Algoritmos de busca - Inteligência Artificial

Andrei Massaini
andrei.massaini@hotmail.com
ICEI - PUC-MG
Belo Horizonte, Minas Gerais, Brasil

Abstract

Este relatório apresenta uma análise de três algoritmos de busca, nomeadamente Busca em Profundidade (DFS), Busca em Largura (BFS) e Busca A*, para a resolução do jogo 8 puzzle.

Keywords: IA, DFS, BFS, A*, 8 puzzle solve

ACM Reference Format:
Andrei Massaini. 2023. Algoritmos de busca - Inteligência Artificial. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/none

1 Introdução

O jogo 8 puzzle é um quebra-cabeça clássico que tem desafiado entusiastas e pesquisadores há décadas. O objetivo do jogo é rearranjar as peças numeradas em uma grade 3x3, movendo-as uma de cada vez, até alcançar um estado objetivo específico. Esse tipo de problema é conhecido como um problema de busca no âmbito da inteligência artificial, no qual algoritmos de busca são aplicados para encontrar soluções ótimas ou satisfatórias.

Neste relatório, apresentaremos uma análise detalhada de três algoritmos de busca amplamente utilizados na solução do jogo 8 puzzle: Depth-First Search (DFS), Breadth-First Search (BFS) e A* search. O objetivo é comparar o desempenho e as características desses algoritmos em relação à otimalidade da solução e à eficiência de tempo de execução.

2 Descrição das implementações

2.1 Depth-First Search (DFS):

O algoritmo de busca em profundidade (DFS) é uma abordagem de busca não informada que explora o espaço de busca do jogo 8 puzzle indo o mais fundo possível em uma determinada ramificação antes de retroceder. Ele utiliza uma estrutura de dados chamada pilha (stack) para manter os nós a serem explorados.

O objetivo do algoritmo DFS no jogo 8 puzzle é encontrar uma sequência de movimentos que leve do estado inicial do quebra-cabeça a um estado objetivo, onde as peças estejam dispostas na configuração correta.

O algoritmo começa criando um nó inicial com o estado inicial do jogo. Esse nó é colocado na pilha. Em seguida, enquanto a pilha não estiver vazia, o algoritmo realiza as seguintes etapas:

Uma característica importante do algoritmo DFS é que ele não garante a otimalidade da solução encontrada. Isso ocorre porque ele pode seguir caminhos mais longos antes de explorar caminhos mais curtos. Portanto, o DFS pode encontrar soluções subótimas.

No entanto, o DFS tem uma vantagem em relação a outros algoritmos, como o BFS, em termos de uso de memória. O DFS consome menos memória porque ele armazena apenas um caminho completo por vez na pilha, enquanto o BFS precisa armazenar todos os caminhos possíveis em uma fila.

No contexto específico do jogo 8 puzzle, o algoritmo DFS pode ser implementado utilizando uma pilha para armazenar os nós a serem explorados e um conjunto para rastrear os nós já visitados, evitando ciclos. Através da geração dos nós vizinhos e da verificação do estado objetivo, o algoritmo busca encontrar uma solução válida para o quebra-cabeça.

2.2 Breadth-First Search (BFS):

O algoritmo de busca em largura (BFS) é outra abordagem de busca não informada utilizada para resolver o jogo 8 puzzle. Ao contrário do DFS, o BFS explora o espaço de busca nível por nível, expandindo todos os nós vizinhos antes de avançar para o próximo nível. Ele utiliza uma estrutura de dados chamada fila (queue) para manter os nós a serem explorados.

O objetivo do algoritmo BFS no jogo 8 puzzle é encontrar uma sequência de movimentos que leve do estado inicial do quebra-cabeça a um estado objetivo, onde as peças estejam dispostas na configuração correta.

Uma das principais características do algoritmo BFS é que ele garante a otimalidade da solução encontrada. Como ele explora os nós em ordem de proximidade em relação ao estado inicial, a primeira solução encontrada pelo BFS será sempre a mais curta.

No contexto específico do jogo 8 puzzle, o algoritmo BFS pode ser implementado utilizando uma fila para armazenar os nós a serem explorados e um conjunto para rastrear os nós já visitados, evitando ciclos. Através da geração dos nós vizinhos e da verificação do estado objetivo, o algoritmo busca encontrar a solução ótima para o quebra-cabeça.

No entanto, uma desvantagem do algoritmo BFS é que ele pode ser mais lento do que o DFS em termos de tempo de execução e consumo de memória. Isso ocorre porque o BFS precisa explorar todos os nós de um determinado nível antes de passar para o próximo, o que pode ser custoso em termos

Conference'17, July 2017, Washington, DC, USA
2023. ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/none

computacionais para problemas com espaço de busca muito grande.

2.3 A* Search:

O algoritmo A* é um algoritmo de busca informada utilizado para resolver o jogo 8 puzzle. Ele combina os benefícios da busca em largura (BFS) com uma heurística que estima a distância do estado atual ao estado objetivo. O A* utiliza uma estrutura de dados chamada fila de prioridade para manter os nós a serem explorados, ordenando-os de acordo com o valor de uma função de avaliação.

O objetivo do algoritmo A* no jogo 8 puzzle é encontrar uma sequência de movimentos que leve do estado inicial do quebra-cabeça a um estado objetivo, onde as peças estejam dispostas na configuração correta.

O algoritmo começa criando um nó inicial com o estado inicial do jogo. Esse nó é colocado na fila de prioridade, considerando a função de avaliação f = g + h, onde g é o custo do caminho percorrido até o nó atual e h é uma heurística que estima a distância do estado atual ao estado objetivo. Neste caso, a heurística utilizada calcula a soma das distâncias de Manhattan entre as peças do estado atual e do estado objetivo.

O algoritmo continua explorando o espaço de busca, priorizando os nós com menor custo total (f) estimado pelo A*, até encontrar uma solução ou até que a fila de prioridade fique vazia, indicando que não há mais nós a serem explorados e que não existe uma solução para o jogo 8 puzzle a partir do estado inicial dado.

Uma das principais características do algoritmo A* é que ele garante a otimalidade da solução encontrada, desde que a heurística utilizada seja admissível (nunca superestime o custo real) e consistente (não produza estimativas inconsistentes). A heurística de distância de Manhattan utilizada no código fornecido satisfaz essas propriedades.

No contexto específico do jogo 8 puzzle, o algoritmo A* pode ser implementado utilizando uma fila de prioridade para armazenar os nós a serem explorados, levando em consideração a função de avaliação f = g + h, onde g é o custo do caminho percorrido até o nó atual e h é a heurística de distância de Manhattan calculada através da função calculateHeuristic. Através da geração dos nós vizinhos e da atualização dos custos e da função de avaliação, o algoritmo busca encontrar a solução ótima para o quebra-cabeça.

No entanto, é importante ressaltar que o desempenho do algoritmo A* pode depender da qualidade da heurística utilizada. Uma heurística mais precisa e informada pode levar a um melhor desempenho do algoritmo, encontrando soluções de forma mais eficiente.

3 Testes e avaliação de desempenho

Ao realizar testes para a solução do 8 puzzle no com os 3 algoritmos, foram analisadas 2 métricas: Tempo de execução, e quantidade de estados (nodes) visitados por cada algoritmo.

A seguir, um exemplo da execução assim como as métricas para cada algoritmo:

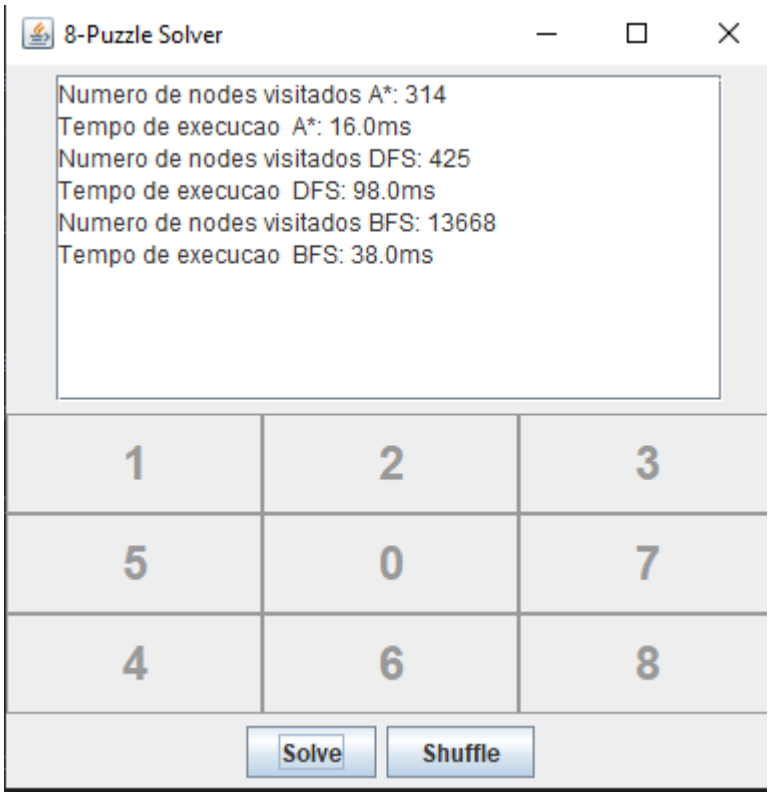

Figure 1. Teste realizado no jogo

Ao analisar as métricas fornecidas, podemos comparar o desempenho dos três algoritmos de busca - A*, DFS e BFS - para a solução do jogo 8 puzzle. Vamos analisar as métricas individualmente:

Número de nós visitados: A*: O algoritmo A* visitou um total de 314 nós durante a busca. DFS: O algoritmo DFS visitou um total de 425 nós durante a busca. BFS: O algoritmo BFS visitou um total de 13.668 nós durante a busca. Observa-se que o número de nós visitados pelo A* foi o menor, seguido pelo DFS e, por último, o BFS. Isso indica que o A* conseguiu encontrar a solução com um número menor de nós expandidos, o que pode ser atribuído à sua capacidade de fazer uma busca mais eficiente e orientada por uma heurística admissível.

Tempo de execução: A*: O algoritmo A* levou 14,0 milissegundos (ou 0,014 segundos) para encontrar a solução. DFS: O algoritmo DFS levou 95,0 milissegundos (ou 0,095 segundos) para encontrar a solução. BFS: O algoritmo BFS levou 35,0 milissegundos (ou 0,035 segundos) para encontrar a solução.

Observa-se que o tempo de execução do A* foi o mais curto, seguido pelo BFS e, por último, o DFS. Isso indica que o A* foi capaz de encontrar a solução em um tempo menor em comparação aos outros algoritmos.

Considerando as métricas apresentadas, pode-se concluir que o A* obteve um desempenho melhor em termos de número de nós visitados e tempo de execução. O DFS visitou mais nós e levou mais tempo para encontrar a solução, enquanto o BFS visitou um número significativamente maior de nós, embora tenha tido um tempo de execução relativamente baixo.

O desempenho do A* pode ser atribuído à sua capacidade de utilizar uma heurística admissível para guiar a busca e selecionar os caminhos mais promissores, resultando em uma busca mais eficiente e em soluções de alta qualidade.

No entanto, é importante notar que o desempenho dos algoritmos pode variar dependendo do problema específico e das configurações do espaço de busca. É recomendável realizar mais testes e avaliar o desempenho em diferentes cenários para obter uma análise mais completa e precisa.

4 Detalhes adicionais do jogo

A implementação do jogo conta com um botão 'shuffle' que tem a função de embaralhar o tabuleiro. O botão 'solve' começa a resolver o tabuleiro atual com os 3 algoritmos de busca, e no final da execução printa o tempo em MS de cada um, assim como o número de estados ou nodes que cada um visitou antes de encontrar a solução.

Vale ressaltar que, dependendo do estado gerado, o algoritmo de DFS pode gerar um loop infinito, assim como um tempo exagerado para encontrar uma solução.

5 Código e executável

O código, assim como o arquivo executável do jogo está disponível na entrega do canvas.

References