Fiche pratique 2 : Introduction MiniZinc

0. MiniZinc

MiniZinc est un logiciel open-source facilitant l'écriture de programmes d'optimisation. Nous nous en servirons principalement pour la programmation par contraintes, et vous devrez vous familiariser avec la syntaxe et l'utilisation du logiciel.

Vos principales ressources comprennent :

- le tutoriel MiniZinc officiel,
- le *'cheat sheet'* MiniZinc,
- la présente introduction,
- les corrections de laboratoire.

1. Tutoriel

C'est la référence pour toutes vos interrogations à propos de MiniZinc ; vous pouvez en trouver le lien sur Moodle. La partie '2.1. Basic Modelling in MiniZinc' vous sera particulièrement utile pour vos premiers programmes.

2. *Cheat Sheet*

Un rappel fort pratique de la syntaxe est toujours à portée de main dans l'IDE de MiniZinc dans l'onglet 'Help/Cheat Sheet...'. Ouvrez-le et faites-en un réflexe !

3. Introduction à la syntaxe

Vous trouverez ici quelques exemples simples pour vous débuter en MiniZinc et comprendre la structure données/variables/contraintes/objectif en programmation mathématique. La documentation reste la référence pour tout ce qui est un peu plus avancé !

Données et variables

Les données et les variables sont déclarées en début de fichier `.mzn` ; on leur donne un type (*int*, *float* ou un intervalle *lb..ub*).

```
%donnée entière
int: N = 3;

#tableau N×N, chargé par un fichier .dzn
array[1..N,1..N] of float: Distance;
```

Les variables sont différenciées par l'ajout de *var*. L'objectif du programme est de leur assigner une valeur pour satisfaire le problème ou lui trouver une solution optimale.

```
%variable entière
var int: x;

#liste de taille N de variables booléennes
array[1..N] of var 0..1: y;

#tableau de taille N×10 de variables décimales
array[1..N,1..10] of var float: z;
```

Contraintes

On ajoute des contraintes avec *constraint* puis l'expression à satisfaire. *forall(i in 1..N)(%expression)* permet d'appliquer la contrainte à tous $i \in [1, N]$, *sum(%indices)(%expression)* permet d'effectuer une sommation.

```
%x = y1 + y2
constraint x = y[1]+y[2];

%∀i ∈ [1, N], yi ≤ ∑j∈[1,10] zij
constraint forall(i in 1..N)( y[i] <= sum(j in 1..10)( z[i,j] ) );

%borne supérieure sur les zij défini par le maximum d'un tableau
constraint forall(i in 1..N,j in 1..10)( z[i,j] <= max(Distance) );
```

Résolution

Un objectif peut être de trouver des valeurs admissibles pour toutes les variables qui satisfassent les contraintes (problème de satisfaction).

```
solve satisfy;
```

Un autre objectif peut être de minimiser/maximiser une valeur tout en satisfaisant les contraintes (problème d'optimisation).

```
%minimisation de la somme des yi
var int : objective = sum(i in 1..N)(y[i]);
solve minimize objective;
```

Contraintes globales

Certaines contraintes habituelles en programmation par contraintes sont implémentées par MiniZinc ; leur utilisation est souvent pratique pour les utilisateurs et efficace pour les solveurs. Vous pouvez trouver une liste des contraintes globales de MiniZinc dans la documentation ; familiarisez-vous par exemple avec *alldifferent*, *count* ou *among*.

```
%il faut inclure le fichier globals
include "globals.mzn";

%définition d'un tableau de variables qui doivent prendre des valeurs différentes dans [1, M]
%(réalisable si N ≤ M)
array[1..N] of var 1..M: x;
constraint alldifferent(x);

%si uniquement les P premières variables doivent être différentes
constraint alldifferent([x[i] | i in 1..P]);
```

4. Solveurs

Programmation linéaire et par contraintes

Prenez garde au solveur que vous utilisez : *COIN-BC* résout des programmes linéaires, tandis ce que *Chuffed* et *Gecode* sont spécialisés en programmation par contraintes. *Gecode Gist* est un outil visuel qui vous permet de suivre l'évolution de l'arbre de recherche pendant la résolution.

Les autres options, les détails et toutes les références associées aux solveurs se trouvent dans la documentation.

Statistiques et réglages avancés

L'onglet *'Show configuration editor'* vous permet de modifier les options associées au solveur. Notamment, *Solving* > *User defined behavior* vous permettra de demander au solveur de trouver toutes les solutions d'un problème de satisfaction plutôt qu'une seule. De plus, *Output solving statistics* demande au solveur de renvoyer des statistiques utiles à propos de la résolution, telles que le nombre de variables, de solutions trouvées, de noeuds explorés, de backtrack, etc. Vous n'aurez pas besoin de toucher aux options de compilation ou de parallélisation.