TP6 - Sur une balançoire

Paul Chaignon - Ulysse Goarant

28 mars 2014

Listing 1 – balancoire.ecl

```
:- lib(ic).
:- lib(ic_symbolic).
:- lib(branch_and_bound).

:- local domain(personnes(ron, zoe, jim, lou, luc, dan, ted, tom, max,
   kim)).

/**
 * Question 6.1
 */
/**
 * famille(?Famille, ?Poids)
 */
famille(Famille, Poids):-
  Famille = [](ron, zoe, jim, lou, luc, dan, ted, tom, max, kim),
  (foreachelem(Personne, Famille) do
    Personne &:: personnes
  ),
  Poids = [](24, 39, 85, 60, 165, 6, 32, 123, 7, 14).

/**
 * places(?Places)
 */
places(Places):-
  famille(_, Poids),
  dim(Poids, [Taille]),
  dim(Places, [Taille]),
  Places #:: [-8.. -1, 1..8].

/**
 * nb_chaque_cote(Places, ?NbGauche)
 */
nb_chaque_cote(Places, NbGauche):-
  (foreachelem(Place, Places), fromto(0, InGauche, OutGauche, NbGauche)
      do
    OutGauche #= InGauche + (Place #< 0)
  ).

/**
 * moment_total(?Places, ?Poids, ?MomentTotal, ?SumMomentNorms)
 */
moment_total(Places, Poids, MomentTotal, SumMomentNorms):-
  (foreachelem(Place, Places), foreachelem(Poid, Poids),
  fromto(0, In, Out, MomentTotal), fromto(0, InGauche, OutGauche, SumMomentNorms) do
    Out #= In + Place * Poid,
    OutGauche #= InGauche + abs(Place) * Poid
  ).

/**
 * extremites(?Places, ?PlusAGauche, ?PlusADroite)
 */
extremites(Places, PlusAGauche, Min, PlusADroite, Max):-
  dim(Places, [Taille]),
  (for(I, 1, Taille), param(Places),
  fromto(0, InMin, OutMin, Min), fromto(0, InGauche, OutGauche, PlusAGauche),
  fromto(0, InMax, OutMax, Max), fromto(0, InDroite, OutDroite, PlusADroite) do
    Place is Places[I],

    Sup #= (Place #> InMax),
    OutDroite #= (Sup * I) + (neg(Sup) * InDroite),
    OutMax #= (Sup * Place) + (neg(Sup) * InMax),

    Inf #= (Place #< InMin),
    OutGauche #= (Inf * I) + (neg(Inf) * InGauche),
    OutMin #= (Inf * Place) + (neg(Inf) * InMin)
  ).

/**
 * differents(?Places)
 * Verifie qu'il n'y a pas deux personnes a la meme place.
 */
differents(Places):-
  dim(Places, [Taille]),
  (for(I, 1, Taille), param(Taille, Places) do
    (for(J, I+1, Taille), param(Places, I) do
      Places[I] #\= Places[J]
    )
  ).

/**
 * pose_contraintes(?Places, ?Famille, ?Poids, ?SumMomentNorms)
 * Verifie qu'il y a 5 personnes de chaque cote.
 * Verifie que la balancoire est equilibree.
 * Verifie que les parents encadrent les enfants et
 * que les deux plus jeunes sont juste devant leurs parents.
 */
pose_contraintes(Places, Famille, Poids, SumMomentNorms):-
  differents(Places),

  nb_chaque_cote(Places, 5),

  moment_total(Places, Poids, 0, SumMomentNorms),

  ic:min(Places, PosGauche),
  ic:max(Places, PosDroite),
  Places[8] #= PosGauche,
  Places[4] #= PosDroite,

  PosDan is Places[6],
  PosMax is Places[9],
  (PosDan#=PosGauche+1 and PosMax#=PosDroite-1) or
  (PosMax#=PosGauche+1 and PosDan#=PosDroite-1).

/**
 * resoudre(?Places)
 */
resoudre(Places, SumMomentNorms):-
  famille(Famille, Poids),
  places(Places),
  pose_contraintes(Places, Famille, Poids, SumMomentNorms),
  labeling(Places).


/**
 * Question 6.4
 */
resoudre_opti(Places, SumMomentNorms):-
  minimize(resoudre(Places, SumMomentNorms), SumMomentNorms).


/**
 * Version optimisee 1.
 * Commence par restreindre les variables les plus contraintes
 * parmi celles de domaine minimal.
 */
resoudre_v1(Places, SumMomentNorms):-
  famille(Famille, Poids),
  places(Places),
  pose_contraintes(Places, Famille, Poids, SumMomentNorms),
  search(Places, 0, most_constrained, indomain_split, complete, []).

resoudre_opti_v1(Places, SumMomentNorms):-
  minimize(resoudre_v1(Places, SumMomentNorms), SumMomentNorms).


/**
 * Version optimisee 2.
 * Commence par les positions au centre de la balancoire.
 */
resoudre_v2(Places, SumMomentNorms):-
  famille(Famille, Poids),
  places(Places),
  pose_contraintes(Places, Famille, Poids, SumMomentNorms),
  search(Places, 0, input_order, indomain_middle, complete, []).

resoudre_opti_v2(Places, SumMomentNorms):-
  minimize(resoudre_v2(Places, SumMomentNorms), SumMomentNorms).


/**
 * Version optimisee 3.
 * Combine les versions 1 et 2.
 */
resoudre_v3(Places, SumMomentNorms):-
  famille(Famille, Poids),
  places(Places),
  pose_contraintes(Places, Famille, Poids, SumMomentNorms),
  search(Places, 0, most_constrained, indomain_middle, complete, []).

resoudre_opti_v3(Places, SumMomentNorms):-
  minimize(resoudre_v3(Places, SumMomentNorms), SumMomentNorms).


/**
 * Version optimisee 4.
 * L'ordre des variables est adapte au probleme
 * pour placer en premier les personnes les plus lourdes
 */
getVarList(Places, [Luc, Tom, Jim, Lou, Zoe, Ted, Ron, Kim, Max, Dan]):-
  Ron is Places[1],
  Zoe is Places[2],
  Jim is Places[3],
  Lou is Places[4],
  Luc is Places[5],
  Dan is Places[6],
  Ted is Places[7],
  Tom is Places[8],
  Max is Places[9],
  Kim is Places[10].

resoudre_v4(Places, SumMomentNorms):-
  famille(Famille, Poids),
  places(Places),
  pose_contraintes(Places, Famille, Poids, SumMomentNorms),
  getVarList(Places, VarList),
  search(VarList, 0, occurrence, indomain_middle, complete, []).

resoudre_opti_v4(Places, SumMomentNorms):-
  minimize(resoudre_v4(Places, SumMomentNorms), SumMomentNorms).


/**
 * Tests
 */
/*
places(Places).
  Places = [](_315{-8 .. 8}, _333{-8 .. 8}, _351{-8 .. 8}, _369{-8 ..
     8}, _387{-8 .. 8}, _405{-8 .. 8}, _423{-8 .. 8}, _441{-8 .. 8},
     _459{-8 .. 8}, _477{-8 .. 8})
  Yes (0.00s cpu)

places(Places), nb_chaque_cote(Places, NbG, NbD).
  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})
  NbG = NbG{0 .. 10}
  NbD = NbD{0 .. 10}
  There are 38 delayed goals. Do you want to see them? (y/n)
  Yes (0.00s cpu)

places(Places), famille(_, Poids), moment_total(Places, Poids,
    MomentTotal).
  Places = [](_473{-8 .. 8}, _491{-8 .. 8}, _509{-8 .. 8}, _527{-8 ..
     8}, _545{-8 .. 8}, _563{-8 .. 8}, _581{-8 .. 8}, _599{-8 .. 8},
      _617{-8 .. 8}, _635{-8 .. 8})
  Poids = [](24, 39, 85, 60, 165, 6, 32, 123, 7, 14)
  MomentTotal = MomentTotal{-4440 .. 4440}
  There are 10 delayed goals. Do you want to see them? (y/n)
  Yes (0.00s cpu)

places(Places), extremites(Places, PlusAGauche, PlusADroite).
  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})
  PlusAGauche = PlusAGauche{0 .. 55}
  PlusADroite = PlusADroite{0 .. 55}
  There are 238 delayed goals. Do you want to see them? (y/n)
  Yes (0.01s cpu)

resoudre_opti(Places, Moment).
  Found a solution with cost 2914
  Found a solution with cost 2858
  Found a solution with cost 2808
  Found a solution with cost 2722
  Found a solution with cost 2716
  Found a solution with cost 2708
  Found a solution with cost 2694
  Found a solution with cost 2602
  Found a solution with cost 2594
  Found a solution with cost 2524
  Found a solution with cost 2474
  Found a solution with cost 2430
  Found a solution with cost 2392
  Found a solution with cost 2344
  Found a solution with cost 2296
  Found a solution with cost 2218
  Found a solution with cost 2196
  Found a solution with cost 2154
  Found a solution with cost 2142
  Found a solution with cost 2064
  Found a solution with cost 1958
  Found a solution with cost 1890
  Found a solution with cost 1748
  Found a solution with cost 1744
  Found a solution with cost 1704
  Found a solution with cost 1604
  Found no solution with cost -1.0Inf .. 1603
  Places = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
  Moment = 1604
  Yes (1.38s cpu)

resoudre_opti_v1(Places, Moment).
  Found a solution with cost 1890
  Found a solution with cost 1604
  Found no solution with cost -1.0Inf .. 1603
  Places = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
  Moment = 1604
  Yes (0.17s cpu)

resoudre_opti_v2(Places, Moment).
  Found a solution with cost 2554
  Found a solution with cost 2352
  Found a solution with cost 2276
  Found a solution with cost 2106
  Found a solution with cost 1944
  Found a solution with cost 1866
  Found a solution with cost 1750
  Found a solution with cost 1704
  Found a solution with cost 1604
  Found no solution with cost -1.0Inf .. 1603
  Places = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
  Moment = 1604
  Yes (1.00s cpu)

resoudre_opti_v3(Places, Moment).
  Found a solution with cost 1890
  Found a solution with cost 1604
  Found no solution with cost -1.0Inf .. 1603
  Places = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
  Moment = 1604
  Yes (0.15s cpu)

resoudre_opti_v4(Places, Moment).
  Found a solution with cost 1696
  Found a solution with cost 1604
  Found no solution with cost -1.0Inf .. 1603
  Places = [](3, -1, 2, 6, 1, -4, -3, -5, 5, -2)
  Moment = 1604
  Yes (0.10s cpu)
*/
```

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.