Devoir 3 : Programmation par contraintes

Correction

1 À la découverte de contraintes globales

1.

```
count(array[int] of var int: x, var int: y, var int: c)
```

x est une liste de variables entières et y et c sont des variables entières. y est une valeur susceptible de se trouver dans la liste x. Cette contrainte globale impose que le nombre d'occurrences de y dans la liste x soit égal à c, ce qui peut s'écrire mathématiquement :

$$c = |\{v \in x \mid v = y\}|$$

Exemple :
Dans un problème de sac à dos, la liste x est indexée par les items que l'on peut emporter et $x[i]$ est la quantité de l'item i que l'on emporte. Si l'on fixe y à 0, la contrainte *count* impose que la variable c soit le nombre d'items que l'on emporte pas du tout. Si par exemple on souhaite avoir au moins un exemplaire de chaque item dans le sac, on peut imposer que c soit nulle.

2.

```
global_cardinality(array [int] of var int: x,
                   array [int] of int: cover,
                   array [int] of var int: counts)
```

x, *cover* et *counts* sont des listes de variables entières. Les éléments de *counts* sont des valeurs susceptibles de se trouver dans la liste x. Cette contrainte impose que pour chaque élément de *cover* se trouvant en position i dans cette liste, le nombre d'occurrences de cet élément dans la liste x soit égal au i^{eme} élément de *count*. Cette contrainte peut se réécrire :

```
forall (i in 1..length(cover)) count(array [int] of var int: x, var int: cover[i],
var int: counts[i])
```

On peut l'écrire mathématiquement sous la forme :

$$\forall i \in \{\ 1,\ ...,\ |cover|\ \},\ count_i = |\{v \in x \mid v = cover_i\}|$$

Exemple :
Dans l'exercice 2, on se sert de cette contrainte : x est la liste des chiffres d'un code, *cover* est l'ensemble des valeurs qu'ils peuvent prendre et *count* le nombre d'occurrences de chaque valeur dans le code. Cette contrainte permet d'imposer que chaque valeur apparaît au plus 2 fois grâce à la définition que l'on fait préalablement de *count*.

3.

```
cumulative(array [int] of var int: s,
           array [int] of var int: d,
           array [int] of var int: r,
           var int: b)
```

On considère que l'on a un ensemble de tâches à effectuer. s, d et r sont des listes de variables entières donnant respectivement la date de début de chaque tâche, sa durée et la quantité de ressource qu'elle nécessite. b est une variable entière. Cette contrainte impose qu'à tout moment dans le processus, la somme des demandes en ressource des tâches en train d'être effectuées n'excède pas b.

Exemple : On souhaite préparer un très grand gâteau au chocolat, les tâches sont les étapes de la recette. Certaines tâches peuvent être faites en même temps (couper le chocolat et battre les blancs en neige) mais il y a un ordre à respecter (préchauffer le four avant d'enfourner le gâteau). Les tâches sont donc ordonnées et se superposent par moments. Notre ressource est la main d'œuvre. Comme c'est un très grand gâteau, il faut souvent plusieurs personnes sur chaque tâche. s contient les dates de début de chaque tâche, d leur durée et r le nombre de personnes nécessaires pour les effectuer. On a un nombre fixe de cuisiniers et cuisinières égal à b. La contrainte *cumulative* permet de s'assurer que notre recette ne prévoit à aucun moment d'avoir besoin de plus de main d'œuvre que ce dont on dispose.

2 Un code secret de la plus haute importance

1.

Données :

$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$: ensemble des valeurs possibles pour les chiffres du code

$C = \{N, B, V\}$: ensemble des couleurs possibles pour les chiffres du code

$n = 5$: nombre de chiffres dans le code

Variables de décision :

x_i, $1 \leq i \leq n$: valeur du chiffre en position i dans le code

c_i, $1 \leq i \leq n$: couleur du chiffre en position i dans le code

Contraintes :

- $\forall i \in \{1, ..., n\},\ x_i \in X$: la valeur d'un chiffre du code est dans l'ensemble X
- $\forall i \in \{1, ..., n\},\ c_i \in C$: la couleur d'un chiffre du code est dans l'ensemble C
- $\forall i \in \{1, ..., n-1\},\ x_i \leq x_{i+1}$: les valeurs sont en ordre croissant
- $\forall i \in \{1, ..., n\},\ x_i \neq 5 \Rightarrow c_i \in \{N, B\}$: la couleur d'un chiffre autre que 5 est noir ou blanc

— $\forall i \in \{1, ..., n\},\ x_i = 5 \Rightarrow c_i = V$: la couleur d'un 5 est vert

— $\forall i \in \{1, ..., n-1\},\ x_i = x_{i+1}$ et $x_i \neq 5 \Rightarrow c_i = N$ et $c_{i+1} = B$: si un chiffre d'une valeur autre que 5 est présent deux fois, le premier est noir et le second est blanc

— $\forall k \in X,\ |\{i \in \{1, ..., n\} | x_i = k\}| \leq 2$: chaque valeur apparaît au plus deux fois

— $|\{i \in \{1, ..., n-1\} | x_i = x_{i+1}\}| \geq 1$: deux chiffres du code ont la même valeur

— $c_1 = c_2 = c_3$ et $\forall i \in \{3, ..., n-1\},\ c_i \neq c_{i+1}$: les trois premiers chiffres du code sont les seuls chiffres adjacents ayant la même couleur

— $x_{n-2} + x_{n-1} + x_n = 7$: la somme des valeurs des trois derniers chiffres vaut 7

Fonction objectif :

Ici il n'y a pas de fonctions objectif, on cherche simplement une solution qui satisfasse toutes les contraintes.

2.

Il n'y a qu'une seule solution possible, c'est le code 0N,1N,2N,2B,3N. On peut donc désamorcer la bombe à coup sûr.

3.

Cette fois il y a 4 solutions possibles :
1N,2N,2B,3B,6B,
1N,2N,3B,5V,8B,
1N,2N,4B,5V,7B,
et 0B,2B,3N,3B,6B.
On a donc 1 chance sur 4 de réussir à désamorcer la bombe.

3 Tournoi de tennis au CEPSUM

1. Modélisation PPC

Données :

T : ensemble des équipes participant au tournoi

W : ensemble des semaines sur lesquelles s'étale le tournoi

$N = 2$: nombre maximum de matchs par semaine qu'une équipe peut jouer

Variables de décision :

$x_{ij} \in W \cup \{0\}$, $(i,j) \in T^2$: numéro de la semaine où l'équipe i rencontre l'équipe j (par convention, $\forall i \in T$, $x_{ii} = 0$)

Contraintes :

- $\forall (i,j) \in T^2$, $i \neq j$, $x_{ij} \geq 1$: toutes les équipes doivent se rencontrer
- $\forall i \in T$, $w \in W$, $|\{j \ \in T \mid \ = w\}| \leq N$: une équipe joue au plus N matchs par semaine
- $\forall (i,j) \in T^2$, $x_{ij} = x_{ji}$: i contre j et j contre i sont le même match donc le problème est symétrique

Fonction objectif :

$$\min \max_{i,j \in T} x_{ij}$$

On souhaite minimiser le nombre de semaines nécessaires pour faire se rencontrer toutes les équipes.

2. Modélisation sur MiniZinc

L'ensemble W est modélisé par le array *"weeks"* dans MiniZinc. Il s'agit d'énumérer les semaines de 1 à l'entier N_s, où N_s est une borne supérieure sur le nombre de semaines du tournois. On peut le voir comme le pire nombre de semaine (on fixe une borne numérique suffisamment large). On modélise ensuite le problème à l'aide de MiniZinc.

3. Résultats : comparaison des solveurs Chuffed et Gecode

Pour 5 équipes

On peut organiser le tournoi sur 2 semaines de la façon suivante :

$$x = \begin{bmatrix} 0 & 1 & 1 & 2 & 2 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 2 & 1 \\ 2 & 1 & 2 & 0 & 1 \\ 2 & 2 & 1 & 1 & 0 \end{bmatrix}$$

Le solveur *Chuffed* est ici meilleur que *Gecode* (il est 5 fois plus rapide). Pourtant *Chuffed* visite beaucoup plus de noeuds (277 contre 15 pour *Gecode*) et cherche plus en profondeur. On peut expliquer la différence de temps de résolution par le nombre de feuilles mortes (failures) que rencontre *Gecode* (5 contre 3 pour *Chuffed*).

	Chuffed	*Gecode*
nodes	277	15
failures	3	5
solveTime	0.001	0.005
peakDepth	24	10

TABLE 1 – Performances pour 5 équipes

Pour 10 équipes

On peut organiser le tournois sur 5 semaines de la façon suivante :

$$x = \begin{bmatrix} 0 & 3 & 3 & 2 & 1 & 1 & 4 & 4 & 5 & 5 \\ 3 & 0 & 1 & 5 & 2 & 3 & 5 & 2 & 4 & 4 \\ 3 & 1 & 0 & 2 & 5 & 5 & 3 & 2 & 4 & 4 \\ 2 & 5 & 2 & 0 & 4 & 1 & 1 & 5 & 3 & 3 \\ 1 & 2 & 5 & 4 & 0 & 5 & 1 & 4 & 3 & 3 \\ 1 & 3 & 5 & 1 & 5 & 0 & 4 & 3 & 2 & 2 \\ 4 & 5 & 3 & 1 & 1 & 4 & 0 & 3 & 2 & 2 \\ 4 & 2 & 2 & 5 & 4 & 3 & 3 & 0 & 1 & 1 \\ 5 & 4 & 4 & 3 & 3 & 2 & 2 & 1 & 0 & 1 \\ 5 & 4 & 4 & 3 & 3 & 2 & 2 & 1 & 1 & 0 \end{bmatrix}$$

	Chuffed	*Gecode*
nodes	4994	53
failures	3154	10
solveTime	0.083	0.001
peakDepth	149	45

TABLE 2 – Performances pour 10 équipes

Pour 10 équipes, c'est *Gecode* qui devient meilleur en visitant beaucoup moins de noeuds que *Chuffed* (53 contre 4994 pour *Chuffed*, soit presque 100 fois moins ...). Le taux d'échec est beaucoup plus élevé du côté de *Chuffed* (63% contre 19% chez *Gecode*). Cela explique l'écart de temps de calcul, qui devrait se creuser de plus en plus lorsqu'on augmente le nombre d'équipes.

Pour 13 équipes

On peut organiser le tournoi sur 6 semaines de la façon suivante :

$$x = \begin{bmatrix} 0 & 1 & 1 & 2 & 3 & 3 & 4 & 2 & 4 & 5 & 5 & 6 & 6 \\ 1 & 0 & 2 & 3 & 2 & 1 & 4 & 5 & 3 & 4 & 5 & 6 & 6 \\ 1 & 2 & 0 & 3 & 3 & 1 & 2 & 4 & 6 & 6 & 4 & 5 & 5 \\ 2 & 3 & 3 & 0 & 6 & 6 & 2 & 1 & 1 & 4 & 4 & 5 & 5 \\ 3 & 2 & 3 & 6 & 0 & 6 & 1 & 5 & 1 & 5 & 2 & 4 & 4 \\ 3 & 1 & 1 & 6 & 6 & 0 & 5 & 2 & 5 & 3 & 2 & 4 & 4 \\ 4 & 4 & 2 & 2 & 1 & 5 & 0 & 6 & 5 & 1 & 6 & 3 & 3 \\ 2 & 5 & 4 & 1 & 5 & 2 & 6 & 0 & 4 & 1 & 6 & 3 & 3 \\ 4 & 3 & 6 & 1 & 1 & 5 & 5 & 4 & 0 & 6 & 3 & 2 & 2 \\ 5 & 4 & 6 & 4 & 5 & 3 & 1 & 1 & 6 & 0 & 3 & 2 & 2 \\ 5 & 5 & 4 & 4 & 2 & 2 & 6 & 6 & 3 & 3 & 0 & 1 & 1 \\ 6 & 6 & 5 & 5 & 4 & 4 & 3 & 3 & 2 & 2 & 1 & 0 & 1 \\ 6 & 6 & 5 & 5 & 4 & 4 & 3 & 3 & 2 & 2 & 1 & 1 & 0 \end{bmatrix}$$

	Chuffed	*Gecode*
nodes	55284	100
failures	45357	17
solveTime	1.01	0.002
peakDepth	303	78

TABLE 3 – Performances pour 13 équipes

Là encore, *Gecode* est largement meilleurs. Il explore 500 fois moins de noeuds, et son taux d'échec est de 17% contre 80% pour *Chuffed*. Le temps de calcul est divisé par 500 en utilisant *Gecode*. Le choix du solveur est donc plutôt simple.

4. Passage à l'échelle du problème

On ne remarque pas une résolution beaucoup plus longue que précédemment pour 13 équipes (on est dans le même ordre de grandeur avec *Gecode*). Néanmoins, sans nuire à la généralité, on ajouter une contrainte qui impose que la première équipe rencontre les autres dans l'ordre et ainsi réduire la symétrie du problème.