Relatório IA 2021/2022 (P3)

Mara Alves 95625
Margarida Rodrigues 95627

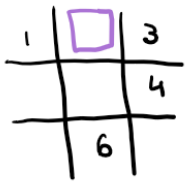
Obtivemos os seguintes resultados ao testar o nosso programa utilizando o código fornecido no ficheiro search.py e também a função time() do python:

	DFS			BFS			A*			GREEDY		
input	**t**	**G**	**E**	**t**	**G**	**E**	**t**	**G**	**E**	**t**	**G**	**E**
1	0.0005	6	6	0.0005	6	6	0.0005	6	6	0.0005	6	6
2	0.007	71	38	0.1375	1306	1306	0.2305	1306	1306	0.0535	355	307
3	0.0045	37	31	0.0105	81	81	0.0105	81	81	0.0110	81	81
4	0.0405	290	286	0.0495	360	360	0.0540	360	360	0.0240	169	152
5	0.0080	89	34	0.2725	2324	2324	0.4815	2324	2324	0.0385	279	195
6	0.0475	190	185	0.0890	379	379	0.0999	379	379	0.0805	326	324
7	0.0495	194	176	0.1069	412	412	0.1105	412	412	0.0945	353	348
8	0.0349	118	107	0.1380	374	374	0.1565	374	374	0.0965	276	267
9	0.0359	118	107	0.1330	374	374	0.1459	374	374	0.0955	276	267
10	0.0315	104	100	0.0465	159	159	0.0479	159	159	0.0480	159	159

Legenda: *t = tempo em segundo* *G = número de nós gerados* *E = número de nós expandidos*

Com base nestes resultados, concluímos que obtemos melhor desempenho utilizando DFS como método de procura, pois apresenta os números mais baixos tanto para os nós expandidos e gerados como para o tempo de execução. Observamos ainda que obtemos sempre resultados de tempo inferiores a 1 segundo.

Isto deve-se à forma como implementámos a função actions(). A nossa função actions() gera todas as ações possíveis para uma determinada posição (a primeira posição encontrada vazia) e efetua vários cortes: primeiro cortamos todos os números já posicionados no tabuleiro, de seguida verificamos as adjacências desta posição verificando que não bloqueamos nenhum dos adjacentes, por fim fazemos cortes calculando o maior antecessor e o menor sucessor já posicionados no tabuleiro e calculamos as distâncias de Manhattan para garantir que essa distância é menor ou igual à diferença entre o sucessor/antecessor e aquele que queremos posicionar.
Consequentemente, são geradas menos ações possíveis, logo menos nós para testar e expandir.



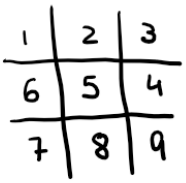
possíveis:

[1, 2, 3, 4, 5, 6, 7, 8, 9]

Neste caso daríamos prioridade ao 2 por ter uma melhor heurística:

$h(2) = 6 - 4 = 2 \leq 5$

$h(8) = 6 - 1 = 5 \leq 5$



Neste estado, $h = 9 - 9 = 0 \leq 0$

é admissível

Obtivemos os piores resultados na procura A* pois com esta implementação, não chegámos a uma heurística que melhorasse o seu desempenho. No entanto refletimos sobre o assunto. Dos vários algoritmos que testámos, aquele que nos fez mais sentido foi calcular o comprimento da sequência que o número a ser posicionado forma com os restantes já posicionados no tabuleiro e calcular a diferença entre o número de posicionados e este valor. A heurística acabou por se tornar pouco eficiente pelos cortes que já tínhamos feito no actions().

Testámos também uma heurística que retornava a diferença entre n^2 (sendo n o valor da dimensão da grelha n*n) e o comprimento da lista de ações para aquele estado. Pois, nas aulas abordámos a heurística de valor menos restritivo que consiste em dada uma variável atribuir-lhe o valor que elimina menos valores do domínio das outras.

Uma otimização que poderíamos fazer seria, em vez de calcular as ações possíveis para a primeira posição vazia, calcularmos as ações possíveis para uma melhor posição, ou seja, uma posição que já tivesse vários números adjacentes. No entanto, não encontrámos maneira eficiente de o implementar.