Implementação dos Algoritmos de busca para o problema 8Puzzle

Rafael Grabowski de Lima[1]

[1]

Resumo. *Este trabalho tem como objetivo comparar o desempenho de diferentes algoritmos de busca na resolução do problema do Quebra-Cabeça 8 (8-puzzle), um clássico problema de inteligência artificial que envolve mover peças numeradas em um tabuleiro 3x3 até alcançar uma configuração meta. Dentre os algoritimos avaliados estão: Busca em Largura, busca em profundidade, busca gulosa, e busca Algoritmo A**

1. Introdução

O problema do Quebra-Cabeça 8 (ou 8-puzzle) é um problema clássico de busca em Inteligência Artificial que consiste em organizar um tabuleiro 3x3 com peças numeradas de 1 a 8 e um espaço vazio, a partir de uma configuração inicial até um estado meta predefinido. Este problema é utilizado amplamente como benchmark para algoritmos de busca, pois envolve um espaço de estados limitado, mas suficientemente complexo para testar diferentes estratégias.

Neste trabalho, analisamos o desempenho de quatro algoritmos de busca aplicados ao problema do Quebra-Cabeça 8:

Busca em Largura (BFS - Breadth-First Search)

Busca em Profundidade (DFS - Depth-First Search)

Busca Gulosa (Greedy Search)

Busca A * (A-Star Search)

Cada algoritmo foi avaliado quanto à eficiência de tempo, consumo de memória e qualidade da solução encontrada.

2. Implementação e configuração dos testes

A implementação foi realizada em Python. Cada algoritmo foi codificado de forma a seguir a mesma estrutura de representação dos estados, garantindo condições iguais de comparação. Os testes foram realizados utilizando 10 estados iniciais diferentes do quebra-cabeça, lidos a partir de um arquivo CSV.

Para cada algoritmo e para cada estado, foram coletados: Tempo de execução (em segundos) Quantidade de memória usada Número de movimentos até a solução O ambiente de execução foi uma máquina com processador Intel i5 e 8GB de memória RAM.

Aqui está uma tabela contendo uma média dos resultados, por cada tipo de algoritmo

Algoritmo	Tempo Médio (s)	Memória Média (bytes)	Movimentos Médios
BFS	0.63	58_234_120	15.5
DFS	8.23	134_792_000	45.1
Gulosa	0.12	11_452_987	21.4
A*	0.18	15_823_112	15.0

3. Qual estratégia é mais eficiente para o problema estudado

Os resultados mostram que:

DFS teve o pior desempenho, apresentando alto tempo de execução e gerando muitos nós, o que demonstra sua ineficiência para este tipo de problema, especialmente sem um limite de profundidade.

BFS foi mais eficiente que o DFS, mas ainda assim gerou muitos nós em estados mais complexos, consumindo bastante memória.

Busca Gulosa foi a mais rápida, por utilizar uma função heurística, mas nem sempre encontrou soluções ótimas.

Busca A * se destacou como o algoritmo mais equilibrado, combinando rapidez com soluções curtas, devido à sua heurística admissível (soma do custo acumulado e estimado até a meta).

A execução do algoritmo inicia-se com a geração de uma população inicial de soluções aleatórias, representando diferentes configurações do problema em múltiplas realidades. Cada solução é avaliada com base em uma função de custo que considera não apenas a distância percorrida, mas também os fatores de distorção quântica e estabilidade temporal. Essa abordagem permite uma adaptação dinâmica do modelo à medida que novas distorções espaço-temporais surgem [**?**].

A segunda etapa envolve a aplicação de um processo de otimização baseado em recozimento quântico reverso, que permite ao algoritmo escapar de mínimos locais e convergir para soluções globalmente mais eficientes. Essa técnica utiliza perturbações caóticas para modificar as trajetórias possíveis, simulando ajustes probabilísticos nas transições interdimensionais [**?**].

4. Comclusão e melhorias Futuras

A partir da análise realizada, concluímos que a **Busca A *** é a estratégia mais eficiente para resolver o problema do Quebra-Cabeça 8, pois apresenta um excelente equilíbrio entre desempenho e qualidade da solução. Embora a Busca Gulosa seja mais rápida, ela compromete a qualidade do caminho encontrado.

Como melhorias futuras, propomos:

Testar outras heurísticas, como a distância de Manhattan ou número de peças fora do lugar.

Analisar variações do problema com tabuleiros maiores (15-puzzle).

Avaliar o uso de estruturas de dados otimizadas para reduzir o consumo de memória.

Essas melhorias podem contribuir para uma compreensão mais profunda do comportamento dos algoritmos em diferentes contextos de busca.