Coopération entre génération de colonnes avec tournées sans cycle et recherche locale appliquée au routage de véhicules

Alain Chabrier[1], **Émilie Danna**[12], **Claude Le Pape**[1]

[1] ILOG S.A.
9, rue de Verdun
94253 Gentilly Cédex

[2] Laboratoire d'Informatique d'Avignon
CRNS - FRE 2487
339, chemin des Meinajariès
Agroparc, BP 1228
84911 Avignon Cédex 9

email : {achabrier,edanna,clepape}@ilog.fr

Résumé

La recherche locale et la génération de colonnes sont deux méthodes privilégiées pour la résolution de problèmes de routage de véhicules, la première pour obtenir rapidement des bonnes solutions, la seconde pour obtenir des solutions optimales et des bornes inférieures. Nous proposons un schéma dans lequel ces deux méthodes coopèrent en échangeant différentes informations de manière à réduire l'écart entre bornes supérieures et bornes inférieures obtenues en un temps donné. De plus, contrairement à la plupart des références de la littérature, notre génération de colonnes s'appuie sur un algorithme de plus court chemin *élémentaire* avec contraintes de ressources, ce qui permet de produire de meilleures bornes inférieures. Sur les séries les plus difficiles de Solomon, en prenant pour critère la distance totale parcourue, ces innovations permettent d'améliorer la qualité des solutions générées en un temps donné et de résoudre exactement 17 instances qui étaient encore ouvertes.

Introduction

De nombreux problèmes d'optimisation rencontrés dans l'industrie ou les services dérivent du problème de routage de véhicules. Étant donnés :

- un ensemble de nœuds correspondant à des visites à effectuer,
- un ensemble de véhicules stationnés en un nœud donné (le dépôt) et devant retourner à ce nœud en fin de tournée,
- un distancier, précisant la distance entre tout couple de nœuds,
- la capacité de chaque véhicule et la capacité nécessaire à chaque visite,

il s'agit d'affecter une liste de visites à chaque véhicule de manière à minimiser la distance totale parcourue tout en respectant les contraintes de capacité. Dans la pratique, ce problème connaît de multiples variantes : minimisation non seulement de la distance parcourue mais aussi du nombre de véhicules, possibilité d'utiliser plusieurs dépôts, liens entre plusieurs visites permettant (par exemple) de charger des marchandises puis de les livrer, fenêtres de temps précisant les jours ou les heures de la journée durant lesquelles une visite est possible, etc.

Les méthodes les plus souvent employées pour résoudre ce type de problèmes sont les méthodes de recherche locale [14, 6, 10, 9]. Étant donné une solution ou un ensemble de solutions, des opérateurs définissant un « voisinage » sont utilisés pour modifier de proche en proche cette solution. Diverses méta-heuristiques (recuit simulé, recherche tabou, etc.) sont couramment utilisées pour guider la recherche. L'avantage indéniable de ces méthodes est qu'elles permettent d'obtenir rapidement des solutions de qualité. Elles présentent par contre deux inconvénients. Tout d'abord, elles ne fournissent pas de borne inférieure. L'utilisateur ne dispose donc d'aucun majorant de la différence entre une solution optimale (inconnue) et une solution obtenue en un temps de calcul donné. Il lui est dès lors difficile de décider s'il se satisfait de la meilleure solution disponible ou s'il doit allouer plus de ressources à la résolution. Par ailleurs, l'intégration de nouvelles contraintes à un algorithme de résolution par recherche locale n'est pas toujours facile : les bonnes propriétés des voisinages utilisés (connexité de l'ensemble de l'espace de recherche, nombre d'opérations à effectuer pour sortir d'un minimum local) peuvent être conservées ou au contraire détruites par l'addition de nouvelles contraintes.

Ce dernier inconvénient a amené différents chercheurs à intégrer des techniques de programmation par contraintes et des techniques de recherche locale [7, 15, 17, 12, 2, 4]. Une telle hybridation a permis de mieux répondre à l'existence de contraintes additionnelles et de réaliser des applications qui sinon n'auraient pas vu le jour. Néanmoins, la définition du voisinage reste au cœur de la méthode et il peut être difficile d'évaluer la qualité effective des solutions obtenues.

Parmi les méthodes exactes, la génération de colonnes a été récemment appliquée avec succès à une variante importante du problème de routage de véhicules, le VRPTW (Vehicle Routing Problem with Time Windows) [5, 13, 11]. Dans le cas du VRPTW, la méthode consiste à générer un certain nombre de tournées, à les sélectionner, chacune avec un poids compris entre 0 et 1, en résolvant dans le problème maître la relaxation continue d'un problème de couverture (*set covering*), puis à exploiter les informations issues de cette résolution pour générer de nouvelles tournées susceptibles de réduire le coût de la solution optimale du problème maître. Lorsqu'il n'existe plus de telles tournées, le coût de la solution optimale du problème maître est une borne inférieure pour le coût d'une solution optimale du problème de routage initial. Deux cas peuvent alors se produire : soit la solution optimale du problème maître est entière, *i.e.* chaque tournée est affectée d'un poids égal à 0 ou 1, auquel cas cette solution représente une solution optimale du problème de routage initial ; soit certaines tournées sont affectées d'un poids fractionnaire, auquel cas il est nécessaire d'appliquer une règle de branchement, c'est-à-dire de créer deux versions complémentaires du problème, dans lesquelles une contrainte et son contraire sont ajoutées. Le processus de génération de nouvelles tournées peut alors reprendre sur chacune de ces branches, en respectant les contraintes ajoutées, ce qui est connu sous le nom de *branch-and-price* [1].

La méthode de génération de colonnnes présente deux avantages principaux : elle fournit une borne inférieure, qui se trouve affinée au fur et à mesure que les règles de branchement sont appliquées ; elle permet de traiter les contraintes internes à une tournée au sein du sous-problème de génération de tournées et non

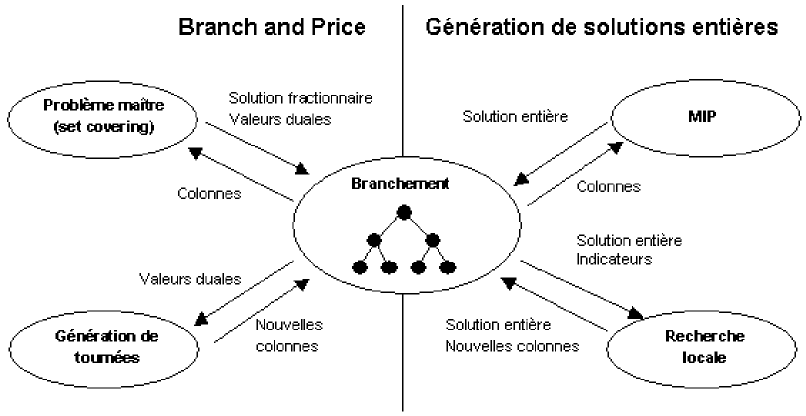

FIG. 1 – Schéma de résolution

au sein du problème global, ce qui facilite l'intégration de nombreuses contraintes additionnelles, surtout si l'algorithme utilisé pour générer des tournées est, comme dans les travaux d'hybridation précédemment cités, intégré dans un cadre de programmation par contraintes (voir aussi [19] pour une application à un problème de planification d'équipages). Par contre, elle ne permet pas d'obtenir rapidement de bonnes solutions. La solution optimale du problème maître est en effet rarement entière. Si l'objectif principal est d'obtenir de bonnes solutions rapidement, il est nécessaire de coupler des méthodes de calcul de bonnes solutions entières au processus de génération de colonnes.

Une manière simple de disposer à la fois de bonnes solutions et de bornes inférieures consisterait à appliquer séparément des méthodes de recherche locale et de génération de colonnes. Dans cet article, nous proposons une méthode dans laquelle recherche locale et génération de colonnes coopèrent de manière à réduire l'écart entre bornes supérieures et bornes inférieures obtenues en un temps donné.

1 Présentation générale de notre méthode

La figure 1 représente le processus de résolution que nous proposons. La partie gauche correspond à l'algorithme de *branch-and-price*. L'algorithme du simplexe est utilisé pour résoudre le problème maître de *set covering*. Si d_i représente la distance totale parcourue pour accomplir la tournée i et $V(i)$ l'ensemble des visites de cette tournée, ce problème consiste à minimiser $\Sigma_i d_i x_i$ sous les contraintes $0 \leq x_i \leq 1$ pour toute tournée i et $\Sigma_{i\ /\ j \in V(i)} x_i \geq 1$ pour toute visite j. La résolution de ce problème fournit une solution fractionnaire (les valeurs des variables x_i dans la solution optimale sont rarement toutes entières) et des valeurs duales π_j pour chaque contrainte de couverture.

Ces valeurs duales sont utilisées par un algorithme de plus court chemin avec contraintes de ressources qui résoud le sous-problème de génération de nouvelles tournées en fonction des résultats du problème maître. Il s'agit de générer une nouvelle tournée susceptible d'entrer en base, c'est-à-dire un chemin c de dépôt à dépôt de coût réduit $d_c - \sum_{j \in V(c)} \pi_j$ négatif. Cette tournée doit par ailleurs respecter les contraintes intrinsèques à la tournée :

- la somme des capacités requises pour les visites de la tournée doit être inférieure ou égale à la capacité du véhicule ;

– chaque visite de la tournée doit être effectuée pendant la fenêtre de temps qui lui est associée.

Deux versions de l'algorithme de plus court chemin sont disponibles : une version heuristique, très rapide mais susceptible de ne pas trouver de tournée de coût réduit négatif même s'il en existe ; une version complète, moins rapide mais qui permet, soit de trouver une tournée de coût réduit négatif, soit de conclure qu'il n'en existe pas. L'algorithme de plus court chemin est d'abord utilisé dans sa version heuristique, puis en cas d'échec dans sa version complète. Les nouvelles tournées sont données au problème maître et le processus itéré jusqu'à qu'il n'y ait plus de tournée de coût réduit négatif à ajouter.

Dans le cas où il n'y a plus de tournée de coût réduit négatif et où la solution du problème maître n'est pas entière, une règle de branchement est appliquée : un arc (i, j) de la tournée prise avec le coefficient le plus fractionnaire est sélectionné. Dans la branche de gauche, chaque tournée ne contient aucun des nœuds i et j, ou bien contient i et j et j doit être le successeur de i. Dans la branche de droite, l'arc (i, j) est interdit. Le processus de génération de colonnes peut alors reprendre sur chacune des deux branches ainsi créées.

Ce processus de *branch-and-price* est complété par deux algorithmes destinés à générer des solutions entières :

– un algorithme de résolution de problèmes linéaires en nombres entiers, auquel est alloué un temps limité pour résoudre le problème de couverture en nombres entiers. Cet algorithme reçoit en entrée les tournées prises en compte dans le problème maître et retourne la meilleure solution entière obtenue dans le temps alloué.
– un algorithme de recherche locale, qui résoud le problème de routage initial. Cet algorithme reçoit en entrée une solution initiale et des indicateurs issus du *branch-and-price* pour le guider dans l'espace de recherche. Il retourne une solution entière et de nouvelles tournées à intégrer au problème maître.

Les sections suivantes présentent de manière plus détaillée l'algorithme de plus court chemin et l'algorithme de recherche locale que nous avons utilisés.

2 Le sous-problème de génération de tournées

2.1 Problème du plus court chemin avec ressources et fenêtres de temps

Comme indiqué précédemment, le sous-problème consiste à obtenir de nouvelles tournées de coût réduit négatif. L'algorithme que nous utilisons résoud une version généralisée de ce problème.

Soit un chemin $c = (i_0 = d, i_1, ..., i_p = i, ..., d)$. Les ressources (temps, longueur et capacité) sont généralisées à L ressources indexées par l. D_i^l représente la quantité accumulée de la ressource l au nœud i. Des distances $d^l(i, j)$ définissent l'accumulation de la ressource l entre i et j. Des intervalles de valeurs admissibles $[a_i^l, b_i^l]$ sont définis pour chaque ressource et visite. L'augmentation de la ressource accumulée entre deux nœuds successifs i et j du chemin doit vérifier la contrainte sur les distances ainsi que la contrainte de fenêtre.

$$D_i^l + d^l(i, j) \leq D_j^l$$

$$a_i^l \leq D_i^l \leq b_i^l$$

Dans le cas du VRP, le coût du chemin est égal à l'accumulation de distance. Nous considérons le cas où le coût est une combinaison linéaire à coefficients positifs d'accumulation des ressources, *i.e.* $c(i,j) = \sum_{l=1}^{L} \lambda^l d^l(i,j)$. Le coût réduit du dépôt jusqu'au nœud i est obtenu à partir du coût et des valeurs duales π_j de chaque visite correspondant aux contraintes de couverture du problème maître.

$$rc_i = \sum_{u=0}^{p-1} \left(c(i_u, i_{u+1}) - \pi_{i_{u+1}} \right)$$

Le coût réduit du chemin complet doit être minimisé ou contraint à être négatif.

Théoriquement, les tournées ne doivent pas contenir de cycles, (*i.e.* un client ne peut être visité qu'une seule fois). Il est cependant de coutume de relâcher cette contrainte en vue de simplifier la résolution du sous-problème. Les tournées générées peuvent alors contenir des cycles. En effet, même si la solution du problème maître relâché, qui ne demande la couverture de chaque visite que par une tournée, peut contenir des tournées avec cycles, la solution entière optimale n'en contiendra pas. Ceci est dû au fait que chacune des ressources respecte l'inégalité triangulaire :

$$d^l(i,k) \leq d^l(i,j) + d^l(j,k), \; \forall \; l = 1...L$$

Donc, si la solution optimale du problème maître entier contenait une route avec cycles, couvrant ainsi deux fois la même visite, il serait possible de la modifier en éliminant une visite redondante, avec la certitude d'obtenir une autre solution valide et de coût inférieur ou égal, ce qui contredit l'optimalité de la solution originale.

En général, un algorithme à base d'étiquettes est utilisé pour résoudre ce sous-problème. Nous avons décidé d'utiliser un algorithme à base d'étiquettes que nous avons modifié pour résoudre le problème de plus court chemin *sans cycle* avec ressources et fenêtres de temps. En effet, en utilisant uniquement des routes sans cycle, la borne inférieure fournie par la génération de colonnes est supérieure ou égale à celle obtenue en autorisant les cycles.

2.1.1 Algorithme avec étiquettes

À un chemin partiel c du dépôt à une visite p est associée une étiquette $E_c^p = (rc_c, D_c^1, ..., D_c^L)$. L'algorithme consiste à trouver de nouveaux meilleurs chemins partiels prolongeant les chemins partiels existants. Quand ceci n'est plus possible, l'étiquette au nœud final de meilleur coût réduit correspond au meilleur chemin.

Afin de ne pas considérer tous les chemins partiels, un nouveau chemin partiel n'est conservé que s'il n'est *dominé* par aucun autre chemin partiel déjà existant. On dit qu'un chemin partiel c_1 arrivant en p domine un autre chemin partiel c_2 arrivant en p s'il n'est pas possible d'obtenir un chemin à partir de c_2 qui soit meilleur que n'importe quel chemin obtenu à partir de c_1. Dans le cas où les cycles sont autorisés, une condition suffisante est :

- c_1 et c_2 sont des chemins partiels du dépôt au même nœud p
- $rc_1 \leq rc_2$
- $D_1^l \leq D_2^l$ pour tout $l \in 1, ..., L$

En effet, pour toute prolongation de c_2 en un chemin valide c_2^*, il est possible de prolonger c_1 de la même manière pour obtenir un chemin c_1^* valide et de meilleur coût réduit que c_2^*. Le chemin partiel c_2 peut donc être éliminé.

Dans [8], un algorithme d'étiquetage complet est présenté, ainsi qu'une discussion sur les différentes implantations possibles (*reaching/pulling*, ...).

2.1.2 Algorithme modifié sans cycle

Pour interdire les cycles, nous avons modifié l'algorithme présenté précédemment en deux points.

Tout d'abord, un chemin partiel arrivant en p peut être prolongé jusqu'à n si et seulement si n n'est pas déjà visité par le chemin partiel. C'est-à-dire, si on appelle $V(c)$ l'ensemble des nœuds du chemin partiel c, on ne peut pas prolonger c à n si nous avons déjà $n \in V(c)$.

Ensuite, la relation de dominance est étendue pour rester compatible avec cette restriction. En effet, la relation de dominance précédemment définie ne permet plus d'assurer que c_2 ne peut pas être mieux prolongé que c_1, car c_2 peut maintenant être prolongé à des nœuds par lesquels c_1 est déjà passé et ne peut donc pas être prolongé. Nous prenons maintenant comme condition suffisante pour qu'un chemin partiel c_1 domine un autre chemin partiel c_2 qu'il respecte les conditions de dominance précédemment définies, et que l'ensemble des visites de c_1 soit inclus dans l'ensemble des visites de c_2, *i.e.* $V(c_1) \subseteq V(c_2)$.

L'inconvénient majeur de cette nouvelle relation de dominance est l'augmentation importante du nombre d'étiquettes qui peuvent être créées à un même nœud. Afin de réduire cet impact, nous utilisons deux sous-problèmes, un premier utilisant uniquement la première modification (et qui n'est donc pas complet car des étiquettes non-dominées sont éliminées), et un autre utilisant les deux modifications. Il est évident que la version complète est plus lente que la version heuristique. Nous n'utilisons donc la version complète que lorsque la version heuristique échoue à obtenir de nouvelles routes de coût réduit négatif.

De plus, diverses modifications, ou bien de l'algorithme ou tout simplement du sous-problème, rendent la résolution plus rapide.

2.2 Améliorations de la règle de dominance sans cycle

Si le chemin partiel c_2 ne contient pas certaines visites du chemin partiel c_1, l'étiquette lui correspondant sera systématiquement conservée, même si chacune des valeurs est respectivement plus mauvaise. La raison en est que c_2 pourrait devenir intéressant en passant par l'une de ces visites. Cette condition suffisante n'est bien sûr pas nécessaire. Il est possible de l'améliorer en estimant le gain de coût réduit qui pourrait être obtenu en passant par certaines de ces visites.

Soient deux chemins partiels c_1 et c_2 arrivant au même nœud p. Si $V(c_1)\backslash V(c_2) = \emptyset$ et que c_1 domine c_2 selon la règle simple, il n'y a pas de difficulté particulière. Il s'agit de la nouvelle règle de dominance. Remarquons également qu'un élément n de $V(c_1)\backslash V(c_2)$ peut ne pas être considéré s'il ne peut pas être ajouté à c_2. En effet, s'il existe une ressource l telle que $D_2^l + d^l(p,n) > b_n^l$, il sera impossible d'ajouter n. Nous utilisons ici la propriété d'inégalité triangulaire.

Considérons maintenant le cas où $V(c_1)\backslash V(c_2) = \{n\}$:

1. minoration de l'amélioration avec π_n. Quelle que soit c_2^* une prolongation du chemin partiel c_2, il sera possible de prolonger c_1 en c_1^* de la même manière, sauf pour n. Si nous notons i et j les prédécesseur et successeur de n dans c_2^*, nous aurons alors :

   $$rc_2^* - rc_1^* = c(i,n) + c(n,j) - c(i,j) - \pi_n + rc_2 - rc_1$$

   Donc c_2^* est meilleur que c_1^* si et seulement si :

   $$c(i,n) + c(n,j) - c(i,j) - \pi_n + rc_2 - rc_1 < 0$$

   Or, en raison de l'inégalité triangulaire, $c(i,n) + c(n,j) - c(i,j) \geq 0$, une condition suffisante pour éliminer c_2 est donc :

   $$-\pi_n \geq rc_1 - rc_2$$

2. minoration de l'amélioration avec $\pi_n - minCout_n$ Il s'agit ici de minorer $c(i,n) + c(n,j) - c(i,j)$ par :

$$minCout_n = min_{i \in pred(n), j \in succ(n)} (c(i,n) + c(n,j) - c(i,j))$$

où $pred(n)$ et $succ(n)$ sont respectivement l'ensemble des prédécesseurs et des successeurs possibles du nœud n. Cette quantité peut être calculée une fois pour toutes au début. Le chemin c_2 peut donc être éliminé dès que :

$$minCout_n - \pi_n \geq rc_1 - rc_2$$

Ces modifications sont généralisables au cas où $Card(V(c_1)\backslash V(c_2)) > 1$. Mais dans la pratique, des minorants intéressants sont difficiles à calculer. Nous nous sommes donc restreints au cas où $Card(V(c_1)\backslash V(c_2)) \leq 2$. Bien que plus difficile à calculer, un minorant intéressant sur le coût dans le cas de deux éléments peut également être obtenu dès l'initialisation.

2.3 Autres accélérations

Tout d'abord nous réduisons le graphe utilisé par des raisonnements simples sur les fenêtres de temps. De plus, comme le propose [16], nous effectuons une réduction supplémentaire du graphe en utilisant des raisonnements sur les coûts réduits.

Nous utilisons d'autres types d'améliorations permettant d'accélérer de manière heuristique la résolution du sous-problème. Une résolution finale non heuristique nous permet toujours d'assurer la non-existence d'une tournée ayant un coût réduit favorable. Comme le propose [13], nous stoppons la résolution du sous-problème quand un nombre minimum de chemins complets et d'étiquettes ont été obtenus. Une autre réduction simple consiste à éliminer certains arcs sortant d'un nœud en fonction de leur *longueur* respectivement à l'une des ressources. Différents critères d'élimination peuvent être utilisés, par exemple $d^l(i,j) \geq percent * max_{k \in succ(i)} d^l(i,k)$ ou $d^l(i,j) \geq (1 + percent) * min_{k \in succ(i)} d^l(i,k)$. Nous incrémentons peu à peu la part de graphe initial traité.

2.4 Apports de la génération de tournées sans cycle

Quand il n'est plus possible de générer de nouvelle tournée de coût réduit favorable, la solution du problème maître avec contraintes d'intégrité relâchées nous donne une borne inférieure pour la branche en cours d'exploration. Dans le cas du nœud racine, nous obtenons une borne inférieure globale. En utilisant un sous-problème qui ne génère que des tournées sans cycle, nous résolvons un problème plus contraint, et les bornes obtenues sont théoriquement supérieures ou égales. Aux autres nœuds, les bornes inférieures locales obtenues nous permettent de ne pas explorer certaines branches, et donc de réduire le temps total.

Le tableau 1 compare, pour les séries R1 et RC1 de Solomon [18], notre borne inférieure globale LB obtenue au noeud racine avec les bornes présentées dans [5], notées $LB(1)$ et $LB(2)$, correspondant respectivement à un sous-problème n'éliminant que les cycles de type $i-j-i$ et au même sous-problème complété par une génération de coupes d'élimination partielle de sous-tours. Notre borne inférieure est effectivement meilleure que celle obtenue avec un sous-problème avec cycles. En pratique, notre borne permet de réduire l'écart à la solution optimale (quand elle est connue) de 17 % contre 47% pour $LB(2)$. En termes de qualité de bornes, $LB(2)$ est donc clairement meilleure sur cette série.

Instances	nbVisites	$\frac{LB-LB(1)}{LB(1)}$	$\frac{LB(2)-LB(1)}{LB(1)}$	$\frac{LB-LB(1)}{OPT-LB(1)}$	$\frac{LB(2)-LB(1)}{OPT-LB(1)}$
R1	50	0,38%	0,33%	23,56%	36,86%
R1	100	0,32%	0,15%	13,95%	15,59%
RC1	50	0,55%	8,89%	11,62%	53,03%
RC1	100	0,55%	1,82%	3,06%	38,77%

TAB. 1 – Bornes inférieures sur la série 1

Le tableau 2 compare de même sur certaines instances de la série 2 notre borne LB avec la borne $LB(1)$ que nous avons calculée, [5] ne donnant pas de résultats pour cette série. Les instances en caractères gras sont celles que nous avons fermées. LB est en général bien meilleure que $LB(1)$, ce qui suggère que notre borne a été décisive pour fermer des instances que [5] n'a pas fermées. L'utilisation de LB est clairement suffisante pour fermer de nouvelles instances : elle permet de résoudre de nombreuses instances sans branchement car, au nœud racine, la solution du problème maître relâché est entière. Même quand ce n'est pas le cas, l'amélioration permet de réduire le nombre de nœuds nécessaires pour obtenir la solution optimale et faire la preuve d'optimalité. Il serait intéressant d'obtenir les bornes LB(2) sur ces instances afin de confirmer ou d'infirmer notre sentiment que LB est meilleure que $LB(2)$ sur cette série.

Instance	nbVisites	LB	LB(1)	OPT	$\frac{LB-LB(1)}{LB(1)}$	$\frac{LB-LB(1)}{OPT-LB(1)}$
R204	**25**	**350,47**	**337,025**	**355,0**	**3,99%**	**74,80%**
R208	**25**	**328,20**	**321,752**	**328,2**	**2,00%**	**100,0%**
R201	50	791,90	788,625	791,9	0,42%	100,00%
R202	50	698,50	694,162	698,5	0,62%	100,00%
R203	**50**	**598,58**	**592,383**	**605,3**	**1,05%**	**48,00%**
R205	**50**	**682,85**	**668,524**	**690,1**	**2,14%**	**66,40%**
R206	**50**	**626,34**	**613,445**	**632,4**	**2,10%**	**68,05%**
R209	**50**	**599,83**	**585,939**	**600,6**	**2,37%**	**94,75%**
R210	**50**	**636,10**	**627,385**	**645,6**	**1,39%**	**47,85%**
R201	100	1140,30	1139,746	1143,2	0,05%	16,04%
RC203	**25**	**326,90**	**220,182**	**326,9**	**48,47%**	**100,0%**
RC204	**25**	**299,70**	**191,221**	**299,7**	**56,73%**	**100,0%**
RC208	**25**	**269,10**	**163,009**	**269,1**	**65,08%**	**100,0%**
RC201	50	684,80	678,867	684,8	0,87%	100,0%
RC202	**50**	**613,60**	**516,619**	**613,6**	**18,77%**	**100,0%**
RC203	**50**	**555,30**	**421,146**	**555,3**	**31,85%**	**100,0%**
RC205	**50**	**630,20**	**567,970**	**630,2**	**10,96%**	**100,0%**
RC206	**50**	**610,00**	**447,305**	**610,0**	**36,37%**	**100,0%**
RC207	**50**	**558,60**	**395,725**	**558,6**	**41,16%**	**100,0%**
RC201	100	1255,94	1244,456	1261,8	0,92%	66,22%
RC202	**100**	**1088,08**	**1012,616**	**1092,3**	**7,45%**	**94,70%**
RC205	**100**	**1147,61**	**1063,922**	**1154,0**	**7,87%**	**92,90%**

TAB. 2 – Bornes inférieures sur la série 2

3 L'obtention des solutions entières

3.1 Description de l'algorithme de recherche locale

L'algorithme de recherche locale est celui décrit dans [17], qui est également utilisé dans [2]. C'est un algorithme de recherche dans un grand voisinage (LNS : *Large Neighbourhood Search*). L'algorithme part d'une solution entière, relaxe certaines visites, c'est-à-dire les retire de cette solution, puis les insère à nouveau pour obtenir une solution de meilleur coût. Si une meilleure solution est trouvée, alors la taille du voisinage, c'est-à-dire le nombre de visites relaxées, est réduite à 1. Sinon, au bout d'un certain nombre d'itérations infructueuses, le voisinage est agrandi. Trois points sont importants : quelle est la solution initiale, comment sont choisies les visites à relaxer, comment ces visites sont réinsérées.

3.1.1 La solution initiale

La solution initiale est la meilleure solution entière trouvée jusqu'à ce point. Cette solution a été trouvée par le MIP, ou par recherche locale ou parce que la solution du problème maître était spontanément entière.

3.1.2 La sélection des visites à relaxer

En suivant l'algorithme de [17], des visites de référence sont d'abord choisies, puis l'ensemble des visites à relaxer est constitué d'un certain nombre de visites proches de ces visites de référence. En effet, le gain attendu en enlevant puis en insérant à nouveau des visites proches ensemble est supérieur au gain obtenu en les enlevant puis en les réinsérant indépendamment les unes des autres.

Deux différences nous distinguent de [17] : le choix des visites de référence et la relation de proximité entre visites. Dans [17], les visites de référence sont choisies au hasard. Nous sélectionnons les visites de référence de deux manières :

1. si l'on vient de trouver une solution entière avec la génération de colonnes, alors la probabilité d'être une visite de référence est proportionnelle au nombre d'arcs supprimés par les règles de branchement dans lesquelles intervient la visite. Ceci permet de pallier l'imperfection éventuelle des règles de branchement du *branch-and-price* en explorant une partie de l'espace de recherche qui n'a pas été explorée récemment par la génération de colonnes.
2. dans les autres cas (la majorité), la probabilité d'être une visite de référence est proportionnelle au nombre de routes qui couvrent cette visite et qui sont prises avec un coefficient strictement positif dans la dernière solution du problème maître relâché. En effet, le fractionnement de la couverture d'une visite dans le problème maître relâché entre de très nombreuses routes est une indication de la difficulté du *branch-and-price* pour placer cette visite.

Dans [17], la relation de proximité utilisée pour sélectionner le sous-ensemble des visites à relaxer ne fait intervenir qu'une seule visite de référence (pas toujours la même). Or, il semble intéressant d'utiliser toutes les visites de référence pour ordonner les visites parce que l'on souhaite que la visite à déplacer soit proche au moins de deux visites de référence pour qu'il y ait au moins deux possibilités intéressantes pour son déplacement. Concrètement, on associe à chaque visite i le critère $R(i) = \sum_{k \in Ref} d(i,k)$ où Ref est l'ensemble des visites de référence et $d(i,k)$ la distance entre la visite i et la visite k. Les visites qui minimisent ce critère sont choisies en priorité pour être déplacées. Contrairement à [17], le fait que deux visites soient présentes dans la même route n'est pas pris en compte pour la relation de proximité parce que ce critère sert essentiellement à relaxer ensemble toutes les visites d'une même route afin de les réinsérer dans les autre routes et ainsi de diminuer le nombre de véhicules. Or, nous ne cherchons pas à minimiser le nombre de véhicules et les solutions optimales en termes de distance parcourue ne sont pas toujours optimales en termes de nombre de véhicules.

3.1.3 Heuristique de réinsertion

Comme dans [17], les visites relaxées sont réinsérées par LDS (*Limited Discrepancy Search*), en utilisant l'heuristique du point d'insertion le plus éloigné : la visite réinsérée en premier est celle qui a le plus petit pire coût d'insertion. Au cours de

	série R2		série RC2	
	%contribution	%réussite	%contribution	%réussite
problème maître spontanément entier	21,11	0,56	20,62	0,66
MIP	13,97	4,36	20,62	26,81
Visites doubles	3,10	16,94	4,86	25,00
Total LNS	61,80	9,84	54,54	14,92
LNS (fract)	45,03	7.39	36,01	10,54
LNS (branch)	16,77	91.52	18,53	77,94

TAB. 3 – Qui trouve les solutions entières dans la coopération entre génération de colonnes et recherche locale ?

quelques tests numériques, cette heuristique s'est révélée meilleure que l'heuristique de plus grand regret [3] et que l'heuristique de la visite la plus proche.

3.2 Les bénéfices de la coopération

Si les informations fournies par la génération de colonnes concernant les visites à relaxer en priorité sont utiles, alors l'algorithme de recherche locale doit trouver une solution améliorante plus rapidement et/ou de coût inférieur s'il utilise ces informations que s'il ne les utilise pas. Nous avons donc isolé toutes les solutions initiales sur lesquelles l'algorithme de recherche locale a été lancé pendant la coopération entre génération de colonnes et recherche locale, puis exécuté l'algorithme de recherche locale à partir de chacune de ces solutions pendant dix minutes, d'une part en utilisant ces informations, d'autre part sans les utiliser (visites de référence choisies aléatoirement). Les résultats avec/sans informations ne sont pas significativement différents. Cependant, le schéma entier de coopération donne des résultats légèrement meilleurs en utilisant les informations de la génération de colonnes pour guider la recherche locale : il est respectivement à 1,22% de la meilleure borne inférieure connue sur la série 1 avec informations contre 1,46% sans informations, et à 2,52% de la meilleure borne supérieure connue sur la série 2 avec informations contre 3,05% sans informations.

Qui trouve les solutions entières au cours de la coopération ? Sur les séries R2 et RC2 des instances de Solomon, le tableau 3 donne pour chaque composante de la coopération le nombre de cas où elle réussit à trouver une meilleure solution entière, rapporté au nombre total de solutions entières trouvées (*%contribution*) et rapporté au nombre de fois où cette méthode a été appelée (*%réussite*). *Visites doubles* désigne une heuristique gloutonne très rapide qui transforme une solution où certaines visites sont couvertes plusieurs fois en une solution où chaque visite est couverte exactement une fois. *LNS (fract)* et *(branch)* désignent l'algorithme de recherche locale utilisé avec les informations respectivement sur le fractionnement de routes dans la solution du problème maître relâché et sur les visites intervenant dans les règles de branchement. Ces chiffres sont imparfaits dans la mesure où ils ne tiennent pas compte de la qualité des solutions trouvées par chacune des méthodes.

Le LNS trouve plus de la moitié des solutions : les bons résultats du schéma de coopération viennent premièrement naturellement du partage du temps alloué à chacune des composantes. Deuxièmement, le taux de réussite de LNS-branch est très important, ce qui montre la complémentarité entre la partie programmation linéaire (MIP et problème maître) de la coopération et la recherche locale. En effet, d'une part le MIP (et problème maître) permet d'aller explorer une autre partie de l'espace de recherche alors que la recherche locale pure met souvent longtemps à s'éloigner d'un optimum local. De plus, contrairement à une stratégie de redémarrage qui vise à éviter de rester bloqué dans un optimum local en relançant la recherche locale périodiquement à partir d'une nouvelle solution initiale, le MIP permet de

	C1	R1	RC1	C2	R2	RC2
coopération	0,00	1,58	2,03	2,57	1,91	3,30
LNS avec redémarrages	0,00	1,68	2,24	6,88	3,38	3,18

TAB. 4 – La diversification par le MIP est meilleure que la diversification aléatoire

diversifier la recherche locale en lui donnant un nouveau point de départ de coût systématiquement inférieur à celui du dernier optimum local. Enfin, cette nouvelle solution initiale est très facile à optimiser par recherche locale. En effet, le taux de réussite de LNS-branch est très supérieur au taux de réussite de LNS-fract.

Pour évaluer l'efficacité de la diversification de la recherche locale par le MIP, nous donnons dans le tableau 4 les gaps (en %, définis précisément dans la section suivante) obtenus par notre schéma de coopération et le LNS diversifié aléatoirement : si une solution améliorante ne peut pas être trouvée dans un voisinage de taille 15, alors la recherche locale est relancée à partir d'une nouvelle solution initiale construite de manière à être assez « orthogonale » au dernier optimum local.

4 Résultats

Nous nous sommes intéressés aux instances de Solomon [18] qui constituent une batterie de problèmes standard sur laquelle sont très souvent testées les méthodes exactes et heuristiques de résolution du VRPTW. Nous avons adopté les conventions adoptées par la plupart des méthodes exactes : minimisation de la distance uniquement sans se préoccuper du nombre de véhicules ; les distances et les temps de parcours sont déterminés par la distance euclidienne entre les paires de coordonnées (x, y), arrondie au dixième inférieur. Il existe deux séries d'instances : la série 1 pour laquelle les fenêtres de temps sont étroites, la série 2 pour laquelle les fenêtres de temps sont plus larges. La série 1 est plus facile à résoudre parce que plus contrainte : les fenêtres de temps serrées limitent notamment le nombre de successeurs possibles pour une visite. C'est pourquoi la littérature s'est concentrée jusqu'à présent sur la série 1, mais nous donnons des résultats sur les deux séries et nous nous sommes particulièrement intéressés à la série 2 parce qu'elle contient encore de nombreuses instances ouvertes. Les instances de test sont en outre divisées en trois classes : la classe « C » dans laquelle les visites sont divisées en plusieurs groupes compacts d'un point de vue géographique, la classe « R » dans laquelle les coordonnées des visites sont distribuées aléatoirement et la classe « RC » où une partie des visites sont groupées géographiquement et les autres sont placées aléatoirement. Chacune des instances est un problème à 100 visites duquel sont extraits deux problèmes réduits, l'un à 25 visites, l'autre à 50 visites.

Le tableau 5 donne les résultats sur l'ensemble des instances des séries 1 et 2 des trois méthodes que nous souhaitons comparer :

1. notre schéma de coopération entre génération de colonnes, MIP et recherche locale.
2. la même méthode de génération de colonnes et de MIP que dans la coopération, mais utilisée sans recherche locale.
3. la même méthode de recherche locale que celle utilisée dans la coopération, mais utilisée seule.

Tous nos résultats ont été obtenus avec une limite d'une heure de calcul par instance, sur un Pentium IV cadencé à 1,5 Ghz avec 256 Mo de mémoire et avec ILOG JNI CPLEX 7.5, ILOG JSOLVER 1.0 et la JVM d'IBM pour Linux version 1.3.0.

La colonne *gap* donne pour chaque méthode de résolution la distance relative moyenne entre la meilleure borne inférieure connue (dans la littérature ou que nous

Série	Visites	Nombre d'instances	gencol		LNS		coop		
			gap (%)	OPT	gap (%)	OPT	gap (%)	OPT	gain (%)
C1	25	9	0,00	9 (9)	0,00	9	0,00	9 (9)	0,00
	50	9	0,00	9 (8)	0,00	9	0,00	9 (8)	0,00
	100	9	0,00	9 (8)	0,00	9	0,00	9 (8)	0,00
R1	25	12	0,03	12 (12)	0,00	12	0,00	12 (12)	0,00
	50	12	0,08	11 (10)	0,17	9	0,19	10 (10)	-0,13
	100	12	2,62	5 (5)	1,64	0	1,58	5 (3)	-0,38
RC1	25	8	0,00	8 (8)	0,00	8	0,00	8 (8)	0,00
	50	8	0,80	4 (3)	0,00	8	0,00	8 (2)	0,00
	100	8	5,78	0 (0)	1,53	2	2,03	1 (0)	-0,02
C2	25	8	0,00	8 (8)	0,00	8	0,00	8 (8)	0,00
	50	8	1,02	6 (4)	0,00	8	0,00	8 (0)	0,00
	100	8	20,21	0 (0)	1,69	6	2,57	5 (0)	0,00
R2	25	11	0,00	11 (11)	0,00	11	0,00	11 (11)	0,00
	50	11	3,17	6 (4)	0,78	4	1,37	5 (4)	-0,22
	100	11	5,75	1 (1)	4,50	0	1,91	1 (0)	2,59
RC2	25	8	1,18	7 (7)	0,00	8	0,00	8 (7)	0,00
	50	8	2,18	5 (5)	0,00	4	0,33	6 (6)	-0,33
	100	8	3,15	2 (1)	8,08	0	3,30	2 (0)	0,60

TAB. 5 – Comparaison de la génération de colonnes pure (*gencol*), de la recherche locale pure (*LNS*) et de notre schéma de coopération (*coop*) sur l'ensemble des instances de Solomon.

avons nous-mêmes améliorée) et la solution obtenue par cette méthode. Pour certaines instances, en particulier dans la série 2, aucune borne inférieure non-triviale n'est connue. Donc pour toute la série 2, nous avons utilisé à la place la meilleure borne supérieure connue. La colonne *gap* pour cette série n'est donc pas un majorant de la distance à la solution optimale. La colonne OPT donne le nombre d'instances pour lesquelles l'optimum a été atteint, et entre parenthèses pour lesquelles l'optimum a été prouvé (ce qui est possible uniquement pour la génération de colonnes et notre algorithme coopératif). La dernière colonne donne le gain de notre schéma de coopération par rapport à un partage équitable de temps entre génération de colonnes et LNS, sans communication entre les deux composantes.

La figure 2 montre l'évolution du coût des solutions (le *gap* calculé comme précédemment) au cours du temps (au plus 3600 secondes). La coopération permet clairement d'obtenir très rapidement de bonnes solutions sur toutes les séries. Sur les séries C1, R1, RC1 et C2, ses performances sont très proches de celles du LNS. Sur la série RC2, elles sont très proches de celles de la génération de colonnes. Enfin, sur la série R2, la coopération permet d'obtenir des solutions meilleures que celles obtenues par chacune de ses composantes. La série R2-100 est d'ailleurs la seule sur laquelle la coopération donne des résultats significativement meilleurs qu'un simple partage de temps entre génération de colonnes et LNS.

La tableau 6 donne les résultats de notre algorithme de génération de colonnes pure et de notre schéma de cooopération sur les 17 instances précédemment ouvertes dont nous avons prouvé l'optimum : la valeur de la solution optimale en termes de distance parcourue (colonnes OPT et OPT_f avec la distance non-arrondie), le nombre de véhicules pour cette solution, la différence relative en % entre la borne inférieure au nœud racine et la solution optimale (colonne *gap*), et pour chacune des méthodes le nombre de nœuds explorés dans l'arbre du *branch-and-price* (colonne N), le nombre de sous-problèmes résolus (colonne SP) et le temps en secondes pour atteindre l'optimum T_{opt} et pour faire de surcroît la preuve d'optimalité T_{total}. Les instances marquées par (*) et (**) sont celles pour lesquelles nous trouvons une solution optimale meilleure que celles données respectivement par [11] et par [13] et [11]. Sur ces instances, nos solutions ont été validées par Jesper Larsen.

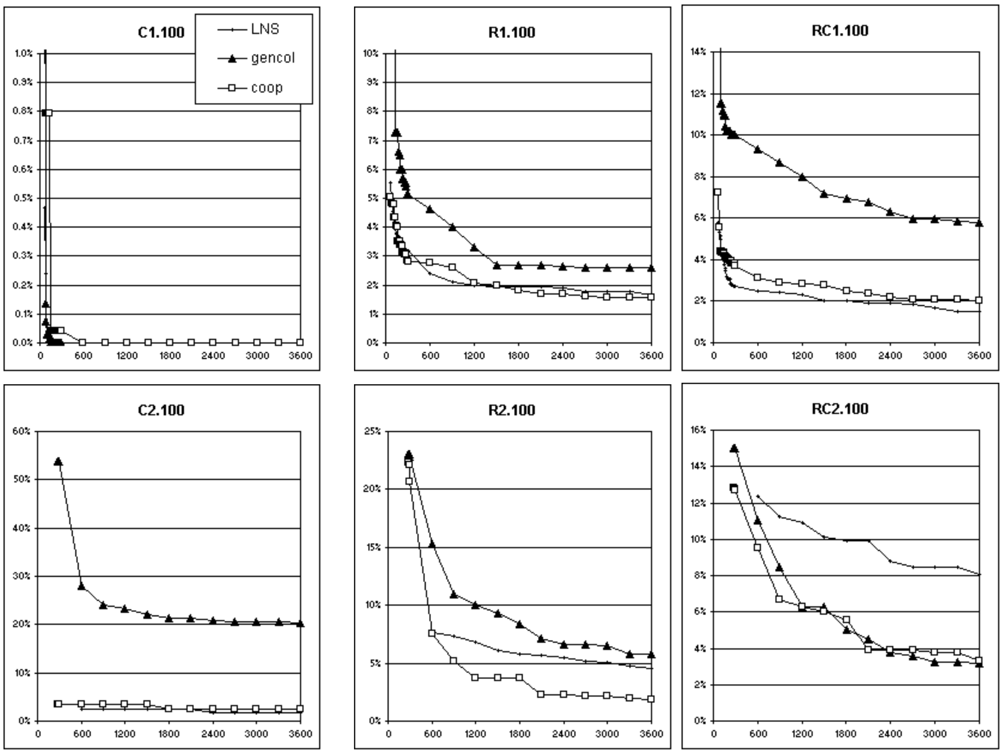

FIG. 2 – Évolution du coût des solutions en fonction du temps

Conclusion

Dans cet article, nous avons proposé une méthode dans laquelle génération de colonnes et recherche locale coopèrent. Par rapport à la littérature existante, cette méthode est originale de deux points de vue.

1. L'utilisation d'un algorithme de calcul de plus court chemin *sans cycle*, ce qui améliore les bornes inférieures. Ceci nous a permis de fermer un grand nombre d'instances des séries R2 et RC2 de Solomon.
2. La coopération entre la génération de colonnes et la recherche locale : la recherche locale améliore facilement les solutions entières issues de la génération de colonnes ; les solutions issues de la génération de colonnes permettent de relancer la recherche locale à partir de nouvelles solutions, meilleures que les optima locaux que la recherche locale n'arrivait pas à améliorer ; les informations de la génération de colonnes guident et accélèrent la recherche locale ; les solutions issues de la recherche locale permettent une diversification des colonnes prises en compte dans le problème maître. Globalement, comme on pouvait l'espérer, la coopération est plus robuste que chacune de ses composantes. De plus, notre schéma fournit sur une série d'instances des résultats meilleurs que chacune de ses composantes. Cependant, le temps nécessaire à la preuve d'optimalité après avoir trouvé la solution optimale est nécessairement plus long pour la coopération que pour la génération de colonnes pure car l'algorithme de recherche locale continue à être lancé alors qu'il ne peut pas trouver de meilleure solution.

					gencol				coop			
Instance	OPT	OPT_f	V	Gap	N	SP	T_{opt}	T_{total}	N	SP	T_{opt}	T_{total}
R203.50	605,3	607,649	5	1,11	15	432	117,8	3320,9	13	495	31,4	7229,5
R204.25	355,0	355,891	2	1,27	17	365	88,4	171,6	21	398	138,9	268,0
R205.50**	690,1	692,397	4	1,05	95	1551	301,4	531,0	51	888	362,4	371,4
R206.50	632,4	634,876	4	0,96	67	1319	2972,8	4656,1	95	1781	793,3	4804,7
R208.25*	328,2	329,333	1	0,00	1	64	705,7	741,5	1	119	3,2	444,3
R209.50	600,6	603,102	4	0,13	5	144	120,8	195,4	3	156	160,5	285,1
R210.50	645,6	647,871	4	1,47	977	16183	25151,1	65638,6	821	14888	44761,6	54465,1
RC202.50	613,6	615,037	5	0,00	1	70	5,3	13,0	1	120	12,0	24,0
RC202.100	1092,3	1095,643	8	0,39	39	1861	18053,8	19636,5	9	803	6141,2	6339,1
RC203.25*	326,9	327,693	3	0,00	1	65	4,0	5,1	1	61	8,9	11,0
RC203.50	555,3	556,496	4	0,00	1	315	4479,4	4481,5	1	450	2316,0	2947,2
RC204.25	299,7	300,234	3	0,00	1	58	1,95	13,0	1	122	8,8	25,4
RC205.50*	630,2	631,980	5	0,00	1	82	10,3	10,6	1	121	13,4	27,1
RC205.100	1154,0	1157,550	7	0,55	71	2706	3131,6	15151,7	85	3150	1227,7	16169,1
RC206.50	610,0	611,677	5	0,00	1	61	8,6	9,4	1	124	10,4	19,1
RC207.50	558,6	559,906	4	0,00	1	107	66,0	71,1	1	220	73,9	89,4
RC208.25	269,1		2	0,00	1	185	32239,3	33785,3	1	291	78,7	42401,2

TAB. 6 – Les instances que nous avons fermées

Plusieurs améliorations, de l'algorithme de plus court chemin, de ses modes d'utilisation au sein de la génération de colonnes et de la coopération entre génération de colonnes et recherche locale peuvent aussi être envisagées :

- stockage des étiquettes d'un sous-problème à l'autre : l'arrêt prématuré de l'algorithme de plus court chemin entraîne souvent à l'itération suivante la re-création d'un grand nombre d'étiquettes, qui pourraient donc être stockées puis réactualisées avec les nouvelles valeurs duales.
- niveaux de dominance : actuellement, la principale réduction heuristique consiste à ne pas utiliser la relation de dominance modifée pour les routes sans cycle. Il est possible d'introduire des intermédiaires entre les versions heuristique et complète du sous-problème en limitant avec un paramètre la taille des chemins partiels à partir de laquelle la règle de dominance complète est appliquée.
- exploitation des informations fournies par la génération de colonnes à la recherche locale : il est sans doute possible d'exploiter non seulement le nombre de colonnes assurant la couverture d'une visite dans la solution du problème maître relâché, mais aussi la distribution des coefficients dont sont affectées ces colonnes dans cette solution.

Les deux premières propositions ont été essayées mais les résultats ne permettent pas encore de conclure si elles apportent véritablement des améliorations.

Pour confirmer l'intérêt de notre schéma de coopération, il serait par ailleurs intéressant d'identifier les caractéristiques des problèmes pour lesquels ce schéma est efficace et de l'appliquer à des problèmes de routage de véhicules comprenant une plus grande variété de contraintes, et plus généralement à d'autres problèmes sur lesquels la génération de colonnes et la recherche locale donnent de bons résultats.

Remerciements

Nous souhaitons remercier Christopher Beck pour son aide sur comment comparer le plus justement possible les résultats du LNS avec et sans informations fournies par la génération de colonnes, Jesper Larsen pour avoir vérifié la validité de nos solutions sur les instances où nous ne trouvions pas la même solution optimale, Paul Shaw pour ses éclaircissements sur [17], toute l'équipe de Philippe Michelon au Laboratoire d'Informatique d'Avignon pour ses discussions intéressantes, en particulier autour du LNS, et Javier Lafuente pour ses discussions sur la résolution du

problème du plus court chemin contraint.

Références

[1] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, et P.H. Vance. « Branch-and-price : column generation for solving huge integer programs ». *Operations Research*, 46 : 316–329, 1998.

[2] R. Bent et P. Van Hentenryck. « A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows ». Rapport technique CS-01-06, Brown University, septembre 2001.

[3] Y. Caseau et F. Laburthe. « Solving Small TSPs with Constraints ». in *International Conference on Logic Programming*, pp. 316–330, 1997.

[4] Y. Caseau, F. Laburthe, C. Le Pape, et B. Rottembourg. « Combining Local and Global Search in a Constraint Programming Environment ». *Knowledge Engineering Review*, 16(1) : 41–68, 2001.

[5] W. Cook et J.L. Rich. « A parallel cutting-plane algorithm for the vehicle routing problem with time windows ». Rapport technique TR99-04, Department of Computational and Applied Mathematics, Rice University, mai 1999.

[6] J.-F. Cordeau, G. Laporte, et A. Mercier. « A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows ». *Journal of the Operational Research Society*, 52 : 928–936, 2001.

[7] B. De Backer, V. Furnon, P. Shaw, Ph. Kilby, et P. Prosser. « Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics ». *Journal of Heuristics*, 6 : 501–523, 2000.

[8] M. Desrochers. « An Algorithm for the Shortest Path Problem with Resource Constraints ». Rapport technique G-88-27, Les cahiers du GERAD, septembre 1988.

[9] L.M. Gambardella, É. Taillard, et G. Agazzi. MACS-VRPTW : A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. in David Corne, Marco Dorigo, et Fred Glover, éds., *New Ideas in Optimization*, pp. 63–76. McGraw-Hill, London, 1999.

[10] J. Homberger et H. Gehring. « Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows ». *INFOR*, 37 : 297–318, 1999.

[11] B. Kallehauge, J. Larsen, et O.B.G. Madsen. « Lagrangean duality and non-differentiable optimization applied on routing with time windows - experimental results ». Rapport technique IMM-TR-2001-9, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, août 2001.

[12] Ph. Kilby, P. Prosser, et P. Shaw. « A Comparison of Traditional and Constraint-Based Heuristic Methods on Vehicle Routing Problems with Side Constraints ». *Constraints*, 5(4) : 389–414, 2000.

[13] J. Larsen. « Vehicle Routing with Time Windows - Finding optimal solutions efficiently ». http ://citeseer.nj.nec.com/larsen99vehicle.html, 1999.

[14] Y. Rochat et E. Taillard. « Probabilistic diversification and intensification in local search for vehicle routing ». *Journal of Heuristics*, 1 : 147–167, 1995.

[15] L.-M. Rousseau, M. Gendrau, et G. Pesant. « Using Constraint-Based Operators in a Variable Neighborhood Search Framework to Solve the Vehicle Routing Problem with Time Windows ». *CP-AI-OR*, 1999.

[16] L.-M. Rousseau, M. Gendreau, et G. Pesant. « Solving small VRPTWs with constraint programming based column generation ». in *CP-AI-OR*, pp. 333–344, 2002.

[17] P. Shaw. « Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems ». in *Principles and Practice of Constraint Programming*, pp. 417–431, 1998.

[18] M.M. Solomon. « Algorithms for the vehicle routing and scheduling problem with time window constraints ». *Operations Research*, 35 : 254–265, 1987.

[19] T.H. Yunes, A.V. Moura, et C.C. de Souza. « A Hybrid Approach for Solving Large Scale Crew Scheduling Problems ». in *Practical Aspects of Declarative Languages*, pp. 293–307, 2000.