Relatório Projeto IA P4 2021/2022

Grupo: al040
Alunos: Rafael Serra e Oliveira (99311) e Tiago Vieira da Silva (99335)

1 Descrição do Problema

O **Problema Takuzu** (ou *Binary Puzzle*) consiste num puzzle $N \times N$ onde cada célula pode conter o número 0 ou 1. O objetivo é preencher todas as células vazias do puzzle tendo em conta as seguintes restrições: Não podem haver 3 números seguidos iguais por linha ou coluna, não podem haver linhas nem colunas iguais e o número de 0s e 1s por linha ou coluna têm que ser iguais (ou com uma diferença de um entre eles para tabuleiros de tamanho ímpares).

2 Descrição da Solução Apresentada

Para resolver o problema acima apresentado, decidimos modelá-lo como um ***Constraint Satisfaction Problem*** (*CSP*). Para tal, iremos ter vários estados (*state*), cada um destes possuindo um tabuleiro (*state.board*) que, por sua vez, possuirá uma representação matricial do tabuleiro (*board.matrix*) e uma matriz de domínios (*board.domains*), que serão as possíveis escolhas de valor para cada posição, conforme as restrições impostas.

No sentido da resolução em si do problema, faremos uma procura a partir de um estado inicial, que possuirá tanto uma representação do tabuleiro como os domínios iniciais possíveis pré-calculados. Por cada nível da árvore iremos selecionar a posição com o menor domínio (usando uma ordem definida como critério de desempate), e iremos expandir os ramos desse estado, que consiste em escolher para a posição em análise cada um dos valores possíveis de acordo com o seu domínio.

As restrições descritas acima serão aplicadas no cálculo dos domínios para cada posição, sendo após, cada ação, depois de definido o valor de uma variável, recalculados os domínios afetados de acordo com as referidas restrições (e apenas esses).

De forma a que a árvore de procura não ganhásse uma largura incomportável, aplicámos o princípio em *CSP*s de considerar apenas uma variável por nível, pelo que o método *actions* da classe *Takuzu*, em vez de devolver todas as jogadas legais em todas as posições, apenas apenas devolve as ações possíveis para jogadas na primeira casa livre do tabuleiro, de acordo com uma ordem definida. Estas correspondem às jogadas de cada um dos valores do seu domínio nessa posição, na forma de tuplos *(linha, coluna, valor)*.

Para além disso, de forma a encurtar a árvore de procura, implementámos uma otimização nesse mesmo método *actions* de forma a que este se comportasse da seguinte forma:

- Se existir alguma posição no tabuleiro com domínio vazio (*i.e.*, de dimensão 0), então não é possível continuar e o método retorna apenas a ação vazia (o tuplo *()*);
- Caso contrário, se existir alguma posição vazia no tabuleiro com domínio unitário (*i.e.*, de dimensão 1), então será obrigatório em algum ponto colocar esse valor nessa posição, pelo que é preferível fazê-lo já no caso de tal ação restringir os domínios de outras posições. Assim, nesse caso, o método devolve apenas essa ação *(linha, coluna, unico_valor_no_dominio)*. Note-se que, no caso de existir mais do que uma posição nessa situação, o algoritmo escolhe a primeira de acordo com a ordem definida;

- Apenas no caso de isso não se verificar, ou seja, se todas as posições livres tiverem domínio de dimensão 2 ($D_{X_{i,j}} = \{0, 1\}$, não estando sujeito a restrições), será necessário fazer uma bifurcação na árvore de procura e portanto são devolvidas as duas ações correspondentes à primeira posição vazia (os tuplos *(i, j, 0)* e *(i, j, 1)*).

No caso das procuras que necessitam de uma **função heurística** para as guiar, como é o caso da **Procura *Greedy*** e da **Procura A***, continuámos a linha de raciocínio de *CSP*s, implementando a heurística **MRV (*Most Restrictive Variable*)**. Assim sendo, o valor da função heurística por nós desenvolvida irá ser, para um dado nó, mais baixo o quão mais restritiva será a variável em análise. Para isso, avaliamos a dimensão do domínio da variável e o número de possíveis domínios de posições adjacentes que poderão ser restringidos pela escolha dessa variável.

Visto que as procuras *A** e *Greedy* já implementadas são apenas procuras *Best First*, sendo que para a *A** se tem $f(n) = g(n) + h(n)$, e para a *Greedy* $f(n) = h(n)$, irá haver escolha de variáveis, sendo assim pertinente aplicar a heurística *MRV*.

Para além da *MRV*, como estamos a modelar o problema como um *CSP*, poderiamos implementar também uma heurística **LCV (*Least Restrictive Value*)**. Para isso, iriamos ter que calcular quais são os domínios que iriam ser restringidos ao colocar um valor. Isso revelou-se ser muito dispendioso e, como tal, optámos por não implementar essa heurística na nossa função h(n), sendo mais eficiente recalcular os domínios apenas na expansão do nó.

3 Análise dos Resultados

De modo a testar os nossos resultados, recorremos à classe *InstrumentedProblem* do código de apoio de forma a registar o número de nós gerados e nós expandidos durante a resolução de cada teste. Para medir o tempo de execução, utilizámos a ferramenta **hyperfine**, que permite fazer um *benchmarking* fidedigno e estatisticamente viável (com parâmetro de *warmup = 1*).

Para os *inputs* dos testes realizados sobre a solução implementada, recorremos a duas baterias de testes distintas:

- **Bateria A:** 13 tabuleiros de dimensões entre 6 e 31, disponibilizados pelo corpo docente.
- **Bateria B:** 205 tabuleiros de dimensões entre 4 e 12, gerados automaticamente e verificados (através de modificação do método *goal_test* para obter todas as soluções) para só ter uma solução possível, disponibilizados publica e livremente num repositório comunitário de alunos, sob a licensa *Unlicense*.

Os resultados da testagem dos respetivos algoritmos encontra-se no final do ficheiro.

Podemos concluir a partir dos dados obtidos que, no geral, o tempo de execução dos vários tipos de procura é muito semelhante, assim como a quantidade de nós expandidos e gerados. As procuras são todas **completas**, visto que a pronfundidade máxima do problema é N^2, sendo N o número de espaços vazios.

A complexidade temporal e espacial do problema será de $\mathbf{O(2^n)}$, visto que o fator de ramificação é, no máximo, 2.

A performance ligeiramente superior da *DFS* em relação à *BFS* é explicado pelo facto de que, visto que a solução do problema encontra-se sempre na profundidade máxima da procura, a BFS irá ter que expandir todos os nós até à profundidade máxima, enquanto que a DFS não terá que obrigatoriamente o fazer. A aplicação das heurísticas levou a um melhoramento dos tempos de execução relativamente a uma *DFS*. Sendo assim, decidimos optar por uma Greedy Search.

Apêndice

Sendo n a dimensão do tabuleiro e f o número de espaços em branco:
Bateria de Testes A

			Tempo de Execução (ms)				Nós Gerados				Nós Expandidos			
Teste	n	f	BFS	DFS	A*	GS	BFS	DFS	A*	GS	BFS	DFS	A*	GS
T01	4	7	111	112	113	113	7	7	7	7	7	7	7	7
T02	6	7	117	117	119	118	7	7	7	7	7	7	7	7
T03	8	42	129	130	127	128	43	42	42	42	43	43	43	43
T04	9	32	159	133	129	129	32	32	32	32	32	32	32	32
T05	10	55	136	138	137	136	59	58	55	55	59	59	57	57
T06	12	79	149	150	150	151	85	81	85	85	85	82	85	85
T07	14	69	155	157	157	158	69	69	69	69	69	69	69	69
T08	15	19	160	159	161	161	19	19	19	19	19	19	19	19
T09	18	139	194	197	194	193	139	139	139	139	139	139	139	139
T10	20	184	227	222	231	228	184	184	184	184	184	184	184	184
T11	21	180	243	233	235	239	180	180	180	180	180	180	180	180
T12	25	166	306	292	294	292	166	166	166	166	166	166	166	166
T13	31	180	461	455	477	459	180	180	180	180	180	180	180	180

Bateria de Testes B (excerto)

			Tempo de Execução (ms)				Nós Gerados				Nós Expandidos			
Teste	n	f	BFS	DFS	A*	GS	BFS	DFS	A*	GS	BFS	DFS	A*	GS
T10_01	10	80	127	125	107	107	288	277	94	94	288	281	99	99
T10_02	10	79	162	150	105	105	614	517	81	81	614	520	89	89
T10_03	10	80	882	737	109	109	6855	5913	107	107	6855	5919	120	120
T10_04	10	79	165	152	123	123	618	518	225	225	618	522	231	231
T10_05	10	81	168	154	118	118	665	550	185	185	665	555	194	194
T10_06	10	78	248	224	126	126	1378	1198	245	245	1378	1202	253	253
T10_07	10	81	683	621	106	106	5054	4784	85	85	5054	4789	95	95
T10_08	10	79	120	118	119	119	229	211	196	196	229	215	200	200
T10_09	10	81	4436	3517	106	106	36334	31767	82	82	36334	31773	96	96
T10_10	10	79	125	122	124	125	274	245	235	235	274	249	240	240
T10_11	10	80	120	119	105	106	232	219	82	83	232	221	91	92
T10_12	10	79	292	261	106	107	1758	1516	91	93	1758	1520	100	102
T10_13	10	79	560	503	140	141	4064	3771	362	362	4064	3775	371	371
T10_14	10	81	195	184	128	128	893	818	261	261	893	822	268	268
T10_15	10	81	738	576	107	108	5604	4415	95	95	5604	4421	107	107
T10_16	10	81	146	142	106	106	454	424	87	87	454	428	95	95
T10_17	10	80	203	179	125	125	960	775	238	238	960	781	249	249
T10_18	10	80	742	595	113	113	5617	4587	135	135	5617	4595	149	149
T10_19	10	78	173	163	155	155	699	629	482	483	699	631	487	487
T10_20	10	78	148	144	107	107	476	450	89	89	476	456	100	100
T10_21	10	79	197	190	106	107	910	865	91	91	910	869	98	98
T10_22	10	80	657	598	106	106	4863	4545	80	80	4863	4550	92	92
T10_23	10	79	311	297	110	110	1925	1864	113	114	1925	1872	127	128
T10_24	10	78	155	150	107	107	545	510	91	91	545	516	101	101
T10_25	10	81	187	181	107	107	817	781	93	93	817	785	100	100
T10_26	10	80	126	125	106	106	285	277	91	91	285	280	96	96
T10_27	10	79	164	147	118	118	624	477	183	183	624	482	194	194
T10_28	10	81	185	161	154	154	800	602	470	470	800	606	476	476
T10_29	10	79	120	115	118	118	231	183	184	184	231	187	189	189
T10_30	10	78	128	121	121	121	303	246	213	213	303	248	217	217
T10_31	10	81	178	158	106	106	734	567	86	86	734	571	95	95
T10_32	10	80	268	241	106	106	1541	1345	85	89	1541	1349	95	98
T10_33	10	78	1787	1383	728	726	14497	11985	4975	4977	14497	11991	4985	4987
T10_34	10	78	136	134	108	108	379	358	101	101	379	364	110	110
T10_35	10	80	226	217	107	106	1154	1108	91	91	1154	1112	100	100
T10_36	10	79	270	245	127	127	1570	1389	254	254	1570	1393	262	262
T10_37	10	78	433	379	110	110	2932	2580	117	117	2932	2586	127	127
T10_38	10	81	225	207	177	177	1147	1017	650	650	1147	1022	659	659
T10_39	10	80	1160	981	106	106	9113	8134	83	83	9113	8139	95	95
T10_40	10	79	290	279	109	109	1731	1691	106	107	1731	1699	120	121
T10_41	10	83	244	235	106	107	1307	1278	91	91	1307	1282	100	100
T10_42	10	81	2101	1662	107	107	17032	14372	88	88	17032	14378	101	101
T10_43	10	78	151	147	113	113	498	474	148	148	498	478	153	153
T10_44	10	79	121	120	121	122	239	230	212	213	239	233	217	218
T10_45	10	79	308	251	108	107	1891	1422	95	95	1891	1428	107	107
T10_46	10	79	156	144	116	116	557	464	169	169	557	467	176	176
T10_47	10	80	192	186	107	107	864	835	91	92	864	839	99	100
T10_48	10	80	180	158	149	150	757	573	433	434	757	577	439	440
T10_49	10	78	142	139	111	111	420	402	126	126	420	406	131	131
T10_50	10	79	127	124	126	126	284	261	251	251	284	265	255	255
T12_01	12	114	1105	881	324	323	6714	5519	1386	1386	6714	5527	1397	1397
T12_02	12	115	8085	6013	3595	3605	50942	41486	21233	21233	50942	41495	21247	21247
T12_03	12	114	9617	6567	3660	3675	59649	45325	21726	21726	59649	45334	21739	21739
T12_04	12	113	1485	1182	831	831	9333	7714	4555	4555	9333	7724	4569	4569
T12_05	12	115	6187	4174	2161	2177	38679	28660	12709	12709	38679	28671	12725	12725