Inteligência Artificial

Projeto

Grupo al012

95569 Eduardo Miranda 95666 Rodrigo Pinto

Introdução:

Neste relatório pretendemos analisar a solução proposta para o projeto de inteligência artificial.

Problema apresentado tem por objetivo descobrir se um certo tabuleiro de jogo numbrix tem solução ou não.

Avaliação experimental dos resultados:

Em seguida são apresentados o tempo de execução do programa, o número de nós expandidos e o número de nós gerados dados 10 inputs diferentes.

	DFS			BFS			A*			Greedy		
	Tempo execução	Nós gerados	Nós expandidos	Tempo execução	Nós gerados	Nós expandidos	Tempo execução	Nós gerados	Nós expandidos	Tempo execução	Nós gerados	Nós expandidos
Input1	0.129s	10	10	0.126s	10	9	0.116s	7	5	0.129s	7	5
Input2	7.567s	64402	64392	9.998s	73489	73489	0.910s	1725	996	0.741s	1284	712
Input3	0.574s	3312	3307	0.660s	4254	4254	0.148s	58	34	0.141s	54	32
Input4	0.298s	1254	1246	0.662s	4399	4399	0.229s	201	116	0.154s	84	51
Input5	0.711s	5406	5402	0.812s	5760	5760	0.346s	558	298	0.201s	198	100
Input6	2.201s	11004	10978	2.631s	12901	12901	1.120s	886	435	0.323s	187	97
Input7	2m10.612s	702821	702797	5m4.340s	1343928	1343928	5.608s	3348	1387	2.260s	823	357
Input8	0.665s	2331	2307	1.720s	8012	8012	1.669s	1320	797	0.758s	585	346
Input9	0.692s	2331	2307	1.774s	8012	8012	1.565s	1320	797	0.755s	585	346
Input10	0.517s	1436	1415	0.853s	2985	2985	0.894s	641	418	0.660s	442	289

Destes resultados podemos observar que todos estes algoritmos encontraram solução para os inputs fornecidos, portanto neste caso todos os algoritmos são completos. Em termos de eficiência conseguimos ver que como A* e Greedy usam uma heurística para encontrar solução acabam por expandir menos nós em relação à BFS e DFS que são algoritmos de procura cega.

Descrição da solução:

A razão pela qual todos os algoritmos completarem os inputs deve-se muito à maneira em que retornamos as actions possíveis. Na nossa solução as actions que retornamos são as posições para o sucessor ou antecessor do menor número encontrado que não os tem no tabuleiro.

Figura 1: Actions retornadas por diferentes tabuleiros

```
59  0  61  0  65  0  73  0  77
 0  0   0  0   0  0   0  0   0
55  0   0  0   0  0   0  0  79
 0  0   0  0   0  0   0  0   0
23  0   0  0   0  0   0  0  41
 0  0   0  0   0  0   0  0   0
19  0   0  0   0  0   0  0  37
 0  0   0  0   0  0   0  0   0
17  0   1  0   3  0   5  0  35
```

```
[(5, 3, 3), (3, 3, 3)]
[(5, 3, 3), (3, 3, 3)]
[(5, 3, 3), (3, 3, 3)]
```

```
[(8, 1, 2), (8, 3, 2), (7, 2, 2)]
[(8, 1, 2), (8, 3, 2), (7, 2, 2)]
[(8, 1, 2), (8, 3, 2), (7, 2, 2)]
[(8, 1, 2), (8, 3, 2), (7, 2, 2)]
```

```
25  0  27  0  37  0  43  0  45
 0  0   0  0   0  0   0  0   0
23  0   0  0   0  0   0  0  51
 0  0   0  0  56  0   0  0   0
19  0   5  4  57 62   0  0  69
 0  0   0  0  58  0   0  0   0
17  0   0  0   0  0   0  0  71
 0  0   0  0   0  0   0  0   0
15  0   9  0  77  0  79  0  81
```

O fator que influencia a eficiência do A* e greedy foi a nossa heurística. Esta heurística divide-se em dois passos, primeiro calcula a manhattan distance entre cada numero no tabuleiro e o seu sucessor mais próximo, caso esta distância seja superior ao valor (sucessor-atual) retorna infinito pois nunca será possível conectar os dois com os valores intermédios *(Figura 2)*. Para além disto ainda executa uma função chamada ***checkPath*** em que caso não exista caminho ou o comprimento do caminho entre o atual e o sucessor seja maior que a subtração entre eles retorna infinito *(Figura 3)*, para descobrir o caminho a função executa uma BFS. Caso nada disto se verifique retornamos o valor ***abs(k-manhattan_distance((i,j),numbers.get(board.matrix[i][j] + k)))*** isto retorna 0 se o valor k(que é a subtração entre o sucessor e o atual) e a manhattan distance entre os dois forem iguais, sendo este o melhor caso. Quanto maior a diferença entre estes dois maior a heuristica retornada.

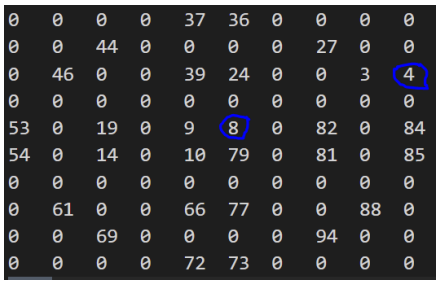

Figura 2:Exemplo em que a distância é maior que a subtração

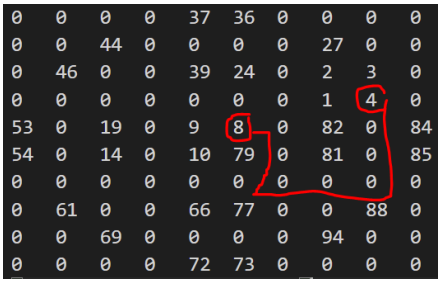

Figura 3: Exemplo onde o comprimento do caminho é maior que a subtração entre os dois

A segunda parte da nossa heuristica é um pequeno algoritmo que deteta se no estado atual existem *"bolhas"*. Isto é se existir um conjunto de zeros rodeado por uma sequência de números em que todos eles já têm o sucessor e o antecessor no tabuleiro *(Figuras 4 e 5)*. Para isto o algoritmo usa uma BFS para obter todos os valores da fronteira e faz a verificação dos sucessores e antecessores posteriormente.

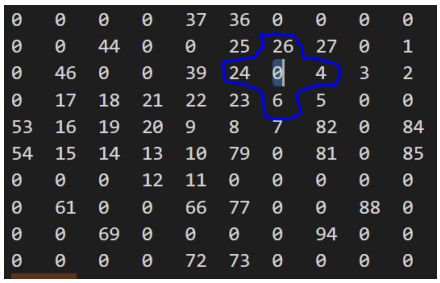

Figura 4: Exemplo de bolha

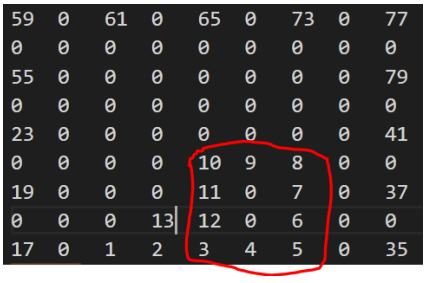

Figura 5: Outro exemplo de uma bolha

Antes de criarmos esta heurística tentámos resolver este problema com uma simples heuristica que apenas verificava se todos os números no tabuleiro tinham sucessor e antecessor e penalizávamos caso isso não se verificasse. No entanto esta heuristica muito ineficiente pois fazia com que explorásse estados que não faziam sentido como por exemplo estados com bolhas. Outra heuristica que testámos foi apenas a primeira parte da atual mas esta sofria do mesmo problema.