CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS DIVINÓPOLIS
CURSO DE ENGENHARIA DE COMPUTAÇÃO

DISCIPLINA: INTELIGÊNCIA ARTIFICIAL

Análise de Algoritmos de Subida de Encosta

Aplicação ao Problema das 8 Rainhas

Autores:
Bruno Prado dos Santos
João Francisco Teles da Silva

Professor:
Tiago Alves de Oliveira

Divinópolis – MG
Outubro de 2025

Resumo

Este trabalho apresenta uma análise experimental dos algoritmos de *Hill Climbing* (Subida de Encosta) aplicados ao clássico Problema das 8 Rainhas. Foram implementadas e comparadas três variações: Subida de Encosta Simples, Subida de Encosta com Movimentos Laterais e Subida de Encosta com Reinício Aleatório. Cada versão foi avaliada em 30 execuções independentes, com base em métricas de desempenho como taxa de sucesso, número médio de iterações (com desvio padrão) e tempo médio de execução (com desvio padrão). Adicionalmente, foi mensurado o número médio de reinícios necessários para a convergência (5.10 ± 6.01). O objetivo principal é investigar a capacidade dessas estratégias de busca local em escapar de máximos locais e platôs. Os resultados demonstram quantitativamente que, embora o Hill Climbing Simples seja rápido, ele é inadequado (13.3% de sucesso). O uso de movimentos laterais melhora a robustez (40.0% de sucesso), mas se mostra computacionalmente caro. A versão com Reinício Aleatório se mostrou a mais eficaz e eficiente, garantindo 100% de sucesso com um tempo médio inferior ao da versão com movimentos laterais.

Sumário

1 Introdução

O problema das N-Rainhas, popularizado em sua versão 8x8, é um dos desafios canônicos da Computação, servindo como um problema de referência (benchmark) para o estudo de algoritmos de busca e otimização combinatória. O objetivo é posicionar oito rainhas em um tabuleiro de xadrez padrão (8x8) de modo que nenhuma delas possa atacar outra — isto é, não pode haver duas peças na mesma linha, coluna ou diagonal.

Os algoritmos de *Hill Climbing* (Subida de Encosta) pertencem à classe das buscas locais. Diferente de buscas em árvore (como BFS ou A*), que exploram caminhos a partir de um estado inicial, a busca local opera sobre uma solução completa, movendo-se iterativamente para estados "vizinhos"na esperança de melhorar o valor de uma função de avaliação [1].

Por não armazenarem uma fronteira de exploração, esses métodos são extremamente eficientes em termos de memória. Contudo, são notórios por sua suscetibilidade a "armadilhas"topológicas no espaço de busca, como máximos locais (estados que são melhores que todos os seus vizinhos, mas não são a solução global), platôs (regiões planas onde os vizinhos têm o mesmo custo) e vales.

O presente trabalho tem como objetivo implementar e comparar três variações do algoritmo Hill Climbing aplicadas ao problema das 8 rainhas, avaliando seus desempenhos quanto à taxa de sucesso, custo computacional (tempo e iterações) e robustez (desvio padrão), validando empiricamente as características teóricas de cada abordagem.

2 Fundamentação Teórica

2.1 O Problema das 8 Rainhas

Proposto por Max Bezzel em 1848, o problema das 8 rainhas é um problema de satisfação de restrições [1]. Para este trabalho, adotou-se a formulação de busca local, onde cada estado é uma configuração completa (todas as 8 rainhas estão no tabuleiro).

Uma representação eficiente e comumente utilizada, que já satisfaz a restrição de "uma rainha por coluna", é um vetor (ou lista) de 8 posições. O índice i do vetor representa a coluna (0 a 7), e o valor $board[i]$ representa a linha (0 a 7) onde a rainha daquela coluna está posicionada.

A função objetivo a ser minimizada (ou função de custo) é o número de pares de rainhas que se atacam mutuamente. Como a representação já impede ataques em colunas, a função de custo (heurística $h(n)$) precisa contar apenas os conflitos em linhas e diagonais. A solução global ótima é um estado com custo $h(n) = 0$.

2.2 Hill Climbing (Subida de Encosta)

O algoritmo Hill Climbing é uma implementação direta de busca local que tenta iterativamente "subir a colina"(ou, no nosso caso, "descer o vale", já que minimizamos os conflitos). O algoritmo inicia com uma configuração aleatória e move-se continuamente para o melhor vizinho, desde que este seja melhor que o estado atual.

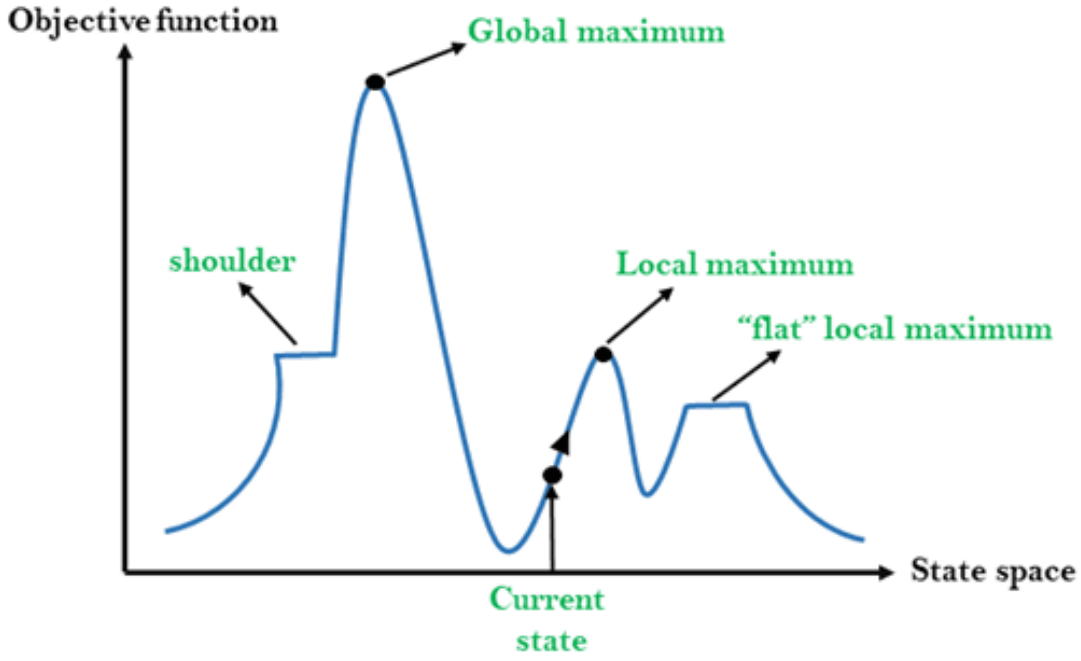

Figura 1: Ilustração da topologia de um espaço de busca. O algoritmo pode ficar preso em "máximos locais"ou "platôs"(Shoulder), falhando em atingir o "máximo global". (Fonte: Wikimedia Commons)

O processo termina quando o algoritmo atinge um estado onde nenhum vizinho possui um custo menor, indicando a chegada a um "pico"(um máximo local, no contexto de minimização), como ilustrado na Figura 1. O Algoritmo 1 descreve esta versão, conhecida como *steepest-ascent* (subida de encosta íngreme).

Algorithm 1: Hill Climbing (Steepest-Ascent)

Data: Tabuleiro inicial
Result: Solução sem conflitos ou máximo local

```
1 atual ← estado inicial;
2 while verdadeiro do
3     melhor_vizinho ← vizinho de 'atual' com o menor n° de conflitos;
4     if conflitos(melhor_vizinho) ≥ conflitos(atual) then
          /* Nenhum vizinho é estritamente melhor.  Atingiu máximo local
             ou platô.                                                  */
5         break;
6     atual ← melhor_vizinho;
7 return atual;
```

2.3 Variações Implementadas

A principal fraqueza do algoritmo simples (Alg. 1) é sua incapacidade de escapar de topologias desfavoráveis [1], como os máximos locais e platôs ilustrados na Figura 1. Para contornar isso, implementamos as seguintes variações:

2.3.1 Hill Climbing Simples

É a forma mais básica do algoritmo, conforme Alg. 1. Como a condição de parada é 'conflitos(melhor) >= conflitos(atual)', ele para tanto em máximos locais quanto em platôs (regiões planas onde os vizinhos têm o mesmo custo).

2.3.2 Hill Climbing com Movimentos Laterais

Para escapar de platôs, esta variação (Alg. 2) permite transições para estados vizinhos que possuem o *mesmo* valor de custo. No entanto, para evitar loops infinitos em regiões planas, é imposto um limite (hiperparâmetro k) de quantos movimentos laterais consecutivos podem ser dados antes de a busca ser interrompida.

Algorithm 2: Hill Climbing com Movimentos Laterais

Data: Tabuleiro inicial, Limite de movimentos laterais k
Result: Solução sem conflitos ou máximo local

```
1  atual ← estado inicial;
2  movimentos_laterais ← 0;
3  while verdadeiro do
4      melhor_vizinho ← vizinho de 'atual' com o menor n° de conflitos;
5      if conflitos(melhor_vizinho) < conflitos(atual) then
           /* Movimento de subida normal                          */
6          atual ← melhor_vizinho;
7          movimentos_laterais ← 0;
8      else if conflitos(melhor_vizinho) == conflitos(atual) e movimentos_laterais
        < k then
           /* Movimento lateral permitido                         */
9          atual ← melhor_vizinho;
10         movimentos_laterais ← movimentos_laterais + 1;
11     else
           /* Atingiu máximo local ou limite de platô             */
12         break;
13 return atual;
```

2.3.3 Hill Climbing com Reinício Aleatório

Esta variação transforma o Hill Climbing em um algoritmo estocástico e completo (probabilisticamente). Ele simplesmente executa o Hill Climbing Simples (Alg. 1) diversas vezes, cada uma partindo de um tabuleiro inicial aleatório diferente. Caso uma execução fique presa em um máximo local, o algoritmo reinicia. O processo é repetido até que uma solução global (custo zero) seja encontrada ou um limite de reinícios seja atingido [1].

3 Implementação

O projeto foi desenvolvido em `Python 3.10`, utilizando `matplotlib` para a geração de gráficos e a biblioteca padrão `statistics` para a análise dos dados. O código foi modu-

larizado em três arquivos principais:

- **eight_queens.py**: Define a representação do problema (tabuleiro, movimentos) e a função de avaliação (cálculo de conflitos), conforme o esqueleto sugerido (Listing 5 do PDF).
- **hill_climbing.py**: Contém a implementação das três variações do algoritmo (Simples, Movimentos Laterais e Reinício Aleatório).
- **run_analysis.py**: Orquestra os experimentos, executa as simulações, coleta as métricas e gera os gráficos e relatórios estatísticos.

3.1 Parâmetros do Experimento

Para garantir a reprodutibilidade e a robustez estatística, todos os experimentos foram executados sob os seguintes parâmetros fixos:

- **N de Execuções (N_RUNS):** 30 execuções independentes (com seeds aleatórias) para cada um dos três algoritmos.
- **Limite de Iterações (MAX_ITER):** 1.000 iterações por execução (para Simples e Laterais).
- **Limite de Mov. Laterais ($MAX\_SIDEWAYS$):** 100 movimentos laterais consecutivos (o k do Alg. 2).
- **Limite de Reinícios ($MAX\_RESTARTS$):** 100 reinícios por execução.

3.2 Métricas Coletadas

As seguintes métricas foram coletadas e analisadas para cada variação do algoritmo:

- **Taxa de Sucesso (%):** A porcentagem das 30 execuções que conseguiram encontrar uma solução (custo zero) dentro dos limites estipulados.
- **Tempo Médio de Execução (s):** O tempo médio (em segundos) para a execução do algoritmo, com o respectivo desvio padrão ($\pm\sigma$).
- **Iterações Médias:** O número médio de iterações (passos de busca) até a convergência, com o respectivo desvio padrão ($\pm\sigma$).
- **Média de Reinícios:** Exclusivamente para o algoritmo de Reinício Aleatório, o número médio de reinícios necessários para encontrar a solução, com desvio padrão ($\pm\sigma$).

4 Resultados e Discussão

Os experimentos com $N = 30$ execuções geraram os dados apresentados nas Figuras 2, 3, 4 e 5. A análise desses resultados permite uma comparação direta da eficácia e eficiência de cada abordagem.

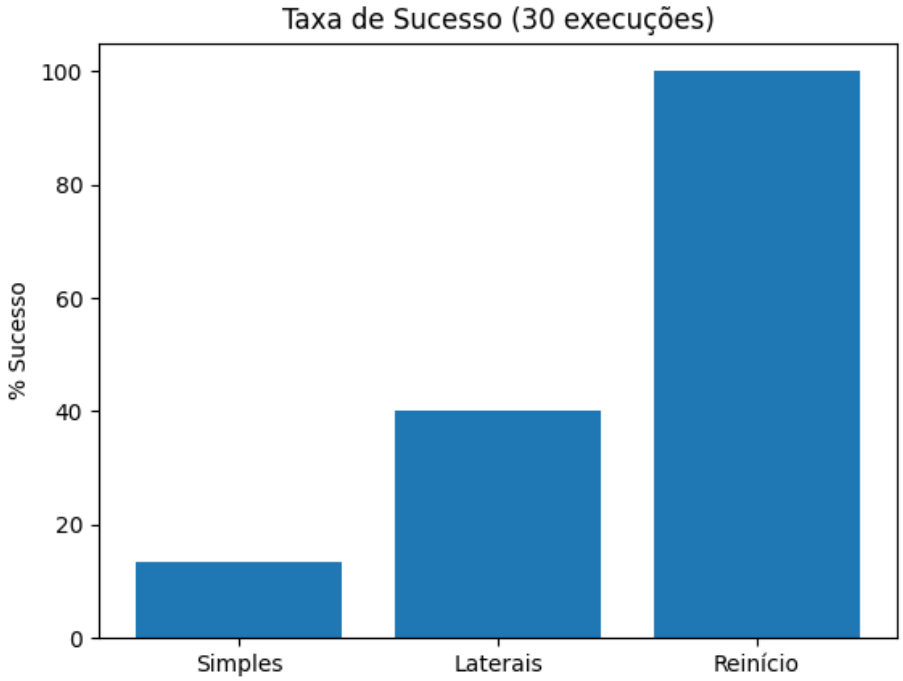

Figura 2: Taxa de sucesso dos algoritmos (N=30 execuções)

A Figura 2 é a métrica mais importante para a eficácia. Como esperado teoricamente, o **Hill Climbing Simples** demonstrou ser inadequado, com uma taxa de sucesso de apenas **13.3%**. Isso ocorre porque o algoritmo (Alg. 1), em sua forma pura, fica "preso"no primeiro máximo local ou platô que encontra.

O **Hill Climbing com Movimentos Laterais** (Alg. 2) melhora esse resultado, alcançando **40.0%** de sucesso. Isso comprova que parte das "armadilhas"são platôs que esta variação consegue atravessar. Por fim, o **Reinício Aleatório** foi o único a obter **100%** de sucesso, confirmando ser uma abordagem probabilisticamente completa.

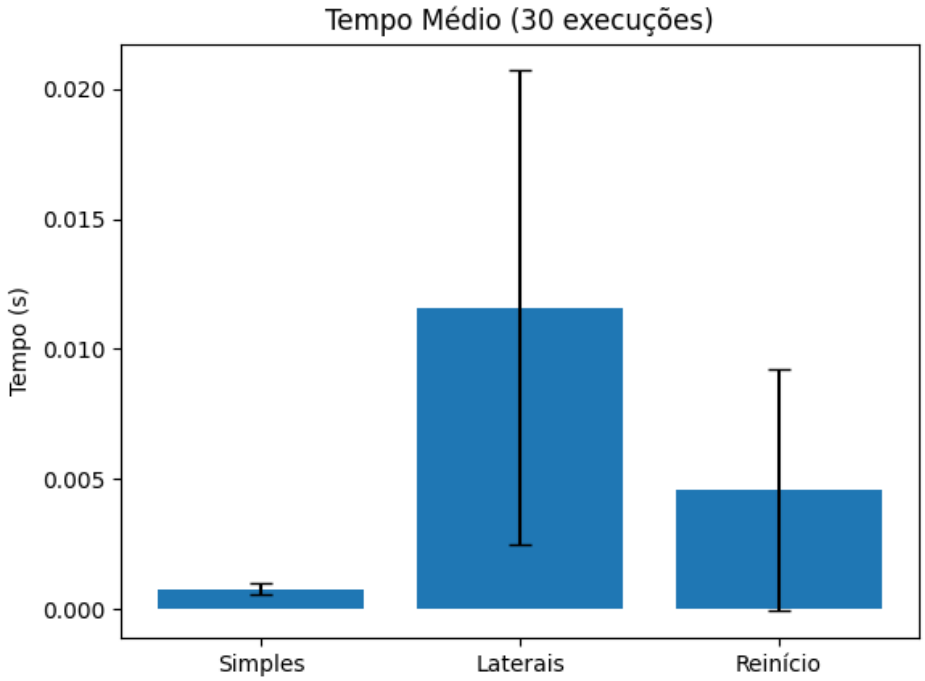

Figura 3: Tempo médio de execução com desvio padrão (N=30)

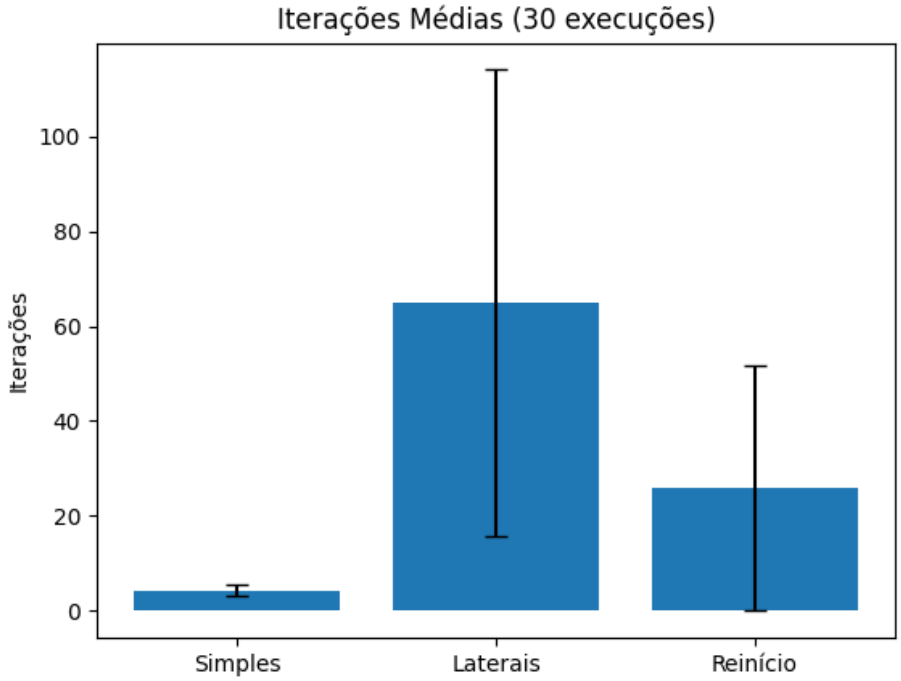

Figura 4: Iterações médias por variação com desvio padrão (N=30)

As Figuras 3 e 4 analisam o custo computacional e revelam um resultado contraintuitivo. O Hill Climbing Simples é o mais rápido (0.0008s) e com menos iterações (4.30),

pois ele "desiste"rapidamente.

Contrariando a intuição, o **Hill Climbing com Movimentos Laterais** foi o algoritmo mais custoso, tanto em tempo médio (**0.0116s**) quanto em iterações médias (**64.83**). O altíssimo desvio padrão (49.23 iterações) sugere que, quando ele encontra um platô, ele gasta muito esforço computacional atravessando-o.

O **Reinício Aleatório** emergiu como a solução mais equilibrada. Embora reinicie, o custo de cada tentativa é baixo (o mesmo do Simples, 4.30 iterações). O custo total é significativamente *menor* que o custo de atravessar platôs. Isso explica por que ele é mais rápido (**0.0046s**) que a versão 'Laterais', mesmo garantindo 100% de sucesso.

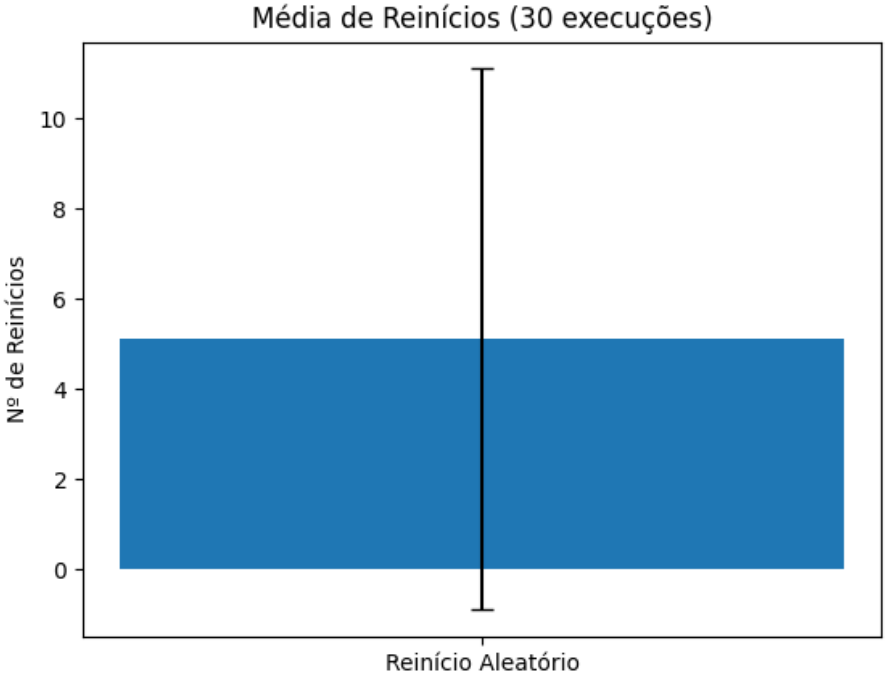

Figura 5: Média de reinícios para a versão "Reinício Aleatório"

A Figura 5 isola o custo do Reinício Aleatório. Em média, foram necessários apenas **5.10 reinícios** (com um alto desvio padrão de 6.01) para encontrar uma solução. Este é o custo direto pago pela completude do algoritmo.

Finalmente, a análise da execução de demonstração ('seed=42') confirmou esta dinâmica perfeitamente. O tabuleiro inicial (10 conflitos) é um platô ou máximo local que "prendeu"o Hill Climbing Simples (falhou com 1 conflito). A versão com Movimentos Laterais conseguiu escapar (8 iterações) e encontrar a solução. A versão 'Reinício' (5 iterações) também encontrou a solução, demonstrando um caminho rápido.

5 Conclusão

Este trabalho implementou e avaliou experimentalmente três variações do algoritmo Hill Climbing no problema das 8 Rainhas. Os resultados estatísticos confirmaram as propriedades teóricas de cada abordagem.

O Hill Climbing Simples (Alg. 1), embora rápido, é inadequado para este problema devido à sua incapacidade de escapar de máximos locais e platôs, apresentando uma taxa de sucesso de apenas **13.3%**. A adição de Movimentos Laterais (Alg. 2) aumenta a robustez (taxa de sucesso de **40.0%**), mas se mostrou a estratégia computacionalmente mais cara (0.0116s).

O Hill Climbing com Reinício Aleatório provou ser a abordagem ideal, atingindo **100%** de taxa de sucesso e sendo mais rápido (0.0046s) que a versão com movimentos laterais. O custo dessa completude foi quantificado em uma média de **5.10 reinícios** por solução.

O estudo evidencia que, para este espaço de busca, a estratégia de "falhar rápido e reiniciar"(Random-Restart) é superior à estratégia de "exploração exaustiva de platôs"(Movimentos Laterais).

Referências

[1] RUSSELL, S.; NORVIG, P. *Inteligência Artificial: Uma Abordagem Moderna*. 4ª ed. Pearson, 2021.

[2] AIMA-Python. *Repositório de Implementação dos Algoritmos de "AI: A Modern Approach"*. Disponível em: `https://github.com/aimacode/aima-python`. Acesso em: 29 out. 2025.

[3] SANTOS, B. P.; SILVA, J. F. T. *Trabalho 2 IA: 8 Rainhas com Hill Climbing*. Repositório de código. Disponível em: `https://github.com/joaofranciscoteles/Trabalhos-Busca-e-8-Rainhas-.git`. Acesso em: 22 out. 2025.

Créditos e Declaração de Autoria

Autores e Papéis

A divisão de tarefas para a elaboração deste trabalho foi a seguinte: (Conforme o Sr. solicitou, apenas a estrutura está aqui para o Sr. preencher os nomes que preferir)

- **[Autor 1 - Bruno Prado dos Santos]**:
  - Implementação do módulo `run_analysis.py`
  - Elaboração do `README.md`
  - Redação do Relatório (`.tex`)
- **[Autor 2 - João Francisco Teles da Silva]**:
  - Implementação do módulo `run_analysis.py`
  - Implementação do módulo `eight_queens.py`
  - Implementação do módulo `hill_climbing.py`

Uso de Inteligência Artificial

Declaramos que ferramentas de Inteligência Artificial (Gemini, Google) foram utilizadas para auxiliar nas seguintes tarefas:

- Revisão textual e gramatical do relatório.
- Geração de comentários de documentação (docstrings) para o código Python.

Nenhuma parte da lógica central de implementação dos algoritmos de busca (`hill_climbing.py`) ou da representação do problema (`eight_queens.py`) foi gerada por IA.

Recursos Externos

- AIMA-Python [2]: Consultado para referência de implementação e boas práticas em Python.
- Russell & Norvig [1]: Livro-texto base para os conceitos teóricos das variações do Hill Climbing.

Declaração

Confirmamos que o código e o relatório entregues foram desenvolvidos pela equipe, respeitando as políticas de integridade acadêmica da disciplina.