EasyChair Preprint

№ 2646

Best Practices to Model Routing Problems Using LocalSolver

Tiphaine Rougerie

February 12, 2020

Modélisation de problèmes de tournées de véhicules avec LocalSolver

Tiphaine Rougerie

LocalSolver, 36 avenue Hoche, Paris, France
trougerie@localsolver.com

Mots-clés : *solveur, routing, tournées, recherche locale*

1 Introduction

LocalSolver est un solveur d'optimisation mathématique de type *model & run* [1]. Son formalisme d'entrée lui permet d'accepter tout modèle utilisant les opérateurs mathématiques usuels (arithmétiques, logiques, relationnels, etc.) avec des variables continues, entières ou ensemblistes. La recherche de solutions est faite par une approche heuristique basée sur des techniques de recherche locale dont l'objectif est de trouver rapidement des solutions de qualité. Des techniques d'optimisation globale sont utilisées en complément de la recherche locale pour accélérer la recherche de solutions, pour calculer des bornes et arrêter la recherche lorsque la solution optimale est trouvée.

Les variables de liste ont été introduites dans LocalSolver 5.5 pour modéliser de façon plus compacte des problèmes avec une notion d'ordre ou de séquence. Une liste est une variable de décision riche qui représente une sous-permutation de l'ensemble $\{0, 1, ..., n-1\}$. Le formalisme de modélisation de LocalSolver a ensuite été étendu pour ajouter des opérateurs variadiques permettant de manipuler plus facilement des listes. L'objectif de cet exposé est de donner des retours d'expérience sur la modélisation des problèmes de tournées de véhicules industriels.

2 Voyageur de commerce et tournées de véhicules

Le problème du voyageur de commerce [2] peut être défini à partir d'un ensemble de villes et d'une distance entre chaque paire de villes. Il consiste à trouver la plus courte séquence permettant de visiter toutes les villes et de revenir au point de départ. Le modèle LocalSolver s'écrit avec une liste contrainte à contenir toutes les villes et sur laquelle on somme les distances entre les villes consécutives.

```
cities <- list(nbCities);
constraint count(cities) == nbCities; // Assign all cities
minimize sum(1..nbCities-1, i => distanceWeight[cities[i-1]][cities[i]])
                + distanceWeight[cities[nbCities-1]][cities[0]];
```

Les problèmes de tournées de véhicules nécessitent de modéliser les séquences de plusieurs véhicules qui doivent se partager un ensemble de clients, tout en minimisant la distance parcourue. Le modèle nécessite d'avoir une variable de liste par véhicule et de contraindre toutes les listes à former une partition de l'espace des clients.

```
customersSequences[k in 1..nbTrucks] <- list(nbCustomers);
constraint partition[k in 1..nbTrucks](customersSequences[k]);
```

Le calcul de la longueur de chaque tour est similaire au problème du voyageur de commerce et l'objectif devient la somme des longueurs de chaque tour.

3 Extensions

Les problèmes de tournées de véhicules sont en pratique rarement purs et des contraintes supplémentaires viennent s'ajouter. Les contraintes de capacité peuvent s'appliquer afin de prendre en compte la contenance des véhicules, de limiter le temps de travail d'un chauffeur, etc. Ces contraintes s'expriment comme une somme sur les éléments d'une liste.

```
routeQuantity <- sum(0..c-1, i => demands[sequence[i]]);
constraint routeQuantity <= truckCapacity;
```

Certains clients peuvent n'être accessibles que par certains camions. Ces compatibilités s'écrivent par des contraintes d'appartenance ou d'exclusion à une liste.

```
constraint !(contains(sequence[i], c)); // Client c cannot be assigned to vehicule i
```

Au-delà de ces contraintes classiques, des variantes plus spécifiques seront présentées : des problèmes avec fenêtre de temps, le *pickup and delivery* qui impose des contraintes de précédence entre les éléments du tour, des problèmes où les temps de trajet varient en fonction de la charge du véhicule, etc. Ces variantes peuvent enfin s'adapter en *prize-collecting VRP*, qui autorise à ne pas livrer certains clients.

Ces différents problèmes inspirés de cas clients seront l'occasion de détailler les bonnes pratiques de modélisation avec LocalSolver.

Références

[1] F. Gardi, T. Benoist, J. Darlay, B. Estellon, and R. Megel. *Mathematical Programming Solver Based on Local Search*. Wiley, 2014.

[2] M. R. Garey, and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. WH Freeman, 1979.