Relatório Projeto Inteligência Artificial P4 2021/2022

Grupo: al006

Alunos: Guilherme Almeida Patrão (99230) e João Domingos Baracho (99248)

Descrição do Problema

O projeto tem como objetivo desenvolver um programa que resolva o problema Takuzu utilizando técnicas de procura de IA. O jogo consiste num puzzle lógico que envolve a colocação de valores (0 e 1) em células de uma grelha quadrada em que o objetivo é preencher toda a grelha de modo a que as seguintes condições se verifiquem: número igual de 1s e 0s na mesma linha (ou diferença de 1 em casos de grelha ímpar), todas as linhas e colunas são diferentes e não há mais do que dois números iguais adjacentes, quer seja horizontal ou verticalmente.

A forma como resolvemos o problema passa por o formalizar como um problema de procura, com todos os conjuntos de propriedades que este contém, tal como aprendemos nas aulas teóricas. Cada estado corresponde a uma configuração de um tabuleiro e cada ação corresponde a colocar um valor numa dada posição (linha i e coluna j). Internamente, cada tabuleiro (classe Board) é representado por uma lista de listas em que cada lista pertencente à lista principal corresponde a uma linha do tabuleiro.

Após executar a função result (aplicar uma ação a um estado existente e, consequentemente, gerar um estado novo), podemos utilizar a goal_test para verificar se o tabuleiro é uma solução. Para tal, é necessário verificar as condições referidas no primeiro parágrafo deste documento.

A função heurística escolhida, para a procura gananciosa e procura A*, é uma função que calcula o número de linhas e colunas não completas ainda, ou seja, ainda não preenchidas totalmente. Previamente, tínhamos usado uma função heurística que retornava a quantidade de espaços por preencher no tabuleiro. No entanto, esta heurística seria inútil, pois cada nó iria gerar nós como todos o mesmo número de posições por preencher e, portanto, com o mesmo valor da função heurística, tornando este valor irrelevante. É claro que, seguindo a mesma lógica para a heurística escolhida, podemos concluir que também será irrelevante visto que um estado gera apenas dois, um com 0 numa posição e outro com 1 nessa mesma posição, tendo ambos o mesmo número de linhas e colunas incompletas. No entanto, no caso de não usarmos restrições na geração de ações (com base em CSPs), esta heurística é, evidentemente, melhor.

Relatório Projeto Inteligência Artificial P4 2021/2022

Grupo: al006

Alunos: Guilherme Almeida Patrão (99230) e João Domingos Baracho (99248)

Testes e Análise Experimental de Dados

Utilizando os testes públicos dados para testagem do projeto, fizemos uma comparação entre os tempos de execução, nº de nós expandidos e nº de nós gerados para cada um dos 4 algoritmos de procura: DFS (procura profundidade primeiro), BFS (procura largura primeiro), Greedy Search (procura gananciosa) e A* Search (procura A*). Os resultados foram os seguintes:

	TEMPO DE EXECUÇÃO (em segundos)				**NÚMERO DE NÓS EXPANDIDOS**				**NÚMERO DE NÓS GERADOS**			
Teste Nº	DFS	BFS	Greedy Search	A*	DFS	BFS	Greedy Search	A*	DFS	BFS	Greedy Search	A*
1	0,153	0,135	0,128	0,143	7	7	7	7	7	7	7	7
2	0,145	0,144	0,141	0,135	1	1	1	1	1	1	1	1
3	0,156	0,139	0,143	0,162	113	130	73	130	116	130	80	130
4	0,142	0,134	0,161	0,136	15	14	11	14	15	15	14	15
5	0,148	0,151	0,136	0,157	168	225	105	225	171	226	111	226
6	0,157	0,146	0,166	0,155	81	211	197	211	87	211	200	211
7	0,133	0,137	0,148	0,147	35	81	71	81	40	81	73	81
8	0,144	0,139	0,134	0,146	2	2	2	2	2	2	2	2
9	0,147	0,143	0,151	0,145	50	84	80	84	56	84	82	84
10	0,142	0,141	0,138	0,168	91	99	66	99	93	99	68	99
11	0,145	0,148	0,167	0,159	115	192	131	192	122	192	135	192
12	0,151	0,165	0,147	0,145	24	28	25	28	25	28	28	28
13	0,153	0,161	0,149	0,156	29	29	27	29	29	29	28	29

Podemos observar que, embora a procura Greedy e A* sejam procuras informadas e DFS e BFS sejam procuras cegas, os tempos de execução são todos semelhantes. Isto deve-se ao facto de, tal como explicado no ponto anterior, a heurística apenas ser impactante no caso de não haver restrições já aplicadas, de um modo semelhante a Constraint Satisfaction Problems (CSPs). Como nas ações são apenas retornadas no máximo duas, em que a única diferença entre estas é a peça a ser usada (a posição é igual), qualquer uma delas terá o mesmo valor da função heurística, pois terão o mesmo número de linhas e colunas incompletas.

Podemos observar que todos os algoritmos usados são completos. Observando a tabela e calculando as médias para cada conjunto de dados podemos concluir que, em média, **BFS** é o algoritmo que demora menos tempo, sendo o **A*** o que demora mais. **DFS** é o algoritmo que, em média, expande menos nós e os algoritmos **BFS** e **A*** são os que expandem mais (ambos empatados com o mesmo valor médio). Podemos também concluir que o algoritmo que menos nós gera é o **DFS** e, novamente, um empate entre **BFS** e **A*** no que toca ao número médio de nós gerados.

Em termos de eficiência, qualquer um dos algoritmos seria uma boa escolha visto que, para esta implementação, não diferem muito entre si. No entanto, tendo em conta o tempo, quantidade de nós gerados e expandidos, o mais eficiente seria o algoritmo de procura de profundidade primeiro, **DFS**.