Análise Comparativa de Algoritmos de Busca Aplicados ao Problema do Quebra-Cabeça dos 8 Números

Iuker de Souza Santos[1]

[1]Universidade Tuiuti do Paraná
Curitiba – PR

iuker.santos@utp.edu.br

Resumo. *Este trabalho apresenta uma análise comparativa entre quatro algoritmos de busca — Busca em Largura (BFS), Busca em Profundidade (DFS), Busca Gulosa e A* — aplicados à resolução do problema clássico do quebra-cabeça dos 8 números. Foram testadas 10 instâncias distintas, e os algoritmos foram avaliados com base no número de movimentos até a solução, tempo de execução e uso de memória. O estudo destaca a importância da escolha adequada da estratégia de busca em problemas de planejamento e oferece sugestões para trabalhos futuros, como a aplicação de novas heurísticas e algoritmos mais avançados.*

1. Introdução

O problema do quebra-cabeça dos 8 números consiste em um tabuleiro 3x3 com oito peças numeradas de 1 a 8 e um espaço vazio. O objetivo é reorganizar as peças a partir de um estado inicial até atingir o estado final desejado, normalmente com as peças em ordem crescente da esquerda para a direita e de cima para baixo.

Para resolver esse problema, diversas estratégias de busca podem ser aplicadas, cada uma com características distintas em termos de desempenho, eficiência de tempo e uso de memória. Este relatório analisa quatro algoritmos:

- Busca em Largura (Breadth-First Search - BFS)
- Busca em Profundidade (Depth-First Search - DFS)
- Busca Gulosa (Greedy Search)
- Busca A* (A-Star Search)

2. Descrição da implementação

A implementação dos algoritmos foi realizada em Python, utilizando estruturas de dados eficientes para representação dos estados do tabuleiro, controle de visitados e expansão de nós. As heurísticas utilizadas foram:

- Para A* e Busca Gulosa: Distância de Manhattan.

As instâncias de teste foram compostas por 10 configurações iniciais diferentes. Para cada uma, foram medidos:

- Número de movimentos até a solução.
- Tempo de execução.
- Memória utilizada.

3. Configuração dos Testes

Os testes foram realizados utilizando uma implementação em Python 3.11, com suporte às bibliotecas `time`, `psutil` e `queue`, para medir com precisão o tempo de execução, uso de memória e controle da estrutura dos nós.

3.1. Ambiente de Execução

- Sistema Operacional: Windows 11 64 bits
- Processador: Intel Core i5-12400f @ 2.5GHz
- Memória RAM: 16 GB
- Python: Versão 3.11.5

3.2. Métricas Avaliadas

Durante a execução de cada algoritmo sobre 10 instâncias distintas do quebra-cabeça dos 8 números, foram coletadas as seguintes métricas:

- **Movimentos**: Quantidade de movimentos realizados para alcançar a solução;
- **Tempo**: Tempo total de execução do algoritmo até encontrar a solução;
- **Memória**: Pico de uso de memória RAM durante a execução do algoritmo.

3.3. Heurísticas Utilizadas

As heurísticas adotadas nas estratégias de busca informada foram:

- **Busca Gulosa**: Distância de Manhattan;
- **Busca A***: Soma do custo do caminho percorrido ($g(n)$) com a distância de Manhattan ($h(n)$).

3.4. Instâncias de Teste

Foram utilizadas 10 configurações iniciais diferentes do quebra-cabeça 8-puzzle, com graus variados de dificuldade (quantidade de movimentos até a solução). As instâncias foram escolhidas de forma manual para incluir tanto casos simples quanto desafiadores.

3.5. Execução

Cada algoritmo foi executado separadamente sobre todas as instâncias. Para cada execução, foram reiniciadas as estruturas internas de controle e medidas as métricas desde o início do algoritmo até a chegada na solução ou interrupção por não encontrar um caminho viável.

4. Tabela comparativa com os resultados

Tabela 1. Resultados totais dos algoritmos de busca nas 10 instâncias do quebra-cabeça dos 8 números.

Algoritmo	Movimentos Totais	Tempo Total	Memória Total
BFS	63	0.0128s	0.16 MB
DFS	468	1.3509s	88.94 MB
Gulosa	53	0.0034s	0.00 MB
A*	68	0.0031s	0.07 MB

5. Discussão

Com base nos resultados totais obtidos, observamos diferenças marcantes entre os algoritmos analisados.

5.1. Busca em Largura (BFS)

A BFS foi eficaz para instâncias simples, mas torna-se rapidamente limitada pelo con- sumo de memóia.

- Efetiva para resolver instâncias simples, retornando sempre a solução mais curta.
- Consome pouca memória em estados iniciais próximos da meta.
- Em instâncias mais complexas, a memória cresce rapidamente, tornando-a inviável para problemas de maior escala.

5.2. Busca em Profundidade (DFS)

A DFS teve o pior desempenho, com tempo elevado, uso de memória muito alto e falhas em encontrar soluções em duas instâncias.

- Se mostrou ineficiente na maioria dos testes.
- Tempo de execução elevado e uso de memória extremamente alto.
- Não conseguiu encontrar solução em duas das dez instâncias, indicando sua limitação em problemas com muitos caminhos possíveis.

5.3. Busca Gulosa

Embora a Busca Gulosa tenha apresentado tempo e mem´oria ainda mais baixos, ela resultou em um total menor de movimentos, o que indica soluc¸ ˜oes mais curtas, por´em nem sempre otimizadas.

- Extremamente rápida e econômica em memória.
- Resolveu todas as instâncias com excelente desempenho.
- Por considerar apenas a heurística, pode gerar caminhos subótimos.

5.4. Busca A*

A Busca A* demonstrou ser a mais eficiente em termos de tempo de execução e uso de memória, além de apresentar uma quantidade de movimentos próxima ao mínimo ideal.

- Mostrou o melhor equilíbrio entre tempo, memória e qualidade da solução.
- Resolveu todas as instâncias com caminhos curtos e ótimo desempenho.
- Utiliza heurística e custo acumulado, garantindo soluções ótimas com tempo de execução ainda muito baixo.

De forma geral, os algoritmos informados (Gulosa e A*) são mais apropriados para esse tipo de problema, enquanto os não-informados (BFS e DFS) são limitados por uso de memória ou ineficiência em problemas maiores.

6. Conclusão

Conclui-se que o algoritmo A* é o mais eficiente para resolver o quebra-cabeça dos 8 números, pois:

- Garante soluções ótimas (menor número de movimentos);
- Possui baixo tempo de execução;
- Apresenta consumo de memória muito pequeno.

A Busca Gulosa também é viável, especialmente em aplicações que exigem rapidez e podem tolerar soluções subótimas, devido à sua simplicidade e velocidade. Ela prioriza caminhos promissores com base em heurísticas, sendo útil quando o tempo de resposta é mais crítico do que a garantia de uma solução ótima.

Com isso, a Busca em Largura (BFS) é prática apenas em casos simples, devido ao seu alto consumo de memória. Já a Busca em Profundidade (DFS) é inadequada para esse tipo de problema, pois pode se perder em caminhos profundos sem solução ou exigir mecanismos de corte de profundidade para ser viável.

6.1. melhorias futuras

Apesar dos resultados promissores obtidos, ainda há diversas possibilidades para expandir e aprimorar o projeto. As limitações observadas durante os testes, bem como o potencial de aplicação dos algoritmos em cenários mais desafiadores, indicam caminhos interessantes para desenvolvimento futuro. Nesse contexto, destacam-se as seguintes melhorias futuras possíveis:

- Uma das possibilidades é experimentar diferentes heurísticas, como o número de peças fora do lugar ou a distância de Manhattan, buscando avaliar qual delas oferece melhor desempenho em diferentes cenários.
- Aplicar os algoritmos a problemas mais desafiadores, como o 15-puzzle ou até mesmo versões tridimensionais, pode trazer novas perspectivas.
- Otimizar a estrutura de armazenamento dos nós, visando reduzir o consumo de memória e acelerar a execução dos algoritmos.
- Implementação e comparação com métodos mais sofisticados, como o IDA*, algoritmos genéticos e abordagens baseadas em aprendizado por reforço.
- Avaliar o desempenho em ambientes com recursos limitados, como dispositivos móveis ou sistemas embarcados, também pode ser útil para medir a viabilidade prática das soluções.
- Incluir visualizações interativas para representar a busca em tempo real e incorporar testes automatizados e benchmarks pode tornar o projeto mais robusto, acessível e didático para estudos futuros.

As possibilidades de aprimoramento deste projeto são amplas e promissoras. A exploração de novas heurísticas, a aplicação a problemas mais complexos e a comparação com algoritmos mais avançados não apenas ampliam o alcance da análise, como também oferecem oportunidades para tornar as soluções mais eficientes e versáteis. Além disso, otimizações estruturais e testes em ambientes com restrições de hardware contribuem para aproximar o projeto de aplicações reais. Essas perspectivas não apenas reforçam a relevância da pesquisa, como também abrem caminho para investigações mais profundas no campo da inteligência artificial aplicada à resolução de problemas combinatórios.

Referências

[1] Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson Education.

[2] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

[3] Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1), 97–109.

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