Hybridation entre recherche incomplète et apprentissage de conflits pour un problème de routage avec sélection de clients

Trong-Hieu Tran[1], Cédric Pralet[2], Hélène Fargier[1]

[1] IRIT-CNRS, Université de Toulouse, Toulouse, France
{trong-hieu.tran,helene.fargier}@irit.fr

[2] ONERA/DTIS, Université de Toulouse, F-31055 Toulouse, France
cedric.pralet@onera.fr

Mots-clés : *recherche incomplète, apprentissage de conflits, OBDD, OPTW.*

1 Problématique

Les méthodes de recherche dites incomplètes sont souvent utilisées en pratique pour traiter en temps contraint des problèmes d'optimisation difficiles, en s'appuyant par exemple sur des métaheuristiques type recherche tabou pour s'échapper des optima locaux. Nous nous intéressons ici au développement d'une nouvelle stratégie de recherche permettant d'améliorer la qualité des solutions produites ou de produire ces solutions plus rapidement. L'idée poursuivie consiste à réaliser une *recherche incomplète guidée par les conflits* rencontrés lors de la résolution, un conflit correspondant dans notre cas à une instanciation d'un sous-ensemble des variables de décision qui conduit à une violation des contraintes du problème ou à une sous-optimalité de la solution. L'objectif est de mémoriser ces conflits *sur le long terme* et de les utiliser pour d'une part éviter de revisiter inutilement certaines zones de l'espace de recherche et d'autre part tenter d'orienter la recherche vers les solutions les plus prometteuses. Cette idée de recherche guidée par les conflits est à rapprocher de la méthode CDCL (*Conflict Driven Clause Learning*) travaillant sur des problèmes SAT [1, 2].

La nouvelle méthode proposée est appliquée au problème de routage connu sous le nom d'OPTW (*Orienteering Problem with Time Windows* [4]), dans lequel un véhicule doit maximiser le profit collecté en visitant un sous-ensemble de clients candidats, sachant (1) qu'il existe pour chaque client une fenêtre temporelle autorisée pour la visite, une durée de visite et un profit associé, et (2) qu'un temps de transition est requis pour passer d'un client à un autre.

2 Techniques mises en œuvre

Nous considérons comme point de départ une méthode de recherche basique qui part d'un plan vide pour le véhicule et qui tente d'insérer à chaque étape un nouveau client dans le plan. Cette phase d'insertion est itérée jusqu'à ce que tous les clients aient été considérés. Une phase de perturbation est ensuite appliquée, en retirant $x\%$ des clients du plan, suite à quoi des insertions itératives de clients sont à nouveau tentées. L'algorithme s'arrête lorsque la meilleure solution connue n'a pas été améliorée depuis un nombre fixé de phases de perturbation-insertion.

Au sein de cette méthode élémentaire, nous ajoutons des techniques d'apprentissage de conflits ainsi qu'une heuristique de choix du prochain client à insérer basée sur ces conflits :

— Lorsqu'une tentative d'insertion d'un client c échoue, nous calculons très rapidement une explication de cet échec d'insertion. Cette explication prend la forme d'un sous-ensemble de clients $\{c_1, \ldots, c_k\}$ minimal pour l'inclusion et qui contient des clients pour lesquels aucun ordre de visite réalisable n'a été trouvé dans le temps imparti pour raisonner.

— Pour l'heuristique de sélection d'un nouveau client à insérer à chaque étape, nous tentons de choisir un client c de manière à maximiser une évaluation de la récompense globale qu'il serait possible d'obtenir en sélectionnant c, étant donné tous les clients déjà sélectionnés et tous les conflits de sélection déjà mémorisés.

Plusieurs obstacles doivent toutefois être surmontés pour obtenir une approche efficace, avec (1) la question de la taille mémoire requise pour stocker tous les conflits rencontrés, (2) la question de la rapidité de la requête d'extraction d'un meilleur client en prenant en compte les conflits, (3) la question de la rapidité des opérations de modification de la base de conflits.

Pour surmonter ces obstacles, la sélection d'un client c est exprimée par une variable booléenne x_c, un conflit $\{c_1, \ldots, c_k\}$ est vu comme une clause $\neg x_{c_1} \vee \ldots \vee \neg x_{c_k}$, et l'ensemble des conflits est stocké sous la forme d'un OBDD (*Ordered Binary Decision Diagram* [3]), qui correspond à une structure de données pouvant être exponentiellement plus compacte qu'une table listant tous les conflits et qui nous permet de sélectionner en temps linéaire en la taille de l'OBDD un client "optimal" étant donné les conflits mémorisés et les profits associés aux clients. Par ailleurs, pour limiter l'impact d'opérations standards de réoptimisation de la taille d'un OBDD, les conflits trouvés par l'algorithme sont ajoutés par *batch* et non un par un.

3 Résultats obtenus

La méthode proposée a été implémentée en langage Python, avec pour la partie gestion des OBDD une utilisation de la librarie PyCUDD. Des expérimentations ont été réalisées sur des instances classiques d'OPTW (`https://www.mech.kuleuven.be/en/cib/op`). Beaucoup de meilleures bornes connues ainsi que des améliorations par rapport à la méthode d'ILS [5] ont été retrouvées, avec des temps de calcul compétitifs avec l'état de l'art (cf. table 1).

Instance	ILS+GRASP [4]	ILS [5]	Notre résultat	#conflits	#OBDD (noeuds)	CPU(s)
c107	**370**	360	**370**	7350	1789	5.59
r101	**198**	182	**198**	5720	1708	3.38
r104	**303**	297	299	15936	2129	9.72
r107	**299**	288	294	11661	2347	7.94
r109	**277**	276	**277**	15118	4645	9.55

TAB. 1 – Quelques résultats obtenus sur des instances classiques d'OPTW (tests réalisés sur un processeur Intel Core i5-10210U, 1.6 GHz, 16 GB de RAM)

Divers travaux futurs sont envisagés, notamment la réduction de la taille de la base de conflits via l'utilisation de langages de compilation autres que les OBDD ou l'approximation de la forme compilée. Un autre point concerne le traitement des incertitudes, avec comme motivation applicative des problèmes de sélection d'observations pour un satellite où des informations météo de dernière minute peuvent modifier les profits associés aux observations.

Références

[1] G. Audemard, J.-M. Lagniez, B. Mazure, and L. Sais. Integrating conflict driven clause learning to local search. In *Int. Workshop on Local Search Techniques in Constraint Satisfaction*, 2009.

[2] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, chapter 4 : Conflict Driven Clause Learning. IOS Press, 2009.

[3] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8) :677–691, 1986.

[4] W. Souffriau, P. Vansteenwegen, G. V. Berghe, and D. Van Oudheusden. The multi-constraint team orienteering problem with multiple time windows. *Transportation Science*, 47 :53–63, 2013.

[5] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12) :3281–3290, 2009.