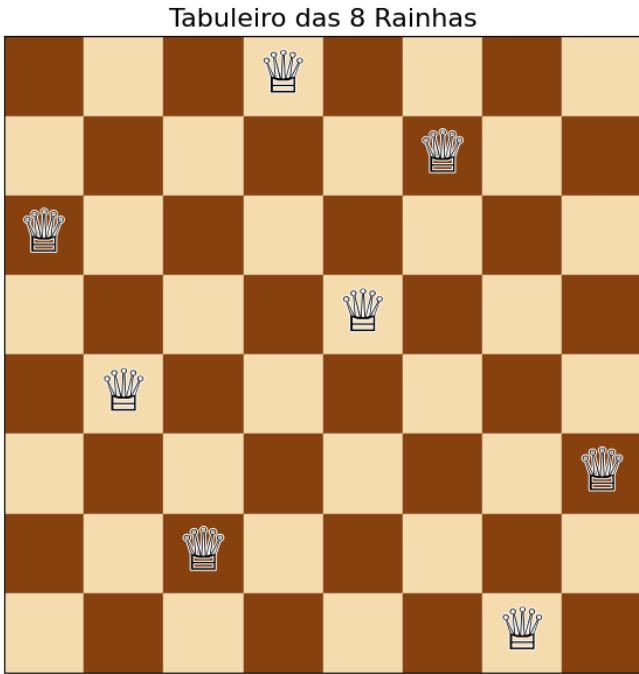


Imagem 29: Exemplo de solução encontrada para o problema das 8 Damas.

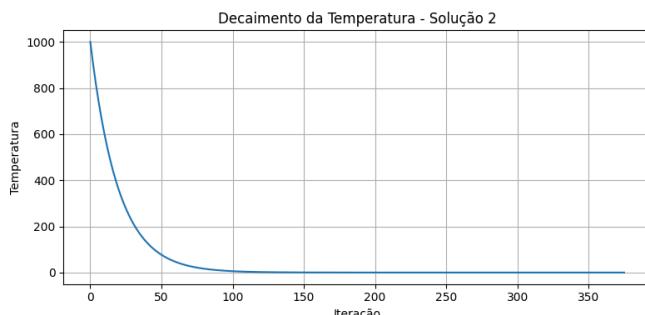


Imagem 30: Gráfico de desempenho da Têmpera Simulada ao resolver o problema da imagem 29.

No total, foram realizadas:

- 92 soluções únicas encontradas.
- 759 execuções necessárias
- Tempo total: 55.66 segundos

D. Análise

Os objetivos específicos desta etapa incluíram não apenas a identificação de uma solução válida, mas a busca por todas as 92 soluções únicas possíveis do problema. Além disso, o trabalho visou analisar o comportamento do algoritmo ao longo das execuções, incluindo o decaimento da

temperatura e a representação visual dos tabuleiros das soluções encontradas.

Eficiência e Convergência

- O algoritmo alcançou soluções ótimas ($f(x) = 28$) rapidamente, com média inferior a 300 iterações por execução.
- A temperatura inicial foi ajustada para ($T_0 = 1000$) com taxa de decaimento ($\alpha = 0.95$), resultando em convergência suave e controlada.
- A maioria das execuções resultou em soluções ótimas sem necessidade de alcançar o número máximo de iterações (10000), evidenciado por curvas de temperatura que cessam por volta da 200^a-300^a iteração.

Sobre as 92 Soluções

- Foram encontradas todas as 92 soluções únicas válidas exigidas no enunciado.
- Cada solução foi validada numericamente e visualmente com uso da biblioteca *python-chess*.
- As soluções foram armazenadas em estrutura de conjunto (*set*) para evitar duplicatas e garantir unicidade.
- Exemplo de soluções:
 - Solução 01: (6, 3, 1, 8, 4, 2, 7, 5)
 - Solução 47: (5, 7, 2, 6, 3, 1, 4, 8)
 - Solução 92: (8, 2, 5, 3, 1, 7, 4, 6)

Observações Técnicas

- A geração inicial das soluções como permutação de 1 a 8 evitou conflitos de linha e acelerou a convergência.
- A função de perturbação, baseada na troca de duas rainhas, foi suficiente para explorar eficientemente o espaço de busca.
- A estratégia de aceitação probabilística baseada na função de Boltzmann permitiu escapar de mínimos locais, o que foi essencial para a diversidade de soluções.

Visualização e Evidência de Desempenho

- A cada nova solução, foi exibido:
 - O vetor da solução;
 - O gráfico de decaimento da temperatura;
 - A representação visual do tabuleiro.
- Os gráficos de temperatura apresentaram comportamento exponencial, típico de Têmpera Simulada bem configurada.
- O tempo total de execução para encontrar as 92 soluções ficou dentro de limites razoáveis no ambiente do Google Colab.

E. Conclusão do problema de domínio discreto

A Parte 2 do trabalho foi concluída com total sucesso. Todas as 92 soluções únicas foram obtidas com uma abordagem

consistente, estruturada e alinhada aos princípios de otimização probabilística. O algoritmo de Têmpera Simulada mostrou-se eficaz, estável e adequado para esse tipo de problema combinatório.

IV. REFERÊNCIAS

- [1] IEEE Conference Templates. IEEE. Disponível em: <https://www.ieee.org/conferences/publishing/templates.html>
- [2] Cirillo Souza Barbosa, Paulo M., "Busca e Otimização", Universidade de Fortaleza, 2025.
- [3] NumPy Documentation, <https://numpy.org/pt/>
- [4] Matplotlib Documentation, <https://matplotlib.org/>
- [5] OpenAI, "ChatGPT — Large language model," 2023. Disponível em: <https://openai.com/chatgpt>. Acesso em: 15 junho 2025.
- [6] Google DeepMind, "Gemini: Advanced AI models for natural language understanding," 2024. Disponível em: <https://deepmind.com/research/ai/gemini>. Acesso em: 15 junho 2025.