TP 3 – Programmation linéaire en nombres entiers et programmation par contraintes

1 Flow-shop

Le *Flow shop scheduling problem* est un problème de planification de n tâches sur m machines.

Première situation

Une série de 8 pièces détachées doivent être construites en passant successivement dans 3 machines. Pour cela, la construction de chaque pièce doit passer sur toutes les machines dans le même ordre : m_1, m_2 et m_3. Une machine ne peut traiter qu'une tâche à la fois. Le passage d'une tâche dans une machine est appelé *opération*. La durée de chaque opération est connue (Tableau 1). L'objectif est de minimiser le *makespan*, c'est à dire le temps final d'exécution pour lequel toutes les tâches sont terminées.

1. Modélisez ce problème en programmation linéaire en nombres entiers. Cela est-il facile/naturel ? Attention à la linéarité !
2. Modélisez ce problème en programmation par contraintes. Quel paradigme de modélisation préférez-vous ?
3. Résolvez ce problème avec la programmation par contraintes, des fichiers .dzn et .mzn vous sont donnés. Regardez dans la section *Scheduling constraints* des contraintes globales de l'API pour voir si une contrainte spécifique peut être utilisée pour empêcher le chevauchement des activités sur une même machine. Un array peut être construit par compréhension (ex : `[start[i,j] | i in 1..p]`).

Deuxième situation

Jusqu'à présent, on a supposé que le délai entre la fin et le début de deux opérations successives d'une même pièce est négligeable. Cependant, cela n'arrive que rarement en pratique. On introduit maintenant un temps de latence entre l'exécution de deux opérations successives d'une même tâche. Ce temps varie d'une pièce à l'autre et qui est dépendant de la prochaine machine utilisée (Table 2). En guise d'illustration, ce temps pourrait être plus important pour une tâche nécessitant un temps de préparation avant sa réalisation.

La valeur à l'entrée (p, m) dans la table est le temps minimum de transition avant de réaliser une opération de la pièce p sur la machine m par rapport à la fin de l'opération de p sur $m-1$. Par exemple, le temps $(p, 1)$ est la durée avant la première opération de la pièce p.

Finalement, deux autres contraintes sont ajoutées :

1. Les tâches relatives à la pièce 4 doivent être effectuées directement après celles de la pièce 8, et cela sur toutes les machines.
2. La pièce 3 doit être terminée avant la pièce 5.

Modifiez le modèle précédent afin de ternir compte de cette nouvelle situation et résolvez le. Un autre fichier .dzn vous est donné.

Pièce/Machine	1	2	3
1	1	3	2
2	3	6	4
3	4	9	4
4	1	2	11
5	5	7	6
6	3	6	1
7	3	5	6
8	1	1	3

TABLE 1 – Durée des opérations pour les paires pièce/machine.

Pièce/Machine	1	2	3
1	2	1	2
2	1	3	1
3	12	1	8
4	4	1	1
5	1	2	3
6	2	1	0
7	3	2	1
8	1	3	1

TABLE 2 – Temps de transition pour les paires pièce/machine.

2 Défi

Résolvez l'équation $ABCDE * A = EEEEEE$ ou chaque lettre représente un chiffre entier différent. Serez-vous plus rapide à la main ou avec MiniZinc ? La contrainte `all_different` pourra vous être utile pour ce problème (n'oubliez pas la ligne `include "globals.mzn";` qui indique que l'on souhaite importer de nouvelles contraintes en début de fichier).