Relatório Numbrix - Grupo al020

Inteligência Aritficial P3 2021/2022

1 Descrição do Problema e da Solução

No âmbito da UC de Inteligência Artificial, foi-nos proposto solucionar o problema *Numbrix*. Neste existe uma grelha $N \times N$, onde cada célula pode conter um número positivo inteiro. O objetivo é preencher a grelha com uma sequência de números adjacentes horizontal ou verticalmente, com base numa instância inicialmente dada que já contém posições fixas de números em células.

Como auxílio de resolução ao problema, foram usados os vários algoritmos de procura informada e não informada disponibilizados no ficheiro *search.py* — mais concretamente, os seguintes: *breadth_first_tree_search*, *depth_first_tree_search*, *astar_search*, *greedy_search*.

A **solução** utilizada para a resolução do problema foi uma que cria ilhas ordenadas de números no tabuleiro (seja o conjunto dessas ilhas **I**). O algoritmo implementado tenta sucessivamente ligar o maior número da menor ilha (I_0) ao menor número da segunda menor (I_1), até que convirja em 1 só. Define-se uma ilha de números, I_j, como uma sequência de números adjacentes que são células vizinhas no tabuleiro, como abaixo exemplificado.

2	3	0
1	0	0
0	0	0

(a) $\mathbf{I} = \{\{1, 2, 3\}\}$

2	0	4
1	0	5
0	0	0

(b) $\mathbf{I} = \{\{1, 2\}, \{4, 5\}\}$

5	4	1
0	3	0
0	0	0

(c) $\mathbf{I} = \{\{1\}, \{3, 4, 5\}\}$

As **ações** a serem executadas a partir de um dado estado dependem de $\#\mathbf{I}$ (seja $x = min(I_0)$ e $y = max(I_0)$):

- ➔ Se $\#\mathbf{I} > 1$, para cada célula adjacente livre[1] de x meter $z = y + 1$.
- ➔ Se $\#\mathbf{I} = 1$, no caso de $x \neq 1$ meter $z_1 = x - 1$ para cada célula adjacente livre da posição de x e, analogamente, no caso de $y \neq N^2$ meter $z_2 = y + 1$ para cada célula adjacente livre da posição de y.

O **resultado** das ações consiste em copiar as estruturas do tabuleiro e auxiliares, definir a nova posição dada pelas ações e realizar a possível convergência de ilhas.

A **heurística** implementada tem em conta os seguintes aspetos (seja ainda $x = max(I_0)$, $y = min(I_1)$ e $g_{min} = min(I_0)$ e $g_{max} = max(I_N)$, assumindo que **I** contém N ilhas):

- ➔ Se não existem $g_{min} - 1$ posições livre desde a posição de g_{min} ou $N^2 - g_{max}$ desde a posição de g_{max}, devolve ∞.
- ➔ Se $\#\mathbf{I} > 1$, se $Manhattan(pos_x, pos_y) > x - y$, ou se a posição de y não é atingível por x, ou se a ação cria *dead spaces*[2] no tabuleiro, devolve ∞.
- ➔ Para cada célula livre no tabuleiro, contar o seu número de células livres adjacentes e elevar esse número a 1.4. No fim, devolver a soma desta contagem de todas as células livres.

[1] Uma célula diz-se livre se não está preenchida por nenhum número no tabuleiro.

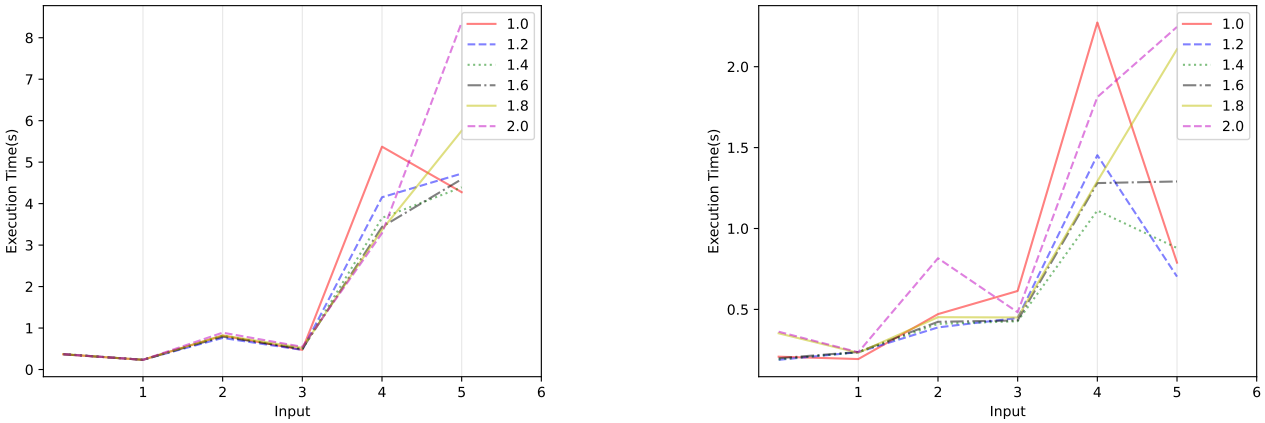
[2] Um *dead space* no tabuleiro é um sub-conjunto de posições livres que torna impossível qualquer tipo de preenchimento coerente com as restrições do problema.

2 Implementação e Análise dos Resultados

Dados os testes de *input/output* fornecidos pelo corpo docente, obtiveram-se os seguintes resultados aplicando o algoritmo implementado:

	Tempo de Execução (s)				Nós Gerados				Nós Expandidos			
Input	BFS	DFS	A*	GS	BFS	DFS	A*	GS	BFS	DFS	A*	GS
1	0.0004	0.0003	0.0008	0.0008	10	7	7	7	10	5	5	5
2	0.4837	0.1165	0.3179	0.3314	7280	1218	779	834	7270	1199	493	516
3	0.0852	0.0360	0.0158	0.0147	2186	991	59	54	2184	980	34	32
4	0.3192	0.1758	0.0630	0.0611	4473	2898	230	230	4473	2887	118	118
5	0.2385	0.0176	0.1479	0.0454	5760	430	516	188	5760	420	302	107
6	0.1693	0.0633	0.1000	0.0998	2857	1092	199	199	2857	1076	111	112
7	106.1534	39.4617	0.4971	0.3778	1343928	714831	928	744	1343928	714802	437	365
8	0.5891	0.2635	0.4280	0.4117	4611	2653	599	588	4584	2634	386	380
9	0.6193	0.2668	0.4232	0.4134	4611	2653	599	588	4584	2634	386	380
10	0.1804	0.0536	0.2072	0.1835	2303	957	338	312	2303	943	232	215

Primeiramente, o uso da heurística foi um elemento nitidamente diferenciador. Porém, dada a impossibilidade de apurar qual dos algoritmos a usar a heurística (*A-Star Search* ou *Greedy Search*) era o mais eficiente, criou-se uma ferramenta em *Python* que gerava ficheiros *input/output* que nos permitissem ver a fundo como otimizar a heurística. Para o efeito, geraram-se 6 instâncias de tabuleiros 10×10 e utilizaram-se ambos os algoritmos com a mesma heurística, com incrementos de 0.2 no expoente da heurística, j, tal que $j \in [1, 2]$.



(a) *A-Star Search* (b) *Greedy Search*

Figura 1: Comparação do expoente da heurística entre *A-Star Search* e *Greedy Search*

Verificou-se que para os mesmos testes o algoritmo *Greedy Search* conseguia uma redução significativa de tempo de execução. Para além dessa análise experimental feita, destacou-se o facto de o expoente 1.4 ser em média o que mais reduzia o tempo de execução. Por fim, considerando todos os fatores supramencionados, decidiu-se optar pela implementação com **procura informada que usa o algoritmo *Greedy Search* e expoente 1.4 na heurística**.