Programmation par Contraintes. Correctif des quelques exercices.

Ruslan Sadykov

9 février 2022

Les règles de Golomb

*Une **règle de Golomb** est un ensemble d'entiers naturels dans lequel les distances entre les éléments sont deux à deux distinctes. Pour un entier n donné, une règle de Golomb est « optimale »si sa longueur (distance maximale) est minimum parmi toutes les règles de Golomb possibles.*

Exemples de règles de Golomb optimales :

$n = 2$	*0 1*
$n = 3$	*0 1 3*
$n = 4$	*0 1 4 6*
$n = 5$	*0 2 7 8 11*

Modéliser le problème de recherche d'une règle de Golomb optimale pour un n donné comme un CSP. Essayer d'utiliser les contraintes globales. Indiquez et éliminer les symétries de votre modèle si possible.

On va utiliser les variables entières x_i, $i = 1, \ldots, n$, x_i est égale à i-ème entier dans la règle cherchée. Le domaine D_{x_i} de x_i est $\{0, 1, \ldots, M\}$, où M est la borne supérieure pour la distance maximale. On peut prendre $M = 2^n$, comme

$$1, 2, 4, 8, 16, \ldots, 2^n$$

est une règle de Golomb.

D'abord, les éléments doivent être dans l'ordre croissant, d'où les contraintes

$$x_1 + 1 \leq x_2; \quad x_2 + 1 \leq x_3; \quad \ldots \quad x_{n-1} + 1 \leq x_n.$$

On doit minimiser la distance maximale, donc la fonction objective est

$$\min x_n - x_1$$

La contrainte principale est que les distances entre les éléments sont deux à deux distinctes. Pour la modéliser, nous introduisons les variables d_{ij}, $1 \leq i < j \leq n$, avec les mêmes domaines $D_{d_{ij}} = \{0, 1, \ldots, M\}$ telles que

$$d_{ij} = x_j - x_i, \quad 1 \leq i < j \leq n.$$

La contrainte est alors

$$\texttt{all} - \texttt{different}\Big(\{d_{ij}\}_{1\leq i<j\leq n}\Big)$$

Les symétries. D'abord, si on ajoute une constante à tous les éléments d'une règle de Golomb optimale, cette règle reste une règle de Golomb optimale. Pour éliminer cette symétrie, on peut fixer $x_1 = 0$.

Deuxièmement, si on « retourne »une règle de Golomb optimale :

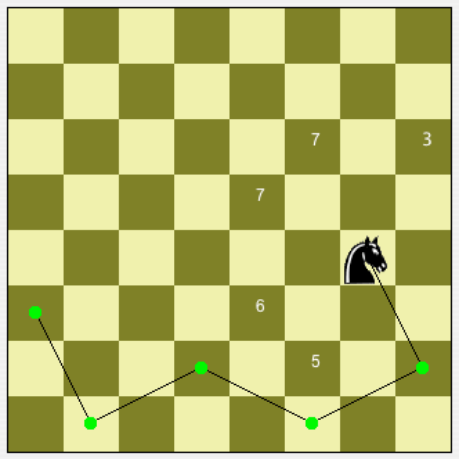
$$x_i = x_n - x_i, \quad i = 1, \ldots, n,$$

elle reste valide est optimale. Pour éliminer cette symétrie, on peut ajouter la contrainte

$$x_2 - x_1 \leq x_3 - x_2.$$

Le cavalier d'Euler

Modéliser le problème suivant :



Comment faire visiter par un cavalier chaque case d'un échiquier pour le faire revenir à son point de départ sans jamais repasser sur la même case ?

On va utiliser les variables entières x_i et y_i, $i = 0, \ldots, 64$, pour désigner les coordonnées (x_i, y_i) de la case où se trouve le cavalier après le i-ème pas. Le domaine de ces variables est $\{0, 1, \ldots, 7\}$. On va également utiliser les variables entières n_i, $i = 0, \ldots, 64$, pour désigner le numéro de la case où se trouve le cavalier après le i-ème pas :

$$n_i = 8 \cdot x_i + y_i.$$

Le domaine des variables n_i est $\{0, 1, \ldots, 63\}$.

Comme le cavalier ne peux pas repasser sur une même case (sauf pour la case de départ), toutes les valeurs des variables n sauf n_0 doivent être différentes :

$$\texttt{all} - \texttt{different}(n_1, n_2, \ldots, n_{64}).$$

Cavalier doit revenir à son point de départ :

$$n_0 = n_{64}.$$

Pour définir les règles de movement du cavalier, on peut utiliser les contraintes explicites qui donnent les quadruples possibles pour les variables $x_i, y_i, x_{i+1}, y_{i+1}$:

$$(x_i, y_i, x_{i+1}, y_{i+1}) \in \left\{ \begin{array}{c} (i, j, i \pm 1, j \pm 2)_{0 \leq i,j \leq 7}, \\ (i, j, i \pm 2, j \pm 2)_{0 \leq i,j \leq 7} \end{array} \right\} \quad \forall i = 0, \ldots, 63.$$

Alternativement, les contraintes arithmétiques peuvent être utilisées :

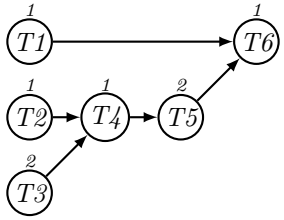
$$\left\{ \begin{array}{ll} \mid x_i - x_{i+1} \mid + \mid y_i - y_{i+1} \mid = 3, & \\ x_i \neq x_{i+1}, & \forall i = 0, \ldots, 63. \\ y_i \neq y_{i+1}. & \end{array} \right.$$

Comme le cavalier peut commencer son parcours à partir de n'importe quelle case, pour réduire la symétrie on peut fixer le début de parcours :

$$n_0 = 0.$$

Planification des tâches

On doit planifier six tâches dans un délai de 6 heures :



Ici, la tâche en début de flèche doit s'effectuer avant la tâche en fin de flèche. Les durées des tâches sont affichées au-dessus. Chaque tâches ne peut commencer qu'en début d'une heure. La tâche T1 ne peut pas être planifier au même heure que la tâche T4.

1. *Modélisez ce problème comme un CSP.*
2. *Rendez ce CSP arc-consistant et arc-B-consistant.*
   Est-ce qu'il y a une différence ?

Modélisation. Soit p_i est la durée de la tâche i. On va utiliser les variables s_i pour désigner le temps de début de la tâche i. Le domaine D_{s_i} de s_i est $\{0, 1, \ldots, 6 - p_i\}$, comme on doit finir toutes les tâches en 6 heures.

Les contraintes principales sont les contraintes de précédence :

$$s_1 + 1 \leq s_6 \tag{1}$$
$$s_2 + 1 \leq s_4 \tag{2}$$
$$s_3 + 2 \leq s_4 \tag{3}$$
$$s_4 + 1 \leq s_5 \tag{4}$$
$$s_5 + 2 \leq s_6 \tag{5}$$

En plus, la tâche 1 ne peut pas être exécuter au même heure que la tâche 4 :

$$s_1 \neq s_4.$$

Propagation. Dans la table suivant, on donne les valeurs qui ne sont pas ni arc-consistantes ni arc-B-consistantes avec les contraintes examinées :

Contrainte examinée	Valeurs supprimée
(1)	$D_{s_1} = \{0, 1, 2, 3, 4, \not{5}\}$, $D_{s_6} = \{\not{0}, 1, 2, 3, 4, 5\}$
(2)	$D_{s_2} = \{0, 1, 2, 3, 4, \not{5}\}$, $D_{s_4} = \{\not{0}, 1, 2, 3, 4, 5\}$
(3)	$D_{s_3} = \{0, 1, 2, 3, \not{4}\}$, $D_{s_4} = \{\not{1}, 2, 3, 4, 5\}$
(4)	$D_{s_4} = \{2, 3, \not{4}, \not{5}\}$, $D_{s_5} = \{\not{0}, \not{1}, \not{2}, 3, 4\}$
(5)	$D_{s_5} = \{3, \not{4}\}$, $D_{s_6} = \{\not{1}, \not{2}, \not{3}, \not{4}, 5\}$
(2)	$D_{s_2} = \{0, 1, 2, \not{3}, \not{4}\}$
(3)	$D_{s_3} = \{0, 1, \not{2}\}$
(4)	$D_{s_4} = \{2, \not{3}\}$
(2)	$D_{s_2} = \{0, 1, \not{2}\}$
(3)	$D_{s_3} = \{0, \not{1}\}$

Les domaines sont $D_{s_1} = \{0, 1, 2, 3, 4\}$, $D_{s_2} = \{0, 1\}$, $D_{s_3} = \{0\}$, $D_{s_4} = \{2\}$, $D_{s_5} = \{3\}$, $D_{s_6} = \{5\}$. Les valeurs au bord de ces domaines sont arc-consistantes, et donc CSP est arc-B-consistant. Par contre, valeur 2 dans domaine D_{s_1} n'est pas arc-consistante avec la contrainte $s_1 \neq s_4$.

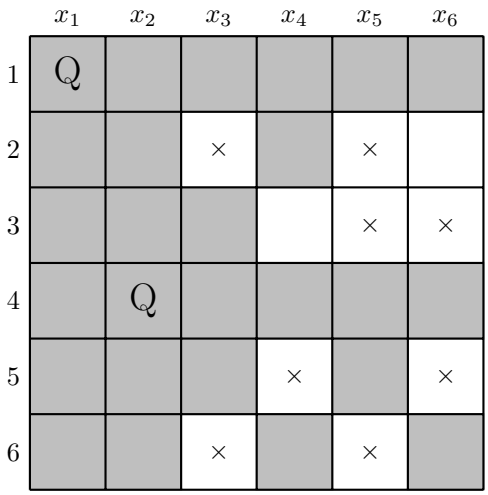
Donc, si rend ce CSP arc-consistant, on supprime une valeur de plus.

Problème de reines

1. *Résolvez le problème de 6 reines avec l'algorithme de Forward Checking.*
2. *Si on maintient l'arc-consistance après chaque suppression d'une valeur, est-ce qu'on peut résoudre le problème plus vite ?*

1. La résolution est très long à détailler ici. Vous pouvez aller voir la résolution du problème de 4 reines vue au cours. C'est exactement pareil.

2. Si on maintient l'arc-consistance après chaque suppression d'une valeur, on détectera les inconsistances (domaines vides) plus tôt, et donc on aura moins de noeuds dans l'abre de recherche. C'est aussi long à détailler ça ici. On va l'illustrer sur un exemple. Imaginons que deux reines sont déjà placées sur l'échiquier :



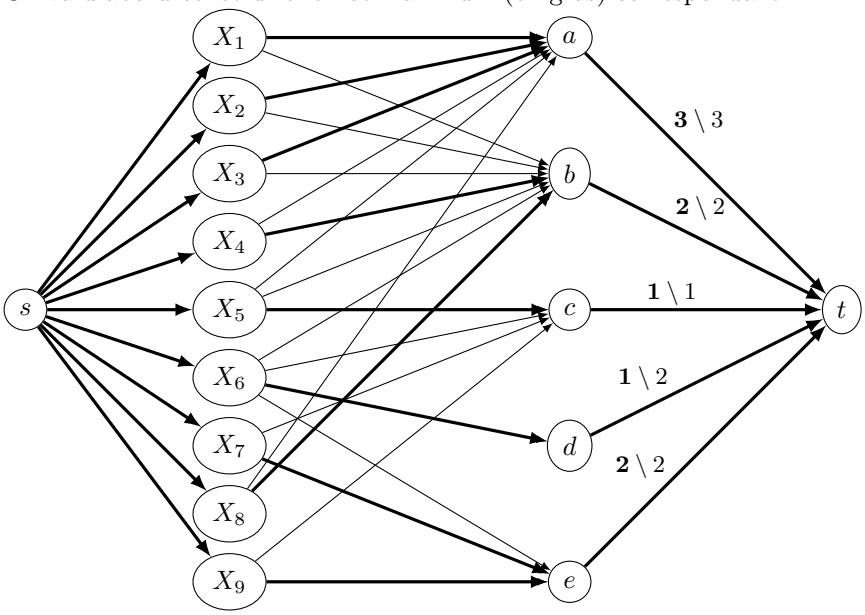
Les cases foncés représentent les valeurs supprimées des domaines des variables selon l'algorithme *Forward Checking*. On peut remarquer que les domaines des variables non-fixées (x_3 à x_6) ne sont pas vides et donc, on ne peut pas savoir si cette solution partielle peut être étendue à une solution complète. Donc, on est obligé à descendre plus bas dans l'arbre de recherche.

Si on maintient l'arc-consistance, on peut supprimer plus des valeurs dans les domaines des variables non-fixées. D'abord, on peut noter que les valeus $(x_6, 3)$ et $(x_6, 5)$ n'ont pas de support dans le domaine de x_4. Après cette suppression, les valeurs $(x_3, 2)$, $(x_5, 2)$, et $(x_5, 3)$ deviennent arc-inconsistantes, ce que entraine la suppression des valeurs $(x_3, 6)$, $(x_4, 5)$ et $(x_5, 6)$. Les domaines des variables x_3 et x_5 sont maintenant vides, et donc le CSP est arc-inconsistante après ce placement de deux premières reines. Par conséquent, on peut remonter dans l'arbre de recherche, comme cette branche ne contient pas d'une solution complète. Les valeurs arc-inconsistantes sont marquer sur le dessin par un « × ».

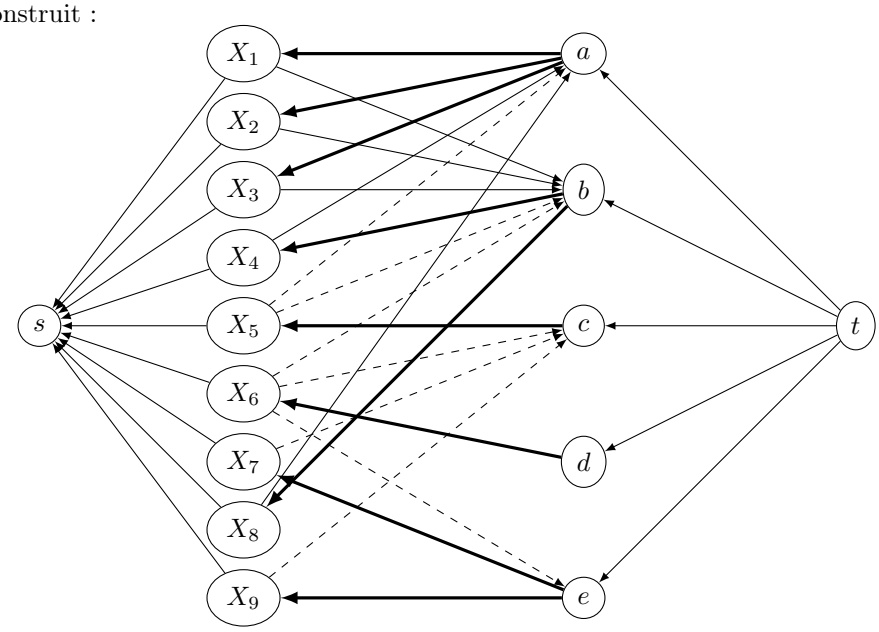
Au même temps, si on maintient l'arc-consistance, on passera plus de temps dans chaque noeud pour maintenir l'arc consistance. Quelle variante (maintenir ou pas l'arc-consistance) permettra résoudre le problème plus vite n'est pas claire. Il faut faire une expérience, i.e. implémenter les deux algorithmes (*Forward Checking* et *Maintaining Arc Consistency*) et les comparer expérimentalement.

Contrainte globale gcc

On va d'abord construire le flot maximum (en gras) correspondant :



Pour propager la contrainte gcc, on utilise le graphe résiduel associé au flot construit :



Les arcs pointillées (entre variables et valeurs) ne font partie du flot et n'appartient pas à aucun circuit alterné. Ces arcs ne peuvent pas appartenir à aucun flot maximum dans ce graphe. Les valeurs correspondes à ces arcs ne sont pas

consistantes avec la contrainte `gcc`, et doivent être supprimées des domaines des variables. Donc, les valeurs inconsistantes sont (X_5, a), (X_5, b), (X_6, b), (X_6, c), (X_6, e), (X_7, c), (X_9, c).

Contrainte globale `element`

On considère le CSP
— *Variables :* X, Y, Z.
— *Domaines :* $D_X = \{5, 6, 7, 8\}$, $D_Y = \{1, 3, 5\}$, $D_Z = \{2, 4\}$.
— *Contrainte :* $X = v_Y + w_Z$, *où* $v = [1, 5, 3, 4, 1]$ *et* $w = [4, 1, 4, 2, 5]$
Rendez ce CSP localement consistant.

Pour simplifier la présentation de la résolution, on va introduire deux variables : $V = v_Y$ et $W = w_Z$. Après la propagation de ces deux contraintes `element`, $D_V = \{1, 3\}$ et $D_W = \{1, 2\}$. On examine maintenant la contrainte $X = Y + W$. La somme de V et W peut prendre les valeurs 2, 3, 4, et 7, donc les valeurs 6, 7 et 8 peuvent être retirées de D_X. Les domaines D_W et D_V sont aussi changés : $D_V = \{3\}$, et $D_W = \{2\}$. Donc, on doit propager de nouveau les contraintes $V = v_Y$ et $W = w_Z$, ce que nous donne $D_Y = \{3\}$ et $D_Z = \{2\}$ (et $D_X = \{5\}$).

Goûter dans une crèche

Il y a l'heure de goûter pour 8 enfants. Chacun a droit à un fruit. Il y a 2 pommes, 2 poires, 1 orange, 1 pamplemousse, et 3 bananes. Les fruits préférés des enfants sont :

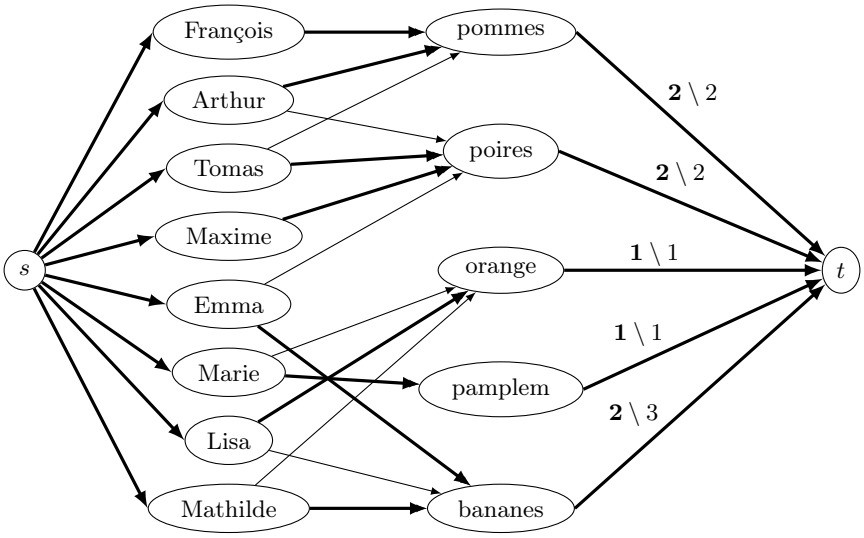
François	*pommes*
Arthur et Tomas	*pommes, poires*
Maxime	*poires*
Emma	*poires, bananes*
Marie	*oranges, pamplemousses*
Lisa et Mathilde	*oranges, bananes*

Si Emma prend une poire, est-ce que les autres enfants pourront chacun choisir un fruit préféré ? Répondez à cette question en utilisant l'algorithme de propagation d'une contrainte globale.

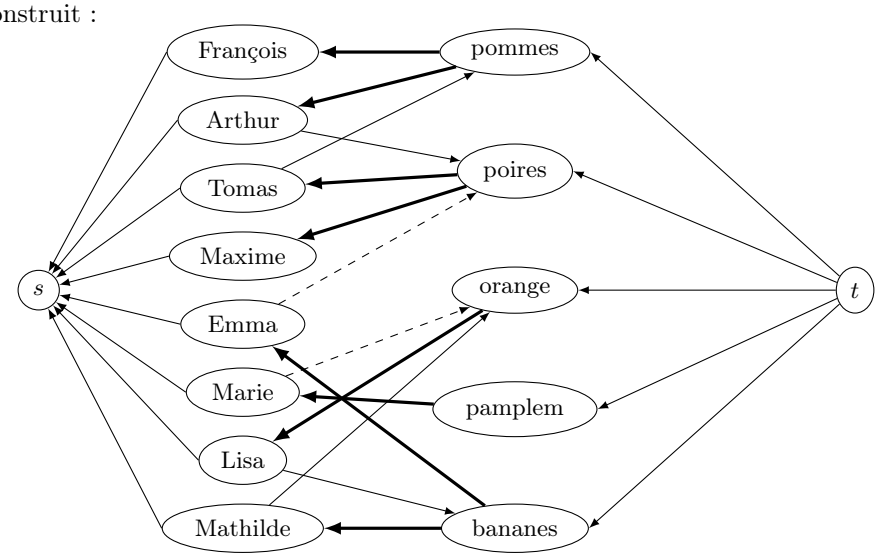
On va modéliser ce problème avec la contrainte de cardinalité globale. On introduit une variables pour chaque enfant. Chaque fruit consideré comme une valeur est dans le domaine de d'une variable-enfant si cet enfant préfère ce fruit. Comme le nombre de fruits disponibles est limité, pour chaque valeur, il y a une borne supérieur pour le nombre de fois cette valeur peut être prise. Donc, on peut modéliser cette situation avec la contrainte globale `gcc`. Après la propagation de cette contrainte, les valeurs restées dans les domaines des variables vont indiquer

quels fruits que les enfants pourront prendre sans empêcher les autres à prendre leurs fruits préférés.

D'abord on construit le flot maximum (en gras) correspondant à notre contrainte `gcc` :



Pour propager la contrainte `gcc`, on utilise le graphe résiduel associé au flot construit :



Les arcs pointillées (entre variables et valeurs) ne font partie du flot et n'appartient pas à aucun circuit alterné. Ces arcs ne peuvent pas appartenir à aucun flot maximum dans ce graphe. Donc, Emma ne peut pas prendre une poire et Marie ne peut prendre une orange sans empêcher les autres à prendre leurs fruits préférés.

Representing integers

What domains are represented by

1. $\{[[x \leq 6]], \neg[[x \leq 2]]\}$
2. $\{[[x \leq 9]], \neg[[x \leq 4]], \neg[[x = 6]], \neg[[x = 8]]\}$
3. $\{[[x = 4]]\}$
4. $\{[[x \leq 5]], \neg[[x \leq 4]]\}$
5. $\{[[x \leq 7]], \neg[[x \leq 1]], \neg[[x = 8]]\}$
6. $\{[[x = 4]], [[x = 7]]\}$
7. $\{\neg[[x \leq 7]], [[x \leq 1]]\}$

Domains are represented by conjunctions of literals. So :

1. $D(x) = \{3, 4, 5, 6\}$
2. $D(x) = \{5, 7, 9\}$
3. $D(x) = \{4\}$
4. $D(x) = \{5\}$
5. $D(x) = \{2, 3, 4, 5, 6, 7\}$
6. $D(x) = \emptyset$
7. $D(x) = \emptyset$

Explanations

Give the resulting domain and explanation for each of the following examples

1. $D(x_1) = \{2, \ldots, 4\}$, $D(x_2) = \{1, \ldots, 4\}$ *:* $x_1 + 1 \leq x_2$
2. $D(x_1) = D(x_2) = D(x_3) = D(x_4) = \{1, 2\}$ *:* `all-different`(x_1, x_2, x_3, x_4)
3. $D(x_1) = \{2, 3\}$, $D(x_2) = \{1, 4\}$, $D(b) = \{false, true\}$ *:* $b \Leftrightarrow x_1 = x_2$
4. $D(x_1) = D(x_2) = \{1, \ldots, 4\}$, $D(x_3) = \{3\}$, $D(x_4) = \{1, \ldots, 4\}$ *:* $2x_1 + x_2 + 3x_3 + x_4 \leq 12$

1. By propagation of constraint $x_1 + 1 \leq x_2$, values 1 and 2 are removed from domain $D(x_2)$ and value 4 is removed from domain $D(x_1)$. The reason for the first domain reduction is $x_1 \geq 2 \Rightarrow x_2 \geq 3$. By translating this to literals, we have $\neg[[x_1 \leq 1]] \Rightarrow \neg[[x_2 \leq 2]]$. Thus, the explanation is

$$[[x_1 \leq 1]] \vee \neg[[x_2 \leq 2]]$$

The explanation for the second domain reduction is

$$\neg[[x_2 \leq 4]] \vee [[x_1 \leq 3]]$$

2. Propagation of the `all-different` constraint makes domains of some variables empty, so a conflict is detected. The reason for the conflict is that three variables x_1, x_2, x_3 can take only 2 values (1 and 2) in total, thus they cannot take different values. So, we have $\neg[[x_1 \geq 0]] \wedge [[x_1 \leq 2]] \wedge \neg[[x_2 \geq 0]] \wedge [[x_2 \leq 2]] \wedge \neg[[x_3 \geq 0]] \wedge [[x_3 \leq 2]] \Rightarrow false$. The explanation is then

$$[[x_1 \geq 0]] \vee \neg[[x_1 \leq 2]] \vee [[x_2 \geq 0]] \vee \neg[[x_2 \leq 2]] \vee [[x_3 \geq 0]] \vee \neg[[x_3 \leq 2]]$$

3. Propagation of constraint $b \Leftrightarrow x_1 = x_2$ implies that $b = false$, as there are no values common values in the domains of x_1 and x_2. Reason is $\neg[[x_1 \leq 1]] \wedge [[x_1 \leq 3]] \wedge \neg[[x_2 = 2]] \wedge \neg[[x_2 = 3]] \Rightarrow \neg b$. Thus, the explanation is

$$[[x_1 \leq 1]] \vee \neg[[x_1 \leq 3]] \vee [[x_2 = 2]] \vee [[x_2 = 3]] \vee \neg b$$

4. As value of x_3 is fixed, the constraint can be rewritten as $2x_1 + x_2 + x_4 \leq 3$, and we note that there are no values in the domains of variables which satisfy this constraint at the same time, so there is a conflict. The reason is that $x_1 \geq 1$ and $x_2 \geq 1$ and $x_3 \geq 3$ and $x_4 \geq 1$. Rewriting with literals : $\neg[[x_1 \leq 0]] \wedge \neg[[x_2 \leq 0]] \wedge \neg[[x_3 \leq 2]] \wedge \neg[[x_4 \leq 0]] \Rightarrow false$. The explanation is then

$$[[x_1 \leq 0]] \vee [[x_2 \leq 0]] \vee [[x_3 \leq 2]] \vee [[x_4 \leq 0]]$$

Balanced Incomplete Block Designs

In this problem, we are given five numbers as an input (v, b, r, k, l). The goal is to construct a $0-1$ matrix with v rows and b columns such that there are exactly r ones per row, k ones per column, and the scalar product of every two rows is l.

An example of solution for $(v, b, r, k, l) = (7, 7, 3, 3, 1)$ is

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Model this problem. Find and eliminate the symmetries of your model.

To model this problem, we introduce binary variables x_{ij}, $1 \leq i \leq v$, $1 \leq j \leq b$, which determine whether element (i, j) is one or not. Then domain is $D(x_{ij}) = \{0, 1\}$.

Constraints are pretty straightforward. First, there are r ones per row :

$$\sum_{j=1}^{b} x_{ij} = r, \quad \forall i \in \{1, \ldots, v\}.$$

Then, there are k ones per column :

$$\sum_{i=1}^{v} x_{ij} = k, \quad \forall j \in \{1, \ldots, b\}.$$

Then, the scalar product of every two rows is l :

$$\sum_{j=1}^{b} (x_{ij} \wedge x_{i'j}) = l, \quad \forall i, i' : \ 1 \leq i < i' \leq v.$$

In this constraint, we use a logical « and » ($\wedge$) to model the product of two binary variables (product is equal to one if an only if both elements are ones). Also, many solvers allow the user to sum up binary variables. In this case, the sum is the total number of variables equal to true.

We now determine the symmetries in this model. Obviously, there is the row permutation symmetry, and the column permutation symmetry. However, eliminating these symmetries is not trivial. What we will do is to require that the rows and columns should be lexicographically sorted. A vector a is lexicographically smaller than vector a' (denoted $a <_{\text{lex}} a'$) if, whenever $a_i > a'_i$, then there should exist index $j < i$ such that $a_j < a'_j$. For example, $(1, 1, 0, 0) <_{\text{lex}} (1, 1, 1, 1)$ and $(0, 1, 1, 1) <_{\text{lex}} (1, 0, 0, 0)$.

A solution satisfying the lexicographic constraints to instance $(v, b, r, k, l) = (7, 7, 3, 3, 1)$ is :

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

To impose lexicographic constraints, on can you `lexleq` (« lexicographically smaller or equal ») global constraints :

$$\texttt{lexleq}(x_{i\cdot}, x_{i'\cdot}), \quad \forall i, i' : \ 1 \leq i < i' \leq v,$$

$$\texttt{lexleq}(x_{\cdot j}, x_{\cdot j'}), \quad \forall j, j' : \ 1 \leq j < j' \leq b,$$

where $x_{i\cdot}$ is the vector of variables corresponding to the i-th row of the matrix, and $x_{\cdot j}$ is the vector of variables corresponding to the j-th columna of the matrix.

An alternative to the lexicographic constraints is to introduce auxiliary boolean variables. Let binary variable $y_{ii'j}$ equals one if and only if there exists index j' such that $1 \leq j' \leq j$ and $x_{ij'} < x_{i'j'}$. Then the following constraints impose lexicographic ordering of rows :

$$y_{i,i',j} = y_{i,i',j-1} \vee (x_{i,j-1} < x_{i',j-1}), \quad \forall i, i', j : \ 1 \leq i < i' \leq v, \ 1 \leq j \leq b,$$

$$(x_{i,j} > x_{i',j}) \vee y_{i,i',j-1}, \quad \forall i, i', j : \ 1 \leq i < i' \leq v, \ 1 \leq j \leq b.$$

Similar auxiliary variables and constraints can be introduced to impose lexicographic ordering of columns.

Scene allocation

You are a movie producer and you need to shoot a film which involves a certain number of actors. For each actor, we know the set of scenes he plays in. At most k scenes can be shot per day. Each actor is paid by the day : if he plays in at least one scene shot this day, he is paid for this day. The pay for each actor is different.

The goal is to assign each scene of the film to a certain day and to minimize the total pay to actors. Model this problem. Find and eliminate the symmetries of your model.

Let us define the necessary notation. Let S be the set of scenes. Let D be the number of days. Let N be the number of actors. Each actor $i \in \{1, \ldots, N\}$ is paid daily fee f_i for playing. Let value a_{is} be equal to one if actor i plays at scene s, otherwise $a_{is} = 0$. Let also S_i be the set of scenes in which actor i plays : $S_i = \{s \in S : \ a_{is} = 1\}$.

We now model the problem. For each scene $s \in S$, let us define variables x_s which determines the day in which scene s is shoot. The domain is $D(x_s) = \{1, \ldots, D\}$. We have additional binary variables y_{id} which determine whether actor i participates in a scene shoot at day d.

The first constraint state that at most k scenes are shoot each day. We can model it with the global cardinality constraint : each value can be taken at most k times by variables x :

$$\texttt{GCC}(x, \{1, \ldots, D\}, \{0, \ldots, 0\}, \{k, \ldots, k\})$$

The second set of constraints link variables x and y :

$$y_{id} = \bigvee_{s \in S_i} (x_s = d), \quad 1 \leq i \leq N, \ 1 \leq d \leq D.$$

Here, expression $x_s = d$ can be used as a binary variable, which is equal to true if and only if the expression is satisfied.

The objective function is to minimize the total salary :

$$\min \sum_{d=1}^{D} \sum_{i=1}^{N} f_i y_{id}.$$

The symmetry of the model is quite clear : days are interchangeable. Any permutation of days keeps a feasible solution feasible, and also keeps the value of the objective function the same.

To eliminate the symmetry, we proceed in the following way. Consider scene number 1. When we decide to which day we assign scene 1, all days are empty. So we can assign scene 1 to day 1, as any other assignment would be the symmetric. When we assign scene 2, there are only two distinguishable options : assign it together with scene 1 or not. Therefore, we can safely assign scene 2 either to day 1 or to day 2. So, according to this reasoning, when we assign scene s, we can assign it together with previous scenes or to an yet empty day. So, variable x_s can take any value in $\{1, \ldots, x_{s-1} + 1\}$. Therefore, the following constraints break the symmetry :

$$x_1 = 1,$$

$$x_s \leq x_{s-1} + 1, \quad \forall s \in S \setminus \{1\}.$$