Centro Federal de Educação Tecnológica de Minas Gerais

Engenharia da Computação
Disciplina: Inteligência Artificial

Trabalho 02:

Problema das 8 Rainhas

Alunos: Eduardo Henrique Queiroz Almeida e Joaquim Cézar Santana da Cruz
Professor: Tiago Alves de Oliveira

Divinópolis/MG
outubro de 2025

Conteúdo

1 Introdução **2**

2 Fundamentação Teórica **2**

3 Metodologia Aplicada **4**

4 Análise Comparativa (Resultados) **5**

5 Discussão **8**

6 Divisão de Tarefas e uso de *IA* **10**

7 Conclusão **10**

1 Introdução

Este trabalho apresenta a implementação e análise comparativa de duas variações do algoritmo Hill Climbing aplicadas ao Problema das 8 Rainhas. O objetivo principal é explorar diferentes estratégias para superar ótimos locais e platôs, comuns em algoritmos de busca local. Especificamente, comparamos o desempenho da abordagem que permite movimentos laterais limitados com a técnica de reinícios aleatórios (Random-Restart).

O problema das 8 Rainhas, que consiste em posicionar 8 rainhas em um tabuleiro de xadrez 8x8 de forma que nenhuma delas ataque qualquer outra, é um exemplo clássico de problema de otimização combinatória. Neste estudo, ele foi abordado utilizando técnicas de busca local para encontrar uma configuração válida.

2 Fundamentação Teórica

Problema das 8 Rainhas

O Problema das 8 Rainhas é um desafio matemático conhecido que busca arranjar 8 rainhas em um tabuleiro de xadrez 8x8 de tal maneira que nenhuma rainha possa atacar outra. No xadrez, as rainhas se movem e atacam em linha reta nas direções horizontal, vertical e diagonal [3].

Formalmente, o problema pode ser modelado como um Problema de Satisfação de Restrições (CSP). Neste modelo, existem 8 variáveis, uma para cada coluna (C0 a C7). O domínio para cada variável são as 8 possíveis linhas (0 a 7). As restrições impostas são que nenhuma rainha pode ocupar a mesma linha ou a mesma diagonal que qualquer outra rainha.

Representação do Estado

Para implementar os algoritmos de busca, o estado do tabuleiro foi representado como um vetor (lista em Python) de 8 posições. Cada índice do vetor corresponde a uma coluna do tabuleiro (0 a 7), e o valor armazenado nesse índice indica a linha (0 a 7) onde a rainha daquela coluna está posicionada.

```
1    Board = list[int]
2    # [linha_coluna0, linha_coluna1, ..., linha_coluna7]
3    # Exemplo: [3, 7, 1, 4, 6, 0, 2, 5] significa que a rainha da
         coluna 0 esta na linha 3,
4    # a da coluna 1 na linha 7, e assim por diante.
```

Listing 1: Representação do estado

Função de Avaliação

A qualidade de um estado (uma configuração do tabuleiro) é medida pela função de avaliação, que, neste caso, conta o número de pares de rainhas que estão em conflito. Um conflito ocorre entre duas rainhas (r_1, c_1) e (r_2, c_2) se elas estão na mesma linha $(r_1 = r_2)$ ou na mesma diagonal $(|r_1 - r_2| = |c_1 - c_2|)$. O objetivo do algoritmo é minimizar essa contagem, alcançando a solução ótima quando o número de conflitos é zero.

Algoritmo Hill Climbing

O Hill Climbing é um algoritmo de busca local que opera de forma iterativa, tentando melhorar a solução atual movendo-se para um estado vizinho de melhor qualidade (menor custo). O processo básico envolve gerar os vizinhos do estado atual, avaliar cada um deles usando a função de avaliação, escolher o vizinho com o melhor custo (o menor, neste caso) e torná-lo o novo estado atual. O algoritmo para se não houver nenhum vizinho melhor que o estado atual [1].

Ótimos Locais e Platôs

O Hill Climbing básico sofre de algumas limitações conhecidas. Um **ótimo local** é um estado que é melhor que todos os seus vizinhos, mas não representa a melhor solução global possível. Quando o algoritmo atinge um ótimo local, ele para, pois não encontra nenhum movimento que melhore a situação[1]. A figura 1 representa visualmente a ideia de ótimos locais.

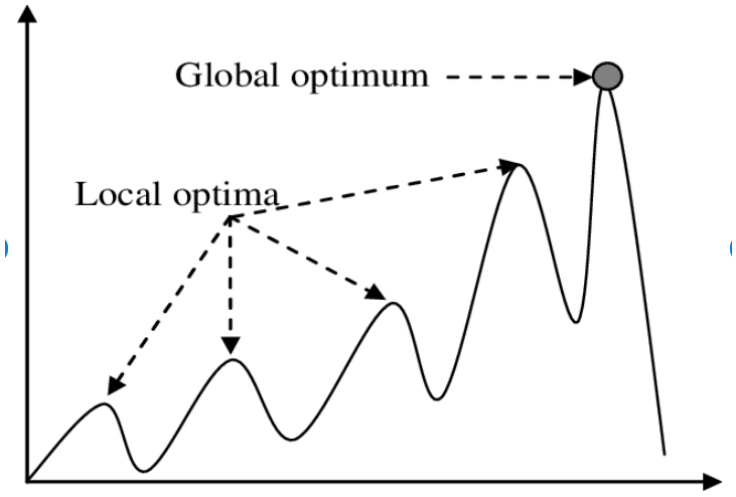

Figura 1: Representação de ótimos (máximos) locais. Estado cujo algoritmo se encontra preso em sua posição atual, pois não tem caminho de melhor custo para progredir.[4]

Outro problema são os **platôs**, regiões no espaço de busca onde vários estados vizinhos têm o mesmo valor de avaliação que o estado atual. Se não houver vizinhos estritamente melhores, o algoritmo também pode parar prematuramente em um platô, mesmo que a solução ótima ainda não tenha sido alcançada.

Estratégias de Escapamento

Para lidar com ótimos locais e platôs, diversas estratégias podem ser incorporadas ao Hill Climbing.

A estratégia de **Movimentos Laterais** permite que o algoritmo se mova para um vizinho que tenha o mesmo custo que o estado atual. Para evitar loops infinitos em platôs, geralmente se impõe um limite no número de movimentos laterais consecutivos permitidos.

A técnica de **Reinícios Aleatórios (Random-Restart)** aborda o problema de ótimos locais de forma diferente. Quando o algoritmo Hill Climbing padrão fica preso

(não encontra melhorias), ele é reiniciado a partir de um novo estado inicial gerado aleatoriamente. O processo é repetido várias vezes, e o melhor resultado encontrado em todas as tentativas é mantido. Esta abordagem aumenta significativamente a probabilidade de encontrar a solução global e torna o algoritmo completo probabilisticamente.

3 Metodologia Aplicada

Operações Implementadas

A implementação se baseou em funções essenciais para manipular o tabuleiro e avaliar os estados. Sendo elas:

- A **função de conflitos** foi implementada para calcular o número total de pares de rainhas em ataque mútuo, com uma complexidade de tempo quadrática ($O(n^2)$, onde n=8);
- A **geração de vizinhos** foi feita através de um generator em Python, que produz iterativamente todos os movimentos possíveis a partir de um estado (mover uma rainha para uma linha diferente em sua coluna), resultando em $8 \times 7 = 56$ vizinhos por estado;
- A função de **aplicação de movimentos** cria uma cópia do tabuleiro atual e modifica a posição de uma rainha conforme o movimento especificado, garantindo a imutabilidade do estado original.

Implementação das Variações do Hill Climbing

A primeira variação, **Hill Climbing com Movimentos Laterais**, prioriza movimentos que reduzem o número de conflitos. Caso não existam movimentos melhores, ela permite até 10 movimentos laterais consecutivos (para vizinhos com o mesmo custo). O algoritmo para se não houver movimentos melhores ou laterais possíveis dentro do limite, ou se atingir o máximo de 1000 iterações totais.

A segunda variação, **Hill Climbing com Random-Restart**, executa repetidamente o Hill Climbing básico (neste caso, sem movimentos laterais permitidos, conforme Tabela 1). Cada execução (tentativa) começa a partir de um novo estado inicial aleatório. O algoritmo armazena a melhor solução encontrada globalmente (menor número de conflitos) ao longo de todas as tentativas. Ele executa no máximo 100 reinícios, com cada tentativa limitada a 100 iterações.

Experimentos

Para comparar as duas abordagens, foram realizados dois experimentos principais, cada um consistindo em 100 execuções independentes para coletar dados estatísticos sobre o desempenho. Para reprodução dos experimentos a cada execução do script python, foi definida uma *random seed* no valor de 42. A Tabela 1 detalha os parâmetros específicos utilizados em cada experimento.

Tabela 1: CONFIGURAÇÃO DOS EXPERIMENTOS

Parâmetro	**Movimentos Laterais**	**Random-Restart**
Execuções	100	100
Max Iterações	1000	100 por reinício
Movimentos Laterais	10	0
Reinícios Máximos	0	100

4 Análise Comparativa (Resultados)

Desempenho Geral

Os resultados agregados das 100 execuções para cada variação do algoritmo são apresentados na Tabela 2. Esta tabela é a base para a análise de eficácia e eficiência, fornecendo as médias que serão contextualizadas pelos gráficos de distribuição na próxima seção. A configuração do experimento (`random.seed = 42`) garantiu que ambas as abordagens fossem testadas exatamente com os mesmos 100 tabuleiros iniciais, permitindo uma comparação justa.

Tabela 2: RESULTADOS COMPARATIVOS AGREGADOS (100 EXECUÇÕES)

Métrica	**Movimentos Laterais**	**Random-Restart**
Taxa de Sucesso	55.0%	100.0%
Tempo Médio por Execução	∼6.0 ms	∼16.0 ms
Passos Médios (Sucesso)	∼7.5 passos	∼22.0 passos (acumulados)
Laterais Médios (Sucesso)	∼3.5	0
Reinícios Médios (Sucesso)	0	∼5.9

Análise Detalhada

Uma análise mais profunda do desempenho é obtida através da visualização da distribuição dos resultados de cada uma das 100 execuções, conforme apresentado nos gráficos a seguir.

Análise da Taxa de Sucesso (Gráfico de Barras)

O Gráfico de Barras da Taxa de Sucesso (Figura 2) ilustra a **confiabilidade** de cada algoritmo. A diferença é gritante:

- **Random-Restart:** Atingiu 100% de sucesso. Conforme esperado, a estratégia de reiniciar a busca a partir de um novo estado aleatório, combinada com um número generoso de 101 tentativas (1 inicial + 100 reinícios), garantiu que uma solução fosse encontrada em todas as execuções. Isso confirma o Random-Restart como uma abordagem **completa** e **confiável** para este problema, superando a limitação fundamental do Hill Climbing simples (que foi usado como base em cada reinício, com 0 movimentos laterais).

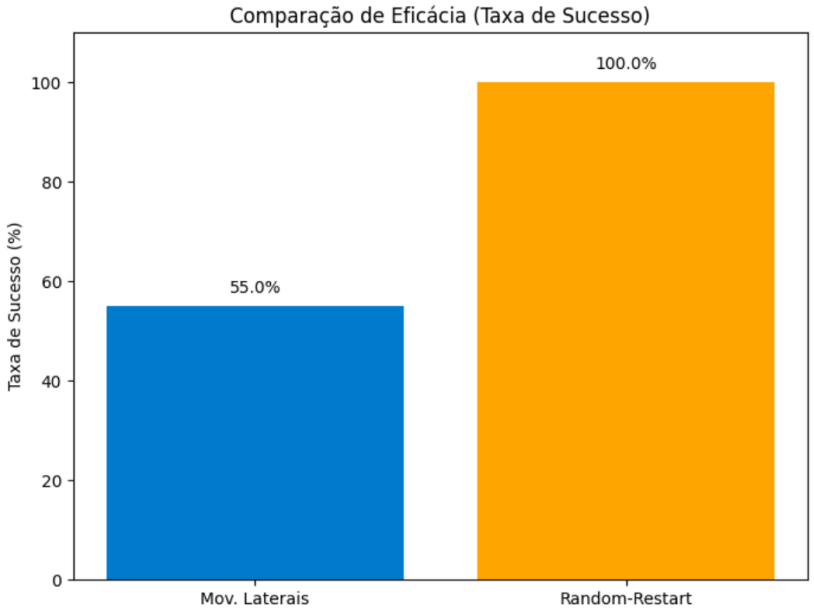

Figura 2: Comparação da Taxa de Sucesso (Eficácia)

- **Movimentos Laterais:** Atingiu apenas 55% de sucesso. Ao limitar a "paciência"do algoritmo a apenas 10 movimentos laterais consecutivos (`LATERAL_MOVES_LIMITS = 10`), sua limitação foi exposta: ele frequentemente fica preso em ótimos locais (platôs extensos ou mínimos locais). Os 45% de falha demonstram que, embora os movimentos laterais ajudem (a taxa de sucesso é bem maior que os ∼14% esperados do Hill Climbing simples), eles **não são uma garantia** de encontrar a solução global. Esta abordagem é, portanto, **não confiável**.

Análise do Tempo de Execução (Box Plot 1)

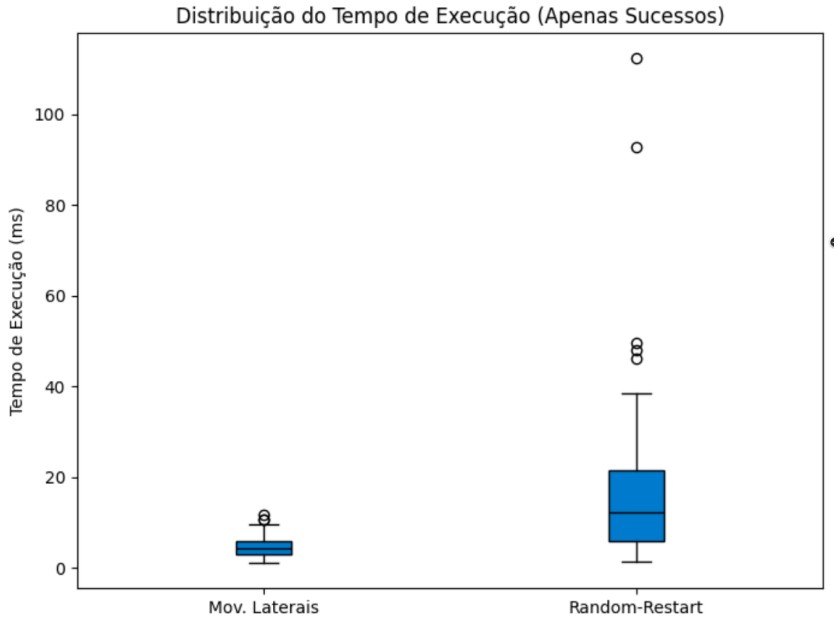

Figura 3: Distribuição do Tempo de Execução (ms) - Apenas Sucessos

O Box Plot do Tempo de Execução (Figura 3) revela a **eficiência em tempo real** das execuções que *encontraram* a solução.

- **Movimentos Laterais:** O box plot mostra uma distribuição **extremamente concentrada** em tempos baixos. A caixa interquartil (IQR), representando 50% dos dados centrais, é muito estreita, com a mediana (linha central) localizada bem abaixo de 5 ms. Existem alguns *outliers* (pontos individuais acima), indicando raras execuções bem-sucedidas que demoraram um pouco mais (até ∼12 ms). Enquanto o box plot nos mostra a **mediana** (o valor central, robusto a *outliers*) como sendo muito baixa, a **média** (∼6.0 ms, Tabela 2) é ligeiramente puxada para cima por essas execuções mais lentas. Isso sugere que, *quando funciona*, o Movimentos Laterais é **muito rápido e consistente** na maior parte do tempo.
- **Random-Restart:** O box plot mostra uma distribuição **muito mais dispersa**. A caixa IQR é significativamente mais larga, indicando maior variabilidade no tempo, e a mediana está claramente mais alta (visualmente em torno de 10-15 ms). Há vários *outliers*, incluindo alguns acima de 90 ms, como a Execução 82 nos logs (51 reinícios, 112 ms). A Tabela 2 reporta uma média de ∼16.0 ms, que é visivelmente maior que a mediana devido à influência desses casos lentos. A explicação para a lentidão e variabilidade é o **overhead das reinicializações**. O tempo total inclui o "trabalho desperdiçado"das tentativas que falharam. Algumas execuções tiveram sorte e encontraram a solução rápido (pontos baixos no gráfico), enquanto outras (os *outliers*) precisaram de muitas tentativas (em média ∼5.9 reinícios por solução, Tabela 2). Este é o **custo temporal da confiabilidade**.

Análise de Passos Totais (Box Plot 2)

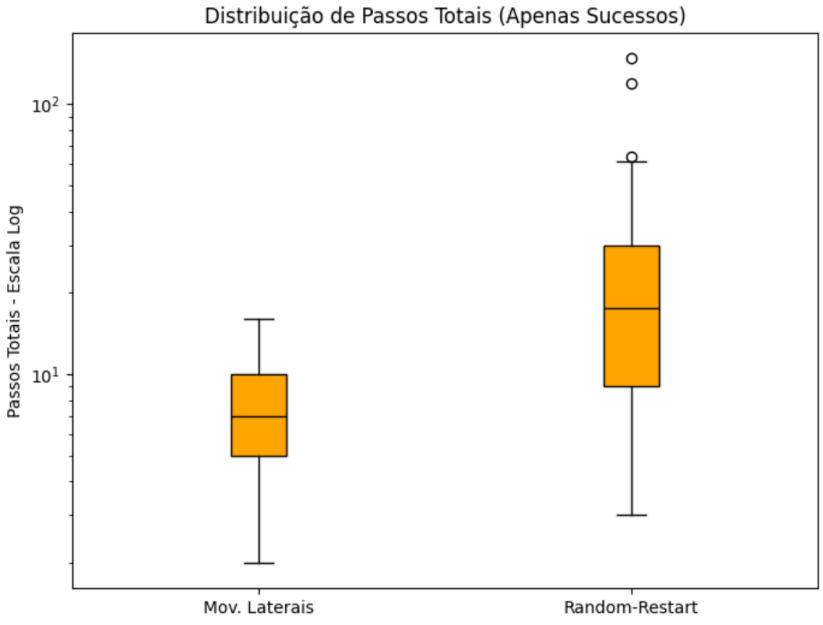

Figura 4: Distribuição de Passos Totais (Escala Log) - Apenas Sucessos

O Box Plot de Passos Totais (Figura 4, note a escala logarítmica no eixo Y) revela a **eficiência da busca** (complexidade algorítmica) das execuções *bem-sucedidas*.

- **Movimentos Laterais:** A Tabela 2 mostra uma média de ∼7.5 passos por solução. O gráfico de box plot confirma isso, com a caixa concentrada em valores baixos (mediana visualmente em torno de 6-7 passos). A métrica "Laterais Médios"(∼3.5, Tabela 2) é crucial: ela revela que aproximadamente 47% (∼3.5 / ∼7.5 × 100%) do esforço do algoritmo, mesmo nas execuções bem-sucedidas, foi gasto "andando de lado"em platôs. Isso mostra que o algoritmo, mesmo quando tem sucesso, ainda explora caminhos que não melhoram o custo.
- **Random-Restart:** A Tabela 2 mostra uma média de ∼22.0 passos acumulados por solução. O gráfico confirma a maior dispersão (caixa IQR mais larga) e a mediana significativamente mais alta (visualmente em torno de 20 passos). Este é o resultado mais paradoxal quando comparado ao gráfico de tempo: o Random-Restart é *mais lento em tempo* (Figura 3), mas também requer *mais passos totais* para garantir a solução. Cada reinício custa passos (mesmo que poucos, nas falhas rápidas do Hill Climbing simples), e esses passos se acumulam ao longo das ∼5.9 tentativas médias. A estratégia de "força bruta"encontra a solução, mas o **custo em termos de exploração total do espaço de estados é significativamente maior**.

Conclusão da Análise

A comparação revela um claro *trade-off*.

1. O **Random-Restart** é a estratégia superior em **confiabilidade**, garantindo 100% de sucesso. No entanto, essa garantia tem um custo: maior **tempo de execução médio** (devido ao *overhead* das reinicializações) e maior **complexidade de busca total** (maior número de passos acumulados, incluindo as tentativas falhas).
2. O **Movimentos Laterais**, com um limite de paciência restrito (10 movimentos), mostra-se **não confiável**, falhando em 45% das execuções. Contudo, *nas vezes em que encontra a solução*, ele o faz de forma **extremamente rápida** (baixo tempo médio) e com **menos passos totais** que o Random-Restart (pois não há passos "desperdiçados"de reinícios).

A escolha entre as duas abordagens dependeria do contexto da aplicação: se a garantia de encontrar a solução é primordial, o Random-Restart é a escolha óbvia, apesar de seu custo computacional. Se a velocidade média (considerando apenas os sucessos) é mais importante e uma taxa de falha é aceitável, o Hill Climbing com movimentos laterais (mesmo limitados) pode ser preferível.

5 Discussão

A comparação entre as duas estratégias de Hill Climbing, evidencia de forma contundente a eficácia de mecanismos de escape para superar as limitações da busca local, bem como a natureza do espaço de busca do problema das 8 Rainhas.

Confiabilidade: 55% vs. 100%

A descoberta mais significativa deste experimento reside na diferença drástica de confiabilidade, claramente visível na Figura 2.

O **Random-Restart** alcançou a taxa de sucesso esperada de **100%**, validando sua robustez teórica. Ao empregar o Hill Climbing simples (sem movimentos laterais) em cada tentativa — um algoritmo conhecido por sua baixa taxa de sucesso individual (∼14%) — a estratégia de permitir 101 tentativas independentes (1 inicial + 100 reinícios) eliminou estatisticamente a possibilidade de falha total. Isso demonstra sua capacidade inerente de escapar de **ótimos locais**, simplesmente abandonando buscas infrutíferas e começando de novo em outra região do espaço de estados.

Em contraste, a abordagem com **Movimentos Laterais**, limitada a apenas 10 movimentos consecutivos em platôs, obteve apenas **55%** de sucesso. Este resultado é fundamental: ele prova que, embora os movimentos laterais sejam úteis (a taxa ainda é muito superior aos ∼14% do Hill Climbing simples), uma "paciência"restrita de apenas 10 é **insuficiente** para navegar consistentemente pelos extensos platôs característicos do problema das 8 Rainhas. Os 45% de falha representam casos onde o algoritmo esgotou seus 10 movimentos laterais permitidos antes de encontrar um estado melhorador, ficando preso. Isso classifica esta abordagem, com este parâmetro específico, como **não confiável**.

Eficiência: Tempo vs. Passos e o Custo do Escape

A análise da eficiência, baseada nas Figuras 3 e 4 e na Tabela 2, revela o *trade-off* associado a cada estratégia de escape.

Movimentos Laterais (Quando Funciona): Nas 55 execuções bem-sucedidas, esta abordagem foi notavelmente **rápida em tempo real**. A Figura 3 mostra uma distribuição muito concentrada em tempos baixos (mediana abaixo de 5 ms), resultando em um tempo médio geral baixo (∼6.0 ms, Tabela 2). Em termos de passos, também foi eficiente, com uma média de apenas ∼7.5 passos por solução (Tabela 2). No entanto, a métrica de "Laterais Médios"(∼3.5) indica que quase metade (47%) desses passos foram laterais, confirmando que mesmo nos sucessos, o algoritmo dedicou um esforço considerável à exploração de platôs. A sua vantagem reside no baixo *overhead*: é uma única busca contínua.

Random-Restart (Sempre Funciona, Mas a Que Custo?): Embora 100% confiável, esta abordagem pagou um preço em eficiência. O tempo médio por execução foi significativamente maior (∼16.0 ms, Tabela 2). A Figura 3 explica o porquê: a distribuição de tempos é muito mais ampla, com uma mediana mais alta e vários *outliers* lentos, refletindo o **overhead das reinicializações**. O algoritmo gasta tempo não apenas buscando, mas reiniciando o processo (gerando tabuleiro, calculando custo inicial) em média ∼6.5 vezes por solução. O custo também é evidente no número de passos: a média de ∼25.0 passos *acumulados* por solução (Tabela 2 e Figura 4) inclui todo o "trabalho desperdiçado"das tentativas que falharam rapidamente ao ficarem presas. Cada reinício, embora necessário para a confiabilidade, adiciona um custo fixo de tempo e passos.

Impacto dos Parâmetros e Limitações

A escolha de *LATERAL-MOVES-LIMITS = 10* foi crucial para expor a fragilidade da abordagem de movimentos laterais e criar um contraste claro com o Random-Restart. Um valor ligeiramente maior (e.g., 20 ou 30) poderia ter melhorado a taxa de sucesso, mas possivelmente à custa de um tempo de execução maior nos casos difíceis, aproximando-a do Random-Restart e obscurecendo a comparação das estratégias de escape puras. O parâmetro *MAX-RESTARTS = 100* provou ser mais que suficiente, garantindo 100% de

sucesso, dado que a média de reinícios necessários foi de apenas ∼6.5.

As limitações deste estudo incluem:

- A exploração limitada do espaço de parâmetros (apenas um valor para *LATERAL-MOVES-LIMITS* foi analisado em detalhe).
- A dependência dos resultados da semente pseudoaleatória específica (‘seed=42‘).
- O número de execuções (100), embora razoável, poderia ser aumentado para maior confiança estatística, especialmente na taxa de sucesso dos Movimentos Laterais.
- A comparação não testou a combinação sinérgica de ambas as estratégias (Random-Restart *com* movimentos laterais permitidos em cada tentativa), que poderia ser uma via promissora para otimizar tanto a confiabilidade quanto a eficiência.

6 Divisão de Tarefas e uso de *IA*

Autores e Papéis

- **Eduardo Henrique Queiroz ALmeida:** Escrita do relatório e suporte no código.
- **Joaquim Cézar Santana da Cruz:** Implementação do código e suporte no relatório.

Uso de *IA*

- **Ferramenta(s)**: Google Gemini
- **Finalidade**: A ferramenta foi utilizada como assistente para a estruturação e revisão textual deste relatório. Ademais, ela foi usada para fazer a função de geração dos gráficos pela biblioteca *matplotlib* no código e para gerar comentários por todos os arquivos do projeto.
- **Declaração de conformidade**: Em conformidade com a política da disciplina, declaramos que nenhuma parte do código de implementação dos algoritmos do hill climbing (lateral moves, random start) foi usada utilizando IA. Para a implementação dos algoritmos, bem como compreensão do problema, foram utilizadas as seguintes referências: [5], [2].

7 Conclusão

Resultados Principais

A análise comparativa demonstrou um trade-off fundamental entre a confiabilidade e a eficiência das duas estratégias de Hill Climbing. A abordagem de Random-Restart provou ser superior em robustez, alcançando 100% de taxa de sucesso nas 100 execuções, confirmando sua capacidade de escapar de ótimos locais. No entanto, essa confiabilidade teve um custo: foi a abordagem mais lenta em média (∼16.0 ms por execução) e exigiu um número maior de passos acumulados (∼22.0), devido ao overhead das ∼5.9 reinicializações necessárias para encontrar a solução. Em contrapartida, a estratégia de Movimentos

Laterais, limitada a 10, revelou-se não confiável, com apenas **55% de sucesso**, indicando que o limite foi insuficiente para navegar pelos platôs do problema em 45% dos casos. Contudo, nos casos em que obteve sucesso, esta foi a abordagem mais rápida (∼6.0 ms) e computacionalmente mais leve, exigindo menos passos (∼7.5), ilustrando que, embora inconstante, ela é eficiente quando encontra um caminho viável.

Perspectivas Futuras

Como continuação deste trabalho, seria interessante implementar e comparar outras variações de busca local. Simulated Annealing por exemplo, oferece um mecanismo probabilístico para aceitar movimentos piores, proporcionando outra forma robusta de escapar de ótimos locais que poderia ser comparada em termos de velocidade e qualidade da solução. A expansão do problema para tabuleiros maiores (N-Queens) permitiria analisar como a escalabilidade afeta o desempenho relativo das estratégias implementadas, visto que a complexidade do espaço de busca cresce drasticamente com N. Adicionalmente, a aplicação de Algoritmos Genéticos representaria uma abordagem populacional contrastante com a busca local ponto a ponto aqui estudada. Uma análise mais detalhada da "paisagem de fitness"do problema das 8 Rainhas poderia ajudar a entender melhor por que certas configurações iniciais levam a ótimos locais difíceis e como as estratégias de escape navegam por essa paisagem.

Referências

[1] A. L. C. Carneiro. Algoritmos de otimização: Hill climbing e simulated annealing, 2020.

[2] L. F. da Silva Marian. Ia - algoritmos de busca local (hill climbing): Trabalho de faculdade ufmt em python — ia, 2023.

[3] P. Evandro Eduardo Seron Ruiz. O problema das oito rainhas, 2006.

[4] Q. Li. Efficient stochastic gradient search for automatic image registration. *National Institute of Advanced Industrial Science and Technology, Japan*, 2007.

[5] D. na Elétrica. Desafio das 8 rainhas - código comentado!, 2021.