Université Paris Diderot - Sorbonne Paris Cité - Master 1 Informatique - Programmation logique par contraintes

Examen du 16 janvier 2013 - Durée : 2 heures

Informations : Tous les documents sont autorisés. Le barème est donné à titre indicatif et peut être modifié.

Exercice 1 (**5 points**)
On considère le problème suivant :
Minimiser $-3X - 2Y$ par rapport à $X \geq 0, Y \geq 0$ et

$$\begin{aligned} X + Y &\leq 4 \\ 2X + Y &\leq 6 \end{aligned}$$

- Visualisez le problème en dessinant un plan (axes : X et Y) avec les contraintes.
- Appliquez l'algorithme simplex (Il est simple d'obtenir une forme simplex de base).
- On suppose qu'on n'est pas très sur du coefficient -3 dans la fonction à minimiser ($-3X-2Y$). On remplace donc -3 par $(-3+\delta_1)$. Appliquez l'algorithme simplex en faisant les *mêmes* opérations (choix des variables et équations à pivoter) qu'en appliquant le simplex sur le problème original. Dans quelles conditions le résultat obtenu permet de conclure ce que c'est le minimum ?
- De la même façon on pourrait avoir une incertitude sur la valeur 4. On considère le problème : Minimiser $-3X - 2Y$ par rapport à $X \geq 0, Y \geq 0$ et

$$\begin{aligned} X + Y &\leq 4 + \delta_2 \\ 2X + Y &\leq 6 \end{aligned}$$

  Appliquez l'algorithme simplex en faisant les *mêmes* opérations (choix des variables et équations à pivoter) qu'en appliquant le simplex sur le problème original. Dans quelles conditions le résultat obtenu permet de conclure ce que c'est le minimum ?

Correction : On transforme d'abord le problème en forme simplex : minimiser $-3X - 2Y$ par rapport à $X, Y, S, T \geq 0$ et $X + Y + S = 4$ et $2X + Y + T = 6$. On obtient min. $-3X - 2Y$ p.r.à $S = 4 - X - Y$ et $T = 6 - 2X - Y$ comme forme simplex de base. En appliquant l'algorithme simplex, il y a deux possibilités : d'abord pivoter X et ensuite Y ou l'inverse. Le résultat est : min. $-10 + T + S$ p.r.à $Y = 2 + T - 2S$ et $X = 2 - T + S$. Le minimum est donc -10.

Avec δ_1 cela donne : min. $-10 + 2\delta_1 + (1 - \delta_1)T + (1 + \delta_1)S$ p.r.à $Y = 2 + T - 2S$ et $X = 2 - T + S$. On peut en déduire le minimum, si $-1 \leq \delta_1 \leq 1$, car les coefficients de la fonction objective doivent être positifs pour avoir une forme résolue.

Avec δ_2 cela donne : min $-10 + T + S - \delta_2$ p.r.à $Y = 2 + 2\delta_2 + T - 2S$ et $X = 2 - \delta_2 - T + S$. On peut en déduire le minimum, si $-1 \leq \delta_2 \leq 2$, car les constantes dans les équations doivent être positives pour avoir une forme résolue.

Exercice 2 (**3 points**)
On considère la contrainte $Z > X \ \wedge \ Y < X \ \wedge \ Z > Y$ avec les domaines (intervalles) $D(X) = [1..3], D(Y) = [0..2], D(Z) = [2..4]$.

- Indiquez, indépendamment, si cette contrainte est nœud-consistante, arc-consistante, borne-consistante, chemin-consistante.
- Si elle n'est pas nœud-consistante (resp. arc-consistante, borne-consistante, chemin-consistante), rendez la nœud-consistante (resp. arc-consistante, borne-consistante, chemin-consistante) sans éliminer des solutions évidemment. Justifiez.

Correction : La contrainte est nœud-consistante (évident car chaque contrainte simple a plus qu'une variable libre), arc-consistante (pour chaque contrainte simple et chaque valeur d'une variable on trouve

une valeur de l'autre variable de sorte que la contrainte simple soit satisfaite), borne-consistante (pour chaque contrainte simple et chaque valeur extrême d'une variable on trouve une valeur de l'autre variable de sorte que la contrainte soit satisfaite) mais pas chemin-consistante. Pour la rendre chemin consistante il faut d'abord considérer les contraintes comme des ensembles de paires R_{XY} etc.
$R_{YX} = \{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$
$R_{XZ} = \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$
et $R_{YZ} = \{(0,2),(0,3),(0,4),(1,2),(1,3),(1,4),(2,3),(2,4)\}$.
Ensuite il faut enlever de R_{YZ} les valeurs $(1,2)$ et $(2,3)$ car pour $Y = 1$ et $Z = 2$ il n'y a pas de valeur pour X qui "marche". Pareil pour $Y = 2$ et $Z = 3$.

Exercice 3 (**3 points**)
Un voyageur débarque aux pays des couleurs. À l'aéroport, il veut changer ses Euros en Coleuros (la monnaie du pays des couleurs). Au guichet on lui donne l'information suivante : Il y a quatre pièces de Coleuros : les vertes, jaunes, bleues et rouges. Deux pièces rouges plus trois pièces bleues valent ensemble 3 Euros 48 cents. Quatre pièces vertes plus une jaune plus une bleue valent 6 Euros 20 cents. Une pièce jaune et deux pièces vertes valent 3 Euros 76. Trois pièce vertes et une pièce jaune et une pièce bleue et une pièce rouge valent 6 Euros 04. Avec ces informations, le voyageur écrit le programme en GPROLOG suivant :

```
colorland(V,J,B,R) :-
    2*R + 3*B #= 348,
    4*V + J + B #= 620,
    J + 2*V #= 376,
    3*V + J + B + R #= 604.
```

En posant la question `?- colorland(V,J,B,R)` GPROLOG répond :

```
B = _#24(0..116)
J = _#126(___..___)
R = _#43(0..174)
V = _#107(___..___)
```

- Donnez les 4 valeurs manquantes à la place des `___` de la sortie de GPROLOG. Justifiez **en détail** en donnant les calculs nécessaires (Plusieurs itérations sont nécessaires).
- Le voyageur souhaite connaître les valeurs exactes des pièces (pas seulement des intervalles). Comment doit-on modifier le programme pour cela ?

Correction : La sortie de GPROLOG est :

```
B = _#24(0..116)
J = _#126(132..248)
R = _#43(0..174)
V = _#107(64..122)
```

GPROLOG utilise la borne consistance. On fait pareil ici. La première équation donne uniquement une information pour `B` et `R`. Regardons les trois autres. La deuxième donne $155 - (max(J) + max(B))/4 \leq V \leq 155 - (min(J) + min(B))/4$. Donc (1) $126 - max(J)/4 \leq V \leq 155 - min(J)/4$ et aussi (2) $504 - 4max(V) \leq J \leq 620 - 4min(V)$. La troisième donne (3) $188 - max(J)/2 \leq V \leq 188 - min(J)/2$ et (4) $376 - 2max(V) \leq J \leq 376 - 2min(V)$. La quatrième donne (5) $(314 - max(J))/3 \leq V \leq (604 - min(J))/3$ et (6) $314 - 3max(V) \leq J \leq 604 - 3min(V)$.

On applique d'abord (1) : $D(V) = [0..155]$ et (4) : $D(J) = [66..376]$. Ensuite (1) : $D(V) = [32..138]$ et (4) : $D(J) = [100..312]$. Ensuite encore (1) : $D(V) = [48..130]$ et (4) : $D(J) = [116..280]$. Ensuite encore (1) : $D(V) = [56..126]$ et (4) : $D(J) = [124..264]$. Ensuite encore (1) : $D(V) = [60..124]$ et (4) :

$D(J) = [128..256]$. Ensuite encore (1) : $D(V) = [62..123]$ et (4) : $D(J) = [130..252]$. Et ensuite (1) : $D(V) = [63..122]$ et (4) : $D(J) = [132..250]$. Et ensuite (1) : $D(V) = [64..122]$ et enfin $D(J) = [132..248]$.

Un autre raisonnement (pas celui de GPROLOG !) : De la 2ème et de la 3ème équation, on déduit : $2V + B = 244$ ce qui donne $D(V) = [64..122]$ et avec la troisième équation on déduit $D(J) = [132..248]$.

Pour avoir la valeur exacte il faut ajouter au programme `fd_labeling([V,J,B,R])`.

Exercice 4 (**7 points**)

On considère le jeu Bokkusu. Le but du jeu est de marquer en noir certaines cases de la grille (carré) donnée. Chaque case a deux valeurs : une valeur A et une valeur B. Les nombres à droite dénotent les valeurs A des cases de chaque ligne correspondante et les nombres en bas dénotent les valeurs B des cases de chaque colonne correspondante (Ces valeurs vont toujours de 1 à la taille de la grille en augmentant de 1 de gauche à droite ou du haut vers le bas). Les valeurs à gauche indiquent la somme des valeurs B des cases noires pour chaque ligne et les valeurs en haut indiquent la somme des valeurs A des cases noires pour chaque colonne. Un exemple d'une grille et sa solution est donné dans la figure 1. Dans la solution de cette exemple on a par exemple $7 = 1 + 2 + 4$ (les cases avec valeur B de 1, 2 et 4 sont marquées noir) et $6 = 2 + 4$ (les cases avec valeurs A de 2 et 4 sont marquées).

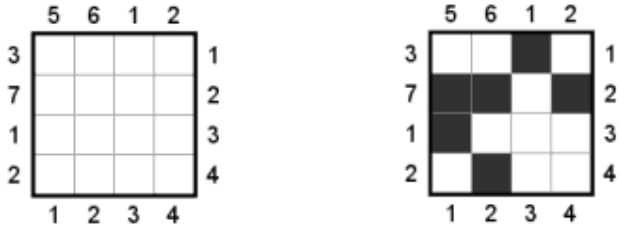

FIGURE 1 – Une grille Bokkusu et sa solution

- Écrivez un programme en GPROLOG qui résout la grille de l'exemple.
- Écrivez un programme en GPROLOG qui résout une grille quelconque (la taille et les valeurs à gauche et en haut sont des paramètres).

Correction :

Pour la grille de l'exemple :

```
bokkusu(L) :-
    L = [X11,X12,X13,X14,
         X21,X22,X23,X24,
         X31,X32,X33,X34,
         X41,X42,X43,X44],
    fd_domain(L,0,1),
    X11 + 2*X12 + 3*X13 + 4*X14 #= 3,
    X21 + 2*X22 + 3*X23 + 4*X24 #= 7,
    X31 + 2*X32 + 3*X33 + 4*X34 #= 1,
    X41 + 2*X42 + 3*X43 + 4*X44 #= 2,
    X11 + 2*X21 + 3*X31 + 4*X41 #= 5,
    X12 + 2*X22 + 3*X32 + 4*X42 #= 6,
    X13 + 2*X23 + 3*X33 + 4*X43 #= 1,
    X14 + 2*X24 + 3*X34 + 4*X44 #= 2.
```

En général :

```
% extrait les variables d'une ligne
```

```
extractRight(S,S,Rest,Rest,[]).
extractRight(S1,S,[X|L],Rest,[X|Vars]) :- S2 is S1+1, extractRight(S2,S,L,Rest,Vars).

% extrait les variables d'une colonne

extractUp(S,_,S,_,[]).
extractUp(S1,S2,S,L,[X|Vars]) :- N is S2 + S1*S, nth(N,L,X),
                                 S3 is S1 + 1, extractUp(S3,S2,S,L,Vars).

% génère un term de la form 1*X + 2*Y + 3*Z + 4*U....

generateTerm(S,S,[X],S*X).
generateTerm(S1,S,[X|Vars],S1*X + T) :- S2 is S1+1, generateTerm(S2,S,Vars,T).

generateUp(_,_,[],_).
generateUp(S1,S,[U|Up],L) :-
    extractUp(0,S1,S,L,Vars), S2 is S1 + 1,
    generateTerm(1,S,Vars,T), T #= U,
    generateUp(S2,S,Up,L).

generateRight(_,[],_).
generateRight(S,[R|Right],L) :-
    extractRight(0,S,L,LRest,Vars),
    generateTerm(1,S,Vars,T), T #= R,
    generateRight(S,Right,LRest).

% on appelle bokkusu avec S: la taille de la grille,
% Up: la liste des valeurs en haut et
% Right: la liste des valeurs a droite

bokkusu(S,Up,Right,L) :-
    NofVars #= S*S,
    length(L,NofVars),        % L la liste de variables
    fd_domain(L,0,1),         % booléennes
    generateUp(1,S,Up,L),     % contraintes en haut
    generateRight(S,Right,L). % contraintes à droite
```

Exercice 5 (**4 points**)
L'entreprise Tetraonics doit embaucher des travailleurs intérimaires pendant 5 jours pour faire face à un pic de demande passager. Chaque travailleur peut travailler soit 2 jours **consécutifs** soit 3 jours **consécutifs**. Au moins 10 travailleurs sont nécessaires les jours 1, 3 et 5, tandis qu'au moins 15 travailleurs sont nécessaires les jours 2 et 4. Un travailleur qui travaille 2 jours consécutifs est payé 125 Euros par jour tandis qu'un travailleur qui travaille 3 jours consécutifs est payé 100 Euros par jour. Un travailleur qui travaille 2 jours ne peut pas commencer le jour 5 et un travailleur qui travaille 3 jours ne peut pas commencer les jours 4 et 5. Tetraonics veut minimiser le coût.

- Modélisez le problème comme un CSP. Il vous est demandé de **modéliser** et non pas de résoudre le problème.
- A cause d'un nombre limité de personnels de supervision pas plus de 10 travailleurs peuvent commencer chaque jour. Modifiez votre CSP en conséquence.
- Les régulations de travail requièrent qu'au moins la moitié de l'argent soit dépensée pour les travailleurs qui travaillent 3 jours. Modifiez votre CSP en conséquence.

- Il y a quatre travailleurs qui veulent travailler les jours 1, 2 et 5. Modifiez votre CSP en conséquence.
- Donnez la requête qui permet à **YAP Prolog** de résoudre le problème.

Correction : On peut par exemple utiliser des variables x_1, x_2, x_3 et x_4 pour le nombre de travailleurs (qui travaillent 2 jours) embauchés les jours 1, 2, 3 et 4 ainsi que y_1, y_2, y_3 pour le nombre de travailleurs (qui travaillent 3 jours) embauchés les jours 1,2,3. Maintenant on peut facilement déterminer le nombre de travailleurs par jour. Une autre possibilité est d'utiliser une variable par jour indiquant le nombre de travailleurs en train de travailler. Dans ce cas, il faut faire attention que ce nombre doit être plus élevé au deuxième jour qu'au premier, etc.
Le CSP : Minimiser $250 * (x_1 + x_2 + x_3 + x_4) + 300 * (y_1 + y_2 + y_3)$ par rapport à
- $x_1 + y_1 \geq 10$
- $x_1 + x_2 + y_1 + y_2 \geq 15$
- $x_2 + x_3 + y_1 + y_2 + y_3 \geq 10$
- $x_3 + x_4 + y_2 + y_3 \geq 15$
- $x_4 + y_3 \geq 10$

Les modifications :
- $x_1 + y_1 \leq 10$ et $x_2 + y_2 \leq 10$ et $x_3 + y_3 \leq 10$ et $x_4 \leq 10$.
- $300 * (y_1 + y_2 + y_3) \geq 250 * (x_1 + x_2 + x_3 + x_4)$
- On obtient (on change uniquement le problème original) : Minimiser $250 * (x_1 + x_2 + x_3 + x_4) + 300 * (y_1 + y_2 + y_3 + 4)$ par rapport à
  - $x_1 + y_1 + 4 \geq 10$
  - $x_1 + x_2 + y_1 + y_2 + 4 \geq 15$
  - $x_2 + x_3 + y_1 + y_2 + y_3 \geq 10$
  - $x_3 + x_4 + y_2 + y_3 + 4 \geq 15$
  - $x_4 + y_3 \geq 10$

Il faut utiliser le prédicat `bb_{inf}` de YAP Prolog dans la librairie `clpr`.