Trabalho Prático 1

Inteligência Artificial

Henrique Rotsen Santos Ferreira

Estruturas de Dados:

Matriz de Estado: O tabuleiro do quebra-cabeça é representado como uma matriz de 3x3, onde cada elemento representa uma peça numerada de 1 a 8 e um espaço vazio representado por 0.

Heurísticas:

Distância de Manhattan (Heurística 1): Calcula a distância de cada peça até sua posição correta no objetivo. A heurística é a soma dessas distâncias para todas as peças. Essa heurística é admissível, pois ela nunca superestima o custo para alcançar o objetivo a partir de qualquer estado. Isso ocorre porque a distância de Manhattan é uma medida direta da quantidade de movimentos necessários para mover cada peça para sua posição correta, sem contar os obstáculos no caminho.

Contagem de Peças Fora do Lugar (Heurística 2): Conta o número de peças que não estão na posição correta no objetivo. Essa heurística também é admissível, pois ela conta o número de peças que estão fora do lugar. Em qualquer configuração do quebra-cabeça, pelo menos uma ação é necessária para mover cada peça para a posição correta. Portanto, essa heurística nunca superestima o custo para alcançar o objetivo.

Algoritmos:

Busca em Largura (BFS): Explora todos os vizinhos de um estado até encontrar a solução. Usa uma fila para gerenciar os estados.

Tempo:
O tempo de execução dependerá da profundidade da solução *d,* e do fator de ramificação *b*. O BFS tem um custo de tempo de $O(b^d)$ no pior caso.
Espaço:
Com uma fila de prioridade, o espaço é limitado pelo número de nós na fronteira. Dado um fator de ramificação de 4, a complexidade de espaço é $O(b^d)$ no pior caso.

Busca em Profundidade com Limites (IDS): Realiza uma busca em profundidade com diferentes limites de profundidade até encontrar a solução.

Tempo:
O tempo de execução dependerá do fator de ramificação *b (que no caso é 4)* e do limite máximo de profundidade d_{max}. O IDS tem um custo de tempo de $O(b \cdot d_{max})$.
Espaço:
O espaço é limitado pelo número de estados mantidos na pilha de recursão. Como a busca em profundidade é utilizada, a complexidade de espaço é $O(b \cdot d_{max})$.

Busca de Custo Uniforme (UCS): Explora os vizinhos com menor custo, onde o custo é a soma dos movimentos realizados. Usa uma fila de prioridade para gerenciar os estados.

Tempo:
O tempo de execução dependerá da função de custo. Com uma fila de prioridade, o tempo é influenciado pelo custo das ações e pela qualidade da heurística. O custo de tempo é geralmente menor que $O(b^d)$ se o custo das ações não for muito pequeno.
Espaço:
O espaço é limitado pelo número de nós mantidos na fila de prioridade. Com a fila de prioridade, a complexidade de espaço é $O(b^d)$ no pior caso.

A* (A-star): Utiliza uma função de avaliação que combina o custo atual com uma heurística que estima o custo restante para a solução. Usa uma fila de prioridade para gerenciar os estados.

Este algoritmo possui a mesma complexidade do UCS *(olhar a análise acima).* A única coisa que muda é que devemos levar em conta o custo de calcular a heurística. No caso do programa, tanto a Heurística 1 quanto a Heurística 2, o custo é linear no tamanho do tabuleiro. Como o tabuleiro tem tamanho 3x3, e este é constante, podemos dizer que calcular as Heurísticas 1 e 2, possuem custo *O(1).*

Greedy Best-First Search: Escolhe o vizinho com a menor heurística (melhor estimativa) sem considerar o custo atual. Usa uma fila de prioridade para gerenciar os estados.

Tempo:
O tempo de execução dependerá da qualidade da heurística. Com a Heurística usada ele foi bastante eficiente, uma vez que ela possui custo *O(1)*.
Espaço:
O espaço é limitado pelo número de estados mantidos na fila de prioridade. Com a heurística usada, a complexidade de espaço pode ser menor que $O(b^d)$.

Hill-Climbing: Escolhe o vizinho com a menor soma de heurísticas. Retorna o caminho seguido até um ótimo local, mesmo que não seja global.

Tempo:
O tempo de execução do Hill-Climbing não possui uma complexidade bem definida, pois não é garantido encontrar a solução ótima. Pode ser rápido em alguns casos, mas pode ficar preso em mínimos locais, que aconteceu muito durante os testes.
Espaço:
Mantém apenas o estado atual e 4 vizinhos na memória. Requer muito menos espaço do que algoritmos de busca global. O custo de espaço é relativamente baixo.

Discussão dos Resultados Obtidos:

BFS e IDS: São métodos completos e garantem encontrar a solução ótima. No entanto, podem consumir muita memória para problemas complexos devido à necessidade de armazenar muitos estados.

Durante os testes, para a pior instância disponibilizada, o BFS fez os 31 passos em 6.97s, já o IDS, fez 43 passos em 4.14s. Isso já era esperado, pois mesmo que o IDS tenha complexidade de tempo menor, ele tem que ficar “recomeçando”, o que gasta mais passos.

Para uma instância média, 8 0 2 5 7 3 1 4 6, que possuí um número de passos esperado de 15, o BFS realizou este número de passos em 0.05579s, já o IDS fez 24 passos em 0.14153s

Para uma instância fácil (2 passos necessários), ambos fizeram os 2 passos, o BFS em 0.00007s e o IDS em 0.00008s

UCS e A*: São algoritmos ótimos em termos de custo, desde que uma heurística admissível seja usada com o A*. O UCS garante a otimalidade, mas pode ser lento se as ações tiverem custos muito pequenos.

Ao testar podemos observar isso, para a pior instância, o UCS fez os 31 passos esperados, porém gastou 9.32s, já o A* fez também 31 passos, porém em 0.69s, o que mostra que caso a Heurística seja consistente, ele é eficientemente ótimo.

Para uma instância média, 8 0 2 5 7 3 1 4 6, que possuí um número de passos esperado de 15, ambos fizeram o mesmo número de passos, porém o UCS gastou 0.04400s e o A* 0.00209s

Para uma instância fácil (2 passos necessários), ambos fizeram os 2 passos, o UCS em 0.00008s e o IDS em 0.00018s

Greedy Best-First Search: Tende a ser rápido, mas não é garantido encontrar a solução ótima. Depende muito da qualidade da heurística.

Durante o teste da pior instância ele fez 141 passos em 0.00962s, isso ocorre já que a Heurística possui custo constante e pequeno.

Para uma instância média, 8 0 2 5 7 3 1 4 6, que possuí um número de passos esperado de 15, o algoritmo guloso fez esses 15 passos em 0.00021s e chego na solução ótima

Para uma instância fácil (2 passos necessários), ele fez os 2 passos em 0.00009s

Hill-Climbing: Pode ser muito eficaz em encontrar soluções rápidas, mas não garante a solução ótima e pode ficar preso em mínimos locais.

Durante o teste de fogo, da pior instância, ele fez 2 passos em 0.00012s, e não chegou na solução ótima, ele trava em um mínimo local, [[8 6 7], [2 5 4], [3 1 0]].

Para uma instância média, 8 0 2 5 7 3 1 4 6, que possuí um número de passos esperado de 15, o Hill-Climbing fez 4 passos em 0.00015s

Para uma instância fácil (2 passos necessários), ele fez 3 passos em 0.00015s, o que é interessante é que ele repetiu o estado objetivo 2x já que a condição de parada dele é (>=).

Tabela de tempo de execução e número de passos

Passos/ Algoritmo	A*	BFS	IDS	UCS	GREEDY	HILL
3	0.00014	0.00012	0.00015	0.00008	0.00011	0.00010
5	0.00019 5	0.00039 5	0.00024 5	0.00027 5	0.00009 5	0.00009 2
7	0.00037 7	0.00119 7	0.00250 9	0.00101 7	0.00331 39	0.00008 2
9	0.00062 9	0.00202 9	0.00262 9	0.00232 9	0.00069 9	0.00013 4
11	0.00063 11	0.00757 11	0.03351 13	0.00724 11	0.00020 11	0.00009 1

13	0.00093 13	0.01796 13	0.42496 27	0.01699 13	0.00016 13	0.00022 3
15	0.00353 15	0.05779 15	0.15062 23	0.05531 15	0.00022 15	0.00021 4
25	5.08897 25	0.06372 25	1.54103 33	6.12650 25	0.00996 63	0.00011 2
28	6.23391 28	0.16263 28	2.77569 40	8.16826 28	0.01389 92	0.00005 1
31	6.74789 31	0.66279 31	4.14604 43	9.28939 31	0.01494 141	0.00015 2