Introdução à Inteligência Artificial

Trabalho Prático 1: 8-Puzzle

Bárbara Martins Ribeiro Duarte
2021074077

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

barbaramrd@ufmg.br

1. Introdução

Neste código, são utilizados diversos algoritmos de busca para resolver o quebra-cabeça 8-puzzle. O quebra-cabeça é representado como uma matriz 3x3, onde cada célula contém um número de 1 a 8 ou 0 (espaço em branco).

2. Implementação

O programa foi desenvolvido na linguagem Python 3.11.3 e executado utilizando a linha de comando.

2.1. Estruturas

1. Matriz **'initial_state**': Uma matriz 3x3 que representa o estado inicial do quebra-cabeça.
2. Matriz constante **'FINAL_STATE**': Uma matriz constante que representa o estado objetivo do quebra-cabeça.
3. Funções:
    a. **find_empty_pos(state)**: Encontra a posição do espaço em branco (0) na matriz.
    b. **is_final_state(state)**: Verifica se um estado é igual ao estado objetivo.
    c. **possible_movements(state)**: Gera os movimentos possíveis a partir de um estado.
    d. **print_solution(solution)**: Imprime a profundidade da solução encontrada.
    e. **print_steps(solution, initial_state)**: Imprime os passos da solução, incluindo o estado inicial e final.

2.2. Algoritmos

Foram implementados 6 algoritmos:

1. **BFS (Breadth-First Search)**
    a. Realiza uma busca em largura, expandindo todos os nós em um nível antes de passar para o próximo nível.
    b. Utiliza uma fila para controlar a ordem de expansão dos nós.
2. **IDS (Iterative Deepening Search)**
    a. Combina busca em profundidade iterativa com busca em largura.
    b. Aumenta gradualmente a profundidade máxima de busca até encontrar a solução.
3. **UCS (Uniform Cost Search)**
    a. Realiza uma busca de custo uniforme, expandindo o nó com o menor custo

acumulado.
   b. Utiliza uma fila de prioridade (heap) para controlar a ordem de expansão dos nós.
4. **A* (A-Star Search)**
   a. Combina custo acumulado com uma heurística para priorizar nós promissores.
   b. Utiliza uma fila de prioridade (heap) com base no custo estimado.
5. **GBFS (Greedy Best-First Search)**
   a. Prioriza nós com base apenas na heurística, sem considerar custos acumulados.
   b. Utiliza uma fila de prioridade (heap) com base na heurística.
6. **HC (Hill Climbing)**
   a. Realiza uma busca local, movendo-se para o vizinho com a melhor heurística local.
   b. Pode ficar preso em ótimos locais locais.

2.3. Heurísticas

Duas heurísticas são utilizadas:

1. **Distância de Manhattan (heuristic_manhattan)**: Calcula a soma das distâncias de Manhattan entre as peças e suas posições finais no estado objetivo. É admissível, pois nunca superestima o custo real para atingir o objetivo.
2. **Número de peças fora do lugar (heuristic_out_of_place)**: Conta o número de peças fora de suas posições finais no estado objetivo. Também é admissível, pois nunca superestima o custo real.

3. Análise comparativa

Figura 1

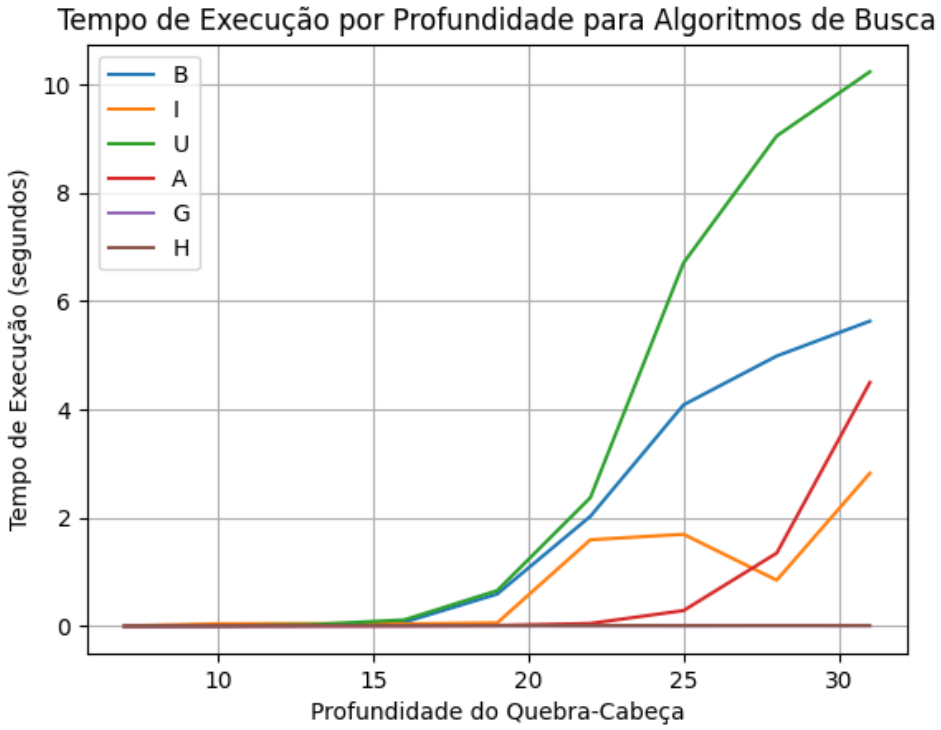

Figura 2

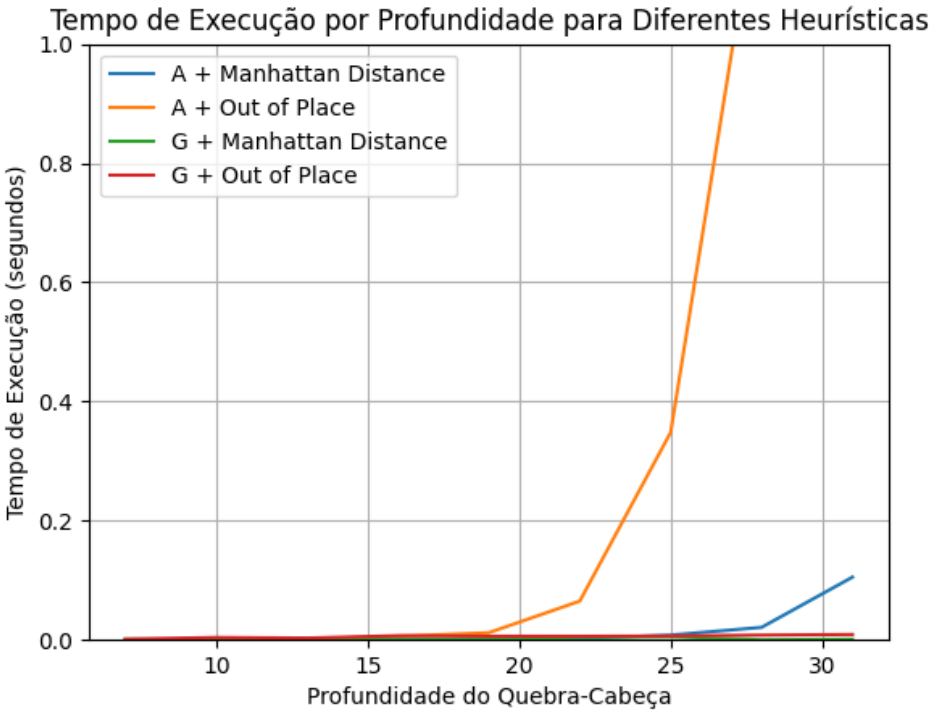

Tabela 1

	BFS	IDS	UCS
7	0,000	0,001	0,001
10	0,003	0,030	0,002
13	0,030	0,050	0,015
16	0,099	0,042	0,142
19	0,662	0,065	0,623
22	2,087	1,670	2,482
25	4,233	1,641	6,796
28	4,800	0,804	9,665
31	6,913	2,859	10,055

Tabela 2

	A* (h1)	A* (h2)	GBFS (h1)	GBFS (h2)
7	0,000	0,001	0,000	0,001
10	0,000	0,009	0,001	0,006
13	0,000	0,009	0,000	0,004
16	0,000	0,003	0,000	0,006
19	0,001	0,011	0,000	0,006
22	0,001	0,061	0,006	0,007
25	0,008	0,353	0,004	0,005
28	0,020	1,370	0,001	0,000
31	0,093	4,279	0,000	0,010

Os gráficos e as tabelas mostram os resultados dos algoritmos BFS, IDS, UCS, A*, GBFS e Hill Climbing para cada profundidade dos casos de teste e mede o tempo de execução. Os algoritmos com informação (A* e GBFS) utilizaram diferentes heurísticas, sendo h1 a distância de manhattan e h2 a de peças fora do lugar. A partir desses dados, podemos concluir que:

1. O tempo de execução aumenta com a profundidade do problema, pois a busca precisa explorar mais estados possíveis.
2. Alguns algoritmos, como BFS e UCS, têm um aumento mais significativo no tempo de execução com o aumento da profundidade, devido à sua natureza de busca completa.
3. Algoritmos de busca heurística, como A*, GBFS e Hill Climbing, geralmente têm tempos de execução mais curtos, pois podem explorar menos estados devido à sua capacidade de fazer escolhas informadas com base nas heurísticas.
4. O GBFS é frequentemente mais rápido do que o A* em termos de tempo de execução, pois faz escolhas mais rápidas e não precisa considerar o custo total do caminho.
5. A escolha do algoritmo depende do equilíbrio entre o tempo de execução e a qualidade da solução desejada. Algoritmos de busca heurística podem ser mais rápidos, mas não garantem a solução ótima.
6. A escolha da heurística é crucial. Se uma heurística admissível e consistente (também chamada de heurística relaxada) estiver disponível, o A* com essa heurística deve ser preferido quando se busca uma solução ótima.
7. O algoritmo IDS (Iterative Deepening Search) pode ser uma opção eficaz para evitar aumentos significativos no tempo de execução, pois controla a profundidade máxima da busca.

4. Resultados obtidos

Os algoritmos de busca implementados demonstram diferentes comportamentos e desempenhos, dependendo das heurísticas e estratégias utilizadas. Alguns algoritmos, como o A* com heurística de distância de Manhattan, tendem a encontrar soluções mais rapidamente e com menos movimentos. Por outro lado, algoritmos como o Hill Climbing podem ficar presos em mínimos locais e não encontrar a solução ótima em todos os casos. A escolha do algoritmo e da heurística adequados depende do problema e das características do estado inicial.