Modélisation ensembliste avec LocalSolver

Thierry Benoist[1], Julien Darlay[1], Bertrand Estellon[2],
Frédéric Gardi[1], Romain Megel[1], Clément Pajean[1]

[1] Innovation 24 – LocalSolver, 75008 Paris, France
{tbenoist, jdarlay, fgardi, rmegel, cpajean}@localsolver.com
[2] Laboratoire d'Informatique Fondamentale, Aix-Marseille Université et CNRS,
Faculté des Sciences de Luminy, F-13288 Marseille Cedex 9, France
{bertrand.estellon}@lif.univ-mrs.fr

Mots-clés : *LocalSolver, planification de tournées, ordonnancement, modélisation*

1 Introduction

LocalSolver est un solveur d'optimisation mathématique de nouvelle génération. De type "model & run", il permet de modéliser un problème d'optimisation avec des opérateurs mathématiques usuels et produit alors des solutions de grande qualité en des temps très courts. Combinant différentes techniques dans le cadre d'une approche heuristique, LocalSolver parvient à traiter des problèmes non linéaires ou combinatoires comportant des millions de variables, sur des ordinateurs standards. Un des grands atouts de ce solver est d'offrir un cadre de modélisation à la fois riche et simple. En effet la plupart des opérateurs mathématiques usuels sont disponibles qu'ils s'agissent d'expressions arithmétiques (sommes, produits, fonction trigonométriques) ou d'expressions logiques (comparaisons, conditions, accès aux éléments d'un tableau...). Dès lors il n'est pas nécessaire de "linéariser" le problème à résoudre : l'utilisateur peut le modéliser de la façon la plus directe possible.

2 Listes variables

En 2015, nous avons introduit des variables de type "Liste" inspirées des Set Based Variables de la Programmation par Contraintes afin de modéliser de façon encore plus compacte les problèmes d'optimisation comportant des notions d'ordre ou de séquence : ordonnancement, tournées, conception de réseaux, etc. Ce concept de liste variable a été introduit dans LocalSolver avec la définition suivante : une variable `list(n)` peut prendre comme valeur toute sous-permutation de l'ensemble $\{0, 1, \ldots, n-1\}$.

3 Opérateurs variadiques

En 2017, afin de simplifier l'écriture des modèles comportant des listes de taille variable, nous ajoutons la possibilité de faire varier l'arité de certains opérateurs en fonction des variables du modèle. Notre objectif a été de permettre aux utilisateurs de pouvoir manipuler les listes variables et les tableaux de taille fixe en utilisant une syntaxe très similaire, c'est-à-dire, sans avoir à apprendre une nouvelle syntaxe. Il est, par exemple, possible de calculer la longueur d'une route dans un problème de tournées de véhicules de la façon suivante :

```
route <- list(clientsCount);
dist <- sum[i in 1..count(route)-1](distMatrix[route[i-1]][route[i]]);
```

4 Tableaux récursifs

Nous avons également introduit la possibilité de définir des tableaux de taille variable récursivement : la valeur d'une case dépend alors des valeurs des cases précédentes et des variables du modèle. Cela permet, par exemple, de calculer la date minimale de livraison des clients dans un problème de tournées de véhicules avec fenêtres de temps :

```
route <- list(clientsCount);
departureTime[i in 0..count(route)-1] <-
     (i == 0) ? startTimeWindow[route[i]]
              : max(startTimeWindow[route[i]],
                    departureTime[i-1] + distMatrix[route[i-1]][route[i]]);
```

Les valeurs du tableau peuvent être ensuite contraintes afin de ne pas dépasser la date maximale de livraison de chaque client :

```
constraint[i in 0..count(route)-1] departureTime[i] <= endTimeWindow[route[i]];
```

Nous montrerons dans cet exposé comment utiliser ces nouvelles fonctionnalités pour construire des modèles très simples et très efficaces pour de nombreux problèmes d'optimisation.