Introdução à Inteligência Artificial
Trabalho Prático 1

Raphael Alves dos Reis

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

cap497@ufmg.br

1 Descrição

O problema do 8-Puzzle é modelado como um espaço de estados discreto, onde cada estado representa uma configuração do quebra-cabeça. Cada estado é uma tupla de 9 números inteiros de 0 a 8, onde 0 representa o espaço vazio no tabuleiro. Por exemplo, o estado inicial padrão é representado como (1, 2, 3, 4, 5, 6, 7, 8, 0).

2 Estruturas de Dados

2.1 Estado:

Representado como uma tupla de 9 números inteiros, onde 0 representa o espaço vazio.

2.2 Operadores/Ações:

Movimentos para cima, para baixo, para a esquerda ou para a direita no tabuleiro.

2.3 Função Sucessora:

Implementada na função `get_neighbors(node)`, que gera todos os estados vizinhos válidos para um estado dado.

2.4 Estado Inicial:

Fornecido como entrada na linha de comando.

2.5 Estado Objetivo:

Sempre definido como (1, 2, 3, 4, 5, 6, 7, 8, 0) para o quebra-cabeça de 8 peças.

2.6 Teste de Objetivo:

Comparação entre o estado atual e o estado objetivo.

2.7 Custo do Caminho:

Em alguns algoritmos como A*, o custo do caminho é levado em consideração ao calcular as prioridades na fila de prioridade. Neste código, o custo do caminho é implícito, representado pela ordem dos estados no caminho.

3 Algoritmos

3.1 Breadth-First Search (BFS)

- **Estrutura de Dados**: Utiliza uma fila para armazenar os estados a serem explorados. Garante que os estados sejam explorados em ordem de profundidade.
- **Modelagem**: Realiza uma busca em largura no espaço de estados, garantindo encontrar a solução mais curta em termos de número de passos.

3.2 Iterative Deepening Search (IDS)

- **Estrutura de Dados**: Utiliza uma busca em profundidade com limitação de profundidade. A profundidade é aumentada iterativamente até encontrar a solução.
- **Modelagem**: Implementa uma busca em profundidade iterativa, aumentando gradualmente a profundidade da busca até encontrar a solução.

3.3 Uniform-Cost Search (UCS)

- **Estrutura de Dados**: Utiliza uma fila de prioridade onde o custo do caminho é a chave de prioridade. Prioriza estados com menor custo de caminho.
- **Modelagem**: Prioriza estados com menor custo de caminho (número de passos) para a expansão.

3.4 A* Search

- **Estrutura de Dados**: Utiliza uma fila de prioridade cuja função de avaliação f(n)=g(n)+h(n) é a chave de prioridade. g(n) é o custo atual do caminho e h(n) é a heurística estimada do estado para o objetivo.
- **Modelagem**: Utiliza uma combinação do custo atual e uma heurística para priorizar estados que têm mais probabilidade de levar à solução.

3.5 Greedy Best-First Search

- **Estrutura de Dados**: Utiliza uma fila de prioridade onde a heurística (no caso, a heurística da Distância de Manhattan) é a chave de prioridade. Prioriza estados com base em quão próximos eles estão do objetivo, independentemente do custo real do caminho percorrido até o momento.
- **Modelagem**: O Greedy Best-First Search prioriza estados que, de acordo com a heurística, parecem mais próximos do estado objetivo. Ele não considera o custo real do caminho percorrido até o momento, tornando-se uma abordagem gananciosa.

3.6 Hill Climbing

- **Estrutura de Dados**: Utiliza uma abordagem de busca local, onde o estado atual é modificado iterativamente para melhorar o valor da função de custo, ou seja, reduzir a discrepância entre o estado atual e o estado objetivo.
- **Modelagem**: O Hill Climbing começa com um estado inicial e, em cada iteração, escolhe o estado vizinho que melhora a função de custo (nesse caso, reduz o número de peças fora do lugar). No entanto, o Hill Climbing pode ficar preso em ótimos locais, não garantindo a solução global ótima.

4 Heurísticas

4.1 Peças Fora do Lugar

- **Modelagem**: Conta o número de peças no estado atual que não estão na posição correta em relação ao estado objetivo.
- **Admissibilidade**: A heurística "Fora do Lugar"conta o número de peças no estado atual que não estão na posição correta em relação ao estado objetivo. Em outras palavras, ela calcula quantas peças estão fora de sua posição final. Esta heurística é admissível porque nunca superestima o custo para alcançar o estado objetivo. Isso ocorre porque, em cada movimento, pelo menos uma peça é colocada na posição correta, reduzindo o número de peças fora do lugar.

4.2 Distância de Manhattan

- **Modelagem**: Calcula a soma das distâncias horizontais e verticais de cada peça à sua posição no estado objetivo.
- **Admissibilidade**: A heurística da "Distância de Manhattan"calcula a soma das distâncias horizontais e verticais de cada peça até sua posição no estado objetivo. Esta heurística também é admissível. Para qualquer estado, a distância de Manhattan até o objetivo é sempre menor ou igual à verdadeira distância, pois um movimento diagonal seria mais curto do que dois movimentos horizontais/verticais. Portanto, a heurística de Manhattan nunca superestima o custo para alcançar o estado objetivo.

5 Exemplos

```
PS  python .\TP1.py B 0 5 2 1 8 3 4 7 6 PRINT
8

  5 2
1 8 3
4 7 6

1 5 2
  8 3
4 7 6

1 5 2
4 8 3
  7 6

1 5 2
4 8 3
7   6

1 5 2
4   3
7 8 6

1   2
4 5 3
7 8 6

1 2
4 5 3
7 8 6

1 2 3
4 5
7 8 6

1 2 3
4 5 6
7 8
```

```
PS  python .\TP1.py B 5 8 2 1 0 3 4 7 6 PRINT
10

5 8 2
1   3
4 7 6

5   2
1 8 3
4 7 6

  5 2
1 8 3
4 7 6

1 5 2
  8 3
4 7 6

1 5 2
4 8 3
  7 6

1 5 2
4 8 3
7   6

1 5 2
4   3
7 8 6

1   2
4 5 3
7 8 6

1 2
4 5 3
7 8 6

1 2 3
4 5
7 8 6

1 2 3
4 5 6
7 8
```

6 Análise

O número de passos e o tempo de execução para cada algoritmo e para cada vetor de exemplo são apresentados nas tabelas 1 (Busca Sem Informação) e 2 (Busca Com Informação e Busca Local).

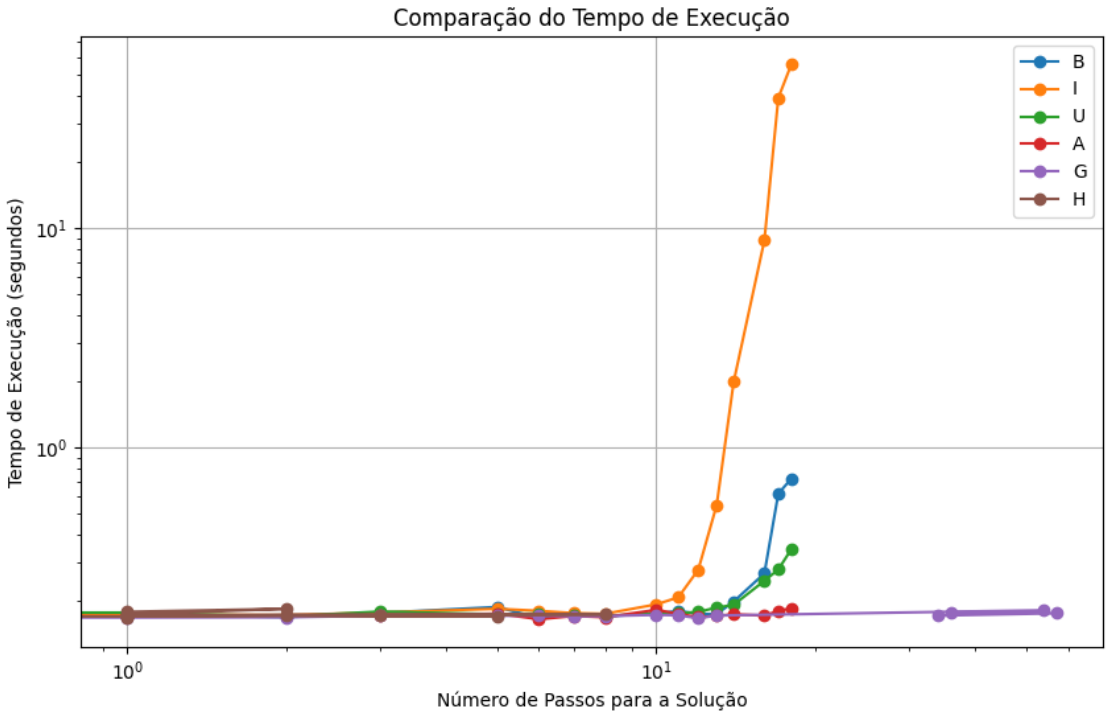
Apesar de serem fornecidos 32 exemplos de entrada, foram mostrados apenas 15 por dois motivos: o primeiro é que para todos os algoritmos, com exceção do IDS, os tempos e números de passos seguem um padrão de forma que é desnecessário continuar para mais exemplos; e o segundo é o tempo do IDS, que aumenta rapidamente, se tornando inviável executar o algoritmo para exemplos com muitos passos.

ID	Vector	B Steps	B Time	I Steps	I Time	U Steps	U Time
0	(1, 2, 3, 4, 5, 6, 7, 8, 0)	0	0.201788	0	0.197470	0	0.189997
1	(1, 2, 3, 4, 5, 6, 7, 0, 8)	1	0.190819	1	0.198872	1	0.230432
2	(1, 2, 3, 4, 0, 5, 7, 8, 6)	2	0.186208	2	0.181038	2	0.182919
3	(1, 0, 3, 4, 2, 5, 7, 8, 6)	3	0.179769	3	0.177157	3	0.187135
4	(1, 5, 2, 0, 4, 3, 7, 8, 6)	5	0.171757	5	0.181038	5	0.177261
5	(1, 5, 2, 4, 8, 3, 7, 6, 0)	6	0.182002	6	0.179995	6	0.199981
6	(1, 5, 2, 4, 8, 0, 7, 6, 3)	7	0.178757	7	0.181955	7	0.183918
7	(5, 8, 2, 1, 0, 3, 4, 7, 6)	10	0.187198	10	0.203425	10	0.184659
8	(5, 8, 2, 1, 7, 3, 4, 0, 6)	11	0.188700	11	0.214440	11	0.183714
9	(5, 8, 2, 1, 7, 3, 0, 4, 6)	12	0.187391	12	0.287726	12	0.186015
10	(5, 8, 2, 0, 7, 3, 1, 4, 6)	13	0.186274	13	0.539352	13	0.192944
11	(5, 8, 2, 7, 0, 3, 1, 4, 6)	14	0.209249	14	2.121032	14	0.211114
12	(8, 7, 2, 5, 0, 3, 1, 4, 6)	16	0.270998	16	8.905442	16	0.250823
13	(5, 0, 8, 7, 3, 2, 1, 4, 6)	17	0.604171	17	38.260044	17	0.287467
14	(8, 7, 2, 5, 4, 3, 1, 6, 0)	18	0.736077	18	55.523407	18	0.327397

Tabela 1: Resultados dos algoritmos de busca sem informação

ID	Vector	A Steps	A Time	G Steps	G Time	H Steps	H Time
0	(1, 2, 3, 4, 5, 6, 7, 8, 0)	0	0.195797	0	0.190519	0	0.190988
1	(1, 2, 3, 4, 5, 6, 7, 0, 8)	1	0.233767	1	0.193923	1	0.186955
2	(1, 2, 3, 4, 0, 5, 7, 8, 6)	2	0.186762	2	0.174252	2	0.177512
3	(1, 0, 3, 4, 2, 5, 7, 8, 6)	3	0.179858	3	0.179242	3	0.179144
4	(1, 5, 2, 0, 4, 3, 7, 8, 6)	5	0.194023	5	0.188615	5	0.186850
5	(1, 5, 2, 4, 8, 3, 7, 6, 0)	6	0.190467	6	0.177334	6	0.191702
6	(1, 5, 2, 4, 8, 0, 7, 6, 3)	7	0.194340	7	0.183432	1	0.181399
7	(5, 8, 2, 1, 0, 3, 4, 7, 6)	10	0.179973	10	0.185392	1	0.176307
8	(5, 8, 2, 1, 7, 3, 4, 0, 6)	11	0.183444	11	0.182796	1	0.181238
9	(5, 8, 2, 1, 7, 3, 0, 4, 6)	12	0.179926	12	0.180532	1	0.177065
10	(5, 8, 2, 0, 7, 3, 1, 4, 6)	13	0.194801	13	0.177402	1	0.175655
11	(5, 8, 2, 7, 0, 3, 1, 4, 6)	14	0.181752	14	0.185197	1	0.179303
12	(8, 7, 2, 5, 0, 3, 1, 4, 6)	16	0.185832	16	0.181492	2	0.181492
13	(5, 0, 8, 7, 3, 2, 1, 4, 6)	17	0.193298	17	0.287467	1	0.188561
14	(8, 7, 2, 5, 4, 3, 1, 6, 0)	18	0.172703	18	0.190850	1	0.181059

Tabela 2: Resultados dos algoritmos de busca sem informação e local



7 Discussão

Os algoritmos BFS e IDS apresentam resultados idênticos, pois ambos garantem encontrar a solução mais curta. UCS também encontra a solução ótima, mas expande menos nós do que BFS e IDS, tornando-o mais eficiente em termos de memória. A* é o único algoritmo que garante a solução mais curta usando uma heurística. Greedy Best-first Search, por outro lado, prioriza estados com base apenas na heurística, o que pode levar a soluções subótimas. O algoritmo de Hill Climbing é um método de busca local que pode ficar preso em mínimos locais. Portanto, sua eficácia depende muito da qualidade da heurística e da escolha do estado inicial. No seu caso, o algoritmo parece ter encontrado uma solução local ótima, mas não necessariamente a solução global mais curta.

8 Conclusão

Para este problema específico, onde a solução mais curta é crucial, o A* é a escolha ideal. Ele garante a solução ótima, levando em consideração tanto o custo acumulado quanto uma heurística informada. Em termos de complexidade espacial e temporal, UCS também é uma boa opção em termos de otimização de memória, enquanto A* atinge um bom equilíbrio entre precisão e eficiência. No entanto, é importante notar que o Hill Climbing, embora seja um algoritmo interessante, pode não ser o mais adequado para problemas nos quais soluções subótimas não são aceitáveis. Para melhorar a busca local, considerar algoritmos mais sofisticados, como Simulated Annealing ou Algoritmos Genéticos, pode ser uma abordagem válida.

Em resumo, o A* é a escolha mais confiável para encontrar a solução mais curta neste contexto. Entretanto, como cada problema pode ter características únicas, é sempre essencial adaptar a escolha do algoritmo às particularidades do problema em questão.