João Filipe Meneses Henriques
João Nuno Santos de Gusmão Guedes

Turn 12

FEUP-PLOG, Turma 3MIEIC05, Grupo 17

– Monograph –

11 de Dezembro de 2011

Springer

Conteúdo

1

Resumo

Este artigo foi elaborado no contexto do curso de Programação em Lógica, e incide sobre a programação em lógica com restrições. A programação em lógica com restrições permite restringir problemas por domínio de soluções e por condições que devem ser cumpridas.

Com o objectivo de avaliar a viabilidade da programação com restrições para resolver problemas de lógica de complexidade relevante, foram desenvolvidos algoritmos de resolução e de geração de novas soluções para dimensões variáveis do jogo Turn 12. Para o efeito fez-se recurso à biblioteca de restrições para o cojunto dos domínios finitos clp(FD) do SICStus Prolog.

O método utilizado compreende a rotação dos dígitos de cada face, sendo o domínio o conjunto de rotações possíveis. A combinação das faces é feita iterativamente e não simultâneamente, por forma a optimizar os recursos.

Obtiveram-se resultados em tempo útil para solucionar o problema original e para um número de digitos não superior a 60, o que permitiu concluir que a metodologia utilizada é adequada para a resolução de problemas do género.

2

Secções

2.1 Introducão

Programação em lógica com restrições é uma junção de dois paradigmas: solução de restrições e programação em lógica. Esta combinação permite uma concepção mais expressiva e flexível — e em alguns casos mais eficiente - de problemas lógicos.

2.1.1 Motivação

A motivação deste trabalho incidiu na compreensão de um paradigma de programação que já nos é familiar, envolvendo uma nova componente de restrições; resolver problemas lógicos com restrições de uma forma geral, tendo a possibilidade de os refinar e otimizar para uma solução particular.

2.1.2 Objectivos

Resolver a versão original do jogo Turn12 recorrendo a restrições; gerar cubos com um número de dígitos variável, e avaliar se estes têm solução ou não segundo as restrições definidas.

2.2 Descrição do problema

O problema centra-se num cubo em que cada face contém dígitos numerados de 3 a 9, aleatoriamente. Na junção das arestas de cada face, a soma dos dois dígitos que se encontram deverá ser igual a 12.

A solução original (Fig. 2.1) com 24 dígitos por face é única.

Na geração de problemas, a resolução que apresentamos contempla dígitos ilimitados, e uma vez que não existe qualquer padrão associado à sequência de dígitos no problema original, estes são gerados aleatoriamente.

No entanto, para garantir a unicidade de solução, em valores demasiado elevados (superiores a 60 por face), as limitações de computação começaram-se a sentir e torna-se improvável gerar uma solução única em tempo útil.

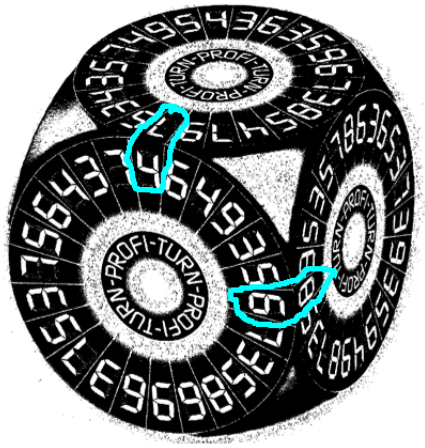

Figura 2.1. Problema original, com os pontos de contacto visíveis assinalados

2.3 Ficheiros de Dados

Os ficheiros de dados são simples ficheiros de texto (*.txt*), contendo em cada linha os dígitos de cada face. As faces encontram-se pela seguinte ordem: Topo, Baixo, Frente, Trás, Esquerda e por fim Direita.

De seguida encontra-se um exemplo deste ficheiro com os valores do problema original:

Exemplo 1 Problema original do turn12, com 24 dígitos por face:

```
356735869637537564374649
343574954363596738547975
353748635975458485676439
349576573795398457964379
357863653739359498735895
348765738453874583769785
```

2.4 Variáveis de Decisão

As variáveis de decisão usadas na solução foram as rotações que podiam ser feitas em cada face do cubo.

Estas rotações têm um domínio compreendido entre 1 e o número de dígitos de cada face.

2.5 Restrições

Independentemente do número de dígitos em cada face, verificou-se que as restrições seriam sempre as mesmas: a soma dos dígitos no ponto de contacto entre duas faces tem de ser 12.

Na implementação foi utilizado no *SICStus Prolog* a biblioteca de restrições para o conjunto dos domínios finitos *clp(FD)*, onde foram definidas variáveis que seriam preenchidas com os valores dos pontos de contacto tendo em conta a rotação aplicada sobre cada face, e restringidas de forma a que a soma dos mesmos seja obrigatoriamente 12.

2.6 Função de Avaliação

Os predicados de avaliação chamam-se *turn12*. Foram implementados três com o mesmo nome na solução, mas o principal aceita como parâmetros de entrada as 6 faces do cubo, que deverão ser cada um uma lista numérica de dígitos. Se for encontrada uma solução, são definidos como parâmetros de saída as rotações respectivas de cada face. São também definidos e agrupados 4 a 4 os valores da solução. Este predicado tem como objectivo apenas retornar a primeira solução encontrada.
O segundo predicado, com o mesmo nome, tem como parâmetro de entrada a localização do ficheiro com a informação de cada face, e só termina até imprimir todas as soluções do problema. Por fim, existe ainda um último predicado que não aceita qualquer parâmetro, e que chama o predicado anteriormente descrito, com um ficheiro numa localização predefinida.

2.7 Estratégia de Pesquisa

A estratégia de pesquisa baseia-se na rotação de cada face até encontrar uma solução. De forma a optimizar a resolução do problema, o algoritmo começa apenas com duas faces, e tenta encontrar através do predicado de etiquetagem *labeling* uma rotação que consiga somar o valor 12 nos seus pontos de contactoo. Quando encontrada, é passada para outra face do cubo, e com recurso a outro predicado de etiquetagem é tentado encontrar uma rotação para esta última que consiga satisfazer as mesmas restrições para os pontos de contacto em comum com as duas faces anteriores. É seguida sempre a mesma estratégia para as restantes faces, fazendo com que a complexidade algorítmica seja sempre a menor possível. A razão para a utilização de vários predicados

de etiquetagem deve-se ao facto de ser utilizado um predicado auxiliar (*shifted_face*), que preenche as variáveis utilizadas para testar as restrições com base na rotação correspondente. Verificou-se que, quando utilizado um único predicado para o mesmo efeito, o *shifted_face* era chamado desnecessariamente mesmo para as faces em que a rotação se mantinha igual e que não tinham originado a falha. Após isolados, o predicado só passou a ser chamado para a face do respectivo labeling, e evitando um varrimento contínuo e dispendioso nas várias listas de dígitos de cada face.

2.8 Visualização da Solução

A visualização da solução é feita através do predicado *project_cube*. Este predicado faz uma projecção (Fig. 2.2) do cubo de 3 dimensões em 2, e tem como parâmetros de entrada os valores da solução encontrada pelo algorítmo de procura. Estes parâmetros são passados em grupos de 4, representando os pontos de contacto de cada face pela seguinte ordem: ponto superior, direito, inferior e esquerdo. A ordem de passagem das faces é : Topo,Traseira, Direita, Esquerda, Dianteira, Inferior.

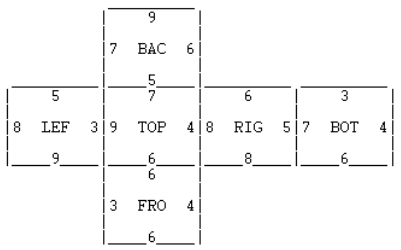

Figura 2.2. Impressão da solução do problema original do turn12.

Adicionalmente, foi desenvolvido o predicado *print_rots*, que imprime o número de rotações (Fig. 2.3) que devem ser dadas em cada face sobre o problema original para chegar à solução, no sentido contrário ao dos ponteiro do relógio.

```
[ Rotations ]
Top   : 14
Bottom: 2
Front : 6
Back  : 23
Left  : 23
Right : 4
```

Figura 2.3. Impressão das rotações do problema original do turn12.

Por fim são imprimidas algumas estatísticas obtidas através do predicado interno *fd_statistics*, que mostra alguns valores úteis que ajudam a perceber o número de *constraints* que foram testadas, *backtracks* ocorridos, entre outros.

2.9 Resultados

De seguida seguem-se dois exemplos da resolução de dois problemas com complexidades diferentes, gerados com o predicado *turn12gen*

Exemplo 2 Problema com 52 dígitos por face:

```
8987698939348595456596537597964569768569756345737893
6756434375953493785343639458956939579893487543689353
3579793975458794354537534574645348648939348956983756
6789345463497649386734675747959348987858395394783478
4636795348358376783754934938935675854758379538676936
6456984789496935693937974554589635464347893474635735
```

```
[ Rotations ]
Top   : 23
Bottom: 51
Front : 19
Back  : 27
Left  : 15
Right : 7
```

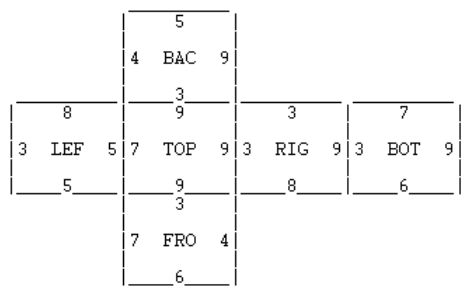

Figura 2.4. Solução do problema com 52 dígitos por face

Exemplo 3 Problema com 72 dígitos por face:

```
894896346958797867878397593785997678947876483959373486864543438685698676
739475798946785793456537485839735954538537834873597678963893563438758346
876768685479345693587896839654685468943463769386734938644896978963478579
467998968397969378534737893478583896345647846478475486739653976367948939
469893649494653679565694856947898675395494956976355856784965975784895785
784837634596534396493678394685453865348743489746597473438967439397839853
```

```
[ Rotations ]
Top   : 67
Bottom: 15
Front : 38
Back  : 58
Left  : 32
Right : 7
```

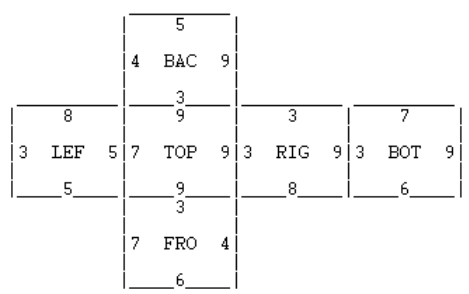

Figura 2.5. Solução do problema com 72 dígitos por face

2.10 Conclusões e Perspectivas de Desenvolvimento

Com este projecto concluímos que através da linguagem *Prolog* e com a ajuda da biblioteca de restrições sobre o conjunto dos domínios finitos *clp(FD)* facilmente conseguimos implementar algoritmos eficientes que conseguem resolver de forma rápida e eficiente problemas que encontramos no dia-a-dia.

O algoritmo desenvolvido, trata de resolver um simples puzzle, mas com uma complexidade que aumenta exponencialmente conforme o número de dígitos de cada face. No problema original, cada face tinha apenas 24 dígitos, mas o algoritmo consegue tratar de qualquer número natural igual ou superior a 1, sendo o hardware a única limitação.

Foi também desenvolvido um algoritmo que gera puzzles, mas como não foi possível encontrar nenhum padrão no problema original, estes são preenchidos aleatoriamente, logo quanto maior for número de dígitos por face mais complexo é conseguir encontrar um problema com uma solução única.

Referências

1. Kajan E (2002) Information technology encyclopedia and acronyms. Springer, Berlin Heidelberg New York
2. Página puzzle turn12, http://www.jaapsch.net/puzzles/turn12.htm
3. Patente puzzle turn12, http://www.jaapsch.net/puzzles/patents/de20112728u.pdf
4. SCStus Prolog, http://www.sics.se/isl/sicstuswww/site/index.html
5. Constraint Logic Programming over Finite Domains, http://www.sics.se/sicstus/docs/latest4/html/sicstushtml/lib_002dclpfdhtml#lib_002dclpfd

3

Anexo

3.1 Turn12

```
1
2  :- use_module(library(clpfd)).
3  :- use_module(library(random)).
4  :- use_module(library(lists)).
5
6
7
8  not(P) :- call(P), !, fail.
9  not(_).
10
11 /*********************************************************
12  * Manipulation of file path.
13  * defines the main path folder, needed in windows
14  * or the installation folder of prolog will be used
15  *********************************************************/
16
17 % uncommet to ignore
18 %main_path('').
19 main_path('C:/Users/Joao Henriques/Desktop/Eng. Informática/
       FEUP/PLOG/turn12/src/').
20
21 construct_full_file_path(File, Path) :-
22   main_path(Dir),
23   atom_concat(Dir, File, Path).
24
25
26 /*********************************************************
27  * Gets and Sets the elements of the list, in pre marked
28  * positions, as the new contact points
29  *********************************************************/
30
```

```
31 get_elements_pos( _, _, -1,-1,-1,-1, A,B,C,D, A,B,C,D ):-!.
32 get_elements_pos( [X|T], Pos, Pos,P2,P3,P4, _,B,C,D, E,F,G,H ) :- !,
33   Pos1 is Pos - 1,
34   get_elements_pos( T, Pos1, -1,P2,P3,P4, X,B,C,D, E,F,G,H ).
35 get_elements_pos( [X|T], Pos, P1,Pos,P3,P4, A,_,C,D, E,F,G,H ) :- !,
36   Pos1 is Pos - 1,
37   get_elements_pos( T, Pos1, P1,-1,P3,P4, A,X,C,D, E,F,G,H ).
38 get_elements_pos( [X|T], Pos, P1,P2,Pos,P4, A,B,_,D, E,F,G,H ) :- !,
39   Pos1 is Pos - 1,
40   get_elements_pos( T, Pos1, P1,P2,-1,P4, A,B,X,D, E,F,G,H ).
41 get_elements_pos( [X|T], Pos, P1,P2,P3,Pos, A,B,C,_, E,F,G,H ) :- !,
42   Pos1 is Pos - 1,
43   get_elements_pos( T, Pos1, P1,P2,P3,-1, A,B,C,X, E,F,G,H ).
44 get_elements_pos( [_|T], Pos, P1,P2,P3,P4, A,B,C,D, E,F,G,H ) :- !,
45   Pos1 is Pos - 1,
46   get_elements_pos( T, Pos1, P1,P2,P3,P4, A,B,C,D, E,F,G,H ).
47
48
49 /*********************************************************
50  * Calculates the positions of new contact points based in
51  * input shift.
52  *********************************************************/
53
54 shifted_face( Face, Shift, TotalElements, Distance, C1, C2, C3, C4 ) :- !,
55   S_F1 is ( Shift mod TotalElements ),
56   S_F2 is ( ( Shift + Distance ) mod TotalElements ),
57   S_F3 is ( ( Shift + ( 2 * Distance ) ) mod TotalElements ),
58   S_F4 is ( ( Shift + ( 3 * Distance ) ) mod TotalElements ),
59   TotalElementsIn is TotalElements - 1,
60   get_elements_pos( Face, TotalElementsIn, S_F1,S_F2,S_F3,S_F4, 0,0,0,0, C1,C2,C3,C4 ).
61
62
63 /*********************************************************
64  * Prints rotations
```

```
65  ************************************************************/
66
67  print_rots(R1,R2,R3,R4,R5,R6) :-
68    write('[ Rotations ]'), nl,
69    write('Top    : '), write( R1 ), nl,
70    write('Bottom: '), write( R2 ), nl,
71    write('Front  : '), write( R3 ), nl,
72    write('Back   : '), write( R4 ), nl,
73    write('Left   : '), write( R5 ), nl,
74    write('Right  : '), write( R6 ), nl.
75
76
77  /************************************************************
78   * Converts a list of numbered characters, defined
79   * with "", to its numeric value
80   ************************************************************/
81
82  string_to_list(String,ListOut):-!,
83    string_to_list(String,[],ListOut).
84  string_to_list([H|T],ListIn,ListOut):-!,
85    Val is H - 48, % o número zero tem o código ascii 48
86    string_to_list(T,[Val|ListIn],ListOut).
87  string_to_list(_,List,List):-!.
88
89
90  /************************************************************
91   * Prints a 2D projection of the cube
92   ************************************************************/
93
94  w_top_line:- write(' ---------').
95  w_top_line_f:- write('---------').
96  w_side_space:- write('          ').
97  w_top_bot_num(N):-write('|    '),write(N),write('    ').
98  w_bot_num_line(N):- write('|____'),write(N),write('____').
99  w_empty_line:- write('|          ').
100 w_middle_nums(TREE_L_DESC,N1,N2):- write('|'),write(N1),
      write('  ') , write(TREE_L_DESC), write('  '),write(N2).
101 w_cube_end:-write('|').
102
103 project_cube(  TT_C1,TT_C2,TT_C3,TT_C4,    BA_C1,BA_C2,BA_C3,
      BA_C4,    RR_C1,RR_C2,RR_C3,RR_C4,
104                 LL_C1,LL_C2,LL_C3,LL_C4,    FF_C1,FF_C2,FF_C3,
                      FF_C4,    BO_C1,BO_C2,BO_C3,BO_C4 ) :-
105
106   %                                       BACK
107   w_side_space,                           w_top_line,
                                        nl,
108   w_side_space,                           w_top_bot_num(BA_C1),
                        w_cube_end, nl,
```

```
109   w_side_space, w_empty_line,
                          w_cube_end, nl,
110   w_side_space, w_middle_nums('BAC',
          BA_C4, BA_C2), w_cube_end, nl,
111   w_side_space, w_empty_line,
                          w_cube_end, nl,
112   w_top_line, w_bot_num_line(BA_C3),
          w_cube_end, w_top_line_f,
          w_top_line, nl,
113
114   % LEFT                          TOP
                                      RIGHT
                                     BOTTOM
115   w_top_bot_num(LL_C1), w_top_bot_num(TT_C1),
                    w_top_bot_num(RR_C1),
          w_top_bot_num(BO_C1), w_cube_end, nl,
116   w_empty_line, w_empty_line,
                          w_empty_line,
                          w_empty_line,
                          w_cube_end, nl,
117   w_middle_nums('LEF', LL_C4, LL_C2), w_middle_nums('TOP',
          TT_C4, TT_C2), w_middle_nums('RIG', RR_C4, RR_C2),
          w_middle_nums('BOT', BO_C4, BO_C2), w_cube_end, nl,
118   w_empty_line, w_empty_line,
                          w_empty_line,
                          w_empty_line,
                          w_cube_end, nl,
119   w_bot_num_line(LL_C3), w_bot_num_line(TT_C3),
                    w_bot_num_line(RR_C3),
          w_bot_num_line(BO_C3), w_cube_end, nl,
120
121   %                             FRONT
122   w_side_space, w_top_bot_num(FF_C1),
                    w_cube_end, nl,
123   w_side_space, w_empty_line,
                          w_cube_end, nl,
124   w_side_space, w_middle_nums('FRO',
          FF_C4, FF_C2), w_cube_end, nl,
125   w_side_space, w_empty_line,
                          w_cube_end, nl,
126   w_side_space, w_bot_num_line(FF_C3),
                    w_cube_end, nl.
127
128
129 /*********************************************************
130  * Processes an input file
131  *********************************************************/
132
133  processStreamLine(Stream, ListIn, ListOut) :-
```

```
134   at_end_of_line(Stream), !,
135   skip_line(Stream),
136   ListOut = ListIn.
137 processStreamLine(Stream, ListIn, ListOut) :-
138   peek_code(Stream, Code),
139   Code = -1, !,
140   skip_line(Stream),
141   ListOut = ListIn.
142 processStreamLine(Stream, ListIn, ListOut) :-
143   at_end_of_stream(Stream), !,
144   ListOut = ListIn.
145 processStreamLine(Stream, ListIn, ListOut) :-
146   get_code(Stream, Code),
147   %write(Code), write(','),
148   Number is Code - 48,
149   Number >= 3,
150   Number =< 9, !,
151   processStreamLine(Stream, [Number|ListIn], ListOut).
152 processStreamLine(Stream,_,_):-!,
153   write('Os valores aceites para as faces têm de estar 3 e
        9, inclusive.'), nl,
154   close(Stream),
155   abort.
156
157 parse_file(Filename, Top, Bottom, Front, Back, Left, Right)
      :-
158   open(Filename, read, Stream),
159   processStreamLine(Stream, [], Top),
160   processStreamLine(Stream, [], Bottom),
161   processStreamLine(Stream, [], Front),
162   processStreamLine(Stream, [], Back),
163   processStreamLine(Stream, [], Left),
164   processStreamLine(Stream, [], Right),
165   close(Stream).
166
167
168 /*********************************************************
169  * Verifies if all list faces, have the same length
170  *********************************************************/
171
172 verifyLinesLength(Top, Bottom, Front, Back, Left, Right,
      ToElements):-
173   length(Top   , ToElements),
174   length(Bottom, L2),
175   length(Front , L3),
176   length(Back  , L4),
177   length(Left  , L5),
178   length(Right , L6),
179   ToElements = L2,
```

```
180   ToElements = L3,
181   ToElements = L4,
182   ToElements = L5,
183   ToElements = L6, !.
184 verifyLinesLength(_,_,_,_,_,_,_,_,_):-!,
185   write('Not all lines have the same with. Aborting.'), nl,
186   abort.
187
188
189 /*********************************************************
190  * Verifies if the length is a multiple of 4
191  *********************************************************/
192
193 verifyLineDistance(Size, Distance):-
194   Size mod 4 =:= 0, !,
195   Distance is floor(Size / 4).
196 verifyLineDistance(_, _):-!,
197   write('The lenght of the lines is not a multiple of 4.
          Aborting.'), nl,
198   abort.
199
200
201 /*********************************************************
202  * turn12 processing
203  *********************************************************/
204
205 turn12p:-
206   turn12('cubo_a.txt').
207
208 turn12(Filename):-
209   construct_full_file_path( Filename, FilePath ),
210   not(turn12_all_poss(FilePath)), write(' no'), nl, nl,
211   fd_statistics.
212
213 /*dump_o_face( Top,Bottom,Front,Back,Left,Right, R1,R2,R3,R4
      ,R5,R6 ) :- !,
214   length(Top, Size),
215   S1 is Size - R1,
216   S2 is Size - R2,
217   S3 is Size - R3,
218   S4 is Size - R4,
219   S5 is Size - R5,
220   S6 is Size - R6,
221   shuffle_cube_face( Top   , S1, ShuffTop    ),
222   shuffle_cube_face( Bottom, S2, ShuffBottom ),
223   shuffle_cube_face( Front , S3, ShuffFront  ),
224   shuffle_cube_face( Back  , S4, ShuffBack   ),
225   shuffle_cube_face( Left  , S5, ShuffLeft   ),
226   shuffle_cube_face( Right , S6, ShuffRight  ),
```

```
227   write_cube_file('C:/Users/Joao Henriques/Desktop/Eng.
          Informática/FEUP/PLOG/turn12/src/cubo_a_t.txt',
228     ShuffTop, ShuffBottom, ShuffFront, ShuffBack, ShuffLeft,
            ShuffRight ).*/
229
230 turn12( Top, Bottom, Front, Back, Left, Right,     R1, R2, R3
        , R4, R5, R6,
231     TT_C1,TT_C2,TT_C3,TT_C4,    BO_C1,BO_C2,BO_C3,BO_C4,
            FF_C1,FF_C2,FF_C3,FF_C4,
232     BA_C1,BA_C2,BA_C3,BA_C4,    LL_C1,LL_C2,LL_C3,LL_C4,
            RR_C1,RR_C2,RR_C3,RR_C4   ) :-
233
234   verifyLinesLength(Top, Bottom, Front, Back, Left, Right,
          TotElems),
235   verifyLineDistance( TotElems, Distance ),
236
237   Rotations = [ R1, R2, R3, R4, R5, R6 ],
238
239   domain(Rotations, 1, TotElems),
240
241
242   % os labelings foram separados de forma a evitar que o
          predicado shifted_face
243   % seja corrido sem ser necessário. Desta forma sempre que
          um shifted_face falha,
244   % volta ao labeling anterior, e gera a rotaçao da face, só
           voltando aos labelings
245   % anteriores assim que o mesmo esgota todas as
          possibilidades do domínio.
246
247   % Face Topo com a face de Trás
248   TT_C1 + BA_C3 #= 12,
249
250   labeling([], [R1,R4]),
251   shifted_face( Top   , R1, TotElems, Distance, TT_C1, TT_C2
          , TT_C3, TT_C4 ),
252   shifted_face( Back  , R4, TotElems, Distance, BA_C1, BA_C2
          , BA_C3, BA_C4 ),
253
254
255   % Face Topo, com Direira e Trás
256   TT_C2 + RR_C4 #= 12,
257   RR_C1 + BA_C2 #= 12,
258
259   labeling([], [R6]),
260   shifted_face( Right , R6, TotElems, Distance, RR_C1, RR_C2
          , RR_C3, RR_C4 ),
261
262
```

```
263   % Face Topo, com Esquerda e Trás
264   TT_C4 + LL_C2 #= 12,
265   LL_C1 + BA_C4 #= 12,
266
267   labeling([], [R5]),
268   shifted_face( Left  , R5, TotElems, Distance, LL_C1, LL_C2
          , LL_C3, LL_C4 ),
269
270
271   % Face Topo, com Frente, Direita, Esquerda e Trás
272   TT_C3 + FF_C1 #= 12,
273   RR_C3 + FF_C2 #= 12,
274   LL_C3 + FF_C4 #= 12,
275
276   labeling([], [R3]),
277   shifted_face( Front , R3, TotElems, Distance, FF_C1, FF_C2
          , FF_C3, FF_C4 ),
278
279
280   % Face de Baixo com o pontos de contacto das faces
          adjecentes
281   BO_C3 + FF_C3 #= 12,
282   BO_C2 + LL_C4 #= 12,
283   BO_C4 + RR_C2 #= 12,
284   BO_C1 + BA_C1 #= 12,
285
286   labeling([], [R2]),
287   shifted_face( Bottom, R2, TotElems, Distance, BO_C1, BO_C2
          , BO_C3, BO_C4 ).
288
289 turn12_all_poss(Filename) :-
290   write('Reading cube file...'),nl,
291   parse_file(Filename, Top, Bottom, Front, Back, Left, Right
          ), !,
292
293   write('Attempting to solve cube...'),nl,
294   turn12( Top,Bottom,Front,Back,Left,Right,     R1, R2, R3,
          R4, R5, R6,
295         TT_C1,TT_C2,TT_C3,TT_C4,    BO_C1,BO_C2,BO_C3,BO_C4,
                FF_C1,FF_C2,FF_C3,FF_C4,
296         BA_C1,BA_C2,BA_C3,BA_C4,    LL_C1,LL_C2,LL_C3,LL_C4,
                RR_C1,RR_C2,RR_C3,RR_C4    ),
297
298   nl, print_rots( R1,R2,R3,R4,R5,R6 ), nl,
299
300   project_cube( TT_C1,TT_C2,TT_C3,TT_C4,    BA_C1,BA_C2,BA_C3
          ,BA_C4,    RR_C1,RR_C2,RR_C3,RR_C4,
```

```
301                     LL_C1,LL_C2,LL_C3,LL_C4,    FF_C1,FF_C2,
                            FF_C3,FF_C4,    BO_C1,BO_C2,BO_C3,BO_C4
                              ),
302
303   %dump_o_face( Top,Bottom,Front,Back,Left,Right,    R1, R2,
          R3, R4, R5, R6 ),!,
304
305   nl,nl,
306
307   write('More possibilities ?'),
308   fail. % falha para verificar se existem mais
          possibilidades
309
310
311
312
313 /*********************************************************
314  *********************************************************
315  * Generating problems
316  *********************************************************
317  *********************************************************/
318
319
320  %
321 /*********************************************************
322  * Generate random number in the domain of the problem
323  * [3, 10-1]
324  *********************************************************/
325
326 random_turn12_n(Number):-
327   random(3, 10, Number).
328
329
330 /*********************************************************
331  * Guarantees there is no adjecent repeated
332  * numbers in each face
333  *********************************************************/
334
335 no_equal_number(N,N, Out) :-!,
336   Out is 3 + ( ( ( N + 1 ) - 3 ) mod 7 ).
337 no_equal_number(_,N,N):-!.
338
339
340 /*********************************************************
341  * Guarantees there is not another not another pattern
342  * equal to the original
343  *********************************************************/
344
345 no_equal_pattern( C1,C2,C3,C4, C1,C2,C3,C4, O4 ) :-!,
```

```
346   no_equal_number(C4, C4, O4).
347 no_equal_pattern( _,_,_,_,_,  _,_,_,_,T4,  T4 ).
348
349
350 /*********************************************************
351  * Fills the cube with random numbers in the domain
352  * of the problem
353  *********************************************************/
354
355 fill_cube_face_int( _,_,_,_,_,  DistBetweenElems,
      DistBetweenElems,  A,B,C,D,  FaceOut ) :- !,
356   append([], D, L1),
357   append(L1, C, L2),
358   append(L2, B, L3),
359   append(L3, A, FaceOut).
360 fill_cube_face_int( C1,C2,C3,C4,  Pos,DistBetweenElems,  [
      HA|A],[HB|B],[HC|C],[HD|D],  FaceOut ) :-
361   random_turn12_n( V1_temp ),
362   no_equal_number( HA, V1_temp, V1 ),
363
364   random_turn12_n( V2_temp ),
365   no_equal_number( HB, V2_temp, V2 ),
366
367   random_turn12_n( V3_temp ),
368   no_equal_number( HC, V3_temp, V3 ),
369
370   random_turn12_n( V4_temp ),
371   no_equal_number( HD, V4_temp, V4_tt2 ),
372
373   no_equal_pattern( C1,C2,C3,C4,  V1,V2,V3,V4_tt2, V4 ),
374
375   NextPos is Pos + 1,
376   fill_cube_face_int( C1,C2,C3,C4,  NextPos,
        DistBetweenElems,  [V1|[HA|A]],[V2|[HB|B]],[V3|[HC|C
        ]],[V4|[HD|D]],  FaceOut ).
377
378 fill_cube_face( C1,C2,C3,C4,  DistBetweenElems,  FaceOut )
      :-
379   fill_cube_face_int( C1,C2,C3,C4,  1,  DistBetweenElems,
          [C1],[C2],[C3],[C4],  FaceOut ).
380
381
382 /*********************************************************
383  * Writes the cube face to the file Stream
384  *********************************************************/
385
386 write_cube_line([], _):-!.
387 write_cube_line([H|T], Stream) :-
388   Char is H + 48, % ascii character 0 is 48
```

```
389   put_code(Stream, Char), !,
390   write_cube_line(T, Stream).
391 write_cube_line(_, Stream) :- !,
392   write('Error writing in file. Aborting.'), nl,
393   close(Stream),
394   abort.
395
396
397 /*********************************************************
398  * Writes the cube faces to file (one per line) in this
399  * order: Top, Bottom, Front, Back, Left, Right,
400  *********************************************************/
401
402 write_cube_file(Filename, Top, Bottom, Front, Back, Left,
       Right) :-
403   open(Filename, write, Stream), !,
404   reverse(Top   ,Top_r   ),
405   reverse(Bottom,Bottom_r),
406   reverse(Front ,Front_r ),
407   reverse(Back  ,Back_r  ),
408   reverse(Left  ,Left_r  ),
409   reverse(Right ,Right_r ),
410   write_cube_line(Top_r   , Stream), nl(Stream),
411   write_cube_line(Bottom_r, Stream), nl(Stream),
412   write_cube_line(Front_r , Stream), nl(Stream),
413   write_cube_line(Back_r  , Stream), nl(Stream),
414   write_cube_line(Left_r  , Stream), nl(Stream),
415   write_cube_line(Right_r , Stream),
416   close(Stream).
417
418
419 /*********************************************************
420  * Predicate that only accepts a cube with a unique
421  * solution. This is possible because the algorithm that
422  * solves the cube starts with a shift of one, if no other
423  * solution is found, the solution will have all six
424  * rotations with the length of the face.
425  *********************************************************/
426
427 turn12_unique_gen( Top, Bottom, Front, Back, Left, Right )
       :- !,
428   turn12( Top, Bottom, Front, Back, Left, Right,    R1, R2,
        R3, R4, R5, R6,
429     -,-,-,-,-,  -,-,-,-,-,  -,-,-,-,-,  -,-,-,-,-,  -,-,-,-,-,  -,-,-,
          - ),
430   !,
431   %print_rots(R1,R2,R3,R4,R5,R6),
432   length( Top, TotalElem ),
433   R1 = TotalElem,
```

```
434   R2 = TotalElem,
435   R3 = TotalElem,
436   R4 = TotalElem,
437   R5 = TotalElem,
438   R6 = TotalElem.
439
440
441 /*********************************************************
442  * Rotates the face list, many times specified.
443  *********************************************************/
444
445 shuffle_cube_face( Face, ShufflePos, FaceOut ) :-
446   shuffle_cube_face( Face, 0, ShufflePos, [], FaceOut ).
447 shuffle_cube_face( [H|T], Pos, ShufflePos, ListIn, FaceOut )
      :-
448   Pos < ShufflePos,
449   append( ListIn, [H], NewList ),
450   NewPos is Pos + 1,
451   shuffle_cube_face( T, NewPos, ShufflePos, NewList, FaceOut
        ).
452 shuffle_cube_face( Tail, _, _, ListIn, FaceOut ) :-
453   append( Tail, ListIn, FaceOut ).
454
455
456 /*********************************************************
457  * Rotates the face with a random number
458  *********************************************************/
459
460 randomly_suffle_face( Face, FaceSize, FaceOut ) :-
461   random(0, FaceSize, Rotation),
462   shuffle_cube_face( Face, Rotation, FaceOut ).
463
464
465 /*********************************************************
466  * Analytics predicates. Just check the sum of each line.
467  * Not needed to solve the problem
468  *********************************************************/
469
470 sum_face([], Sum, Sum).
471 sum_face([H|T], SumIn, SumOut) :-
472   NewSum is SumIn + H,
473   sum_face(T, NewSum, SumOut).
474
475 print_face_stats(F,Sum,FSize):-
476   Div is Sum / FSize,
477   write('    '), write(F), write(' sum is: '), write( Sum ),
        write(' ('), write(Div), write(')'), nl.
478
```

```
479 print_cube_stats( Top, Bottom, Front, Back, Left, Right,
        FaceSize ) :-
480   sum_face(Top, 0, SumTop),
481   sum_face(Back, 0, SumBack),
482   sum_face(Right, 0, SumRight),
483   sum_face(Left, 0, SumLeft),
484   sum_face(Front, 0, SumFront),
485   sum_face(Bottom, 0, SumBottom),
486   print_face_stats('Top', SumTop, FaceSize),
487   print_face_stats('Back', SumBack, FaceSize),
488   print_face_stats('Right', SumRight, FaceSize),
489   print_face_stats('Left', SumLeft, FaceSize),
490   print_face_stats('Front', SumFront, FaceSize),
491   print_face_stats('Bottom', SumBottom, FaceSize),
492   TotalSum is SumTop + SumBack + SumRight + SumLeft +
          SumFront + SumBottom,
493   print_face_stats('Total', TotalSum, FaceSize).
494
495
496 /*********************************************************
497  * Generates a new problem base on the arguments
498  *********************************************************/
499
500 turn12gen(Dist):-
501   turn12gen( 'cubo_a.txt', Dist ).
502
503 turn12gen(Filename, Distance):-
504   Distance > 0,
505
506   construct_full_file_path( Filename, FilePath ),
507
508   repeat,
509   write('Making an attempt to find if current generated
          random numbers can make a solution...'), nl,
510
511   random_turn12_n( TT_C1 ),
512   random_turn12_n( TT_C2 ),
513
514   random_turn12_n( BO_C1 ),
515   random_turn12_n( BO_C3 ),
516
517   random_turn12_n( BA_C2 ),
518   random_turn12_n( BA_C4 ),
519
520   random_turn12_n( FF_C2 ),
521   random_turn12_n( FF_C4 ),
522
523   random_turn12_n( LL_C2 ),
524   random_turn12_n( LL_C4 ),
```

```
525
526   random_turn12_n( RR_C2 ),
527   random_turn12_n( RR_C4 ),
528
529   ContactPoints = [  TT_C2,TT_C4,    FF_C1,FF_C3,    LL_C2,
          LL_C4,
530                      BO_C2,BO_C4,    BA_C1,BA_C3,    RR_C2
                             ,RR_C4   ],
531
532   domain(ContactPoints, 3, 9),
533
534   % têm de ser diferentes, se nao, ao girar a face 180 graus
          ,
535   % e se os cantos opostos forem os mesmos, originaria uma
          nova soluçao
536   % se os quatro forem iguais, originariam 4 novas soluçoes.
537   TT_C1 #\= TT_C3   #\/   TT_C2 #\= TT_C4,
538   BO_C1 #\= BO_C3   #\/   BO_C2 #\= BO_C4,
539
540   FF_C1 #\= FF_C3   #\/   FF_C2 #\= FF_C4,
541   BA_C1 #\= BA_C3   #\/   BA_C2 #\= BA_C4,
542   RR_C1 #\= RR_C3   #\/   RR_C2 #\= RR_C4,
543   LL_C1 #\= LL_C3   #\/   LL_C2 #\= LL_C4,
544
545   TT_C1 + BA_C3 #= 12,
546
547   TT_C2 + RR_C4 #= 12,
548   RR_C1 + BA_C2 #= 12,
549
550   TT_C4 + LL_C2 #= 12,
551   LL_C1 + BA_C4 #= 12,
552
553   TT_C3 + FF_C1 #= 12,
554   RR_C3 + FF_C2 #= 12,
555   LL_C3 + FF_C4 #= 12,
556
557   BO_C3 + FF_C3 #= 12,
558   BO_C2 + LL_C4 #= 12,
559   BO_C4 + RR_C2 #= 12,
560   BO_C1 + BA_C1 #= 12,
561
562   labeling([], ContactPoints),
563
564   repeat,
565
566   write('Attempting to fill cube with a unique solution...')
          , nl,
567   fill_cube_face( TT_C1,TT_C2,TT_C3,TT_C4, Distance, Top
          ),
```

```
568   fill_cube_face( BA_C1,BA_C2,BA_C3,BA_C4, Distance, Back
          ),
569   fill_cube_face( RR_C1,RR_C2,RR_C3,RR_C4, Distance, Right
          ),
570   fill_cube_face( LL_C1,LL_C2,LL_C3,LL_C4, Distance, Left
          ),
571   fill_cube_face( FF_C1,FF_C2,FF_C3,FF_C4, Distance, Front
          ),
572   fill_cube_face( BO_C1,BO_C2,BO_C3,BO_C4, Distance, Bottom
          ),
573
574   FaceSize is Distance * 4,
575
576   turn12_unique_gen( Top, Bottom, Front, Back, Left, Right )
          , !,
577   write('Cube has a unique solution.'),nl,
578
579
580   randomly_suffle_face( Top   , FaceSize, ShuffTop    ),
581   randomly_suffle_face( Back  , FaceSize, ShuffBack   ),
582   randomly_suffle_face( Right , FaceSize, ShuffRight  ),
583   randomly_suffle_face( Left  , FaceSize, ShuffLeft   ),
584   randomly_suffle_face( Front , FaceSize, ShuffFront  ),
585   randomly_suffle_face( Bottom, FaceSize, ShuffBottom ),
586
587   project_cube( TT_C1,TT_C2,TT_C3,TT_C4,   BA_C1,BA_C2,BA_C3
          ,BA_C4,   RR_C1,RR_C2,RR_C3,RR_C4,
588                   LL_C1,LL_C2,LL_C3,LL_C4,   FF_C1,FF_C2,
                        FF_C3,FF_C4,   BO_C1,BO_C2,BO_C3,BO_C4
                         ),
589
590   nl, write('Writing cube to file.'), nl,
591   write_cube_file(FilePath, ShuffTop, ShuffBottom,
          ShuffFront, ShuffBack, ShuffLeft, ShuffRight ),
592   nl,nl,
593   fd_statistics.
594
595 turn12gen(_,_):-
596   write('Distance between elements must be grater than 0.
          Aborting.').
```

Listing 3.1. Código ProLog: turn12.pl