Devoir 3 – Programmation par contraintes

Remise avant le 25 octobre à 23h59 sur Moodle.

Consignes

- Les devoirs doivent être réalisés seul ou par binôme (de préférence par binôme).
- Soumettez un document (format pdf) reprenant votre devoir, ainsi que votre modèle MiniZinc (fichier .mzn).
- Indiquez vos noms et matricules dans le titre des fichiers soumis.
- Veillez à rendre un rapport **structuré**, **clair** et **concis**. Les lacunes de forme seront pénalisées.
- L'utilisation d'une IA générative est **STRICTEMENT INTERDITE**.

Question 1 – À la découverte de contraintes globales (4 pts)

Un des points forts de la programmation par contraintes est son vaste catalogue de contraintes globales. Pour cette question, il vous est demandé de parcourir la documentation MiniZinc afin de comprendre de nouvelles contraintes globales. Concrètement, donnez une explication en quelques lignes des contraintes suivantes et inventez pour chaque un exemple court pour illustrer :

1. `at_most(int: n, array [int] of var int: x, int: v)`
2. `count(array[int] of var int: x, var int: y, var int: c)`
3. `global_cardinality(array[int] of var int: x, array[int] of int: cover, array[int] of var int: counts)`
4. `cumulative(array[int] of var int: s,array[int] of var int: d, array[int] of var int: r, var int: b)`

Attention : il n'est pas suffisant de recopier la définition donnée dans la documentation. A la place, vous devez expliquer la sémantique des contraintes avec vos mots. Donnez également un exemple d'utilisation avec des variables/paramètres que vous définissez (soyez succincts, l'idée est plus importante que la rigueur de l'exemple). Pour cela, vous pouvez utiliser la syntaxe MiniZinc.

1 Question 2 – Un code secret (6 pts)

Vous êtes en mission pour désamorcer une bombe et vous avez pour cela besoin d'un code secret à cinq chiffres. Celui-ci est sécurisé par des règles bien particulières, et vous savez qu'il suit la logique suivante :

1. Chaque chiffre a une valeur, entre 0 et 9, et une couleur, noir ou blanc (N ou B). Il y a donc vingt chiffres possibles ($0N, 1N, \ldots, 9N, 0B, 1B, \ldots, 9B$).
2. Les cinq chiffres du code sont à choix unique dans cette liste (il ne peut y avoir qu'un seul 1 noir, par exemple).
3. Les chiffres sont en ordre croissant, avec le chiffre noir avant le blanc s'il y a deux fois la même valeur.
4. Les 5 font exception pour la couleur : il n'y a pas de 5 noir ni blanc dans la liste des chiffres possibles, mais deux 5 verts.

Ainsi un code pourrait ressembler à $2N$, $2B$, $5V$, $5V$, $9B$.

Vous vous êtes infiltrés dans une salle de commande et après avoir injecté un logiciel espion dans l'ordinateur de contrôle vous obtenez pour le code les indications suivantes :

— Deux chiffres du code ont la même valeur.
— Les trois premiers chiffres du code sont les seuls chiffres adjacents ayant la même couleur.
— La somme des valeurs des trois derniers chiffres du code vaut 7.

Faites vite et bien ! Vous n'avez qu'un seul essai, et les gardes reviennent dans 2 minutes.

1. Modélisez ce problème en utilisant la programmation par contraintes. Formalisez mathématiquement les données, les variables de décision, la fonction objectif ainsi que les contraintes.
2. Modélisez ce problème sur MiniZinc. Pouvez-vous désamorcer la bombe à coup sûr ? Si non, quelle est votre chance de succès ?
3. Vous réalisez qu'il y a une seconde bombe à désamorcer, et obtenez cette fois-ci les renseignements suivants. Pouvez-vous désamorcer la bombe à coup sûr ? Si non, quelle est votre chance de succès ?
   — Il y a un 2 en deuxième position du code.
   — La somme des valeurs des chiffres noirs du code vaut 3.
   — La somme des valeurs des chiffres blancs du code vaut 11.
   — Il n'y a pas de 0 noir, de 1 blanc, de 7 noir ni de 9 dans le code.

Question 3 – Ordonnancement avec ressources multiples (10 pts)

Une boulangerie doit produire une grande quantité de produits avec des ressources très limitées. La taille de l'atelier et le nombre réduit de four sont des contraintes pour la production qui doit donc être minutieusement planifiée. Les boulangeries embauchent habituellement une personne pour produire un planning des tâches. Ce planning est généralement produit le lundi quand les boulangeries sont fermées, et vous avez été embauchés pour produire différents plannings (de 30 à 90 tâches ; vous êtes rémunérés en crédits de cours universitaires).

La boulangerie possède plusieurs ressources de production, mais en nombres limités (avec 3 fours différents, seulement 3 fournées peuvent être effectuées en même temps par exemple). Pour cette boulangerie nous considérerons 4 ressources : les fours, les employés, les robots et les grilles de refroidissement. Vous avez une série de tâches de production à ordonnancer (c-à-d que vous devez décider quand elles vont s'exécuter, à partir d'un temps 0). Les tâches sont caractérisées par une durée, qui est connue, et ne peuvent pas être interrompues. De plus, chaque tâche est sujette à des relations de précédence avec d'autres tâches : on ne peut pas cuire la pâte avant qu'elle ait monté. Les tâches ne peuvent donc être exécutées que si toutes les autres tâches de l'ensemble des précédentes ont déjà été complètement exécutées. Pour chaque tâche dans le fichier `.dzn` on donne `suc[]` la liste des tâches qui doivent être exécutées ultérieurement (pour la tâche i, `suc[i]` est la liste des tâches qui ne peuvent commencer qu'après la fin de la tâche i).

On a également un ensemble de ressources ayant chacune leur propre capacité. Durant leur exécution, chaque tâche a une demande particulière (pouvant être nulle) en chaque ressource. A tout moment,la capacité de chaque ressource ne peut pas être dépassée. Finalement, l'objectif est de minimiser le makespan, c'est à dire le temps final d'exécution.

1. Modélisez ce problème en utilisant la **programmation par contraintes**. Formalisez mathématiquement les données, les variables de décisions, la fonction objectif ainsi que les contraintes.

2. Modélisez ce problème sur MiniZinc, **en utilisant au moins une contrainte globale**. Définissez les données comme indiqué dans les fichiers `boulangerie1.dzn`, `boulangerie2.dzn` et `boulangerie3.dzn` fournis. Chaque fichier représente une instance différente du problème.
3. Résolvez les trois instances en utilisant les solveurs *Gecode* ou *Chuffed*. Reportez les solutions des trois situations dans le rapport (temps de début et coût optimal). Pour vous aider à corriger vos modèles, la solution optimale de la première situation vous est donnée (première ligne du fichier `boulangerie1.dzn`).
4. Finalement, reportez les statistiques de recherche (nodes, failures, solveTime et peakDepth) des trois instances pour les solveurs *Gecode* et *Chuffed*. Comparez les performances de *Chuffed* et *Gecode*. Cette question a pour objectif de vous sensibiliser aux différences qu'il existe entre les solveurs. (Les statistiques de résolution sont obtenues avec *Show configuration editor* > *Options* > *Output* > *Output solving statistics*).