Comparação de Algoritmos de Busca Aplicados ao Problema do 8-Puzzle

Luiz Augusto Manfron Matias

[1]Universidade Tuiuti do Paraná
Curitiba – PR

luiz.matias@utp.edu.br

1. Introdução

O 8-puzzle é um problema amplamente estudado na área de IA, servindo como base para o desenvolvimento e teste de algoritmos de busca e heurísticas. Consiste em um tabuleiro 3x3 com oito peças numeradas de 1 a 8 e um espaço vazio, onde o objetivo é alcançar uma configuração final específica a partir de uma configuração inicial, movendo as peças adjacentes ao espaço vazio. Diversos algoritmos de busca têm sido aplicados à resolução do 8-puzzle, cada um com suas vantagens e desvantagens. A escolha do algoritmo adequado depende de fatores como eficiência computacional, uso de memória e capacidade de encontrar soluções ótimas.

2. Algoritmos de Busca

2.1. Busca em Largura (BFS)

A BFS explora todos os nós em um nível antes de avançar para o próximo, garantindo encontrar a solução com o menor número de passos. No entanto, seu uso de memória pode ser elevado, especialmente em problemas com grandes espaços de estados.

2.2. Busca em Profundidade (DFS)

A DFS explora o caminho mais profundo possível antes de retroceder. Embora utilize menos memória que a BFS, não garante encontrar a solução mais curta e pode entrar em loops infinitos se não houver controle de profundidade.

2.3. Busca Gulosa

A busca gulosa utiliza uma função heurística para estimar a proximidade do objetivo, escolhendo sempre o nó que parece mais promissor. Embora seja rápida, não garante encontrar a solução ótima, pois pode ser enganada por heurísticas mal definidas.

2.4. Busca A*

A busca A* combina a heurística da busca gulosa com o custo real do caminho percorrido, garantindo encontrar a solução ótima se a heurística for admissível. No entanto, seu uso de memória pode ser significativo.

3. Implementação e Configuração dos Testes

3.1. Implementação

Os algoritmos foram implementados em Python, utilizando as seguintes bibliotecas:

csv para leitura dos puzzles a partir de um arquivo CSV.

time para medir o tempo de execução.

psutil para monitorar o uso de memória.

queue.PriorityQueue para gerenciar a fila de prioridades nas buscas informadas.

collections.deque para implementar filas e pilhas eficientes.

Cada algoritmo foi adaptado para trabalhar com o 8-puzzle, representando o estado do tabuleiro como uma string de 9 caracteres, onde '0' representa o espaço vazio.

3.2. Configuração dos Testes

Foram utilizados 10 puzzles diferentes, extraídos de um arquivo CSV. Para cada puzzle, os quatro algoritmos foram executados, e os seguintes parâmetros foram registrados:

Número de passos: quantidade de movimentos necessários para alcançar a solução.

Tempo de execução: medido em segundos.

Uso de memória: medido em kilobytes (KB).

4. Discussão e Resultados

A análise dos resultados obtidos revela detalhes importantes sobre o desempenho dos algoritmos de busca aplicados ao 8-puzzle. A Busca A*, especialmente quando utiliza a heurística da distância de Manhattan, destaca-se por sua capacidade de encontrar soluções ótimas com eficiência tanto em tempo quanto em uso de memória. Essa heurística é admissível e consistente, o que garante a optimalidade das soluções encontradas pelo algoritmo A*.

Por outro lado, a Busca em Largura (BFS) também garante soluções ótimas, mas seu consumo de memória cresce exponencialmente com a profundidade da solução, tornando-a menos prática para instâncias mais complexas. A Busca em Profundidade (DFS), embora consuma menos memória, não garante soluções ótimas e pode entrar em loops, especialmente se não houver mecanismos de controle de profundidade ou detecção de estados repetidos.

A Busca Gulosa, que se baseia exclusivamente na heurística para tomar decisões, apresenta desempenho variável. Embora possa encontrar soluções rapidamente em alguns casos, sua falta de consideração pelo custo acumulado pode levá-la a caminhos subótimos.

É importante notar que o desempenho dos algoritmos pode variar dependendo da configuração inicial do puzzle. Algumas instâncias podem ser resolvidas eficientemente por algoritmos menos sofisticados, enquanto outras exigem abordagens mais robustas como o A*.

Tabela 1. Comparativo de algoritmos para resolução de puzzles 8

Puzzle	Algoritmo	Passos	Tempo (s)	Memória (KB)
1	Largura	2	0.0001	0.00
	Profundidade	30	0.0624	772.00
	Gulosa	2	0.0001	0.00
	A*	2	0.0001	0.00
2	Largura	4	0.0002	0.00
	Profundidade	14	0.0958	896.00
	Gulosa	4	0.0002	0.00
	A*	4	0.0002	0.00
3	Largura	6	0.0007	0.00
	Profundidade	30	0.1604	636.00
	Gulosa	6	0.0002	0.00
	A*	6	0.0003	0.00
4	Largura	8	0.0015	0.00
	Profundidade	N/A	0.1893	2432.00
	Gulosa	40	0.0046	0.00
	A*	8	0.0004	0.00
5	Largura	6	0.0004	0.00
	Profundidade	N/A	0.1833	384.00
	Gulosa	6	0.0003	0.00
	A*	6	0.0003	0.00
6	Largura	6	0.0006	0.00
	Profundidade	6	0.0938	0.00
	Gulosa	6	0.0002	0.00
	A*	6	0.0005	0.00
7	Largura	8	0.0011	0.00
	Profundidade	28	0.1483	128.00
	Gulosa	8	0.0003	0.00
	A*	8	0.0006	0.00
8	Largura	3	0.0001	0.00
	Profundidade	N/A	0.1891	640.00
	Gulosa	3	0.0002	0.00
	A*	3	0.0002	0.00
9	Largura	13	0.0134	0.00
	Profundidade	N/A	0.2564	1408.00
	Gulosa	15	0.0005	0.00
	A*	13	0.0006	0.00
10	Largura	13	0.0193	0.00
	Profundidade	29	0.1087	0.00
	Gulosa	35	0.0097	0.00
	A*	13	0.0020	0.00

5. Conclusão e Possíveis Melhorias Futuras

Este estudo comparativo demonstrou que, entre os algoritmos analisados, a Busca A* com heurística de Manhattan oferece o melhor equilíbrio entre eficiência e optimalidade na resolução do problema do 8-puzzle. Sua capacidade de encontrar soluções ótimas com uso moderado de recursos computacionais a torna uma escolha preferencial para esse tipo de problema.

Contudo, há espaço para melhorias e explorações futuras:

Implementação de Algoritmos Genéticos: Algoritmos genéticos podem ser explorados como alternativas, especialmente para instâncias onde a busca exaustiva é impraticável. Estudos indicam que eles podem oferecer soluções próximas do ótimo com menor custo computacional . ResearchGate

Busca Bidirecional: A implementação de buscas bidirecionais, que simultaneamente exploram o espaço de estados a partir do estado inicial e do estado objetivo, pode reduzir significativamente o tempo de busca .

Heurísticas Aprendidas: A utilização de técnicas de aprendizado de máquina para desenvolver heurísticas adaptativas pode melhorar o desempenho dos algoritmos de busca, ajustando-se dinamicamente às características específicas de cada instância do problema.

Paralelização: A exploração de abordagens paralelas, utilizando múltiplos threads ou processamento distribuído, pode acelerar a busca por soluções, especialmente em ambientes com recursos computacionais adequados.

Em suma, enquanto a Busca A* com heurística de Manhattan se mostra eficaz para o 8-puzzle, a contínua pesquisa e desenvolvimento de novas técnicas e aprimoramentos é essencial para lidar com desafios mais complexos e variados no campo da Inteligência Artificial.