Inteligência Artificial
Trabalho Prático 1 - 8-Puzzle

Gabriel Henrique Souto Pires
gabrielpires@dcc.ufmg.br

1 Modelagem

O 8-puzzle foi modelado em 3 arquivos, um para o tabuleiro, um para modelar os estados e outro contendo os algoritmos de busca. A linguagem utilizada foi Python3 e o sistema operacional em que o programa foi testado é um Xubuntu 18.04.2 LTS, instalado em uma máquina virtual.

1.1 Tabuleiro

O tabuleiro do 8 Puzzle consiste de uma matriz quadrada, em que os elementos são numerados a partir de 1 em ordem crescente e o último elemento é um espaço vazio.

Para esse programa, o tabuleiro foi modelado como uma matriz $N \times N$, onde N é a dimensão da matriz. Inicialmente a matriz é populada com as peças na forma da resposta, ou seja, em ordem crescence a partir de 1 até $N^2 - 1$ com a última posição vazia.

1	2	3
4	5	6
7	8	0

Figura 1: Tabuleiro 3x3 “goal”. A posição 0 representa o espaço vazio.

Isso foi feito para facilitar na criação de casos teste, embaralhando o tabuleiro a partir da solução, uma vez que instâncias desse problema não podem ser geradas ao acaso, já que cerca de 50% das disposições possíveis não tem solução.

Também existe a opção de ler o tabuleiro a partir de uma string de números no formato “1 2 3 ... 8 0”, o que facilitou testar os casos disponibilizados juntamente à especificação do trabalho.

A solução é testada comparando o estado do tabuleiro com uma matriz no formato inicial, ou seja, com as peças em ordem crescende de 1 até $N^2 - 1$ e a última posição vazia. Os outros métodos da classe do tabuleiro são apenas para mover as peças, ou seja, um swap comum das posições da matriz.

1.2 Estados

Os estados são cada um uma instância do problema, ou seja, um tabuleiro de onde podemos encontrar a solução do 8-puzzle. Cada estado tem uma função sucessora que retorna os estados que podem ser derivados dele de acordo com os movimentos possíveis.

Os estados foram modelados de tal forma que:

- Cada estado tem um tabuleiro
- Cada estado tem um estado pai do qual herda os movimentos anteriores
- A profundidade de cada estado é a profundidade do pai + 1
- Os movimentos necessários para se chegar a esse estado são os movimentos para se chegar ao pai mais o movimento para se chegar do pai a esse estado.

A classe do estado também tem as heurísticas. Tanto as heuristicas quanto a função sucessora serão explicados mais adiante.

1.3 Função Sucessora

A função sucessora é uma função que a partir de um estado retorna um par <estado, ação>, que nesse caso é um estado com o movimento que foi feito para se chegar nesse estado a partir do anterior. Mais especificamente, a função sucessora retorna uma lista com todos os estados que podem ser atingidos a partir do estado especificado e o estado pai de cada um, não levando em consideração por exemplo movimentos que não alteram o tabuleiro, como por exemplo no caso de o espaço vazio estar nos cantos do tabuleiro como na Figura 2.

2 Heurísticas

Para esse trabalho foram implementadas duas heurísticas, uma para ser usada no algoritmo A* e outra no algoritmo Greedy Best First Search. A heurística usada no A*

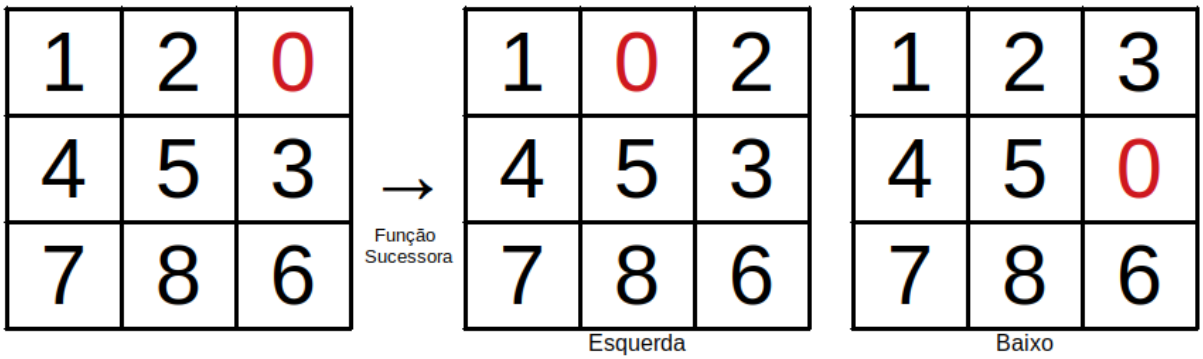

Figura 2: Estados retornados pela função sucessora, note que os tabuleiros resultantes dos movimentos para cima e para a direita são ignorados, uma vez que não haveria alteração no estado.

é chamada Distância de Manhattan, já a usada no Greedy Best First Search é a de N Peças Erradas.

2.1 Distância de Manhattan

Consiste na soma da distância horizontal e vertical das peças até sua posição correta. Foi encontrado que a posição correta horizontal das peças no 8-puzzle é sempre $(n-1)$ e a posição correta vertical é $(n-1)/tamanho$, sendo n o número da peça, então a partir disso podemos calcular a distância da posição correta até onde a peça se encontra. Se mostrou uma ótima heurística ao se aplicar no algoritmo A*.

Essa heurística é admissível, pois, a única coisa que qualquer um movimento pode fazer é mover uma peça apenas um passo mais próximo do objetivo.

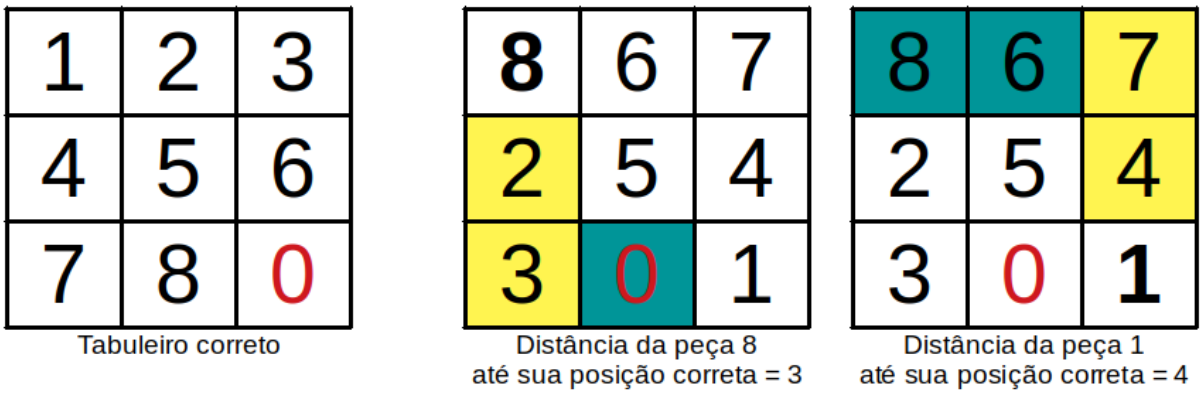

Figura 3: Calculo da distancia de manhattan. As posições em amarelo representam a distância vertical, os azuis representam a distância horizontal.

2.2 N Peças Erradas

É o número de peças que não estão na posição que deveriam estar de acordo com o tabuleiro objetivo. Essa é uma heurística bem simples e fácil de implementar, mas não se mostrou eficiente no algoritmo A*.

Essa heurística é admissível, pois, fica claro que qualquer peça que esteja fora do lugar, precisará ser movida ao menos uma vez.

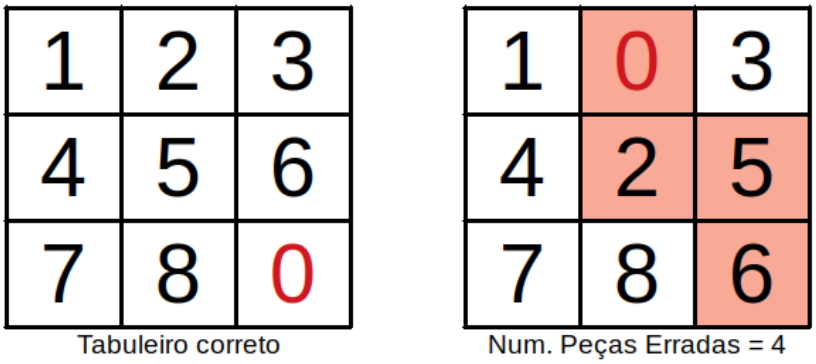

Figura 4: Calculo da heurística das N Peças Erradas. A matriz do problema é percorrida e toda vez que uma posição não contém a peça correta soma-se 1 no resultado.

3 Algoritmos

3.1 Breadth-first search

Algoritmo que expande os nós mais rasos antes, é completo e ótimo para esse problema mas possui complexidade de espaço exponencial e de tempo igual a $O(b^d)$ com d igual ao nível da solução e b igual ao branching factor.

O algoritmo começa atribuindo o estado inicial do problema a uma variável e testando se esse estado é o objetivo, se for retornamos esse estado, caso contrário adicionamos ele na fronteira.

Dentro de um loop infinito o algoritmo expande a fronteira em busca da solução, expandindo primeiro a raiz, e então seus sucessores, e então os sucessores desses estados, e assim por diante. Isso é feito até que não existam mais estados na fronteira ou até a solução ser encontrada.

3.2 Iterative deepening search

Esse algoritmo consiste em chamar o Depth First Search incrementando o limite de profundidade até que uma solução é encontrada. É completo e ótimo pra esse problema, tem complexidade de $O(bd)$ sendo b o branching factor, ou seja, o número máximo de

sucessores que um nó pode ter e d a profundidade máxima. Além disso o IDS possui complexidade de espaço linear.

3.3 Uniform-cost search

Algoritmo que expande os nós com menor custo de caminho $g(n)$, é ótimo para esse problema.

O menor custo de caminho no caso do 8-puzzle foi interpretado como sendo sempre 1. Isso fez com que o Uniform Cost Search não desse prioridade a nanhum nó. Isso aconteceu devido a natureza do problema ser mover sempre as peças por uma quantia fixa (uma posição).

3.4 A* search

Expande os nós com menor valor de $f(n) = g(n)+h(n)$ sendo $h(n)$ o valor da heurística e $g(n)$ o custo de se chegar até o nó. Esse algoritmo é completo e ótimo se escolhermos uma heurística admissível (no caso de buscas em árvores) ou consistente (em caso de grafos) mas possui complexidade de espaço preocupante.

Nesse problema poderíamos considerar $g(n)$ como a profundidade do nó, ou seja, o custo de se chegar até ele a partir do estado inicial, e para $h(n)$ usamos o valor da heurística da Distância de Manhattan.

3.5 Greedy best-first search

Esse algoritmo expande o nó que está mais próximo da solução de acordo com uma heurística escolhida. Nesse caso escolhemos a heurística das N Peças Erradas (nWrongTiles) que foi explicado acima. Assim como o A* esse algoritmo usa uma fila de prioridades para saber quais nós expandir.

Não é completo, uma vez que pode entrar em loops e nem sempre acha a melhor solução. Possui complexidade de tempo de $O(b^m)$ com m sendo a profundidade máxima do nosso problema.

4 Soluções encontradas

As soluções apresentadas a seguir foram encontradas para a última instância do problema que foi disponibilizada juntamente da especificação (solução 31).

BFS:
Nós expandidos: 2063788
Resposta: CCDBEEBDDCEECDDBEBECCDBBECCDDBB
Num. movimentos: 31
Tempo: 183.32 segundos

UniformCost:
Nós expandidos: 2076560
Resposta: ECDCDBEBDCEECDDBEBECCDBBECCDBDB
Num. movimentos: 31
Tempo: 190.43 segundos

A* com a heurística da Distancia de Manhattan:
Nós expandidos: 82936
Resposta: DCCEBBDCCEBEBDCECDBBDCCEBEBDCDB
Num. movimentos: 31
Tempo: 21.42 segundos

GreedyBestFirst com a heurística das N Peças Erradas:
Nós expandidos: 1116
Resposta: ECCDDBBEECCDDBBEECCDDBBEECCDDBEBDCECDBBECDCEBBDCECDBB
Num. movimentos: 53
Tempo: 0.11 segundos

IterativeDeepening:
Nós expandidos: 2076556
Resposta: CCDBEEBDDCEECDDBEBECCDBBECCDDBB
Num. movimentos: 31
Tempo: 1417.87 segundos

5 Análise quantitativa

	#	Nós expandidos	Solução	Moves	Tempo
BFS	4	56	CDBB	4	0.0044
	8	368	BBDCCDBB	8	0.0285
	12	4072	DCCEBBDCC...	12	0.3292
	16	30236	CEBBDCCEB...	16	2.5112
	20	146492	BBECCEBBDC...	20	12.5053
Iterative deepening	4	84	CDBB	4	0.0149
	8	628	BBDCCDBB	8	0.1110
	12	4884	DCCEBBDCCDBB	12	1.0331
	16	43860	CEBBDCCEBBD...	16	9.7072
	20	214284	BBECCEBBDCCEBB...	20	51.6616
Uniform-cost	4	120	CDBB	4	0.0102
	8	684	BBDCCDBB	8	0.0574
	12	5844	DCCEBBDCC...	12	0.5203
	16	51300	CEBBDCCEB...	16	4.5931
	20	321028	BBECCEBBDC...	20	30.3154
A*	4	16	CDBB	4	0.0016
	8	36	BBDCCDBB	8	0.0034
	12	80	DCCEBBDCC...	12	0.0078
	16	508	CEBBDCCEBBD...	16	0.0509
	20	1256	BBECCEBBDC...	20	0.1291
Greedy best-first	4	16	CDBB	4	0.0014
	8	32	BBDCCDBB	8	0.0030
	12	1820	DCEBDCDBEC...	48	0.1973
	16	3220	ECDDBBEECC...	30	0.3463
	20	2644	BEECDDBEEB..	46	0.2824