Partiel de programmation linéaire (PLAM1)

Mercredi 28 mai 2014

Exercice 1 : Décrire le principe de l'algorithme de *branch and bound*. Linéariser la fonction de coût $min \mid x - y \mid$

Exercice 2 : Résoudre le programme linéaire suivant, par la méthode graphique puis par la méthode algébrique :

$\max (5 x_1 + 10 x_2)$ tel que :

$x_1 + x_2 \leq 10$

$2 x_1 + x_2 \leq 15$

$x_1 + 3 x_2 \leq 24$

$x_1, x_2 \geq 0$

Exercice 3 : Le gérant d'un hôtel souhaite renouveler le linge de toilette de son établissement. Il a besoin de 90 draps de bain, 240 serviettes et 240 gants de toilette. Une première entreprise de vente lui propose un lot A comprenant 2 draps de bain, 4 serviettes et 8 gants pour 20 euros. Une deuxième entreprise vend pour 40 euros un lot B de 3 draps de bains, 12 serviettes et 6 gants de toilettes. Formaliser ce problème sous forme d'un programme linéaire en nombres entiers.

Exercice 4 : Dans une usine, on considère un coût dégressif en fonction de la quantité Q produite : si $Q \leq 200$, le coût est de 1500€/unité ; si $200 < Q \leq 500$, le coût est de 1200€/unité ; si $500 < Q$, le coût est de 1000€/unité. Modéliser cette fonction de coût particulière sous forme d'un programme linéaire.

Exercice 5 : Un fermier doit passer la rivière dans une barque juste assez grande pour lui et son chien, ou lui et sa chèvre, ou lui et ses choux. Les choux seront mangés s'il les laisse seuls avec la chèvre, et la chèvre sera mangée s'il la laisse seule avec le chien. Comment faire passer tout ce monde sans dégâts ?

Formaliser ce problème comme un programme linéaire en variables booléennes. Pour vous aider, voici un modèle des contraintes en programmation par contraintes sous OPL Studio :

```
berge_gauche[1][0] == 1;              // La situation de départ (1 = chien, 2 = chevre, 3 = choux)
berge_gauche[2][0] == 1;
berge_gauche[3][0] == 1;
berge_droite[1][Nb_mouvements] == 1;      // La situation finale
berge_droite[2][Nb_mouvements] == 1;
berge_droite[3][Nb_mouvements] == 1;
```

```
// Un élément (chien, chevre ou choux) est à droite ou à gauche
forall(j in 0..Nb_mouvements)
  forall(i in 1..3)
    berge_gauche[i][j] == 1 - berge_droite[i][j];
// Le chien et la chevre ne peuvent être ensemble sur une berge quand le berger n'est pas là.
// La chèvre et le choux ne peuvent être ensemble sur une berge quand le berger n'est pas là.
forall(i in 0..(Nb_mouvements div 2)) {
  berge_gauche[1][2 * i + 1] == 0 || berge_gauche[2][2 * i + 1] == 0;
  berge_gauche[2][2 * i + 1] == 0 || berge_gauche[3][2 * i + 1] == 0;
};
forall(i in 0..(Nb_mouvements div 2)) {
  berge_droite[1][2 * i] == 0 || berge_droite[2][2 * i] == 0;
  berge_droite[2][2 * i] == 0 || berge_droite[3][2 * i] == 0;
};
// La barque va de gauche à droite : un élément (chien, chevre, choux) se déplace ou reste.
forall(i in 0..(Nb_mouvements div 2))
  ( ( berge_droite[1][2 * i + 1] == 1 ) => ( berge_gauche[1][2 * i   ] == 1 ) ) ||
  ( ( berge_droite[2][2 * i + 1] == 1 ) => ( berge_gauche[2][2 * i   ] == 1 ) ) ||
  ( ( berge_droite[3][2 * i + 1] == 1 ) => ( berge_gauche[3][2 * i   ] == 1 ) );
// La barque va de droite à gauche : un élément (chien, chevre, choux) se déplace ou reste.
forall(i in 0..(Nb_mouvements div 2) - 1)
  ( ( berge_gauche[1][2 * i + 2] == 1 ) => ( berge_droite[1][2 * i + 1] == 1 ) ) ||
  ( ( berge_gauche[2][2 * i + 2] == 1 ) => ( berge_droite[2][2 * i + 1] == 1 ) ) ||
  ( ( berge_gauche[3][2 * i + 2] == 1 ) => ( berge_droite[3][2 * i + 1] == 1 ) );
// Un seul élément (chien, chevre, choux) se déplace à la fois.
forall(i in 0..Nb_mouvements-1)
  forall(j,k in 1..3 : j != k)
    ( berge_gauche[j][i] != berge_gauche[j][i+1] )
  => ( berge_gauche[k][i] == berge_gauche[k][i+1] );
```