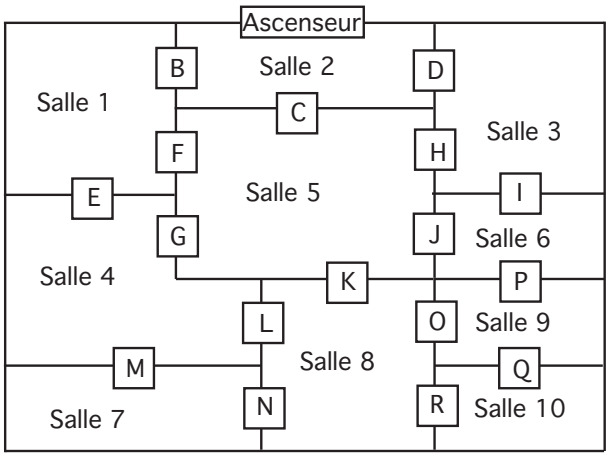
Feuille de travaux pratiques n° 2
GNU MathProg – Modélisation avancée

1 Modélisation implicite

L'écriture d'une matrice définissant chaque contrainte du problème comme nous l'avons fait dans la feuille de TP 1 sera souvent fastidieuse. Pour illustrer cela, Nous considérons le problème suivant :

– Afin de protéger les salles d'un musée, il a été décidé de placer des caméras de surveillance au dessus du linteau de certaines des portes donnant accès aux salles d'exposition. Chaque caméra a été installée dans une alvéole aménagée à cet effet de manière à balayer les 2 salles reliées par la porte grâce à un double objectif. Le but de la direction était d'assurer, au moindre coût, que chaque salle soit placée sous l'oeil d'au moins une caméra. Le plan de ce musée est présenté ci-dessous. Les portes sont indiquées par les lettres B à R. Combien de caméras a-t-il fallu utiliser pour parvenir avec le moins de caméras possible, à surveiller chaque salle avec au moins une caméra tout en plaçant la salle 4 sous l'oeil d'au moins deux caméras ?



– La modélisation de ce problème s'écrit :

$$
\begin{array}{rlll}
\max\ z & = & \displaystyle\sum_{j=B}^{R} x_j & \\
& s.c. & x_B + x_F + x_E & \geq\ 1 \\
& & x_B + x_C + x_D & \geq\ 1 \\
& & x_D + x_H + x_I & \geq\ 1 \\
& & x_E + x_G + x_L + x_M & \geq\ 2 \\
& & x_C + x_F + x_G + x_H + x_J & \geq\ 1 \\
& & x_I + x_J + x_P & \geq\ 1 \\
& & x_M + x_N & \geq\ 1 \\
& & x_K + x_L + x_N + x_O + x_R & \geq\ 1 \\
& & x_O + x_P + x_Q & \geq\ 1 \\
& & x_Q + x_R & \geq\ 1 \\
& & x_B, \ldots, x_R & \in\ \{0,1\}
\end{array}
$$

où x_i pour $i \in \{B, \ldots, R\}$ reprénte la décision de placer (ou pas) une caméra à l'emplacement i.

Pour saisir la modélisation, il n'est pas indispensable (bien que cela puisse éventuellement faciliter les choses) de changer les lettres en nombre. Dans un langage de modélisation et en particulier GNU MathProg, on peut considérer des ensembles dont les indices ne sont pas nécessairement contigüs, voir même des indices qui ne sont pas entiers. Le fichier `.mod` est donné par (pour simplifier, les données sont incluses, mais on peut les extraire facilement) :

```
# Declaration des donnees du probleme

param maxSalle;
set S := 1..maxSalle; # indices des salles

set indCam; # indices des cameras

param obj{indCam}; # Vecteur des coefficients de la fonction objectif
                     (indice sur les cameras)

param coeffcontr{S,indCam}; # Matrice des coefficients definissant les membres
                              de gauche des contraintes (double-indice)

param mdroite{S}; # Vecteur des membres de droites des contraintes
                    (indice sur les salle)

# Declaration d'un tableau de variables binaires

var x{indCam} binary;

# Fonction objectif

minimize cout : sum{j in indCam} obj[j]*x[j];

# Contraintes

s.t. Salle{i in S} : sum{j in indCam} coeffcontr[i,j] * x[j] >= mdroite[i];

# Resolution (qui est ajoutee en fin de fichier si on ne le precise pas)

solve;

# Affichage des resultats

display : x; # affichage "standard" des variables
display : "objectif : ", sum{j in indCam} obj[j]*x[j];

# donnees numeriques dont le debut est indique par le mot-cle "data;"

data;

param maxSalle := 10;

set indCam := B C D E F G H I J K L M N O P Q R;

param obj := B 1 C 1 D 1 E 1 F 1 G 1 H 1 I 1 J 1 K 1 L 1 M 1 N 1 O 1 P 1 Q 1 R 1;

param coeffcontr :  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R :=
                 1  1  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0
                 2  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
                 3  0  0  1  0  0  0  1  1  0  0  0  0  0  0  0  0  0
```

```
                  4  0  0  0  1  0  1  0  0  0  0  1  1  0  0  0  0  0
                  5  0  1  0  0  1  1  1  0  1  0  0  0  0  0  0  0  0
                  6  0  0  0  0  0  0  0  1  1  0  0  0  0  0  1  0  0
                  7  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0
                  8  0  0  0  0  0  0  0  0  0  1  1  0  1  1  0  0  1
                  9  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0
                  10 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1;

param mdroite := 1  1
                 2  1
                 3  1
                 4  2
                 5  1
                 6  1
                 7  1
                 8  1
                 9  1
                 10 1;

# Fin

end;
```

2 Utilisation d'une matrice creuse

Comme on peut le remarquer, la matrice des contraintes est composée pour l'essentiel de zéros. En pratique, c'est un phénomène courant qui est même fortement accentué en grande taille. Une saisie implicite de la modélisation se fera donc de préférence avec une matrice creuse. Certains langages de modélisation proposent une structure de données spécifique pour gérer cela. Ce n'est pas le cas de GNU MathProg, cependant les ensembles permettent de modéliser des matrices creuses. Plusieurs solutions sont possibles, en voici une :

```
# Declaration des donnees du probleme

param maxSalle;
set S := 1..maxSalle; # indices des salles

set indCam; # indices des cameras

set SCam dimen 2; # double-indice des salles et cameras

param obj{indCam}; # Vecteur des coefficients de la fonction objectif
                     (indice sur les cameras)

param coeffcontr{(i,j) in SCam}; # Matrice des coefficients definissant les membres
                                   de gauche des contraintes (double-indice)

param mdroite{S}; # Vecteur des membres de droites des contraintes
                    (indice sur les salle)

# Declaration d'un tableau de variables binaires

var x{indCam} binary;

# Fonction objectif

minimize cout : sum{j in indCam} obj[j]*x[j];
```

```
# Contraintes

s.t. Salle{i in S} : sum{(i,j) in SCam} coeffcontr[i,j] * x[j] >= mdroite[i];

# Resolution (qui est ajoutee en fin de fichier si on ne le precise pas)

solve;

# Affichage des resultats

display : x; # affichage "standard" des variables
display : "objectif : ", sum{j in indCam} obj[j]*x[j];

# Donnees numeriques dont le debut est indique par le mot-cle "data;"

data;

param maxSalle := 10;

set indCam := B C D E F G H I J K L M N O P Q R;

set SCam := (1,B) (1,E) (1,F) (2,B) (2,C) (2,D) (3,D) (3,H) (3,I) (4,E) (4,G) (4,L)
            (4,M) (5,C) (5,F) (5,G) (5,H) (5,J) (6,I) (6,J) (6,P) (7,M) (7,N) (8,K)
            (8,L) (8,N) (8,O) (8,R) (9,O) (9,P) (9,Q) (10,Q) (10,R);

param obj := B 1 C 1 D 1 E 1 F 1 G 1 H 1 I 1 J 1 K 1 L 1 M 1 N 1 O 1 P 1 Q 1 R 1;

param coeffcontr := [1,B] 1 [1,E] 1 [1,F] 1 [2,B] 1 [2,C] 1 [2,D] 1 [3,D] 1
                    [3,H] 1 [3,I] 1 [4,E] 1 [4,G] 1 [4,L] 1 [4,M] 1 [5,C] 1
                    [5,F] 1 [5,G] 1 [5,H] 1 [5,J] 1 [6,I] 1 [6,J] 1 [6,P] 1
                    [7,M] 1 [7,N] 1 [8,K] 1 [8,L] 1 [8,N] 1 [8,O] 1 [8,R] 1
                    [9,O] 1 [9,P] 1 [9,Q] 1 [10,Q] 1 [10,R] 1;

param mdroite := 1  1
                 2  1
                 3  1
                 4  2
                 5  1
                 6  1
                 7  1
                 8  1
                 9  1
                 10 1;

# Fin

end;
```

La nouveauté est ici la déclaration d'un ensemble de dimension 2 `SCam`. Nous utilisons cet ensemble comme un sous-ensemble du produit cartésien de `S` et `indCam`, ce que la déclaration n'indique pas. Cet ensemble est parcouru comme les autres dans la définition des contraintes, un des deux indices étant fixé. La saisie de cet ensemble est maintenant réalisée de manière habituelle (il était juste nécessaire d'en préciser la dimension). Ensuite, la saisie de `coeffcontr` est sans surprise.

On peut encore trouver cela trop long à saisir, et remarquer que dans ce cas particulier, la fonction objectif n'est constituée que de 1, de même que la partie non-creuse de la matrice, et les membres de droite des contraintes (à l'exception de la 4ème contrainte). Dans de nombreux langages de modélisation, on utiliserait une boucle pour

cette saisie. Une boucle `for` existe bien dans GNU MathProg, mais son usage est restreint et ne permet pas la saisie des paramètres. Dans le cas que nous considérons, les valeurs par défaut permettent de saisir les données avec moins d'effort. Le bloc de données devient donc :

```
data;

param maxSalle := 10;

set indCam := B C D E F G H I J K L M N O P Q R;

set SCam := (1,B) (1,E) (1,F) (2,B) (2,C) (2,D) (3,D) (3,H) (3,I) (4,E) (4,G) (4,L)
            (4,M) (5,C) (5,F) (5,G) (5,H) (5,J) (6,I) (6,J) (6,P) (7,M) (7,N) (8,K)
            (8,L) (8,N) (8,O) (8,R) (9,O) (9,P) (9,Q) (10,Q) (10,R);

param obj default 1;

param coeffcontr default 1;

param mdroite default 1:= 4 2; # tout a 1 sauf le membre d'indice 4
                               dont la valeur est 2

# Fin
```

Remarque : La composition de la matrice de contraintes n'est pas un cas si particulier ici, il n'est pas rare de n'y trouver que des 1 dans la partie non-creuse, en particulier pour des problèmes d'optimisation combinatoire.

3 Travail à effectuer

1. Pour éviter des recopiages ennuyeux, les fichiers `camera.mod`, `camera2.mod` et `camera3.mod` sont disponibles sur madoc. Il ne faut bien entendu pas se contenter de leur exécution, il faut comprendre l'intérêt de chacune des variantes proposées, et éventuellement essayer d'autres variations. On pourra aussi placer les données dans un fichier séparé.
2. On pourra remarquer (comme lors du premier TP) que l'utilisation d'une unique matrice (dite de contraintes) pour représenter l'ensemble des membres de gauche des contraintes n'est pas toujours le choix le plus judicieux. Faire les bons choix afin de faciliter l'écriture et la lecture de la modélisation est important pour éviter des erreurs, il est préférable de laisser apparaître les regroupements naturels de contraintes en utilisant le langage de modélisation.
3. On considère le problème suivant.

   La compagnie COSMAR a comme activité principale le chargement et le déchargement de porte-containers dans le port du Havre. À cause de récents problèmes, COSMAR désire renforcer son dispositif de lutte contre l'incendie en installant des systèmes SLIC dans l'entrepôt où sont placés les containers avant ou après toute manutention. La compagnie a identifié 7 zones à protéger plus spécialement. Ces zones sont marquées par une croix dans le diagramme ci-dessous.

		Côté N						
		1	2	3	4	5	6	7
Côté O	8				X			
	9			X				
	10					X		
	11	X	X				X	
	12							
	13				X			

   COSMAR souhaite placer les systèmes SLIC sur les côtés nord et ouest de l'entrepôt. Un système ne peut protéger que les zones situées dans la même rangée (ligne ou colonne) : par exemple, un SLIC installé en 4 protégerait les zones (8 ;4) et (13 ;4).

Les ingénieurs du service de l'incendie du Havre ont par ailleurs demandé que les systèmes SLIC ne soient pas placés dans des lignes consécutives, qu'au moins un SLIC soit placé en ligne 1 ou en colonne 8 et qu'au moins 50% des SLIC soient du côté nord.

Le but du problème est de minimiser le nombre de SLIC que COSMAR doit acheter et installer. Pour cela, on va poser le problème sous la forme d'un programme linéaire, et ensuite le résoudre en utilisant GNU MathProg.