Inteligência Artificial — Trabalho 2

Análise Comparativa de Algoritmos Hill Climbing para o Problema das 8 Rainhas

Rafael Adolfo Silva Ferreira
Engenharia de Computação
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)

Introdução

O problema das Oito Rainhas é um quebra-cabeça clássico em ciência da computação, que consiste em posicionar oito rainhas em um tabuleiro 8x8 de forma que nenhuma ameace a outra. Por ser um problema de otimização combinatória, ele serve como um excelente caso de estudo para algoritmos de busca local. Tais algoritmos exploram o espaço de estados movendo-se de uma solução para uma vizinha, buscando otimizar uma função objetivo — neste caso, minimizar o número de conflitos entre as rainhas [1].

Este relatório detalha a implementação e a análise comparativa de duas variações do algoritmo **Hill Climbing** aplicadas a este problema: **Hill Climbing com Reinício Aleatório** e **Hill Climbing com Movimentos Laterais**, ambas estratégias conhecidas para mitigar o problema fundamental do Hill Climbing de ficar preso em mínimos locais [1]. O objetivo é avaliar a eficácia, a eficiência e a robustez de cada abordagem.

Metodologia

A metodologia foi dividida em três etapas: implementação e testes, execução dos experimentos para coleta de dados e, por fim, a análise estatística dos resultados.

Estrutura e Implementação do Código

A solução foi desenvolvida em Python com uma arquitetura modular, dividida em múltiplos arquivos para garantir a separação de responsabilidades, a testabilidade e a reprodutibilidade do projeto:

- **`eight_queens.py`**: Define o modelo do problema. Contém a representação do tabuleiro, a função de custo (`conflicts`) e a função que gera estados sucessores (`neighbors`). É o núcleo que descreve o ambiente.
- **`hill_climbing.py`**: Implementa a lógica dos algoritmos de busca. As funções `random_restart_hill_climbing` e `hill_climbing_with_sideways_moves` utilizam as definições do problema para navegar pelo espaço de estados. A implementação segue os conceitos descritos por [1].
- **`main.py`**: Orquestra o experimento. Este script importa os algoritmos, executa cada um 100 vezes, cronometra o desempenho e coleta as métricas de sucesso e passos. Ao final, exporta os dados brutos para um arquivo CSV (`results2.csv`).
- **`test_eight_queens.py`** e **`test_hill_climbing.py`**: Asseguram a correção do código através de testes unitários. `test_eight_queens.py` valida se a função de conflitos e a geração de vizinhos

estão corretas. `test_hill_climbing.py` verifica se os algoritmos de busca se comportam como esperado, por exemplo, garantindo que encontram uma solução quando devem.

- **`analysis.py`**: Realiza a análise pós-experimento. Este script lê o arquivo CSV gerado pelo `main.py`, calcula as estatísticas agregadas (média, desvio padrão, etc.) e gera os gráficos (boxplots e barras) utilizados neste relatório para a visualização dos resultados.

Condução dos Experimentos

Para a análise de desempenho, cada um dos dois algoritmos foi executado 100 vezes de forma independente. Foi utilizada uma semente pseudoaleatória (`seed = 42`) no script principal para garantir a reprodutibilidade dos testes. Foram coletadas as seguintes métricas:

- **Sucesso**: um valor booleano indicando se uma solução (0 conflitos) foi encontrada.
- **Tempo de Execução**: o tempo total em segundos para a conclusão.
- **Passos**: o número de reinícios aleatórios ou movimentos laterais realizados.

Os limites para os algoritmos foram definidos em 100 reinícios para o **Reinício Aleatório** e 100 movimentos laterais consecutivos para a outra variação.

Resultados e Análise

Os dados coletados nas 100 execuções de cada algoritmo foram agregados para análise estatística, cujos resultados estão resumidos na Tabela 1.

Tabela 1: Análise comparativa do desempenho dos algoritmos. O tempo e os passos médios referem-se apenas às execuções bem-sucedidas.

Algoritmo	Taxa de Sucesso (%)	Falhas	Tempo Médio (ms)	Desvio Padrão (Tempo)	Passos Médios	Desvio Padrão (Passos)	Soluções Sortudas
Reinício Aleatório	100.0%	0	7.08	6.58	6.54	6.03	0
Movimentos Laterais	96.0%	4	8.91	9.31	13.25	18.20	15

Análise da Taxa de Sucesso e Robustez

A robustez dos algoritmos apresentou uma diferença clara, conforme a Tabela 1. O **Reinício Aleatório** demonstrou ser totalmente confiável, alcançando uma taxa de sucesso de 100% e não registrando nenhuma falha. Em contrapartida, o algoritmo de **Movimentos Laterais** falhou em 4 das 100 execuções, resultando em uma taxa de sucesso de 96%. Embora alta, essa taxa indica que a estratégia de explorar platôs não é infalível e pode, eventualmente, ficar presa em mínimos locais dos quais não consegue escapar dentro do limite de movimentos.

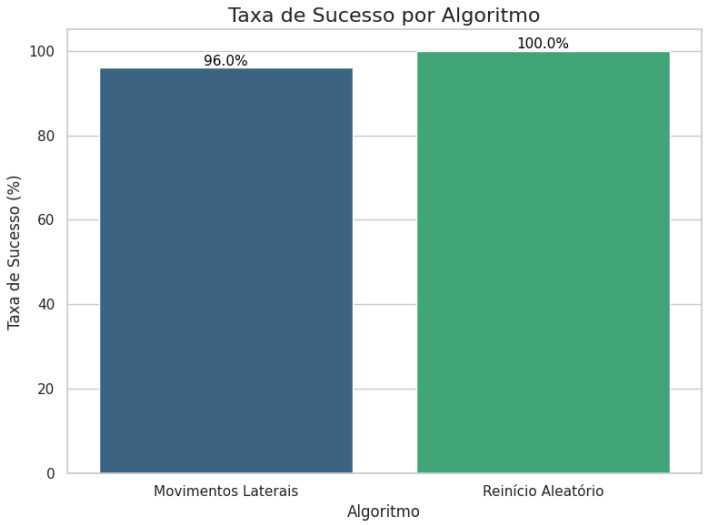

Figura 1: Taxa de sucesso de cada algoritmo em 100 execuções.

Análise do Tempo de Execução

Em termos de eficiência, o **Reinício Aleatório** foi, em média, mais rápido, com um tempo médio de 7.08 ms por solução encontrada, enquanto os **Movimentos Laterais** levaram 8.91 ms. O desvio padrão do tempo também foi menor para o Reinício Aleatório (6.58 ms contra 9.31 ms), sugerindo que, além de mais rápido, seu desempenho é mais consistente, como pode ser observado na Figura 1.

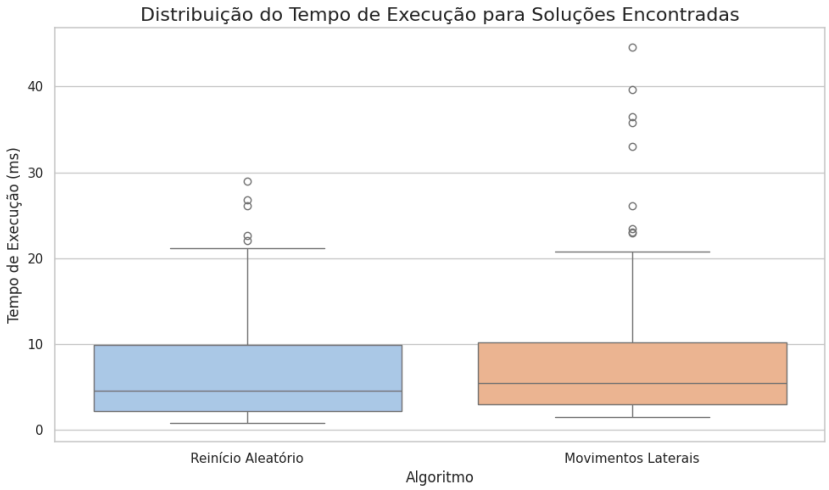

Figura 2: Distribuição do tempo de execução (ms) para encontrar uma solução.

Análise de Passos e Eficiência da Busca

A análise dos passos (Figura 2) e das "soluções sortudas" revela a dinâmica mais interessante deste experimento. O algoritmo de **Movimentos Laterais** precisou, em média, de 13.25 movimentos laterais para encontrar uma solução. Notavelmente, em 15 execuções, ele encontrou a solução sem precisar de **nenhum** movimento lateral (`passos = 0`). Essas 15 "soluções sortudas" indicam que o caminho a partir do estado inicial era uma "descida" direta até a solução, sem ficar preso em platôs.

O resultado do **Reinício Aleatório** é o mais surpreendente: ele registrou zero soluções sortudas. Isso significa que em **nenhuma** das 100 execuções o estado inicial levou diretamente

a uma solução. O sucesso de 100% do algoritmo foi, portanto, inteiramente dependente de sua principal estratégia: abandonar um caminho desfavorável e reiniciar a busca. O número médio de 6.54 reinícios por solução mostra que essa tática não só é necessária, como também é altamente eficiente.

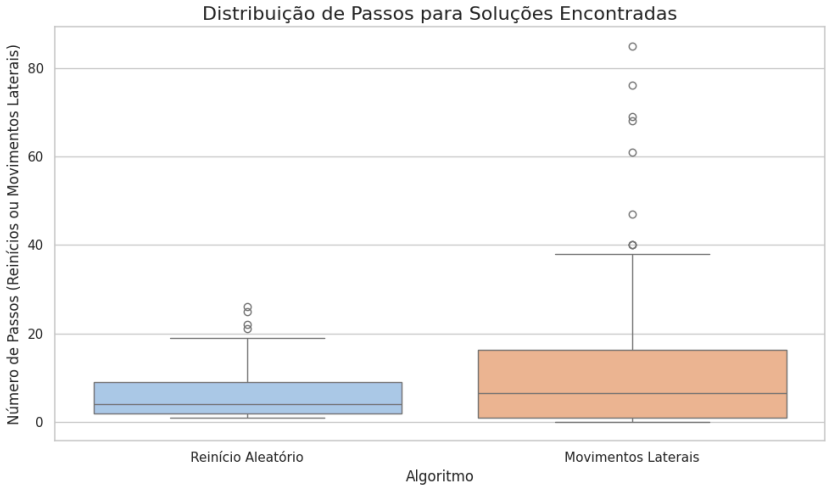

Figura 3: Distribuição do número de passos (reinícios ou movimentos laterais) até a solução.

Conclusão

Ambos os algoritmos foram capazes de resolver o problema das Oito Rainhas, mas a análise comparativa aponta o **Hill Climbing com Reinício Aleatório** como a estratégia superior para este cenário. Ele se mostrou mais robusto (100% de sucesso), mais rápido (7.08 ms em média) e mais eficiente em sua busca (6.54 reinícios em média).

O achado mais significativo foi que o sucesso do Reinício Aleatório não dependeu de sorte no estado inicial — ele nunca encontrou uma solução na primeira tentativa. Sua eficácia deriva puramente da força de sua estratégia de diversificação (reiniciar), que se provou mais vantajosa do que a estratégia de intensificação (explorar platôs com movimentos laterais), que, apesar de funcional, mostrou-se mais lenta e suscetível a falhas.

Créditos e Declaração de Autoria

- **Uso de IA:** A ferramenta Gemini (Google) foi utilizada como assistente de programação para gerar o código de análise de dados e os gráficos, e para auxiliar na estruturação e revisão textual deste relatório. Nenhuma parte da lógica central dos algoritmos de Hill Climbing foi gerada por IA.
- **Declaração:** Confirmo que o código entregue foi desenvolvido por mim, respeitando as políticas da disciplina.
- **Repositório:** https://github.com/Radsfer/ia-lab1.git .

Referências

[1] S. Russell e P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. em Pearson Series in Artifical Intelligence. Pearson, 2020.