Comparação de Algoritmos de Busca na Resolução do Quebra-Cabeça 8-Puzzle

Rafael Luiz Cuypers

[1]Universidade Tuiuti do Paraná
Curitiba – PR

{rafael.cuypers@utp.edu.br}

Resumo. *Este programa implementa e compara quatro algoritmos de busca clássicos — Busca em Largura, Busca em Profundidade, Busca Gulosa e Busca A*, aplicados à resolução do problema do quebra-cabeça 8-puzzle, um jogo de tabuleiro deslizante que consiste em reorganizar peças numeradas de 1 a 8 até alcançar uma configuração final predefinida.*
A avaliação dos algoritmos é conduzida a partir de dez diferentes estados iniciais, utilizando quatro critérios principais de desempenho: número de passos necessários para atingir a solução, quantidade de nós explorados durante a busca, tempo total de execução e consumo de memória.
Os resultados obtidos são consolidados em uma tabela comparativa, permitindo uma análise clara e objetiva da eficiência relativa entre os métodos. Dessa forma, o programa proporciona uma abordagem e compreensão do comportamento dos algoritmos de busca, destacando especialmente os benefícios do uso de heurísticas na resolução de problemas de planejamento e tomada de decisão.

1. Introdução

O ”Quebra cabeça dos 8 números”é um problema clássico no campo da inteligência artificial e da ciência da computação, frequentemente utilizado como base para o desenvolvimento e análise de algoritmos de busca em espaços de estados. Trata-se de um tabuleiro de 3×3 contendo oito peças numeradas de 1 a 8 e um espaço vazio, representado pelo número 0. O objetivo do desafio é reorganizar as peças, a partir de uma configuração inicial qualquer, até alcançar uma configuração final em que os números estejam em ordem crescente e o espaço vazio esteja localizado no canto inferior direito.

Criado pelo matemático americano Samuel Loyd no final do século XIX, esse quebra-cabeça se tornou um objeto de estudo relevante tanto na teoria dos jogos quanto na inteligência artificial. Sua estrutura combinatória bem definida, com regras claras de movimentação, o torna ideal para testar e comparar a eficácia de diferentes algoritmos de busca.

As únicas ações permitidas envolvem mover o espaço vazio nas quatro direções principais (cima, baixo, esquerda e direita), desde que o movimento não ultrapasse os limites do tabuleiro. Isso gera uma estrutura de grafo no espaço de estados, onde cada configuração pode ter entre dois e quatro estados vizinhos, dependendo da posição do espaço vazio.

Neste cenário, os algoritmos de busca têm papel fundamental, permitindo a exploração sistemática e, muitas vezes, guiada por heurísticas, desse espaço de estados.

A avaliação da eficiência desses algoritmos pode ser feita com base em diversos fatores, como tempo de execução, uso de memória e a qualidade da solução (medida pelo número de movimentos necessários).

Neste trabalho, propõe-se a análise de quatro algoritmos de busca aplicados à resolução do quebra cabeça dos 8 números :

A busca em largura (Breadth-First Search, BFS) é um algoritmo que explora um grafo, ou seja, um conjunto de nós e arestas, visitando todos os nós acessíveis a partir de um nó inicial em ordem crescente de distância. Em outras palavras, ele primeiro explora todos os nós que estão a uma distância de 1 do nó inicial, depois todos os nós que estão a uma distância de 2, e assim por diante.

A busca em profundidade (DFS - Depth-First Search) é um algoritmo de travessia em grafos e árvores que explora o grafo o mais profundamente possível antes de voltar atrás e explorar outros ramos. Em outras palavras, tenta ”descobrir”um caminho até o fim de um ramo e depois, se necessário, volta para explorar outros caminhos.

A busca gulosa, ou Greedy Search, é uma técnica de busca em grafos que busca a solução ideal de um problema, tomando decisões baseadas em uma estimativa do quão próximo está do objetivo, sem se preocupar com as consequências futuras dessas decisões. Em outras palavras, a busca gulosa sempre escolhe a opção que parece ser a melhor no momento, sem olhar para o futuro e se arrepender da escolha.

O Algoritmo A* é um algoritmo de busca de caminho, ou seja, uma técnica para encontrar o caminho mais curto entre dois pontos em um grafo (uma estrutura que representa conexões entre objetos). Ele é uma combinação da Busca em Largura (Breadth-First Search) e do algoritmo de Dijkstra, utilizando uma função heurística para estimar a distância entre um nó e o ponto final.

O objetivo deste estudo é não apenas desenvolver esses algoritmos, mas também realizar uma análise comparativa empírica entre eles, utilizando diferentes instâncias do problema. Através dessa comparação, busca-se entender as principais vantagens e limitações de cada técnica de busca no contexto do 8-puzzle.

2. Descrição da implementação e configuração dos testes

O programa desenvolvido em python tem como objetivo comparar o desempenho de quatro algoritmos de busca aplicados ao problema do quebra-cabeça dos 8 números (8-puzzle), que consiste em reorganizar os números de 1 a 8 em uma matriz 3x3, deixando o espaço vazio (representado por 0) na última posição, conforme o estado objetivo: [1, 2, 3, 4, 5, 6, 7, 8, 0].

Foram implementadas as seguintes estratégias de busca: Busca em Largura (BFS): explora todos os estados em ordem de profundidade crescente. Busca em Profundidade (DFS): aprofunda ao máximo cada caminho antes de retroceder, com limite de profundidade de 50 movimentos para evitar loops infinitos. Busca Gulosa: utiliza a heurística da distância de Manhattan para escolher o próximo estado com menor custo estimado até o objetivo. Busca A*: combina a distância já percorrida com a heurística de Manhattan, priorizando estados com menor custo total estimado.

A função pegar-vizinhos foi criada para gerar os possíveis movimentos a partir de

um estado, movimentando o espaço vazio em até quatro direções (cima, baixo, esquerda e direita), desde que dentro dos limites do tabuleiro. A função calcular-distancia calcula a soma das distâncias de Manhattan de cada peça em relação à sua posição correta, sendo usada como heurística nos algoritmos Guloso e A*.

Para a realização dos testes, foram definidos 10 estados iniciais distintos com diferentes níveis de dificuldade, armazenados em uma lista chamada inicios. Para cada um desses estados, os quatro algoritmos foram executados, e os seguintes dados foram coletados:

Número de passos: quantidade de movimentos realizados do estado inicial até o estado objetivo (tamanho do caminho da solução).

Número de nós expandidos: quantidade de estados explorados durante a busca.

Tempo de execução: medido com a biblioteca time, em segundos.

Uso de memória: capturado com a biblioteca tracemalloc, em kilobytes, considerando o pico de memória utilizado durante a execução de cada algoritmo.

Ao final, os resultados são organizados e exibidos em formato de tabela no terminal, permitindo a comparação entre os algoritmos em relação à eficiência (tempo e memória), profundidade da solução e quantidade de estados explorados. Isso possibilita uma análise clara sobre as vantagens e limitações de cada abordagem no contexto do problema do 8-puzzle.

3. Resultados

Comparação dos algoritmos para o 8-Puzzle. (-1 indica que a solução não foi encontrada.)

Nº	Largura				Profundidade				Gulosa				A*			
	Passos	Nós	Tempo	Mem.	Passos	Nós	Tempo	Mem.	Passos	Nós	Tempo	Mem.	Passos	Nós	Tempo	Mem.
1	2	4	0.0000	2.98	50	20355	0.2327	4732.00	2	3	0.0000	0.78	2	3	0.0000	0.78
2	4	25	0.0000	7.62	50	10662	0.1062	1472.69	4	5	0.0000	1.26	4	5	0.0000	1.42
3	6	123	0.0000	38.12	50	15118	0.1698	1961.14	6	7	0.0000	2.31	6	8	0.0000	2.54
4	8	253	0.0000	78.33	50	13081	0.1526	1738.70	8	24	0.0010	7.34	8	14	0.0010	3.59
5	6	57	0.0010	15.49	48	8806	0.0947	1268.16	6	7	0.0000	2.06	6	7	0.0010	2.06
6	6	99	0.0010	33.72	6	123818	1.3976	17392.58	6	7	0.0000	2.81	6	12	0.0000	4.15
7	8	203	0.0000	73.48	8	140587	1.5995	19218.66	8	9	0.0000	3.24	8	14	0.0000	4.66
8	3	15	0.0000	3.89	-1	62744	0.6955	8680.65	3	4	0.0000	1.15	3	4	0.0000	1.15
9	13	2374	0.0390	988.03	49	84401	0.9655	14543.93	13	14	0.0000	4.78	13	21	0.0000	7.38
10	13	3350	0.0680	1332.65	-1	103945	1.2032	15211.34	37	400	0.0000	205.90	13	74	0.0174	24.17

4. Discussão dos Resultados

Busca em Largura (BFS):

Tempo: O tempo de execução para BFS é muito baixo, variando entre 0.0000s e 0.0390s, o que confirma que o algoritmo é rápido na maioria das instâncias.

Memória: O uso de memória é relativamente baixo, variando entre 2.98 KB e 988.03 KB, embora em casos mais difíceis (como o Nº 9) o uso de memória possa ser mais alto.

Conclusão: Apesar de ser eficiente em termos de tempo, o uso de memória pode ser um problema em instâncias mais complexas, como mostrado na tabela.

Busca em Profundidade (DFS):

Tempo: O tempo de execução pode ser mais longo, com valores entre 0.0947s e 1.5995s, dependendo da complexidade da instância.

Memória: O uso de memória para DFS é bem mais alto, variando de 1268.16 KB a 19,218.66 KB, principalmente em instâncias mais difíceis.

Conclusão: Embora o tempo de execução em alguns casos seja razoável, o uso elevado de memória e a falta de garantia de solução ótima são pontos negativos, como visto na tabela.

Busca Gulosa (Greedy):

Tempo: O tempo de execução é muito rápido, variando de 0.0000s a 0.0010s, o que mostra sua eficiência.

Memória: O uso de memória é extremamente baixo, variando de 0.78 KB a 7.34 KB.

Conclusão: A Busca Gulosa é muito eficiente em termos de tempo e memória, mas, como a tabela mostra, ela pode não fornecer a solução ótima, já que os resultados são subótimos em alguns casos.

A* (A-star):

Tempo: O tempo de execução do A* é semelhante ao da Busca Gulosa, variando entre 0.0000s e 0.0174s, o que demonstra alta eficiência.

Memória: O uso de memória do A* é ligeiramente maior que o da Busca Gulosa, variando entre 0.78 KB a 7.38 KB.

Conclusão: O A* é o mais equilibrado, garantindo sempre a solução ótima, com um bom desempenho tanto em tempo quanto em memória, como demonstrado na tabela.

A BFS é ótima para soluções ótimas, mas pode consumir muita memória. A DFS pode ser rápida, mas não garante a melhor solução e usa muita memória. A busca gulosa é rápida e leve, mas pode falhar na qualidade da solução. O A* é o mais eficiente e garante a solução ótima.

5. Conclusão

A partir da implementação e análise comparativa dos algoritmos de Busca em Largura (BFS), Busca em Profundidade (DFS), Busca Gulosa e A*, aplicados ao problema do 8-puzzle, foi possível compreender de forma prática as características, vantagens e limitações de cada abordagem. O algoritmo A* destacou-se como o mais eficiente e equilibrado, oferecendo soluções ótimas com rapidez e uso moderado de memória, graças à combinação entre o custo real do caminho e a heurística da distância de Manhattan. A Busca Gulosa, apesar de muito rápida e econômica em termos de memória, apresentou resultados subótimos em algumas instâncias, justamente por considerar apenas a heurística, sem levar em conta o custo acumulado. A BFS garantiu soluções corretas, porém com uso crescente de memória à medida que a complexidade aumentava. Já a DFS, mesmo apresentando tempos de execução razoáveis, foi a menos eficaz, tanto pela possibilidade de não encontrar soluções quanto pelo consumo elevado de memória, especialmente em casos mais difíceis.

Esses resultados demonstram a importância do uso de heurísticas bem projetadas na resolução de problemas de busca em espaços de estados, como o 8-puzzle. Os algoritmos que se baseiam em informações do problema, como A* e Busca Gulosa, geralmente

superam as abordagens cegas (BFS e DFS), tanto em eficiência quanto em qualidade da solução.

Como possíveis melhorias futuras, sugere-se a exploração de novas heurísticas, como o número de peças fora do lugar ou combinações heurísticas, que podem tornar a busca ainda mais eficaz. A otimização do uso de memória também é um ponto relevante, especialmente para algoritmos como BFS e DFS, que tendem a consumir muitos recursos em instâncias mais complexas. Outra sugestão é o desenvolvimento de uma interface gráfica que permita visualizar a execução dos algoritmos passo a passo, tornando o projeto mais didático e acessível. Além disso, seria interessante aplicar os algoritmos estudados a outros problemas clássicos de busca e planejamento, avaliando sua versatilidade. Por fim, técnicas como paralelização e geração de estados com dificuldade progressiva podem enriquecer ainda mais as análises futuras, tornando o estudo mais abrangente e aplicável a diferentes contextos da inteligência artificial.