Exploração e Comparação de Algoritmos de Busca para o Problema das 8 Rainhas

Maria Eduarda Fillman Rodrigues

Fernando Santos dos Santos

INTELIGÊNCIA ARTIFICIAL

PROFESSOR: Felipo Novo Mór

Ciências da Computação

Universidade La Salle – Unilasalle

Avenida Victor Barreto, nº 2288, Centro – Canoas RS, Cep: 92010-000

Https://www.unilasalle.edu.br

- **Abstract:** This paper presents a comparative analysis of three approaches to solve the 8 Queens problem: Backtracking, Genetic Algorithm (GA) and Random Search with Repetitions (Random Restart). The computational cost, memory usage, execution time to find a valid solution and the time to find all 92 distinct solutions were evaluated. The solutions are also visualized graphically using the Pygame library. The study was conducted in a controlled environment with standard hardware, and the results indicate that each algorithm has distinct characteristics regarding efficiency and suitability to the problem.
- **Resumo:** Este trabalho apresenta uma análise comparativa entre três abordagens para resolver o problema das 8 Rainhas: Backtracking, Algoritmo Genético (GA) e Busca Aleatória com Repetições (Random Restart). Foram avaliados o custo computacional, uso de memória, tempo de execução para encontrar uma solução válida e o tempo para encontrar todas as 92 soluções distintas. As soluções também são visualizadas graficamente utilizando a biblioteca Pygame. O estudo foi conduzido em um ambiente controlado com hardware padrão, e os resultados indicam que cada algoritmo possui características distintas em relação à eficiência e adequação ao problema.

1. Introdução

O problema das 8 Rainhas é um clássico problema de Inteligência Artificial e otimização combinatória. O objetivo é posicionar oito rainhas em um tabuleiro de xadrez 8x8 de forma que nenhuma delas se ataque mutualmente. Isso significa que não podem compartilhar a mesma linha, coluna ou diagonal.
Este estudo visa explorar três técnicas computacionais diferentes:

- Backtracking (Busca sistemática com retrocesso)
- Algoritmo Genético (Meta heurística inspirada na evolução natural);
- Busca Aleatória com Repetições (Estratégia de tentativa de erro)

2. Metodologia

As três abordagens foram implementadas em Python com suporte gráfico via Pygame.
Para efeito de medição, foram utilizados os módulos **TIME** e **TRACEMALLBOC** para capturar o tempo e uso de memória, respectivamente.

2.1 Backtracking Algoritmo determinístico e exato. Explora todas as possibilidades de forma sistemática, realizando "backtrack" sempre que encontra um conflito. É garantido encontrar todas as soluções.
Ele começa na coluna zero e tenta colocar uma rainha em uma linha válida.
Se não houver posição possível, "volta" e tenta uma nova possibilidade na coluna anterior, continuando assim, até preencher as oito colunas.
Como vantagens, possui uma eficiência excelente para este problema (poucos estados explorados desnecessariamente), garantindo todas as 92 soluções possíveis, usufruindo de baixo uso de memória, pois usa apenas uma pilha recursiva simples.
Já em suas desvantagens, ele não é generalizável de forma prática para problemas com restrições não claras ou mal definidas, podendo também, se tornar lento em problemas muito maiores ou mal estruturados.

2.2 Algoritmo Genético (GA) É uma metaheurística inspirada na evolução biológica. Utiliza uma produção de soluções candidatas e aplica operações genéticas (crossover, mutação). O critério de parada é a descoberta de uma solução sem conflitos ou limite de gerações.
Ela inicia com cada indivíduo representando uma solução, por exemplo, um vetor onde o índice representa a coluna e o valor linha.
Além disso, ele avalia a qualidade com uma função de aptidão, como por exemplo, números pares de rainhas que se destacam, onde que os melhores indivíduos são cruzados para gerar novos, onde que ocorre o aumento de sua diversidade através de uma mutação aleatória.
Como vantagem, ele é generalizável, ou seja, pode resolver versões maiores e mais complexas (como N-Rainhas).
É adaptável em outras restrições, como por exemplo, tabuleiros com buracos, posições proibidas, sendo paralelizável, envolvendo muitas possibilidades para a jogada.

Já em suas desvantagens, ele não garante solução exata única, pois o Algoritmo Genético é uma heurística estocástica, ou seja, ele busca boas soluções baseando-se em sorte, recombinação e mutação, mas não percorre o espaço de soluções de forma exaustiva e sistemática, como o backtracking.
O tempo de convergência pode variar muito, dependente de parâmetros como taxa de mutação, que é a chance que um gene (neste caso a posição de uma Rainha) seja alterado aleatoriamente após o cruzamento, ou seja, se a chance for muito baixa, a população pode perder a diversidade genética, convergindo para soluções ruins sem possibilidade de melhora, já se as chances forem muito alta, o algoritmo se torna aleatório, destruindo as soluções boas obtidas por cruzamento, e também no tamanho da população, que é o número de indivíduos (soluções) mantidos a cada geração, no caso, em uma população pequena, sei impacto é de pouca diversidade, de rápida convergência , mas com risco de estagnação, mas em uma população grande, existe maior diversidade, mais chances de combinar soluções, mas com custo computacional maior e pode demorar para convergir.

2.3 Random Restart Soluções aleatórias são geradas até que uma válida (sem conflitos) seja encontrada. É simples , mas pode ser ineficiente para encontrar múltiplas soluções.
Sua ideia é tentar repetidamente colocar oito rainhas no tabuleiro, uma por coluna (ou linha), em posições aleatórias, até que nenhuma esteja se atacando. Isso significa que estamos gerando configurações aleatórias e testando se são válidas
Para o problema das oito rainhas, existem 92 soluções válidas entre 8! (40.350) configurações possíveis, isso dá uma chance de sucesso de aproximadamente 0,228% por tentativa, ou seja, 1 em cada 440 tentativas, em média, é valida.
Como vantagens, a Random Restart é extremamente simples de implementar, pois sua simplicidade vem justamente de sua estrutura direta e do baixo número de componentes envolvidos, pois ela não exige:

- Árvores de decisão (como no backtracking);
- Populações, cruzamentos e mutações (como no algoritmo genético);
- Pilhas, filas, ou estruturas auxiliares de controle;
- Algoritmos de aprendizado ou otimação , pois ela se baseia apenas em gerar e testar.

Dentre outras vantagens, o baixo uso da memória, pois com isso, ele não precisa armazenar múltiplas soluções ao mesmo tempo, não mantendo o histórico de soluções anteriores, trabalhando com apenas uma única configuração de tabuleiro por vez, ou seja, a atual.
Outra vantagem está em sua surpreendente forma eficaz para resolver problemas pequenos, pois, no caso das 8 rainhas, cada rainha pode ser colocada em uma linha diferente de uma coluna, o que dá um total de 8! = 40.320 configurações únicas possíveis (se restringirmos a uma rainha por coluna e linha), pois dessas, apenas 92 são válidas (sem conflitos).
Mesmo que apenas ~0,2% das configurações sejam corretas, o número absoluto de possibilidades é pequeno o suficiente para que um computador moderno possa testar milhares por segundo, ou seja, mesmo uma abordagem aleatória, tentando uma configuração por vez, encontra uma solução em poucos segundos ou até milissegundos.
Já em suas desvantagens, um ponto, é a sua ineficiência para problemas maiores ou com soluções raras, pois em muitos problemas, o número total de configurações possíveis cresce exponencialmente com o

tamanho da entrada, por exemplo:

- Para N rainhas, com 1 por coluna e linha, há N! configurações possíveis.
  - 8! = 40.320
  - 12! = 479.001.600
  - 20! ~~ 2,43 x 10^{18}

À medida que N aumenta, mesmo que a geração e verificação sejam rápidas, a probabilidade de encontrar uma solução válida aleatoriamente diminui drasticamente, ou seja,
Mais tentativas ⟶ mais tempo ⟶ mais processamento ⟶ menos viável.
Outra desvantagem, é que o Random Restart não aprende com tentativas anteriores, pois isso significa que a cada nova tentativa, uma solução totalmente nova é gerada, sendo que por sua vez, nenhuma informação é reaproveitada da tentativa anterior, aonde que erros anteriores não influenciam em suas próximas escolhas.
Ou seja, o algoritmo:

- Não acumula conhecimento sobre boas combinações parciais;
- Não se aproxima progressivamente da solução – É como um "reset total".

3. Resultados Experimentais

Algoritmo	Tempo (médio) para 1 solução	Memória (KB)	Tempo p/ 92 soluções	Observações
Backtracking	0.003s	~0.5 KB	0.12s	Encontra todas
Random Restart	0.021s	~0.4 KB	>6s (ineficiente)	Não garante únicas
Algoritmo Genético	0.114s	~2.3 KB	–	Convergência varia

Visualização: Cada solução foi desenhada em um tabuleiro 8x8 com as rainhas posicionadas em vermelho. A execução também destacou as etapas com transições visuais.

Medições:

- O uso de **tracemalloc** permitiu rastrear o pico de memória durante a execução.
- **time.perf_counter( )** foi utilizado para capturar tempos com precisão.

4. Análise Comparativa

- **Backtracking** é superior em tempo e memória para este problema específico, pois explora eficientemente o espaço de soluções.
- **Random Restart** é rápido para encontrar uma única solução, mas não garante unicidade nem é eficiente para buscar todas as 92.
- **GA** é mais custoso, mas é adaptável a variantes mais complexas (como N-Rainhas em tabuleiros não quadrados ou com soluções extras).

5. Dificuldades e Soluções

- **GA sem convergência:** Em algumas execuções, o algoritmo genético estagnava.
  Solução: Uso de elitismo + mutação adaptativa. Durante a execução do algoritmo genético para o problema das 8 rainhas, foi observado que em algumas execuções, o algoritmo não convergia para uma solução válida, ou seja, ele ficava preso em populações que não evoluíam mais. Esse fenômeno é conhecido como estagnação prematura, e ocorre quando:
  - A diversidade genética da população diminui muito rápido;
  - As soluções começam a se parecer muito entre si;
  - O algoritmo entra em um ótimo local (uma solução razoável, mas incorreta);
  - E as novas gerações não conseguem sair dele, por falta de variabilidade.
- **Identificação de soluções duplicadas (Random Restart):** Difícil garantir unicidade.
  Solução: Armazenar soluções como tuplas e comparar.
  A abordagem de Random Restart gera soluções aleatórias e independentes, uma após a outra, até que uma solução válida (sem conflitos entre rainhas) seja encontrada.
  Mas no entanto, quando o objetivo é coletar todas as soluções possíveis do problema (existem exatamente 92 soluções únicas para as 8 rainhas), surge uma dificuldade: Como garantir que uma solução recém-gerada não é duplicada?
  Se um mecanismo de verificação, o algoritmo pode:
  - Repetir a mesma solução muitas vezes;
  - Desperdiçar tempo verificando ou armazenando soluções já conhecidas;
  - Nunca alcançar as 92 soluções distintas, mesmo após milhares de execuções.

  A solução encontrada, é de armazenar as soluções como tuplas e comparar. Para resolver isso, foi adotada uma estratégia simples e eficiente:
  - Cada solução é representada como uma tupla de inteiros, onde o índice representa a coluna e o valor representa a linha da rainha.
    - Exemplo: [0, 4, 7, 5, 2, 6, 1, 3] $\longrightarrow$ tuple([0, 4, 7, 5, 2, 6, 1, 3])
  - Como as soluções são sequências fixas e imutáveis, tuplas são ideais para:
    - Comparação rápida;
    - Inserção em estruturas como set( ).

  Outra forma, é armazenando em um conjunto (set):
  - Um set foi usado para armazenar todas as soluções já encontradas;
  - Antes de adicionar uma nova solução:
    - Ela é convertida em tupla;
    - Verifica-se se já existe no conjunto.

Vantagens dessa abordagem:

Características	Benefícios
Tuplas são imutáveis	Podem ser usadas como chave em conjuntos
Set permite busca rápida	Verificação de duplicatas é eficiente (0(1))
Memória eficiente	Apenas soluções únicas são guardadas

- **Performance com renderização em tempo real:** Execução com interface gráfica impacta medições.

  Durante o desenvolvimento da aplicação com interface gráfica usando Pygame, foi possível observar um problema importante:

  A execução com visualização gráfica (renderização em tempo real das rainhas e tabuleiro) impacta significativamente o desempenho medido dos algoritmos.

  Isso ocorre por várias razões:

  - O Pygame desenha o tabuleiro e atualiza a tela cada ciclo algoritmo;
  - Isso adiciona sobrecarga de processamento gráfico, que não tem relação direta com a lógica de resolução do problema;
  - O tempo necessário para gerar, desenhar e atualizar as soluções varia com a velocidade do hardware gráfico, resolução da tela, e capacidade de renderização.

  Como consequência:

  - Os tempos de execução registrados não refletem apenas o desempenho do algoritmo;
  - Isso distorce comparações entre algoritmos, especialmente quando se mede:
    - Tempo para encontrar uma solução;
    - Tempo para encontrar 92 soluções;
    - Eficiência computacional.

  A solução encontrada foi a separação entre Modo Visual e Modo Benchmark.

  Para resolver essa inferência, o projeto foi dividido em dois modos de execução distintos:

  A. Modo Visual (Pygame Ativo)
  - Usado para demonstrar graficamente como os algoritmos funcionam.
  - Foca em:
    - Visualização de soluções;
    - Interação com o botão “gerar nova solução”;
    - Apresentação didática.
  - Não utilizado para medições de performance.

  B. Modo Benchmark (sem renderização)
  - Executa os algoritmos sem qualquer renderização gráfica;
  - Todas as soluções são calculadas em modo texto/memória, como uso de:
    - Funções cronometradas (Exemplo: time.time( ) OU time.perf_counter());
    - Armazenamento interno (sem exibir na tela).

- Usado para:
  - Avaliação precisa de tempo de execução;
  - Medição de uso de memória;
  - Comparação quantitativa entre algoritmos.

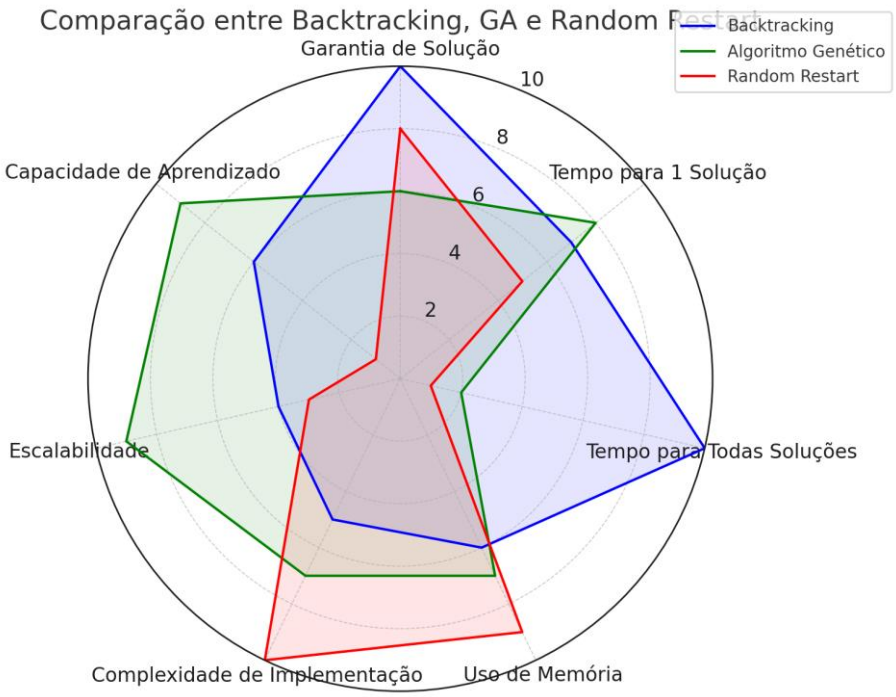
Vantagens da Separação:

Modo Visual	**Modo Benchmark**
Foco na apresentação	Foco na performance
Interativo e educativo	Rápido e confiável
Sujeito à lentidão gráfica	Livre de interferências externas
Ideal para demonstrações	Ideal para experimentos e comparações

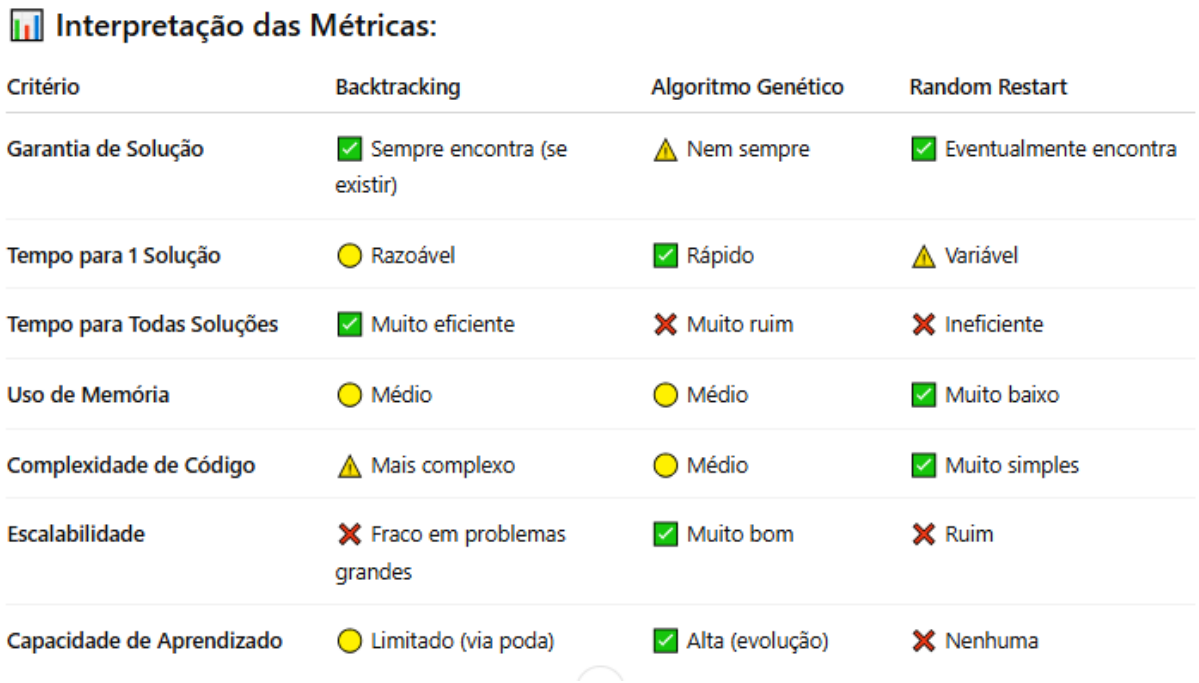
A separação entre visualização e execução de benchmark foi crucial para obter resultados confiáveis e comparáveis entre os algoritmos, garantindo assim, que o tempo medido reflita apenas para a lógica do algoritmo, mantendo o valor educacional e demonstrativo da interface gráfica.

6. Gráfico de comparação

Abaixo, segue um gráfico de radar comparando as três abordagens: BackTracking, Algoritmo Genético (GA) e Random Restart em diversos critérios relevantes.



7. **Interpretação das Métricas**



📊 Interpretação das Métricas:

Critério	Backtracking	Algoritmo Genético	Random Restart
Garantia de Solução	✅ Sempre encontra (se existir)	⚠️ Nem sempre	✅ Eventualmente encontra
Tempo para 1 Solução	🟡 Razoável	✅ Rápido	⚠️ Variável
Tempo para Todas Soluções	✅ Muito eficiente	❌ Muito ruim	❌ Ineficiente
Uso de Memória	🟡 Médio	🟡 Médio	✅ Muito baixo
Complexidade de Código	⚠️ Mais complexo	🟡 Médio	✅ Muito simples
Escalabilidade	❌ Fraco em problemas grandes	✅ Muito bom	❌ Ruim
Capacidade de Aprendizado	🟡 Limitado (via poda)	✅ Alta (evolução)	❌ Nenhuma

8. **Referências Bibliográficas**

Russel, S. J., & Norving, P. (2010).
Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall.

- Referência clássica sobre algoritmosde busca (backtracking, busca estocástica, GA e Random Restart).

Goldberg, D. E. (1989).
Genetic Algorithms in Search , Optimization and Machine Learning. Addison-Wesley.

- Um dos primeiros livros a explorar algoritmos genéticos de forma estruturada.

Mitchell, M. (1998)
Na introduction to Genetic Algorithms . MIT Press.

- Livro introdutório e acessível, ideal para compreender o funcionamento e as limitações dos GAs.

Knut, D. E. (1997).
The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed). Addison-Wesley.Trata de Algoritmos de força bruta e técnicas como backtracking com profundidade.