TP01 - Introdução a Inteligência Artificial

Leandro Freitas de Souza

Outubro de 2023

1 Introdução

O 8-puzzle é um quebra-cabeça clássico jogado em um tabuleiro 3x3, onde 8 peças numeradas são dispostas aleatoriamente, deixando um espaço vazio. O objetivo do jogo é reorganizar as peças para que fiquem em ordem numérica, utilizando movimentos que deslocam as peças para o espaço vazio.

Existem vários algoritmos para resolver esse quebra-cabeça, cada um com diferentes níveis de complexidade e eficiência. Neste trabalho, foram implementados seis algorítmos diferentes para a resolução do problema: Breadth-first search, Iterative deepening search, Uniform-cost search, A* search, Greedy best-first search e Hill climbing.

2 Modelagem dos algorítmos

2.1 Estruturas de dados

Na implementação da solução do 8-puzzle, a estrutura de nó foi criada para representar cada possível estado do quebra-cabeça. Os filhos de um nó representam os estados que podem ser alcançados através de uma única movimentação. Isso permite a construção de uma árvore de estados do jogo e a navegação por seus diferentes estados.

Cada nó possui os seguintes atributos:

- **state**: corresponde ao estado do quebra-cabeça do nó correspondente.
- **depth**: profundidade na qual o nó se encontra na árvore gerada.
- **parent**: referencia o nó pai, que é o estado anterior do quebra cabeça a partir do qual o nó atual foi alcançado.
- **heuristic**: valor da função de heurística para o nó.

Além disso, foram criados algoritmos para que cada nó consiga gerar seus possíveis filhos e calcular seus valores de heurística, para facilitar o cálculo dos algorítmos de busca.

2.2 Algorítmos

Breadth-First Search

O algorítmo de Breadth-First Search foi implementado utilizando uma fila, de forma a navegar pela a árvore de possibilidades por níveis. Isso nos garante que esse algorítmo encontrará a solução de menor nível da fila, o que no caso do 8-puzzle também será a solução ótima. No entanto, esse algorítmo apresenta alta complexidade de tempo para casos em que a solução mínima está em níveis mais baixos da árvore, uma vez que percorrerá todas as soluções anteriores.

Iterative Deepening Search

O Iterative Deepening Search é uma combinação do BFS com o DFS que usufrui de qualidades de ambos os algorítmos. Além de ser completo, como o BFS, e com certeza encontrará a solução ótima para o 8-puzzle, temos que possui menor complexidade de espaço uma vez que não precisa de percorrer todos os níveis anteriores para encontrar uma solução. Porém, o IDS repete o trabalho nas camadas anteriores a cada iteração, o que também é um problema para casos em que a solução mínima está distante da raíz.

Uniform-Cost Search

Esse algorítmo prioriza estados de menor custo para a busca. Para esse problema, o custo representa o número de movimentos realizados. Assim, o UCS também é completo e no caso do 8-puzzle, se comportará de maneira similiar à do BFS, uma vez que cada nível da árvore representa uma classe de distância para a raiz.

A*

O algorítmo A* utiliza de funções de heurísticas, combinadas com as funções de custo, para encontrar a solução ótima para o problema de forma mais rápida. Para que o algorítmo funcione, a heurística deve ser consistente, uma vez que caso contrário, o custo do algorítmo seria piorado. Uma vez que a desigualdade triangular é mantida, a otimalidade do algorítmo se mantém, e com uma maior eficiência, uma vez que menos estados são explorados até encontrar uma solução ótima.

Greedy Best-First Search

Esse algorítmo não utiliza as funções de custo e prioriza a buscas nos nós cuja função de heurística seja a melhor. Essa mudança resulta na perda de otimalidade da solução, uma vez que os custos são desconsiderados. No entanto, a complexidade de tempo é significativamente reduzida, o que o torna uma escolha viável.

Hill Climbing

Esse algorítmo, que também olha apenas para a função de heurística é utilizado para encontrar mínimos locais na árvore de busca, ou seja, nós nos quais todos os seus vizinhos possuem heurísticas maiores que ele. A solução também não será ótima, uma vez que pode ser encontrado um mínimo local. Além disso, foi feita uma modificação no algorítmo de modo a permitir deslocamentos laterais, para que ele não fique preso em mínimos locais com potencial de melhora.

3 Heurísticas

Para os algorítmos de busca com informação, foram implementadas duas heurísticas diferentes:

- **Número de posições fora do lugar**: para um estado, conta quantas casas estão fora de seus respectivos lugares.
- **Distância de Manhattan**: para cada estado, soma a distância de manhattan de cada peça para sua posição no estado solução.

Temos que ambas as heurísticas são consistentes, uma vez que as estimativas geradas serão sempre menores que o custo real, e, logo, não superestimam o custo e nem quebram a desigualdade triangular da árvore.

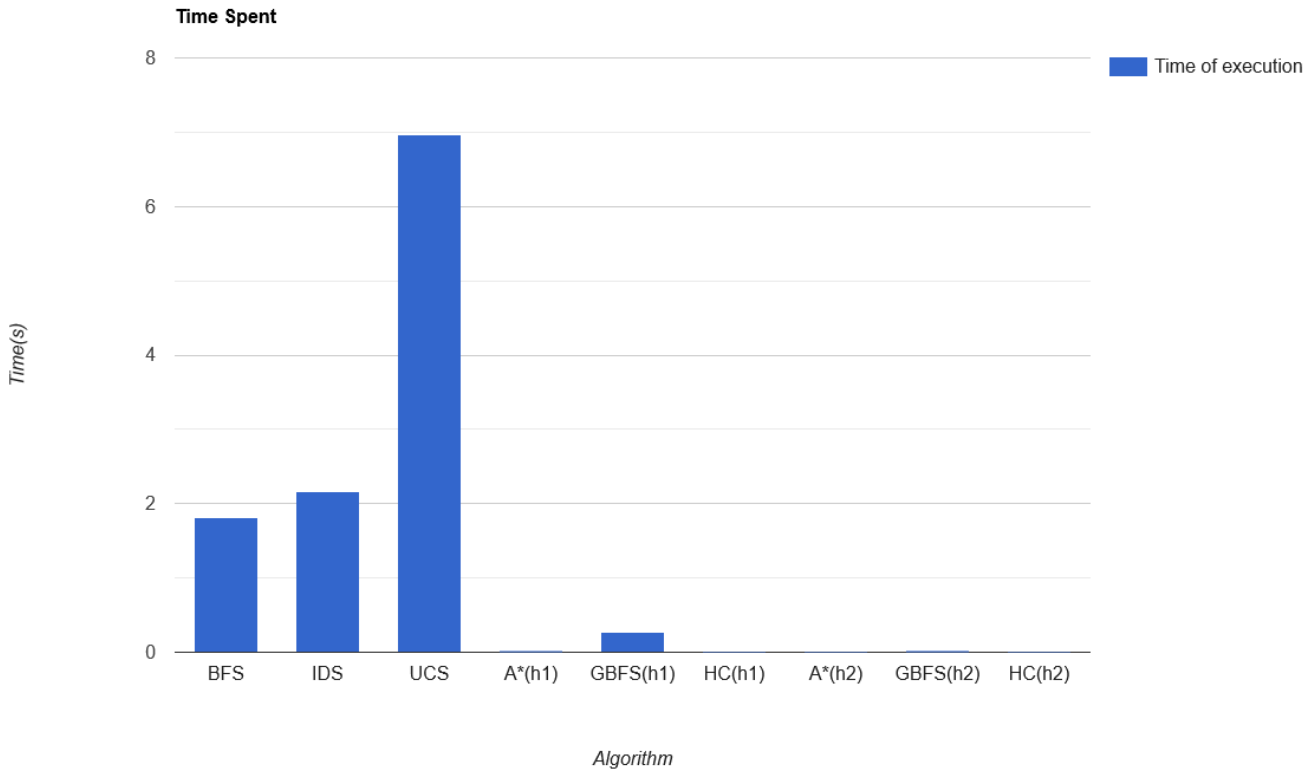
4 Exemplos de soluções

4.1 Caso 1

5	8	2
7		3
1	4	6

Solução: 14

Tempos de execução



A partir da imagem, podemos ver que os algorítmos de busca sem informação levaram mais tempo para serem executados, tendo levado 1.82, 2.17 e 6.96 segundos para serem executados, respectivamente. Para os algorítmos de busca com informação, temos que todos os tempos de execução foram abaixo de 1 segundo, com uma melhora visível nos casos em que a heurística da distância de manhattan é utilizada.

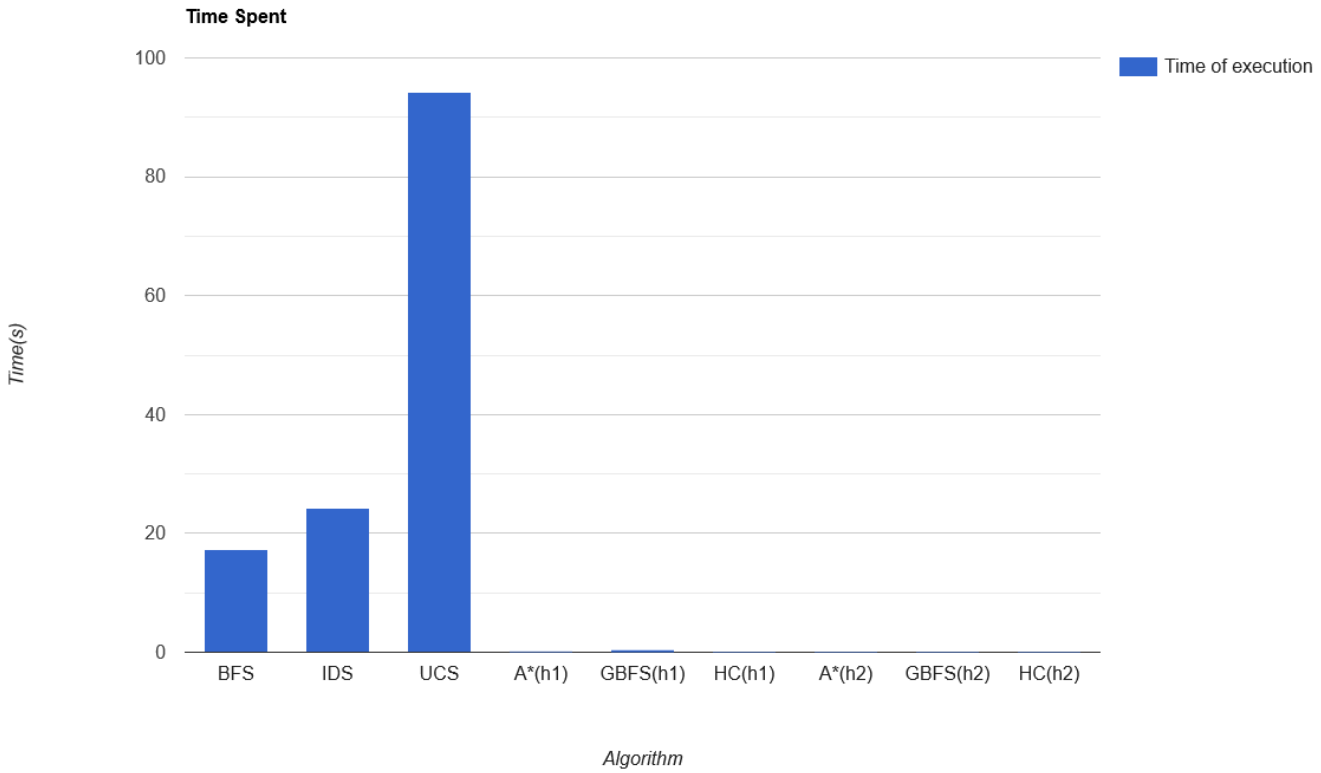
Além disso, na execução do Greedy Best-First Search, foi encontrada, para ambas as heurísticas, uma solução de 90 passos para o problema, cerca de 6.43 vezes pior que a solução ótima. No algoritmo de Hill Climbing, a solução parou em um mínimo local.

4.2 Caso 2

5		8
7	3	2
1	4	6

Solução: 17

Tempos de execução



Nesse caso, todas as buscas com informação se mantiveram com tempo abaixo de 1s, e o GBFS apresentou solução melhor quando utilizada a heurística da Distância de Manhattan. Os tempos de execução da heurística de manhattan continuaram ligeiramente melhores.

5 Conclusão

A partir da realização desse trabalho, pode-se concluir que a resolução do problema 8-puzzle pode se dar de várias maneiras, variando de acordo com os requisitos do solucionador. Além disso, a criação de heurísticas consistentes para esse tipo de problema melhora de forma muito significativa o tempo de execução do algorítmo, mesmo mantendo a qualidade da solução, com o algorítmo A*.