Un GRASP×ELS simple et efficace pour le CARP

Philippe Lacomme [1], Christian Prins [2]

[1] Université Blaise Pascal, LIMOS (UMR 6158), F-63178 Aubière, France
placomme@isima.fr
[2] Université de Technologie de Troyes, ICD-LOSI (UMR 6281), F-10004 Troyes, France
christian.prins@utt.fr

***Mots-clés** : capacitated arc routing problem (CARP), evolutionary local search (ELS).*

1 Introduction

Le CARP est un problème de tournées NP-difficile rencontré par exemple en collecte de déchets urbains. Soit un graphe non-orienté $G = (V, E)$ avec n nœuds, dont un dépôt de véhicules de capacité Q, et m arêtes. Chaque arête $e = [i, j]$ est munie d'une demande q_e et d'un coût c_e. On a $t \leq m$ arêtes à traiter ou tâches (celles de demandes non nulles). Le but est de déterminer un ensemble de tournées de coût total minimal, dans lequel chaque tâche est traitée une seule fois, dans un des deux sens possibles. En pratique, chaque tournée peut être codée comme une liste de tâches (chacune ayant un sens de traversée), reliées implicitement par un chemin de coût minimal.

2 Métaheuristique de type GRASP × ELS

Nous proposons une métaheuristique hybride GRASP×ELS, c'est-à-dire un GRASP dont la recherche locale est remplacée par une recherche locale évolutionnaire (ELS), plus puissante. Cette métaheuristique alterne entre des solutions codées par des tours géants et des solutions complètes. Le GRASP effectue ng itérations. Chaque itération construit un tour géant de départ T avec une heuristique randomisée $HR(T)$, inspirée par l'heuristique de Clarke et Wright pour les problèmes de tournées sur nœuds. Ce tour est découpé optimalement en tournées (sous contrainte de la séquence) par une procédure $Split(T, S)$ (cf. référence [1]) puis la solution S est améliorée par une recherche locale $LS(S)$. La recherche locale effectue des déplacements d'une ou deux tâches, des échanges d'une ou deux tâches et des mouvements de type 2-OPT, dans une ou deux tournées. A chaque itération elle exécute le premier mouvement améliorant trouvé.

Chaque ELS effectue ni itérations. Chaque itération convertit la solution actuelle S en un tour géant T (procédure $Concat(S, T)$), en concaténant les tâches des tournées. Puis ne enfants sont générés. Chaque enfant est obtenu en prenant une copie T' de T, en la perturbant par des échanges aléatoires de tâches (procédure $Perturb(T', T'')$), en découpant T'' en tournées ($Split(T'', S'')$) puis en appliquant la recherche locale ($LS(S'')$). Si le meilleur enfant S' obtenu est meilleur que S, il met à jour S ($S \leftarrow S'$) sinon S est inchangée. On recommence ce processus à l'itération suivante. La meilleure solution $\bar{S}$ trouvée au cours des ELS successives est renvoyée à la fin.

La méthode est raffinée par un niveau de perturbation variant dynamiquement : le nombre d'échanges aléatoires de tâches est initialisé à une valeur minimale p_{min}. Si le meilleur enfant S' n'améliore pas la situation actuelle, il est incrémenté mais sans dépasser une valeur p_{max}. Sinon il est remis à p_{min}. Nous avons ajouté aussi un processus de recuit déterministe qui accepte le meilleur enfant s'il est au plus à α % du coût de la solution actuelle. Ce facteur de dégradation α est initialisé à une valeur α_0 au début de chaque ELS. Ensuite, il décroît linéairement de façon à s'annuler après un pourcentage δ des ni itérations de l'ELS. Dans les dernières itérations, l'ELS fonctionne donc de manière habituelle, en n'acceptant que des améliorations.

3 Evaluations numériques

Le GRASP×ELS est programmé en Visual C++ en trois versions V1, V2 et V3, testées sur un PC sous Windows 7 Professionnel équipé d'un processeur Xeon à 3.5 GHz (7000 Mflops). La V1 utilise $ng = 10$ itérations du GRASP, $ni = 100$ itérations par ELS, $ne = 10$ enfants par itération des ELS, une dégradation maximale dans le recuit $\alpha_0 = 20\%$, une annulation de la dégradation après $\delta = 67$ itérations (2/3 de ni) et des niveaux minimal et maximal de perturbation $p_{min} = 1$ et $p_{max} = 2$. La version V2 est identique sauf $ne = 40$ enfants.

Méthode	**MAENS**	**Ant-CARP**	**GRASP**	**HMA1**	**HMA2**	**GRASP×ELS**
Processeur	Xeon E5335 2 Ghz	Pentium III 1 Ghz	Intel Core 2.3 Ghz	AMD-OPT 4184 2.8 Ghz	AMD-OPT 4184 2.8 Ghz	XEON 3.5 GHz
Mflops	4000	1500	1591	7678	7678	7000
Facteur	0.57	0.21	0.22	1.09	1.09	1

TAB. 1 – Puissances relatives des machines utilisées.

Nous avons utilisé 5 ensembles d'instances de la littérature pour une comparaison avec les 5 meilleures méthodes publiées, en tenant compte (table 1) des puissances relatives des machines. Les 23 instances Gdb, les plus petites, ont 7 à 27 nœuds et 11 à 55 arêtes. Les 34 instances Val ont 24 à 50 nœuds et 34 à 97 arêtes. Les 24 instances Egl ont 77 à 140 nœuds et 98 à 198 arêtes, dont seules 51 à 190 sont à traiter. Les instances Luc-C ont 32 à 97 nœuds et 42 à 140 arêtes, dont 32 à 107 à traiter. Enfin, les Luc-E ont 26 à 97 nœuds et 35 à 142 arêtes, dont 28 à 107 à traiter. Les méthodes comparées sont l'algorithme mémétique MAENS de Tang *et al.* (2009), la méthode à colonie de fourmis Ant-CARP de Santos *et al.* (2010), le GRASP avec path-relinking d'Usberti *et al.* (2013) et les algorithmes HMA1 et HMA2 de Chen *et al.* (2016). Par manque de place nous renvoyons à [2] qui donne les références pour toutes ces instances et métaheuristiques.

Instances	**Gdb**		**Val**		**Egl**		**Luc-C**		**Luc-E**	
Critères	Ecart	Durée	Ecart	Durée	Ecart	Durée	Ecart	Durée	Ecart	Durée
ANT-CARP	0,037	/	0,295	/	0,522	/	3,422	/	0,29	/
GRASP	0,000	1,12	0,342	19,59	0,415	183,31	/	/	/	/
MAENS	0,000	/	0,336	/	0,422	/	3,294	/	0,14	/
HMA1	0,000	1,29	0,257	17,17	0,270	226,12	3,285	40,19	0,05	46,45
HMA2	/	/	0,232	41,99	0,220	734,76	3,259	59,09	0,01	127,03
GRASP×ELS V1	0,000	0,544	0,104	2,14	0,670	41,27	0,340	9,96	0,232	9,50
GRASP×ELS V2	x	x	0,061	3,99	0,527	82,34	0,332	13,88	0,252	13,99
GRASP×ELS V3	x	x	0,054	3,82	0.526	91,43	0,262	14,40	0,149	13,839

TAB. 2 – Résultats (/ : information non disponible, écarts moyens aux bornes inférieures en %, durées en secondes).

4 Résultats et conclusion

Le tableau 2 indique les résultats : la méthode proposée est beaucoup plus rapide que toutes les autres métaheuristiques (parfois 10 fois plus) et donne un écart moyen plus petit aux bornes inférieures. Les cases marquées "/" désignent des informations non disponibles car non fournies par les auteurs, tandis que les "x" indiquent que nous n'avons pas testé les versions V2 et V3 car la V1 donne des résultats optimaux. Nous comptons maintenant développer une version coopérative multi-thread pour améliorer encore plus les performances et traiter de plus grandes instances.

Références

[1] P. Lacomme, C. Prins, W. Ramdame-Chérif. Competitive memetic algorithms for arc routing problems. Annals of Operations Research, 13 : 159-195, 2004.

[2] Y. Chen, J.K. Hao, F. Glover. A hybrid metaheuristic approach for the capacitated arc routing problem. *European Journal of Operational Research*, 253: 25-39, 2016.