

# TP6 - Sur une balançoire

PAUL CHAIGNON - ULYSSE GOARANT

28 mars 2014

Listing 1 – balançoire.ecl

```
1 :- lib(ic).
2 :- lib(ic_symbolic).
3 :- lib(branch_and_bound).
4
5 :- local domain(personnes(ron, zoe, jim, lou, luc, dan, ted, tom, max,
    kim)).
6
7 /**
8  * Question 6.1
9  */
10 /**
11  * famille(?Famille, ?Poids)
12  */
13 famille(Famille, Poids):-
14     Famille = [(ron, zoe, jim, lou, luc, dan, ted, tom, max, kim),
15     (foreachelem(Personne, Famille) do
16         Personne &:: personnes
17     ),
18     Poids = [(24, 39, 85, 60, 165, 6, 32, 123, 7, 14)].
19
20 /**
21  * places(?Places)
22  */
23 places(Places):-
24     famille(_, Poids),
25     dim(Poids, [Taille]),
26     dim(Places, [Taille]),
27     Places #:: [-8.. -1, 1..8].
28
29 /**
30  * nb_chaque_cote(Places, ?NbGauche)
31  */
32 nb_chaque_cote(Places, NbGauche):-
33     (foreachelem(Place, Places), fromto(0, InGauche, OutGauche, NbGauche)
34         do
35         OutGauche #= InGauche + (Place #< 0)
36     ).
37 /**
38  * moment_total(?Places, ?Poids, ?MomentTotal, ?SumMomentNorms)
39  */
40 moment_total(Places, Poids, MomentTotal, SumMomentNorms):-
41     (foreachelem(Place, Places), foreachelem(Poid, Poids),
```

```

42   fromto(0, In, Out, MomentTotal), fromto(0, InGauche, OutGauche,
43       SumMomentNorms) do
44       Out #= In + Place * Poid,
45       OutGauche #= InGauche + abs(Place) * Poid
46   ).
47 /**
48  * extremités(?Places, ?PlusAGauche, ?PlusADroite)
49  */
50 extremités(Places, PlusAGauche, Min, PlusADroite, Max):-
51     dim(Places, [Taille]),
52     (for(I, 1, Taille), param(Places),
53         fromto(0, InMin, OutMin, Min), fromto(0, InGauche, OutGauche,
54             PlusAGauche),
55         fromto(0, InMax, OutMax, Max), fromto(0, InDroite, OutDroite,
56             PlusADroite) do
57             Place is Places[I],
58             Sup #= (Place #> InMax),
59             OutDroite #= (Sup * I) + (neg(Sup) * InDroite),
60             OutMax #= (Sup * Place) + (neg(Sup) * InMax),
61             Inf #= (Place #< InMin),
62             OutGauche #= (Inf * I) + (neg(Inf) * InGauche),
63             OutMin #= (Inf * Place) + (neg(Inf) * InMin)
64         ).
65 /**
66  * différents(?Places)
67  * Verifie qu'il n'y a pas deux personnes a la meme place.
68  */
69 différents(Places):-
70     dim(Places, [Taille]),
71     (for(I, 1, Taille), param(Taille, Places) do
72         (for(J, I+1, Taille), param(Places, I) do
73             Places[I] #\= Places[J]
74         )
75     ).
76 /**
77  * pose_contraintes(?Places, ?Famille, ?Poids, ?SumMomentNorms)
78  * Verifie qu'il y a 5 personnes de chaque cote.
79  * Verifie que la balançoire est equilibree.
80  * Verifie que les parents encadrent les enfants et
81  * que les deux plus jeunes sont juste devant leurs parents.
82  */
83 pose_contraintes(Places, Famille, Poids, SumMomentNorms):-
84     différents(Places),
85     nb_chaque_cote(Places, 5),
86     moment_total(Places, Poids, 0, SumMomentNorms),
87     ic:min(Places, PosGauche),
88     ic:max(Places, PosDroite),
89     Places[8] #= PosGauche,
90     Places[4] #= PosDroite,
91
92
93
94
95
96

```

```

97 PosDan is Places[6],
98 PosMax is Places[9],
99 (PosDan#=PosGauche+1 and PosMax#=PosDroite-1) or
100 (PosMax#=PosGauche+1 and PosDan#=PosDroite-1).
101
102 /**
103  * resoudre(?Places)
104  */
105 resoudre(Places, SumMomentNorms):-
106     famille(Famille, Poids),
107     places(Places),
108     pose_contraintes(Places, Famille, Poids, SumMomentNorms),
109     labeling(Places).
110
111
112 /**
113  * Question 6.4
114  */
115 resoudre_opti(Places, SumMomentNorms):-
116     minimize(resoudre(Places, SumMomentNorms), SumMomentNorms).
117
118
119 /**
120  * Version optimisee 1.
121  * Commence par restreindre les variables les plus contraintes
122  * parmi celles de domaine minimal.
123  */
124 resoudre_v1(Places, SumMomentNorms):-
125     famille(Famille, Poids),
126     places(Places),
127     pose_contraintes(Places, Famille, Poids, SumMomentNorms),
128     search(Places, 0, most_constrained, indomain_split, complete, []).
129
130 resoudre_opti_v1(Places, SumMomentNorms):-
131     minimize(resoudre_v1(Places, SumMomentNorms), SumMomentNorms).
132
133
134 /**
135  * Version optimisee 2.
136  * Commence par les positions au centre de la balançoire.
137  */
138 resoudre_v2(Places, SumMomentNorms):-
139     famille(Famille, Poids),
140     places(Places),
141     pose_contraintes(Places, Famille, Poids, SumMomentNorms),
142     search(Places, 0, input_order, indomain_middle, complete, []).
143
144 resoudre_opti_v2(Places, SumMomentNorms):-
145     minimize(resoudre_v2(Places, SumMomentNorms), SumMomentNorms).
146
147
148 /**
149  * Version optimisee 3.
150  * Combine les versions 1 et 2.
151  */
152 resoudre_v3(Places, SumMomentNorms):-
153     famille(Famille, Poids),
154     places(Places),

```

```

155     pose_contraintes(Places, Famille, Poids, SumMomentNorms),
156     search(Places, 0, most_constrained, indomain_middle, complete, []).
157
158 resoudre_opti_v3(Places, SumMomentNorms):-
159     minimize(resoudre_v3(Places, SumMomentNorms), SumMomentNorms).
160
161
162 /**
163  * Version optimisee 4.
164  * L'ordre des variables est adapte au probleme
165  * pour placer en premier les personnes les plus lourdes
166  */
167 getVarList(Places, [Luc, Tom, Jim, Lou, Zoe, Ted, Ron, Kim, Max, Dan]):-
168     Ron is Places[1],
169     Zoe is Places[2],
170     Jim is Places[3],
171     Lou is Places[4],
172     Luc is Places[5],
173     Dan is Places[6],
174     Ted is Places[7],
175     Tom is Places[8],
176     Max is Places[9],
177     Kim is Places[10].
178
179 resoudre_v4(Places, SumMomentNorms):-
180     famille(Famille, Poids),
181     places(Places),
182     pose_contraintes(Places, Famille, Poids, SumMomentNorms),
183     getVarList(Places, VarList),
184     search(VarList, 0, occurrence, indomain_middle, complete, []).
185
186 resoudre_opti_v4(Places, SumMomentNorms):-
187     minimize(resoudre_v4(Places, SumMomentNorms), SumMomentNorms).
188
189
190 /**
191  * Tests
192  */
193 /*
194 places(Places).
195     Places = [][_315{-8 .. 8}, _333{-8 .. 8}, _351{-8 .. 8}, _369{-8 ..
196         8}, _387{-8 .. 8}, _405{-8 .. 8}, _423{-8 .. 8}, _441{-8 .. 8},
197         _459{-8 .. 8}, _477{-8 .. 8})
198     Yes (0.00s cpu)
199
200 places(Places), nb_chaque_cote(Places, NbG, NbD).
201     Places = [][_413{-8 .. 8}, _431{-8 .. 8}, _449{-8 .. 8}, _467{-8 ..
202         8}, _485{-8 .. 8}, _503{-8 .. 8}, _521{-8 .. 8}, _539{-8 .. 8},
203         _557{-8 .. 8}, _575{-8 .. 8})
204     NbG = NbG{0 .. 10}
205     NbD = NbD{0 .. 10}
206     There are 38 delayed goals. Do you want to see them? (y/n)
207     Yes (0.00s cpu)
208
209 places(Places), famille(_, Poids), moment_total(Places, Poids,
210     MomentTotal).
211     Places = [][_473{-8 .. 8}, _491{-8 .. 8}, _509{-8 .. 8}, _527{-8 ..
212         8}, _545{-8 .. 8}, _563{-8 .. 8}, _581{-8 .. 8}, _599{-8 .. 8},

```

```

207     _617{-8 .. 8}, _635{-8 .. 8})
208 Poids = [(24, 39, 85, 60, 165, 6, 32, 123, 7, 14)
209 MomentTotal = MomentTotal{-4440 .. 4440}
210 There are 10 delayed goals. Do you want to see them? (y/n)
211 Yes (0.00s cpu)
212 places(Places), extremities(Places, PlusAGauche, PlusADroite).
213 Places = [][_413{-8 .. 8}, _431{-8 .. 8}, _449{-8 .. 8}, _467{-8 ..
      8}, _485{-8 .. 8}, _503{-8 .. 8}, _521{-8 .. 8}, _539{-8 .. 8},
      _557{-8 .. 8}, _575{-8 .. 8})
214 PlusAGauche = PlusAGauche{0 .. 55}
215 PlusADroite = PlusADroite{0 .. 55}
216 There are 238 delayed goals. Do you want to see them? (y/n)
217 Yes (0.01s cpu)
218
219 resoudre_opti(Places, Moment).
220 Found a solution with cost 2914
221 Found a solution with cost 2858
222 Found a solution with cost 2808
223 Found a solution with cost 2722
224 Found a solution with cost 2716
225 Found a solution with cost 2708
226 Found a solution with cost 2694
227 Found a solution with cost 2602
228 Found a solution with cost 2594
229 Found a solution with cost 2524
230 Found a solution with cost 2474
231 Found a solution with cost 2430
232 Found a solution with cost 2392
233 Found a solution with cost 2344
234 Found a solution with cost 2296
235 Found a solution with cost 2218
236 Found a solution with cost 2196
237 Found a solution with cost 2154
238 Found a solution with cost 2142
239 Found a solution with cost 2064
240 Found a solution with cost 1958
241 Found a solution with cost 1890
242 Found a solution with cost 1748
243 Found a solution with cost 1744
244 Found a solution with cost 1704
245 Found a solution with cost 1604
246 Found no solution with cost -1.0Inf .. 1603
247 Places = [(3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
248 Moment = 1604
249 Yes (1.38s cpu)
250
251 resoudre_opti_v1(Places, Moment).
252 Found a solution with cost 1890
253 Found a solution with cost 1604
254 Found no solution with cost -1.0Inf .. 1603
255 Places = [(3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
256 Moment = 1604
257 Yes (0.17s cpu)
258
259 resoudre_opti_v2(Places, Moment).
260 Found a solution with cost 2554
261 Found a solution with cost 2352

```

```

262 Found a solution with cost 2276
263 Found a solution with cost 2106
264 Found a solution with cost 1944
265 Found a solution with cost 1866
266 Found a solution with cost 1750
267 Found a solution with cost 1704
268 Found a solution with cost 1604
269 Found no solution with cost -1.0Inf .. 1603
270 Places = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
271 Moment = 1604
272 Yes (1.00s cpu)
273
274 resoudre_opti_v3(Places, Moment).
275 Found a solution with cost 1890
276 Found a solution with cost 1604
277 Found no solution with cost -1.0Inf .. 1603
278 Places = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
279 Moment = 1604
280 Yes (0.15s cpu)
281
282 resoudre_opti_v4(Places, Moment).
283 Found a solution with cost 1696
284 Found a solution with cost 1604
285 Found no solution with cost -1.0Inf .. 1603
286 Places = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
287 Moment = 1604
288 Yes (0.10s cpu)
289 */

```

### Question 6.3

L'élimination de la symétrie réduit seulement le domaine des variables et ne change pas fondamentalement la recherche de solutions. Cette dernière prend toujours autant de temps.

### Question 3.1

Le labeling original d'ECLiPSe n'est pas optimisé pour ce problème. Il se contente de parcourir les variables dans l'ordre données par l'utilisateur et leurs valeurs dans l'ordre croissant. C'est à l'utilisateur d'adapter la recherche en fonction du problème.