Université de Montréal

Gestion de flotte avec fenêtres horaires : approches de résolution mixtes utilisant la programmation par contraintes

par

Louis-Martin Rousseau

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures en vue de l'obtention du grade de Docteur ès sciences (Ph.D.) en informatique.

Octobre 2002

©, LOUIS-MARTIN ROUSSEAU, 2002

QA
76
U54
2003
v. 002

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée :

Gestion de flotte avec fenêtres horaires :
approches de résolution mixtes utilisant
la programmation par contraintes

présenté par :
Louis-Martin Rousseau

a été évalué par un jury composé des personnes suivantes :

Professeur Jean-Yves Potvin
président-rapporteur

Professeur Michel Gendreau
directeur de recherche

Professeur Gilles Pesant
co-directeur de recherche

Professeur Jacques Ferland
membre du jury

Professeur Philippe Michelon
Examinateur externe

Résumé

La présente thèse porte sur l'application de la programmation par contraintes au domaine des problèmes de gestion de flotte de véhicules avec contraintes de temps. Ces problèmes de transport, d'une importance économique capitale, sont étudiés depuis plusieurs années tant par le milieu académique qu'industriel. Il existe plusieurs variantes de ce problème, mais toutes ont en commun le même objectif : desservir un ensemble de clients à l'aide d'un ensemble de routes en respectant des contraintes horaires restreignant les moments où chaque client peut être servi. Toutefois, les problèmes réels sont de plus en plus difficiles à résoudre, et ce, en raison du nombre grandissant de contraintes complexes à satisfaire.

Heureusement, l'utilisation de la programmation par contraintes, un paradigme d'optimisation expressif et flexible, permet de modéliser facilement la majorité des contraintes rencontrées dans les problèmes réels. En combinant cette approche à diverses techniques issues de la recherche opérationnelle on peut obtenir ainsi des méthodes efficaces et polyvalentes pour faire face aux problèmes de transport les plus complexes.

Cette thèse s'articule en trois volets qui abordent différentes facettes du problème de gestion de flotte avec contraintes de temps. Le premier volet étudiera la résolution de problèmes statiques de gestion de flotte à l'aide de méthodes intégrant la programmation par contraintes et la recherche locale, de façon à obtenir de bons résultats pour des problèmes de grandes tailles. Le volet deuxième considérera la résolution de manière exacte des même problèmes en proposant une intégration de la programmation par contraintes à une méthode de génération de colonnes. Finalement le troisième volet se penchera sur la gestion de flotte en temps réel et plus particulièrement lorsque des contraintes complexes rendent difficile la résolution avec des méthodes traditionnelles.

Mots clés : Gestion de flotte, tournées de véhicules avec fenêtres de temps, programmation par contraintes, métaheuristiques, génération de colonnes, méthode hybride, temps réel, contraintes de synchronisation.

Abstract

This thesis looks upon the application of constraint programming to solve fleet management problems under time constraints. These transportation problems, of great economic importance, have been the subject of considerable research from both academia and industry. There exist numerous variants of these problems but all have the common objective of servicing a set of customers with a set of routes, while respecting time considerations at each visit. However real life problems are becoming harder to solve since the number of complex constraints to satisfy is constantly growing.

Fortunately using constraint programming, a flexible and expressive optimization paradigm, allows easy modeling of most constraints encountered in real life problems. By combining this approach with known operations research techniques it is possible to devise efficient and flexible solution methods, that can handle most complex transportation problems.

The thesis is structured around three major themes, dealing with different aspects of time constrained fleet management problems. The first theme is the study of static fleet management problems with the combination of constraint programming and local search, in order to obtain good results on large size problems. The second will consider solving the same problems with an exact method, here a hybrid of constraint programming and column generation. Finally the real time fleet management problem will be considered, but more precisely a variant that is very hard to solve using only traditional methods.

Key words : fleet management, vehicle routing with time windows, constraint programming, metaheuristics, column generation, hybrid method, real time, synchronization constraints.

Table des matières

Liste des tableaux

Table des figures

Remerciements

S'il y a trois ans j'ai décidé de ne pas compléter ma maîtrise et passer directement au doctorat, c'est d'abord grâce à Michel Gendreau et Gilles Pesant, mes directeurs de recherche. Ils ont su par leurs conseils, leur expérience et leur passion rendre chacune des cinq années que nous avons passées ensemble plaisante et stimulante. Je tiens à les remercier particulièrement de m'avoir permis de voyager et de rencontrer plusieurs chercheurs étrangers avec qui j'entretiens aujourd'hui des liens particuliers ; c'est dans ces voyages que j'ai souvent puisé la motivation de poursuivre une carrière académique.

Je remercie aussi tous les membres présents et passés du Centre de Recherche sur les Transports (Guy Bordeleau, Daniel Charbonneau, Lucie-Nathalie Cournoyer, Martine Gemme, Pierre Girard, Lucie L'Heureux, Luc Rocheleau, Clairette Simard, Claudine St-Pierre, Josée Vignola) pour leur compréhension, leur collaboration et leur compétence.

Merci à ma famille, Lucie, Jean-Marc et Isabelle, de m'avoir donner confiance en moi ; une confiance qui est plus que nécessaire lorsque les résultats se font attendre. Merci particulièrement à mon père pour sa grande disponibilité et nos nombreuses conversions sur mon sujet de thèse qui firent débloquer mes recherches à plusieurs reprises.

Finalement merci à mon amour, Anne, qui me rend si heureux. Elle fut ma lumière dans le noir, littéralement, et sans elle il certain que je n'aurais pas le plaisir de rédiger ces lignes aujourd'hui.

Chapitre 1

Introduction

1.1 Motivation

Les problèmes de confection de tournées de véhicules occupent une place centrale dans la gestion de la distribution et ils ont une importance économique majeure. Il est estimé que les entreprises dépensent en moyenne 10% de leurs revenus à des fins de distribution. Quelques études de cas, documentées dans *Interfaces*, démontrent l'importance de la part des revenus consacrés à la distribution : 7,5% chez Dupont(3500 clients), 15% chez Air Products and Chemicals (500 clients), 10,7% chez Kraft(150-220 clients), etc. Du point de vue scientifique, l'étude des problèmes de tournées de véhicules a contribué de façon importante à l'avancement des domaines de l'optimisation et de l'algorithmique. Depuis plus de 40 ans la communauté scientifique tente de résoudre efficacement ces problèmes complexes ; elle dispose aujourd'hui d'un arsenal de méthodes très efficaces.

Au cours des dernières années, les nouvelles technologies ont engendré un nouveau type de problèmes de transport : la répartition de flotte en temps réel. Celui-ci constitue une importante part des problèmes de transport rencontrés par les entreprises d'aujourd'hui, qu'elles soient publiques, parapubliques ou privées. Les problèmes de gestion des véhicules d'urgence, de transport adapté, de gestion de services de réparation et de courrier rapide en sont tous des exemples bien connus.

Les méthodes de recherche opérationnelle développées au fil des ans présentent certaines lacunes; si elles s'avèrent très efficaces lorsqu'elles traitent des problèmes abstraits et théoriques, elles le sont moins avec les problèmes réels de l'industrie. Ceci s'explique par le fait que les problèmes concrets de tournées de véhicules ou de répartition de flotte sont des problèmes sujets à des contraintes particulières, qu'il est difficile de modéliser avec les méthodes traditionnelles. La grande variété de contraintes auxquelles on peut faire face dans un problème réel rend ardu le développement d'algorithmes réutilisables.

La programmation par contraintes est un paradigme émergeant qui permet de modéliser facilement les contraintes les plus complexes. Ses puissants mécanismes de propagation et d'exploration en font un outil de choix en optimisation combinatoire. De plus, cette approche permet la séparation entre la modélisation du problème et les méthodes de résolution. On peut donc réutiliser facilement les algorithmes en ne modifiant que le modèle, rendant ainsi les méthodes de résolution facilement réutilisables d'un problème à l'autre. Par contre, ces méthodes de résolution se révèlent souvent trop lentes lorsqu'on s'attaque à des problèmes de répartition de flotte de taille réaliste.

Le domaine de recherche traitant des méthodes hybrides est actuellement en pleine évolution. L'approche proposée consiste donc en une hybridation de méthodes découlant de la programmation par contraintes et des techniques d'optimisation plus classiques de la recherche opérationnelle. Il est possible de combiner les puissants outils de propagation et de gestion des contraintes tant à des heuristiques connues de recherche locale qu'à des méthodes exactes, et ce, de façon à profiter des avantages complémentaires des deux méthodes.

La programmation par contraintes propose un langage de haut niveau permettant d'exprimer facilement la majorité des contraintes inhérentes à un problème. De plus, elle offre des outils qui permettent de filtrer bon nombre de solutions non-réalisables sans les examiner explicitement. La recherche opérationnelle s'est, de son coté, penchée sur l'aspect relatif à l'optimisation du problème. Plusieurs techniques ont été développées pour optimiser des problèmes dont les contraintes ont une forme particulière (linéaire, convexe, en nombre entier, etc.). La force de la recherche opérationnelle, contrairement à celle de la programmation par contraintes, est moins le filtrage des solutions non-réalisables que celui des solutions non-optimales. L'intégration des deux paradigmes se présente donc comme un défi intéressant qui pourrait déboucher sur une génération de méthodes de résolution.

1.2 Organisation de la thèse

Le document est présenté de la façon suivante : la section 1.3 donne une description du paradigme de la programmation par contraintes et la manière dont elle peut être utilisée pour résoudre des problèmes d'optimisation combinatoire. Y est incluse également une description et une formulation du problème de confection de tournées de véhicules avec fenêtres de temps. Des détails concernant l'implantation de certaines contraintes y sont présentés.

Le chapitre 2 passe en revue la littérature pertinente concernant les combinaisons de programmation par contraintes et de recherche opérationnelle qui permettent de résoudre les problèmes de gestion de flotte. Cette section traitera de la littérature abordant les méthodes heuristiques de résolution, les algorithmes exacts et la problématique associée au contexte du temps réel.

Le chapitre 3 détaillera le premier volet du programme de recherche, soit l'intégration de la programmation par contraintes à des méthodes de recherche locale. Des opérateurs utilisant ce paradigme pour effectuer des améliorations locales à une solution réalisable y seront présentés. Ils seront utilisés pour construire une méthode complète de résolution incluant notamment des phases de construction, amélioration, diversification et post-optimisation. Les résultats obtenus lors de tests sur des problèmes de référence démontreront clairement l'efficacité de la combinaison proposée.

Le chapitre 4 abordera la résolution de manière exacte du problème de confection de tournées de véhicules avec fenêtres de temps. L'approche choisie est celle d'une hybridation de la programmation par contraintes avec une méthode de décomposition de Dantzig-Wolfe et d'une résolution par génération de colonnes. La programmation par contraintes sera utilisée afin de résoudre le sous-problème associé à la génération de colonnes, de

façon à bénéficier de sa flexibilité et de son expressivité.

Le chapitre 5 se penchera sur l'utilisation de la programmation par contraintes dans le cadre de problèmes de gestion de flotte en temps réel. Un problème complexe de transport dynamique sera présenté afin de démontrer la puissance de modélisation et de résolution des méthodes hybrides proposées. Afin de permettre la comparaison avec d'autres techniques de résolution, une section sera consacrée aux détails permettant de construire un ensemble de problèmes tests à partir de problèmes bien connus de la littérature.

1.3 Préliminaires

1.3.1 Programmation par contraintes

La dernière décennie a vu naître un nouveau paradigme issu des domaines de l'informatique et de l'intelligence artificielle : la programmation par contraintes. Cette approche qui a fait ses preuves dans les domaines de l'ordonnancement, de la planification et des transports ([22], [21], [80], [82]) a récemment fait l'objet d'un livre [56] ainsi que d'un article de survol [47]. Bien que les premières recherches aient été effectuées à la fin des années 70 il aura fallu attendre dix ans pour que cette technique connaisse ses premiers succès grâce à la programmation logique [52]. Jaffar et Maher [46] définissent alors le paradigme de la programmation logique par contraintes (Constraint Logic Programming : CLP) qui peut être spécialisé et définit sur plusieurs domaines (les nombres réels, les domaines finis, etc). Toutefois plus récemment certains langages de programmation par contraintes se sont séparés de la programmation logique (ie ILOG Solver [66, 44]) tout en maintenant l'expressivité des contraintes ; c'est pourquoi on utilise maintenant simplement le terme programmation par contraintes (Constraint Programming : CP).

L'approche de la CP pour résoudre des problèmes combinatoires est de les modéliser par un ensemble de variables prenant leur valeur dans un ensemble fini d'entiers et liées par un ensemble de contraintes mathématiques ou symboliques. L'efficacité de ce paradigme repose sur de puissants algorithmes de propagation de contraintes qui éliminent du domaine des variables les valeurs qui engendrent des solutions irréalisables. Si la propagation de contraintes n'est pas suffisante à elle seule pour identifier une solution réalisable, une recherche arborescente est entreprise afin de réduire davantage le domaine des variables définissant le problème. à chaque noeud de l'arbre de recherche une variable est d'abord choisie selon une certaine politique (statique ou dynamique) puis fixée à une des valeurs possibles de son domaine. Une solution est identifiée lorsque le domaine de chaque variable ne contient qu'une seule valeur alors qu'une impasse est détectée lorsque le domaine d'une variable devient vide. Il est généralement possible de guider la recherche de solutions réalisables en incorporant de l'information sur la nature du problème dans les politiques de sélection de variables et de valeurs.

Il est facile d'étendre cette méthode à la solution de problèmes d'optimisation combinatoire, c'est à dire l'identification de la solution réalisable qui minimise ou maximise une certaine fonction. Lorsqu'une feuille de l'arbre de recherche contient une solution réalisable au problème combinatoire à résoudre, on modifie le problème en y ajoutant une contrainte supplémentaire stipulant que la valeur de l'objectif doit être inférieure à la valeur de l'objectif de la dernière solution identifiée. Ainsi on obtient des solutions dont la qualité est sans cesse croissante jusqu'à ce qu'on puisse prouver que la dernière solution identifiée est optimale.

1.3.2 Problème de tournées de véhicules

Le problème de confection de tournées de véhicule (Vehicle Routing Problem : VRP) se définit comme suit : soit un ensemble de clients ayant chacun une demande fixe pour un produit, un ensemble de véhicules et un dépôt. Le problème consiste à construire un ensemble de tournées débutant et se terminant au dépôt et parcourant le minimum de distance de sorte que chaque client soit visité par exactement un véhicule. De plus, on doit généralement considérer des contraintes additionnelles telles une capacité et/ou une durée maximale pour chaque tournée sans quoi la solution ne comporterait qu'un seul véhicule et ce problème serait celui du voyageur de commerce (Traveling Salesman Problem : TSP).

Il existe plusieurs variantes de ce problème ; l'une d'elles impose des contraintes temporelles sur chaque visite, communément appelées fenêtres de temps (Vehicle Routing Problem with Time Windows : VRPTW). Ces contraintes stipulent que chaque client doit être visité pendant ses heures d'ouverture ; s'il est permis d'attendre lorsqu'on arrive trop tôt, les retards sont, eux, interdits. Une variante de ce problème propose de permettre les arrivées tardives mais de pénaliser le coût de la solution en fonction des retards engendrés ; on qualifiera de "souples" ces fenêtres de temps alors que les fenêtres horaires qu'on doit respecter sont dites "dures" ou "rigides". Dans le cadre de ce projet, seules les fenêtres rigides seront étudiées.

Le problème de confection de tournées de véhicules avec fenêtres de temps illustré en 1.1 est un problème NP-difficile. En fait, même la construction d'une solution réalisable lorsque la flotte de véhicules est limitée est en soi un problème NP-complet (Savelsbergh [75]).

Formulation du problème

En programmation par contraintes, le problème est typiquement modélisé avec des variables représentant les successeurs de chaque client. Le domaine

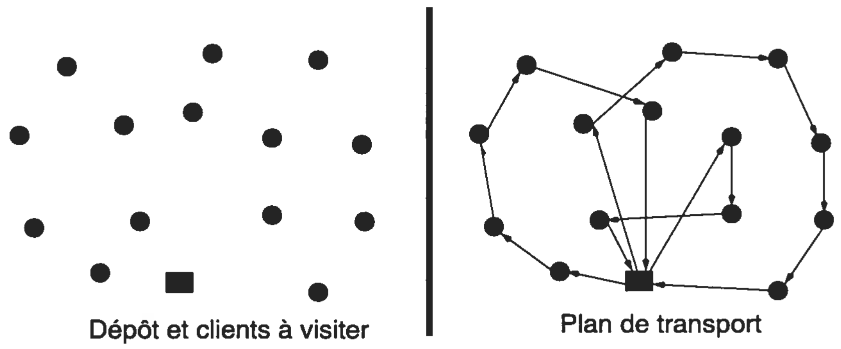


FIG. 1.1 – Problème VRPTW

associé à une variable successeur d'un noeud i contient donc l'ensemble de tous les noeuds qui peuvent être visités directement après i. Des variables à domaine fini sont également utilisées pour modéliser le temps de passage et la capacité accumulés lors de la visite de chaque noeud. Pour obtenir une solution au problème il est nécessaire de fixer les variables successeurs tout en s'assurant que toutes les contraintes sont respectées. Pour chaque véhicule on ajoute deux copies du dépôt, l'une comme point de départ et l'autre comme point d'arrivée.

Paramètres

N	L'ensemble des clients.
I	L'ensemble des dépôts initiaux
F	L'ensemble des dépôts finaux
d_{ij}	La distance du client i au client j.
t_{ij}	Le temps de déplacement du client i au client j.
a_i, b_i	Les bornes de la fenêtre horaire du client i.
s_i	Le temps de service du client i.
l_i	La demande en produit du client i.
k	La capacité des véhicules.

Variables

$S_i \in N \cup F$	$\forall i \in N \cup I$	Successeur du client i.
$T_i \in [a_i, b_i]$	$\forall i \in N \cup I \cup F$	Temps de passage au client i.
$L_i \in [0, k]$	$\forall i \in N \cup I \cup F$	Charge du véhicule après visite du client i.

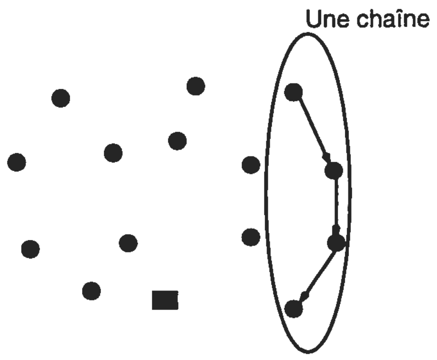


FIG. 1.2 – Illustration d'une chaîne

Contraintes

$AllDiff(S)$		Conservation du flot.
$S_i \neq i$	$\forall i \in I \cup N$	Boucles simples interdites.
$S_i = j \Rightarrow T_i + s_i + t_{ij} \leq T_j$	$\forall i \in I \cup N$	Contraintes de temps.
$S_i = j \Rightarrow L_i + l_i = L_j$	$\forall i \in I \cup N$	Contraintes de capacité.

Objectif

$\min \sum_{i \in N \cup I} d_{iS_i}$	On minimise la distance entre i et son sucesseur.

La contrainte *AllDiff* est généralement utilisée afin de forcer la conservation du flot dans le graphe associé à une solution. La modélisation avec des variables représentant les successeurs assure déjà que chaque noeud n'aura qu'un seul arc sortant ; en interdisant que deux variables prennent la même valeur, on s'assure aussi que chaque noeud n'aura qu'un seul arc entrant. Cette contrainte, qui repose sur un algorithme de couplage dans un graphe bipartie et qui maintient la cohérence d'arc de façon incrémentale, est détaillée dans [71].

Les contraintes qui imposent le respect des fenêtres de temps et de la capacité interdisent la création de sous-tournées. Toutefois cette manière de procéder s'avère peu efficace puisque les sous-tournées ne sont détectées que lorsqu'elles ont été crées et entraînent ainsi des échecs et retours arrières inutiles. Une contrainte dédiée spécialement à l'élimination des sous-tournées fut donc créée afin de prévenir leur formation. La contrainte *NoSubTour* maintient le noeud initial et final de chaque chaîne (une chaîne est un ensemble de noeud liés ensemble, illustré à la figure 1.2). Lorsque deux chaînes sont jointes (*i.e.* lorsqu'une variable est fixée et qu'un nouvel arc est introduit) les noeuds initiaux et finaux sont mis à jour et le nouveau noeud initial est retiré du domaine de la variable successeur du nouveau noeud final ; ceci élimine par le fait même toute possibilité de création de sous-tournées.

L'ajout de contraintes redondantes, comme $NoSubTour$, est une pratique courante en programmation par contraintes. Ces contraintes ne modifient en rien l'espace des solutions, mais elles permettent d'éliminer un nombre important de noeuds en élaguant des branches complètes de l'arbre de recherche. Certaines contraintes, qui furent développées dans le cadre de travaux concernant le problème du voyageur de commerce avec fenêtre de temps (TSPTW), furent donc ajoutées au présent modèle. Ces contraintes effectuent du filtrage notamment au niveau des fenêtres de temps afin que le maximum d'information disponible dans le domaine des variables non fixées soit utilisées pour accélérer la recherche. Le détail de ces contraintes et de $NoSubTour$ sont explicités dans [62].

Chapitre 2

Revue de littérature

Ce chapitre passe en revue la littérature pertinente au projet proposé. Les deux premières sections sont consacrées aux méthodes permettant de résoudre et d'identifier de bonnes solutions aux problèmes de confection de tournées de véhicules avec fenêtre de temps. La troisième section traite du problème de répartition de flotte en temps réel.

2.1 CP et méthodes heuristiques

Les méthodes heuristiques ont été développées afin de permettre d'identifier rapidement de bonnes solutions à des problèmes de taille raisonnable. Les problèmes industriels sont souvent de trop grandes tailles pour être résolus de manière exacte, et même lorsque cela est possible, le temps nécessaire à l'obtention d'une solution optimale est habituellement prohibitif. La recherche s'est donc orientée vers des méthodes pouvant fournir de bons résultats en un court laps de temps. Il existe un très grand nombre d'algorithmes et de méthodes s'attaquant au VRPTW (voir l'excellente revue de Braysy [3]). Cette section vise à décrire les différents principes guidant la recherche de solutions à l'aide de la programmation par contraintes et de méthodes heuristiques.

Quelques implantations de méthodes hybrides ont déjà démontré leur efficacité à résoudre le problème VRPTW. De Backer et Furnon [14] ont proposé d'utiliser la programmation par contraintes pour valider des solutions générées heuristiquement. Leurs travaux ont démontré que la validation a *posteriori* des solutions n'est efficace que si certaines contraintes simples sont intégrées directement dans les heuristiques afin de filtrer plus rapidement certaines solutions irréalisables.

Leurs travaux, ainsi que ceux de Shaw, Kilby et Prosser ([49],[15]), ont de plus permis l'implantation d'une libraire de plusieurs métaheuristiques connues (la recherche avec Tabou et la recherche locale guidée) qui travaillent conjointement avec un moteur de programmation par contraintes. Cette librairie [45] de fonctions permet d'identifier de bonnes solutions au VRPTW et allie la rapidité des heuristiques utilisées à la flexibilité de la programmation par contraintes. Caseau et Laburthe ont également travaillé à l'intégration de la CP et des métaheuristiques pour résoudre le VRPTW. Dans [7]

ils utilisent un algorithme de CP résolvant le problème du commis voyageur avec fenêtre de temps pour optimiser individuellement chacune des tournées du problème. Par la suite, dans [8] ils proposent des opérateurs simples d'insertion (ou de retrait) utilisant la programmation par contraintes pour ajouter (ou enlever) des clients à une tournée. Ils utilisent ensuite ces opérateurs pour construire automatiquement, à l'aide d'une technique d'apprentissage, une métaheuristique qui sera spécialement adaptée au problème à résoudre. Leurs recherches ont donc mené à des méthodes complexes pour résoudre le VRPTW. Elles sont basées sur des opérations simples effectuées en CP, ce qui permet encore une fois d'allier rapidité et flexibilité.

Parallèlement, Shaw [76] propose la recherche à voisinage étendue (Large Neighborhood Search : LNS) afin de résoudre le problème en ré-optimisant une déconstruction du problème de confection de tournées avec fenêtres de temps. la méthode, similaire au "shuffling", est itérative et effectue successivement les opérations suivantes. Elle retire d'abord de la solution courante un ensemble de clients possédant des caractéristiques communes ; ensuite, à l'aide de la programmation par contraintes, elle résout le problème qui consiste à les réinsérer de manière à améliorer la solution.

Pesant et Gendreau [60] et [61] ont toutefois été les premiers à proposer d'utiliser la programmation par contraintes pour explorer le voisinage défini par un opérateur. L'évaluation systématique des voisins d'une solution est donc remplacée par un problème d'optimisation qui est résolu à l'aide de la CP. Cette méthode a l'avantage de permettre de tirer profit des mécanismes de propagation et de filtrage de la programmation par contraintes et d'ainsi réduire considérablement le nombre de voisins qui devront être évalués. Pesant *et al* . ont appliqué cette méthode au problème du voyageur de commerce avec fenêtre de temps en modélisant le voisinage de l'opérateur GENI (introduit par Gendreau *et al.* dans [37]) comme un problème de CP [63].

C'est de cette dernière méthode que naîtra le premier volet du projet proposé, où il s'agira d'appliquer ces principes aux problèmes de confection de tournées de véhicules avec fenêtres horaires.

2.2 CP et méthodes exactes

Il existe plusieurs méthodes de recherche opérationnelle qui peuvent résoudre de façon exacte le problème de confection de tournées de véhicules avec fenêtres de temps. La grande majorité des méthodes exactes entrent dans la catégorie des méthodes dites de décomposition ; les méthodes de décomposition lagrangienne et celles de décomposition de Dantzig-Wolfe ont obtenu les meilleurs résultats. Une description complète des différentes méthodes est donnée dans la revue proposée par Cordeau *et al.* [12]

Traditionnellement, la méthodologie utilisée pour résoudre des problèmes d'optimisation en programmation par contraintes est la séparation et l'évaluation progressive (Branch and Bound : BnB). Ce qui veut dire que lorsqu'une solution réalisable est identifiée, une contrainte stipulant que la prochaine solution devra être de moindre coût est ajoutée. On peut parfois élaguer d'importantes sections de l'arbre de recherche lorsque la borne inférieure associée à ces sections est supérieure à la meilleure solution réalisable identifiée. Cet algorithme est dit "exact" parce qu'il permet de prouver que la meilleure solution trouvée est la solution optimale du problème, et ce, même si on n'évalue pas explicitement toutes les solutions possible. Toutefois, cette méthodologie ne peut être appliquée directement au VRPTW (exception faite des problèmes jouets) parce que celui-ci estgénéralement de trop grande taille.

2.2.1 Applications au TSPTW

La programmation par contraintes a toutefois été appliquée au problème du voyageur de commerce avec fenêtres de temps (Traveling Salesman Problem with Time Windows : TSPTW), une restriction du VRPTW où l'on ne dispose que d'un véhicule. Les équipes de Caseau et Laburthe [5] et celle de Pesant *et al.* [62] ont implanté avec succès des algorithmes exacts permettant de résoudre le TSPTW. Ces méthodes utilisaient la programmation par contraintes pour modéliser et résoudre le problème en entier. Des algorithmes sophistiqués ont été développés pour guider la recherche, effectuer un filtrage efficace et calculer des bornes inférieures. Toutefois, aucun algorithme de recherche opérationnelle n'a été utilisé.

Quelques années plus tard Focacci, Lodi et Milano [27] [28] ont proposé de résoudre le TSPTW en incorporant à la CP des techniques classiques de recherche opérationnelle. Ils ont donné le nom de *contraintes d'optimisation* à ces techniques. L'idée de Focacci *et al.* fut d'adapter les techniques de fixation de variables avec coûts réduits (*reduced cost fixing*) au contexte de la programmation par contraintes. A chaque noeud de l'arbre de recherche, ils calculent d'abord la valeur optimale du problème d'affectation sous-jacent au TSPTW, constituant une borne inférieure au problème originel. L'algorithme polynomial qu'ils utilisent pour résoudre les problèmes d'affectation fournit également les coûts réduits associés à chaque arc. On peut interpréter le coût réduit d'un arc non utilisé dans la solution optimale au problème d'affectation comme étant le coût supplémentaire à payer si on désirait en faire l'utilisation. Il est donc possible d'utiliser les coûts réduits afin d'effectuer un filtrage supplémentaire basé sur la qualité des solutions plutôt que sur la faisabilité de celles-ci. En résumé, on peut éliminer du domaine d'une variable toute valeur dont l'ajout du coût réduit à la borne inférieure ferait augmenter la valeur de l'objectif au-dessus de la borne supérieure. Cette méthode s'est avérée très performante puisqu'elle a permis d'obtenir de meilleurs résultats que les méthodes utilisant uniquement la programmation par contraintes ou la recherche opérationnelle.

Ils ont par ailleurs poussé leurs recherches plus loin en se penchant sur l'utilisation de méthodes de coupes dans un contexte de programmation par contraintes [29]. Les contraintes d'optimisation reposent sur la résolution de bornes inférieures à l'aide de méthodes de recherche opérationnelle ; plus la qualité de la borne est bonne (*i.e.* plus elle est proche de la solution optimale) plus les contraintes d'optimisation sont efficaces. En incorporant des méthodes de coupes aux techniques de résolution déjà proposées, il devient possible d'améliorer le filtrage. Ainsi des coupes d'élimination de sous-tournées sont ajoutées à la méthode de calcule de borne inférieure afin d'en améliorer la qualité. Les auteurs proposent deux approches pour incorporer ces coupes : dans la première ils proposent d'utiliser la programmation linéaire alors que la deuxième repose sur un algorithme spécialisé. La programmation linéaire a l'avantage de pouvoir traiter n'importe quel type de coupes tant que celles-ci sont exprimées sous une forme linéaire, mais sa complexité élevée (exponentielle en pire cas) rend son utilisation peu efficace. La deuxième approche préconise l'utilisation d'un algorithme spécialisé pour le calcule de la borne inférieure, soit un algorithme primal-dual résolvant le problème d'affectation sous-jacent. Toutefois, l'ajout de coupes vient briser la structure du problème d'affection ; celles-ci sont donc relaxées et le problème est résolu par relaxation lagrangienne. Cette dernière méthode s'avère nettement plus efficace que la programmation linéaire, puisque l'algorithme spécialisé peut être exécuté en temps polynomial ($O(n^3)$ la première fois et $O(n^2)$ par la suite).

L'intégration de la programmation dynamique a aussi été étudiée par Focacci et Milano [30]. La modélisation du TSPTW par programmation dynamique nécessite l'utilisation d'un graphe espace-temps : c'est à dire qu'on définit le successeur de chaque noeud pour chaque position possible q'ils peuvent occuper dans la solution finale (S_{ip} représente le successeur de i lorsqu'il est visité à la position p). Cette façon de procéder pose toutefois un problème si on veut utiliser la programmation par contraintes, puisqu'il ne faut tenir compte que des variables correspondant aux positions réellement occupées par les noeuds. Les auteurs introduisent donc les variables conditionnelles qui sont définies à l'aide d'un domaine d et d'une condition c. Une variable conditionnelle doit prendre sa valeur dans d lorsque c et respecté et n'a pas de valeur sinon. À l'aide de ce nouveau type de variables (et en utilisant les positions comme conditions) Focacci et Milano ont conçu une méthode classique de programmation dynamique basée sur la programmation par contraintes. Ils utilisent ensuite cette méthode comme algorithme de filtrage pour une contrainte globale assurant l'existence d'un cycle hamiltonien reliant ensemble de noeuds.

Finalement, Focacci et Shaw [32] ont appliqué au TSP la notion de dominance déjà connue en programmation dynamique et implantée en CP [31]. La dominance est définie comme suit : une solution (s_1) en domine une autre (s_2) lorsqu'elle-ci contient plus de noeuds ($s_2 \subseteq s_1$) et est de meilleure qualité ($val(s_1) \leq val(s_2)$). L'idée consiste donc à détecter le plus rapidement possible les portions de l'arbre de recherche qui sont dominées et qui ne contiennent pas de solution permettant d'améliorer la borne supérieure. Pour ce faire, il faut conserver la valeur de la meilleure solution trouvée dans le sous-arbre de recherche associé à chaque solution partielle générée. Ensuite, pendant la recherche, s'il est possible de démontrer que la solution partielle courante est dominée par une autre déjà identifiée, on peut élaguer la sous arborescence courante. Afin de comparer les solutions partielles entre elles et d'identifier celles qui sont équivalentes, les auteurs font appelle à des méthodes connues de recherche locales (insertions et relocalisations de certains noeuds). Ils peuvent ainsi réduire considérablement le nombre de solutions à conserver et obtenir une implantation plus efficace.

2.2.2 Génération de colonnes

La programmation par contraintes a déjà été utilisée conjointement avec la recherche opérationnelle dans le cadre d'un projet visant à résoudre un problème d'affectation de personnel. L'approche qui fut choisie fut celle de la génération de colonnes, une méthode aussi efficace pour résoudre les problèmes de confection de tournées de véhicules avec contraintes de ressources [19].

Fahle *et al.*[48] ont utilisé la programmation par contraintes en substitution à la plus classique programmation dynamique afin de générer des colonnes de coût réduit négatif. Ils ont modélisé le sous-problème de plus court chemin en programmation par contraintes avec une seule variable ensembliste (*i.e.* la variable est un ensemble dont on cherche à déterminer le contenu). Les contraintes, la propagation et le filtrage

assurent le respect des conditions de travail alors qu'un algorithme de plus court chemin vérifie que la valeur du chemin traversant tous les éléments de l'ensemble est négative. Cette implantation tire profit du fait que le graphe associé au sous problème est acyclique (puisque le temps est une des dimensions du graphe) ; ce qui n'est malheureusement pas le cas pour le VRPTW. Dans ce cas, au lieu de résoudre un algorithme de plus court chemin en temps polynomial sur l'ensemble des clients retenus dans la variable ensembliste, il faudrait plutôt résoudre un TSP en temps exponentiel. Fahle et Sellmann [23] ont par la suite étendu cette méthode aux cas où les sous-problèmes présentent la structure d'un problème de sac à dos (Knapsack Problem : KP).

Chabrier [9] a proposé une méthode utilisant les structures de buts de la programmation par contraintes pour modéliser le processus de résolution de la génération de colonnes. Chabrier, Danna et Le Pape [10] travaillent présentement à l'intégration de la recherche locale à une méthode de génération de colonnes basée sur la génération de tournées élémentaires. Leur méthode utilise aussi la recherche locale afin d'améliorer les solutions identifiée . L'implantation est réalisée à l'intérieur d'une librairie de programmation par contraintes mais repose essentiellement sur une implantation en programmation dynamique ne permettant pas de bénéficier de la flexibilité de la modélisation intrinsèque à la CP.

L'utilisation de la programmation par contraintes pour résoudre le sous-problème dans le contexte d'une méthode de génération de colonnes fera l'objet du deuxième volet du sujet proposé. Celui-ci détaillera une implantation du problème de plus court chemin permettant de tirer profit de toute l'expressivité de la programmation par contraintes.

2.3 CP et méthodes en temps réel

Les problèmes de gestion de flotte en temps réel occupent une place de plus en plus importante dans l'industrie et les méthodes de recherche opérationnelle s'y intéressent depuis plusieurs années déjà. Des revue de littérature de Gendreau et Potvin [38], Qiu et Hsu [68] (dans le cas de véhicules robots) et Psaraftis [65] ainsi que la thèse de doctorat d'Ichoua [43] (consacrées à ce sujet) font l'inventaire des problèmes en les catégorisant selon une nouvelle taxonomie.

Cette taxonomie s'articule autour de trois caractéristiques : le degré de dynamisme, la présence de routes planifiées et l'importance du repositionnement des véhicules. **Le degré de dynamisme** décrit la fréquence des changements affectant les données, c'est à dire la fréquence d'apparition de nouvelles requêtes et la "stochasticité" des attributs de ces nouvelles requêtes. Ainsi, plus les requêtes sont fréquentes et différentes les unes des autres et plus le problème est dynamique. Le temps disponible pour la prise de décision est aussi un facteur important qui influencent le degré de dynamisme : certaines applications exigent une réponse en quelques secondes alors que d'autres disposent de plusieurs heures. **La présence de routes planifiées** est un autre critère permettant de classifier les différents problèmes de gestion de véhicules en temps réel. Une route planifiée est définie comme étant une séquence de requêtes reçues et affectées à un véhicule, mais qui n'ont pas encore été desservies. Ces routes sont généralement construites à partir des requêtes déjà connues avant que la période de répartition en temps réel ne débute (souvent un ou plusieurs jours avant). **L'importance du repositionnement des véhicules** décrit finalement le rôle joué par le positionnement des véhicules lorsque ceux-ci ont terminé une requête et sont en attente d'une prochaine tâche à effectuer. Ce repositionnement est parfois crucial, notamment lorsqu'il est obligatoire de pouvoir répondre à toutes les requêtes dans un délai prescrit comme c'est le cas dans les applications impliquant des véhicules d'urgence.

Si aucune méthode n'impliquant la programmation par contraintes ne s'adresse directement au problème de gestion de flottes en temps réel, plusieurs travaux ont toutefois été menés dans le domaine de l'optimisation sous contrats de temps.

Afin de s'assurer qu'un algorithme non déterministe est en mesure de respecter un contrat temporel, il faut être mesure d'estimer son temps de résolution. Ces problèmes furent l'objet de travaux de par Knuth [50], Purdom [67] et Chen [11] pour des problèmes de satisfaction et repris par Lobjois et Lemaître [54] pour des problèmes d'optimisation.

Ensuite, diverses approches existent afin de profiter au maximum du temps alloué à la résolution. Les plus populaires sont des stratégies itératives, proposées par Ginsberg et Harvey [39], Harvey [41] et Walsh [83] qui consistent à enchaîner le même algorithme en lui donnant de plus en plus de libertés. Par exemple la recherche à divergence limitée (Limited Discrepancy Search : LDS) proposé par Harvey [41] permet de parcourir un arbre de recherche en limitant le nombre de branches considérées à chaque noeud. En manipulant cette limite on peut ainsi modifie le temps nécessaire au processus de résolution et la qualité de la solution produite. Ces méthodes itératives sont bien adaptées lorsque la résolution est dite interruptible (on peut arrêter l'algorithme à tout moment) mais présentent l'inconvénient de répéter à plusieurs reprises les même opérations lorsque que le contrat de temps est fixe.

D'autres stratégies ont aussi été proposées comme celle de Bresina [4] qui consiste à répéter un grand nombre de fois un algorithme glouton dont l'heuristique est biaisé de façon aléatoire. Allen et Minton [1] propose pour des problème de satisfaction (respectivement Lobjois et Lemaître [53] pour des problèmes d'optimisation) de sélectionner l'algorithme complet estimé comme étant plus rapide parmi une liste d'algorithme possible, et ce, en fonction de l'instance à résoudre. Afin d'améliorer les performances d'un système d'optimisation en ligne pour une application de gestion automatique d'armes, De Givry *et al.*[16] tente d'ajuster le plus exactement possible le temps de résolution de leur algorithme au contrat de temps imposé. Pour ce faire ils estiment régulièrement (en cours de traitement) le temps restant au processus d'optimisation et ils adaptent les paramètres du système de façon à accélérer et ou approfondir la recherche.

Chapitre 3

CP et méthodes heuristiques

Ce chapitre étudie une approche de résolution intégrant la programmation par contraintes à des méthodes de recherche locale. Ces travaux ont mené à la publication de l'article suivant :

Rousseau L.-M., Gendreau M. et Pesant G.. *Using Constraint Programming Based Operators to Solve the Vehicle Routing Problem with Time Windows.* Journal of Heuristics, 8, 2002, pages 43-58.

Dans le cadre de ce projet, j'ai effectué la presque totalité de la recherche et de la rédaction de l'article produit. Des rencontres périodiques avec Michel Gendreau et Gilles Pesant, mes directeurs de recherche, étaient planifiées dans le but d'aider au suivi du projet. Leur expérience a été fort utile à la réalisation de ce projet de recherche.

3.1 Introduction

The Operations Research (OR) field has produced in the last decades numerous algorithms and optimization methods that are both effective and efficient. These methods, based on Mathematical Programming or Metaheuristics, are being used to solve most of today's logistic problems. Recently issued from the field of Artificial Intelligence (AI), the Constraint Programming (CP) paradigm provides very good modeling flexibility, which combined with the complete separation between the model and the search, creates a tool to solve real world problems. However, the usual search method used in CP often shows prohibitive execution times for problems of reasonable size, despite recent research on improving search techniques that minimize this drawback (Harvey and Ginsberg [41], Meseguer and Walsh [57].

The combination of optimization methods issued from Operations Research, Artificial Intelligence and Constraint Programming thus seems to be fertile, as these research area show complementary advantages that could be combined while minimizing drawbacks. Pesant and Gendreau have proposed a general framework for this integration [60], [61] and shown how it could be applied to a Vehicle Routing Problem [63]. Baptiste, Le Pape and Nuijten [2] used algorithms taken from OR to build a Constraint Scheduling System. Caseau and Laburthe [6] defined a language for the design of hybrid algorithms. The Ilog team, following the work of Pascal Van Hentenryck [81] has recently commercialized OPL studio a tool for the rapid development of algorithms integrating Mathematical and Constraint Programming.

This paper presents operators that make use of constraint programming to perform local search. These operators are combined in a Variable Neighborhood Descent (VND) framework to solve the Vehicle Routing Problem with Time Windows (VRPTW).

3.1.1 The VRPTW

Vehicle Routing Problems (VRP) are omnipresent in today's industries, ranging from distribution problem to fleet management. They account for a significant portion of the operational cost of many companies. The VRP can be described as follows : given a set of customers C, a set of vehicles V, and a depot d, find a set of routes, starting and ending at d, such that each customer in C is visited by exactly one vehicle in V. Each customer having a specific demand, there are usually capacity constraints on the load that can be carried by a vehicle. In addition, there is a maximum amount of time that can be spent on the road. The time window variant of the problem (VRPTW) imposes the additional constraint that each customer c must be visited after time b_c and before time e_c. One can wait in case of early arrival, but late arrival is not permitted.

The objective function for this class of problem varies from one instance to the other. The primary objective might be to reduce the number of vehicles, the total travel distance or even the time spent on the road. But most of the times these objectives, are considered simultaneously, in a hierarchical fashion.

3.1.2 Hybrid Approach

Some hybrid algorithms using constraint programming have already proven the efficiency of the paradigm on this problem. De Backer and Furnon [14] have proposed to use constraint programming to validate solutions generated by some heuristics, and Shaw [76] proposes to use CP to re-optimize an insertion-based restriction of the VRPTW. Caseau and Laburthe have developed a method that makes use of incremental local optimization as customers are inserted one by one in the solution. Constraint programming is then used to solve the individual routes as Traveling Salesman Problems and thus validate the insertion [7]. They also proposed, in [8], a set of operators using constraint-based insertions and ejections, which they combine in different metaheuristics using learning techniques.

Our method is somewhat similar to the latter, since we developed a set of constraint-based operators that are later on assembled to generate a solution method. We also follow the line of thought of Pesant and Gendreau [61], in the sense that we propose to use constraint programming to search for improving solutions in a neighborhood.

3.2 Operators

In a traditional metaheuristic context, an operator is defined as a recipe to modify a solution and obtain a potentially better one. An operator defines a neighborhood, that is the set of solutions that can be produced by applying that operator on one solution. A move is a transition from one solution to another one in its neighborhood. It is easy to understand that larger neighborhoods will tend to include better solutions, but the time needed to explore those neighborhoods, if we have to look at every solution, grows rapidly and substantially limits the scope of an operator. The idea, originally introduced by Pesant and Gendreau in [60] and [61], is to explore the set of neighboring solutions with a branch-and-bound search in a constraint programming framework. Propagation and pruning are thus implicitly used to eliminate subsets of neighbors, which limits the number of solutions that are actually visited.

3.2.1 Operator : LNS-GENI

The first operator introduced is inspired by ideas from Large Neighborhood Search presented by Shaw in [76]. The algorithm first removes a subset of customers from the solution, and then looks for a better way to reinsert them in the partial solution. LNS obtains very good results on benchmark problems that are quite constrained, which is the case for Solomon class 1 problems where the time windows and capacities are tight and the solution consists of a good number of small routes. We think that this is due to class 1 problems being mainly partitioning problems, that is for a given list of customers for each route, getting the optimal routing (solving the TSPTW for each route) is easy. Solving problems where the routing component is more present like Solomon's class 2 problems is still difficult using LNS. And indeed no computational results were given for that class.

So the idea is to perform LNS but instead of performing simple insertion we use a constraint-programming version of the GENI algorithm. The GENeralized Insertion algorithm [36] and its time window variant [37] try to insert a customer in a given route between any two customers of that route : if the chosen customers are not consecutive, a local optimization is performed to make the insertion possible. The constraint programming version developed by Pesant, Gendreau and Rousseau [63] differs from the original by the fact that instead of looking at all possible local optimizations, it looks for the best one using branch and bound with constraint propagation.

The customers to be removed are chosen randomly but a good bias towards customers generating the longest detour is introduced. The idea is that removing a customer situated far away from the rest of the route will create temporal space in that route and thus facilitate future insertions. The selection strategy also tends to reduce the total travel distance as it tends to replace long arcs by shorter ones. To select the set of customers to be removed, we first order them according to the detour they generate in their route, and then select them randomly with the i^{th} customer having a selection probability of $2i/(n^2+n)$, where n is the number of customers.

To reintroduce the removed customers, we proceed following a *first fail* strategy, attempting the reinsertion of the most constrained customers first. Customer constrainedness can be defined by various means, usually depending on the problem instance that is addressed. Choices include demand, relative position or time window width : in our case the key constraints appeared to be time windows, the insertion order is thus based on time window width. Figure 3.1 illustrates a GENI where the optimal insertion point for c is between c_i in c_j.

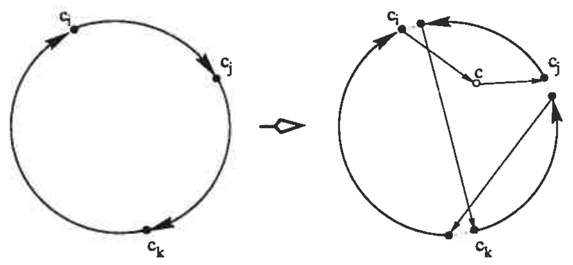


FIG. 3.1 – A GENeralized Insertion

3.2.2 Operator NEC

Ejection Chains are well known in the Vehicle Routing field as they provide a powerful and complex way of moving customers from one vehicle to another. The principle behind ejection chains relies on the fact that some customer c_i of vehicle v_k could possibly be advantageously transferred to vehicle v_l if some customer c_j of v_l was not there. So the idea is to remove c_j of v_l in order to introduce c_i and place c_j elsewhere, which might cause another customer to be ejected and replaced, and so on. The process is called an ejection chain, as it is a series of insertion and ejection that ends with some customer being inserted without the need of an ejection.

The tricky part is to search for such a chain that has negative total differential cost, that is, when all customers have been reinserted, the total travel distance is less than the one before the chain was started. To do so, several methods exist. Glover [40], Rego and Roucairol [72],Thompson and Psaraftis [79] and Gendreau *et al.* [34] have all worked on this problem, suggesting different techniques using graph theory. For instance, Gendreau *et al.* [34] propose to use a graph, where nodes would be the different vehicles and the arcs would be the possible customer transfers, and to solve a negative cycle problem to find an improving ejection chain.

The Naive Ejection Chain (NEC) operator we proposed is defined in the same way as the general algorithm except for the ejection chain search process. In fact, NEC does not really search for an ejection chain that will reduce the total travel distance but rather for a chain that will permit to remove a given customer from a given vehicle. The idea behind such a choice is to try to remove all the customers from a given vehicle in order to eliminate it.

The algorithm is defined as follows. To eliminate a vehicle v_l we first remove one of its customer c_i (which we will call the floating customer) and we try to insert it in another route. If this succeeds then the ejection chain is complete. If no insertion is possible, we try to identify another customer c_j in vehicle v_k such that c_i could be inserted in v_k if c_j was ejected. If such c_j exist we perform the insertion-ejection and restart the process with c_j as the floating customer. In the case where no ejection allows the insertion of the floating customer, we backtrack and chose another floating customer. In order to limit the search space, we always accept the first completed ejection chain we find. Figure 3.2 illustrates this process.

Some restrictions have to be applied in order to prevent the algorithm from cycling and to generate the desired effect of vehicle reduction. Firstly, if we want to eliminate one vehicle we must forbid any insertion in that vehicle anywhere along the chain. Secondly, there are several strategies that could be used to prevent cycling, we chose to simply store the value (Customer) and thus prevent each customer from being moved more then once.

We could have also stored the couple (Customer,Vehicle) and prevent the same customer from being inserted twice in the same vehicle, which would have been a more refined strategy. Unfortunately, this generates a neighborhood too large to be searched efficiently.

Finally, we need an ejection-candidate selection algorithm for which se-

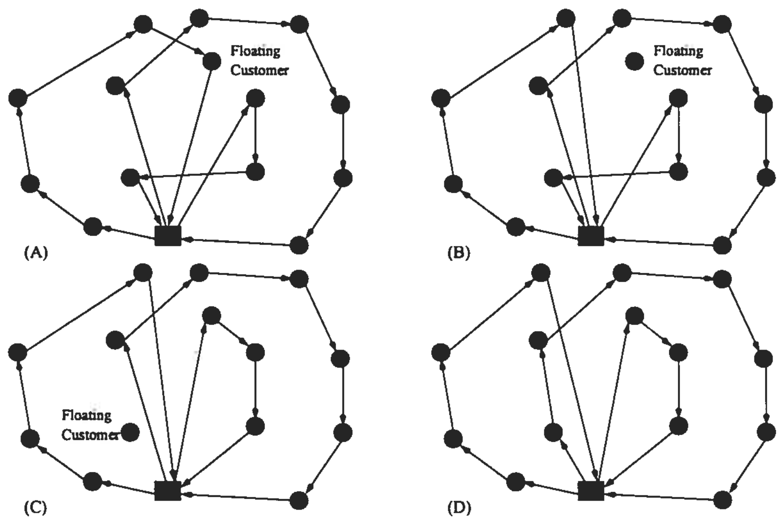


FIG. 3.2 – NEC (A) Original solution (B) *Floating customer* ejected (C) An insertion-ejection and new *Floating customer* (D) New Solution

veral strategies can be considered. We could decide, for instance, to look for ejection-candidate by looking at only one route at a time, and if no possible ejection is found, then consider another route. We could also evaluate each customer independently from their route, sorting them with a particular criterion. A logical criterion could be the distance between the candidate and the floating customer, the similitude of their time windows or, in instances where capacities are a key factor, the demand of the candidate. We experimented with those strategies without much success, therefore we considered a criterion similar to one we already had used in LNS-GENI. In the final implementation, we choose the ejection-candidate that generates the maximum detour in its route, hoping that its ejection will create sufficient temporal space for the insertion of the floating customer, as well as reduce the total travel distance.

The whole search process is performed using the natural backtracking mechanism of a CP solver. The operations of inserting and removing a customer are modeled as constraints on a working copy of the current solution and the ability to validate (prune) feasible (infeasible) solutions is left to the original model. To find an ejection chain, starting with a given customer c already removed from its vehicle, we simply need to solve a Constraint Satisfaction Problem (CSP) similar to :

$$NEC(c) : -Insert(c,T) \vee (Remove(C,T) \wedge Insert(c,T) \wedge NEC(C))$$

where T and C are variables representing any tour or customer.

3.2.3 Operator SMART

The SMAll RouTing operator bears its name because it actually solves a smaller VRP that is a restriction of the original one. The general concept is that, instead of removing customers, we remove arcs from the problem thus creating an incomplete solution to the original problem. This incomplete solution could be interpreted as a smaller VRP in two different ways. The first one is to suppose that the freed arcs are all consecutive. The last customer before and the first customer after the freed sequences of all routes would then become the new depots, and all we would need to do is solve (exactly or not) the smaller problem. A second way of interpreting the relaxation is to consider the general case where the removed arcs are randomly distributed across the solution : we could then replace each of the remaining sub-sequences by a single customer. After adjusting the distance table we would then have a smaller asymmetric VRP.

We now examine in detail the version of the SMART operator where sequences of consecutive arcs are removed. In order to make the move useful, we need to insure that some exchange will be possible, which means that the freed sequences should be close to each other either geographically, temporally or both. The removing policy we use is the following. First identify randomly a *primary pivot*, which will define the neighborhood. Then, remove a certain number of customers before and after that *pivot* thus creating a hole in its tour. The next step is to identify the one customer in each other tour that can be visited in that hole while generating a minimal detour : such customers are named *secondary pivots*. Apply the same treatment to the *secondary pivots*, removing a number of preceding and following customers. Figure 3.3 illustrates this procedure.

As for the second variant of the SMART procedure, arcs are freed randomly with a good bias toward the longer arcs. This procedure insures that intact sequences (those remaining after all chosen arcs have been removed) are short and thus movable; it also helps to accelerate the descent process of reducing the total distance traveled as it replaces longer arcs by shorter

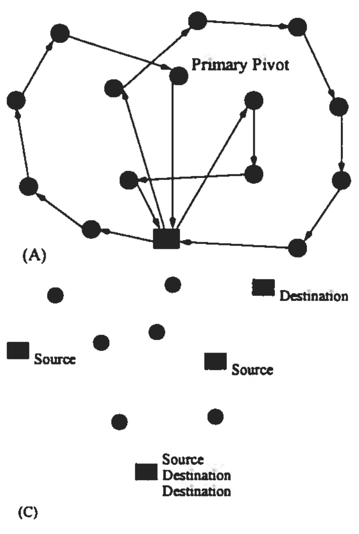


FIG. 3.3 – SMART (A) Original solution (B) *Pivots* and arcs to remove (C) New SMART problem (D) New Solution

ones.

If we choose the size of a SMART problem correctly, we can manage to solve it exactly. To do so we use a modified version the TSPTW model developed by Pesant *et al.* [62]. For upper bound, we use the previous total distance, before the operator was invoked, and for lower bounds three different methods. A nearest neighbor bound and minimum spanning tree are calculated at each node, adding tightness to a more complex regret based nearest neighbor described by Caseau and Laburthe in [5], which is calculated only once per problem. In order to increase the neighborhood size we consider, we choose to explore it only partially using Limited Discrepancies Search (LDS) with a bounded number of discrepancies. In combination with a good value selection algorithm, we can explore much more rapidly a large search space without a significant decrease in solution quality.

It is important to understand that the two variants presented above are only representations of the same problem. We could generalize the procedure by looking at its implementation : to each arc in a solution corresponds a certain variable (in a CP sense). What we do is simply fix the variable associated with the remaining arcs and solve the problem over the variable associated with the ones that are freed. This method allows the treatment of complex constraints since the intact sequences are not really replaced by single customers or depots. It also permits the use of several different arc-removing strategies since the solving process is independent from the arcs that have been removed. Therefore we could use one of the two variants presented, designed a new one or combine multiple strategies during the search. For simplicity we will restrict ourselves to the two variants presented.

3.3 Method

We want to use the operators presented above to solve the Vehicle Routing Problem with Time Window. We thus need to find a way to combine them in order to take advantage of their diverse properties and neighborhood searching capabilities. We propose a construction algorithm, a minimum escaping strategy a and diversification procedure all using the described operators. We also propose to use post-optimization on the best-found solution to furthermore increase its quality.

3.3.1 Construction Method

The main problem encountered while solving routing problems is the reduction of the number of vehicles needed to visit all customers. This problem arises from the fact that it is hard to represent this objective in a cost function since two solutions using the same number of vehicles cannot be easily differentiated. Furthermore, this objective is usually in conflict with the attempt to minimize the travel distance. Therefore, apart for the case where an operator gives a solution using fewer vehicles, it is quite hard to guide the search toward vehicle reduction.

The idea of this method is to concentrate on constructing an initial solution that has a minimal number of vehicles instead of a rather small total travel distance. To do so we use the Naive Ejection Chains (NEC) presented in the previous section. This operator tries to find a feasible solution where one vehicle visits one less customer. Therefore, if applied repeatedly to the same vehicle, it has a good chance of removing it completely. Our construction method thus can be defined as follows. First start by constructing a solution where each vehicle visits only one customer (number of vehicles = number of customers). Then, successively apply the NEC operator to all vehicles, by increasing order of customers visited, in order to maximize the chance of removing the route. The number of times the NEC operator is applied is a parameter of the search construction method, but basically very good solutions are obtained after about 5 executions on all routes.

3.3.2 VND framework

The SMART and LNS-GENI operators presented above are used in a local descent strategy, as they never allow the objective function to increase; it is known that such strategies get trapped in local minimum when no better solution can be found in a neighborhood. To circumvent such a problem, let us consider the Variable Neighborhood Descent (VND) scheme introduced by Mladenovic and Hansen [59]. The VND strategy is to use one operator until we are sufficiently sure that we are trapped in a local minimum, then to start applying the second one until itself can no longer improve the solution. A VND will oscillate in this way between two or more operators, hoping that changes in neighborhood structure will permit an escape from most local minimums. For this principle to work well, we think that the operators used should be very different with respect to the neighborhood structure they generate. Fortunately, it is the case with the operators we proposed, one being centered on customer insertion and the other on arc exchange. We do not include the NEC operator in the VND process as it possesses no mechanism to force local descent.

3.3.3 Improvement and Diversification Phases

Even though the VND scheme permits the search to escape local minimums, some solutions are local minimums for all the VND operators. Such solutions entrap the search process and no escape is possible. When this situation occurs we propose to use a diversification strategy to reposition the search in another area of the solution space. To diversify, we propose to use the NEC operator again, but in a more limited way. When an ejection chain is completed, a small number of customers usually change vehicle. This is a desirable effect because it permits to "shuffle" a certain number of customers around. However because we don't want to loose too much of the good solution at hand, we limit the number of successful ejection chains completed, for instance we accept only the first 5 or 10 completed chains.

The search process thus becomes a two-phase process. The first phase is an Improvement procedure that uses the VND scheme in a descent strategy until it reaches a local minimum (for all the VND operators). The second phase, being the diversification procedure described above, permits the Improvement phase to become effective again, as the solution it produces is no longer local minimums. The global search process alternates between those two phases a predefined number of times, but always keeping the best solution found after the Improvement phase.

3.3.4 Post-Optimization

During the search process we consider multiple exchanges that involve customers belonging to different routes. Partitioning being a key factor in a Vehicle Routing Problem, it needs to be addressed constantly. However at the end of the search process, we can assumed that nothing more can be gained by looking at inter-route exchanges and concentrate on intra-route moves, which are much more simple. In fact, solving a VRP route by route is like solving a series of independent Travelling Salesman Problems (TSP). By regarding each route as TSP and trying to improve them, two cases arise.

The first is when the considered route or TSP as a rather small number of customers (typically less than 20) in which case we can attempt to solve it exactly. To do so we use the original version of TSPTW model developed by Pesant *et al.* [62]. As the execution time distribution tends to have a very long tail, we allow a maximum time limit after which we consider that the TSP belongs to the case.

The second case includes all routes that contain too many customers to be solved exactly. To try to improve those routes we use the US part of the GENIUS-CP [63], which tries to remove a customer and reinsert it in better place. This method is very fast and allows some final local optimization to be performed.

3.4 Results

We use the 56 Solomon problems as benchmarks for our method, firstly because they provide a good variety of instances in terms of constraint tightness, and secondly because they are probably the most popular benchmark for algorithms solving the VRPTW, making it easier to compare with other approaches. All tests were performed on a Sun Ultra10 workstation using Ilog Solver 3.2 as our constraint solver.

3.4.1 Construction

	Construction Method			
Class	Solomon's Insertion I1 [77]		NEC operator	
	Distance	Vehicles	Distance	Vehicles
R1	1393.92	13.42	1419.85	12.17
C1	909.81	10.00	1525.72	10.00
RC1	1561.98	13.50	1575.35	11.75
R2	1280.15	3.18	1290.54	3.09
C2	696.57	3.13	883.75	3.00
RC2	1567.67	3.75	1578.46	3.75

TAB. 3.1 – Averages by class of best initial solutions obtained by compared methods

We easily note that the proposed algorithm is very good in generating a solution with a small number of needed vehicles. When we compare it with Solomon's insertion algorithm, we see that the NEC operator always finds a solution using fewer vehicles. However Solomon's method runs in a fraction of a second instead of minutes for our strategy, but running the Solomon method longer could not produce significantly better results.

Since the NEC operator is not designed to reduce the total travel distance, one could think that the solutions produced would have been of very poor quality. But in fact experimentation showed that this method produces solution comparable to those produced by other specialized algorithms. Except for the C1 and C2 classes, which are easy enough for the Solomon method to almost solve. We think that this is due to the ejection candidate selection algorithm that always tries to eject the customer generating the maximum detour and thus tends to reduce the total travel distance.

3.4.2 Operators

We compare the SMART and LNS-GENI algorithms to determine their relative strengths and weaknesses. Looking at table 3.2 we see that SMART outperforms LNS-GENI in average on all the problem classes. The SMART operator produces very good results considering the fact that it is used in simple descent; but what about LNS-GENI?

We can attribute the worse performance of the LNS-GENI operator to the fact that using GENeralized Insertions actually limits the size of the neighborhood it can search. Because GENIs are much more complicated to

Class	Operators and Combination					
	LNS-GENI		SMART		VN D	
	Dist	Vehi	Dist	Vehi	Dist	Vehi
R1	1276.39	12.17	1245.41	12.17	1225.19	12.08
C1	977.60	10.00	860.11	10.00	834.30	10.00
RC1	1438.79	11.75	1426.80	11.63	1401.76	11.63
R2	999.43	3.09	971.17	3.09	954.07	3.00
C2	697.88	3.00	599.78	3.00	591.06	3.00
RC2	1232.18	3.38	1156.42	3.38	1124.46	3.38

TAB. 3.2 – Averages by class of average solutions obtained by compared operators

evaluate than simple insertions, we can afford to consider only the best insertion point for each removed customer, as opposed to the original LNS [76] which could try to insert a customer in its second or third best insertion point. LNS-GENI is however very useful in combination with the SMART operator as it allows it to reposition in a significantly different area of the solution space, without a decrease in solution quality.

Moreover, it his hoped that, by combining these operators in VND, we can benefit from the strength of both operators while avoiding their weaknesses. It is also hoped that the local minimum escaping property of VND will permit the search to explore a better region of the solution space and thus produce better results. Again looking at table 3.2, we see that this goal is achieved. Furthermore, the figures reported being averages of 20 executions, we note that the VND scheme is very robust, as it produces results very close to those reported in the literature (see table 3.3).

Since the GENI operator is well known in the Operations Research community and many implementations of it exists, it could be interesting to compare the Constraint Programming implementation to a more straightforward version of the algorithm and measure the impact on our application. A very similar experimentation has been performed in [63] and we report here some results. When tested on individual vehicle routes taken from good solutions of Solomon's problems, the CP version of the algorithm was on average four times slower than the specialized heuristic. Using table 3.5 we can thus calculate that the impact on the total run time of the entire algorithm would be a reduction in the order of 35%. However we would lose the flexibility provided by the Constraint Programming model. Each constraint of the problem would have to be separately implemented in the specialized GENI heuristic and we could even lose the ability to model and solve problems with complex and intricate constraints. This argument also stands in the case of the NEC and SMART operators, since the three operators share the same model to describe the problem.

3.4.3 Improvement and Diversification

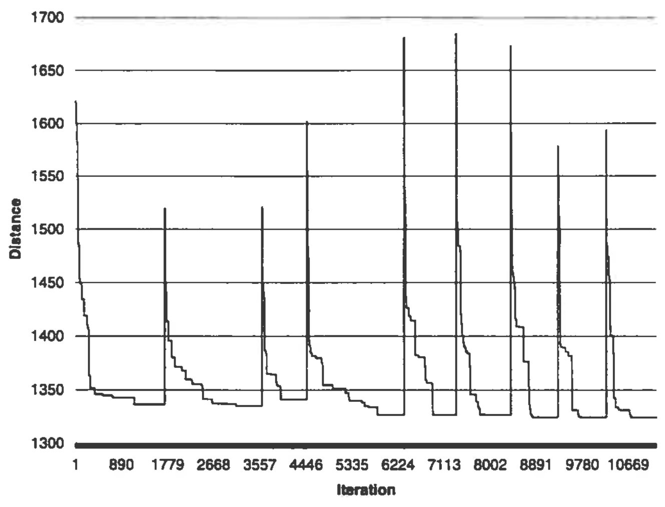


FIG. 3.4 – An example of the Search Process for a particular instance

If we look at the search process presented in figure 3.4, we note that, even though the diversification phases produce solutions which present distance values comparable to those of the construction algorithm, the solution produced after the improvement phase seems to decrease. This reflects the fact that the total distance traveled does not give a proper evaluation of an overall solution quality. The solution produced by the diversification procedure only differs by a few customer exchanges from the previous local minimum solution. The increase in distance is therefore only a result of those few exchanges while the rest of the solution structure is still of good quality. This explains the fact that the Improvement phase seems to be more effective towards the end of the search process, when solutions have a good structure, than at the beginning, when the solutions exhibit more chaotic structure elements.

3.4.4 Post-Optimization

On the 56 problems we have tested, post-optimization allows the improvement of 15 solutions at the end of the search process. However the relative improvement is usually well under 1%.

3.4.5 Comparison

To better evaluate the overall performance of our method, we compare it to other recent methods solving the same problem. We choose to only compare to methods that consider the different distances to be real numbers. The figures presented for our method are the averages by class of the best solutions found for each problem over 10 executions. In the following tables we refer to our Combination of Cooperating Constraint-Programming Operators with the acronym C^3PO.

Comparative Table	Tabu [78]		GLS [49]		LNS [76]		C^3PO	
	Dist	**Vehi**	**Dist**	**Vehi**	**Dist**	**Vehi**	**Dist**	**Vehi**
R1	1209.35	12.17	1200.83	12.67	1209.80	12.00	1210.21	12.08
C1	828.38	10.00	830.75	10.00	828.94	10.00	828.38	10.00
RC1	1389.22	11.50	1388.15	12.12	1366.40	11.75	1382.78	11.63
R2	980.27	2.82	966.56	3.00	—	—	941.08	3.00
C2	589.86	3.00	592.24	3.00	—	—	589.86	3.00
RC2	1117.44	3.38	1133.42	3.38	—	—	1105.22	3.38

TAB. 3.3 – Averages by class of best solutions obtained by compared methods

The solutions produced by our hybrid method seem to be competitive with those published in the literature : we note that both the number of vehicle used and the total traveled distance is around 0.5% of the other methods if not better. We also report new best solutions on 12 problems which are reported in table 3.4 and detailed in annex.

However a note must be taken that C^3PO is somewhat slower than those with which it is compared to by a factor of at least three. More precisely, table 3.5 shows an approximate execution time for each component. The aim of this project was not develop a very fast heuristic but rather to demonstrate that constraint programming could produce state-of-the-art results while providing increased flexibility.

Problem	New Best Solutions				
	Previous Best			New Best	
	Distance	Vehicles	Reference	Distance	Vehicles
R110	1135.07	10	[76]	1124.40	10
R111	1096.73	10	[76]	1096.72	10
RC105	1643.38	13	[78]	1633.72	13
R201	1254.09	4	[49]	1252.37	4
R202	1214.28	3	[78]	1191.70	3
R205	998.96	3	[49]	994.42	3
R206	932.47	3	[78]	929.03	3
R209	923.96	3	[49]	909.86	3
RC202	1162.80	4	[49]	1161.28	4
RC203	1068.07	3	[49]	1066.99	3
RC204	803.90	3	[49]	801.40	3
RC206	1156.26	3	[49]	1153.93	3
RC208	833.97	3	[73]	829.69	3

TAB. 3.4 – New best solutions

Approximate Execution Times	
Component	**CPU time**
The Construction Phase (NEC)	200 seconds
A Local Descent (SMART or LNS-GENI)	500 seconds
Improvement Phase (DVV)	2000 seconds
Diversification Phase (NEC)	50 seconds
Post-Optimization (TSPTW or GENIUS)	100 seconds
Total Execution of C^3PO [1]	11000 seconds

TAB. 3.5 – Approximate execution time for each component

3.5 Conclusion

We have presented a method using constraint programming as a neighborhood searching algorithm for three operators. These operators, combined in Variable Neighborhood Descent and a two phase search process, produce good results on all Solomon's benchmark problems while also proving extra modeling capability. This method is presently able to solve problems with multiple time windows and precedence constraints. Hybrids that are using CP to validate solutions or optimize individual tours would suffer from the introduction of complex and very tight constraints because solutions acceptable by the CP solver would become more rare. As for our method, it would gain in performance, propagation being used more extensively to prune the search space.

Constraint programming has also benefited us from the software developing point of view. Since we have developed all operators on top of a single constraint-programming model of our problem (the VRPTW), any future modification to this problem could simply be achieved by a modification of this model.

The next step consists of exploring incomplete search methods in order to limit the time needed to find improving solutions. Efforts should be made to speed up the search process in both the SMART and the LNS-GENI operators, which would benefit the overall search process.

Chapitre 4

CP et méthodes exactes

Ce chapitre est consacré à la résolution de manière exacte du problème de confection de tournées de véhicules avec fenêtres de temps, et plus précisément à l'intégration de la programmation par contraintes à la génération de colonnes. L'article suivant, déjà paru dans les actes d'un workshop

Rousseau L.-M., Gendreau M. et Pesant G. *Solving small VRPTWs with Constraint Programming Based Column Generation.* Actes de CPAIOR-02.

a été soumis à la revue Annals of Operations Research. Cette version utilise une approche par programmation dynamique développée par Filippo Focacci, d'où son ajout à la liste des auteurs.

Rousseau L.-M., Focacci F., Gendreau M. et Pesant G. *Solving VRPTWs with Constraint Programming Based Column Generation.* soumis à Annals of Operations Research.

Tout comme dans le premier volet, bien que j'ai effectué la majorité de la recherche et de la rédaction de l'article produit, l'expérience de mes directeurs de recherche avec les méthodes de génération de colonnes et de programmation par contraintes fut utile à la réalisation de ce projet de recherche. De plus, le code de programmation dynamique fourni par Filippo Focacci a permis de développer une version plus performante de la méthode.

4.1 Introduction

Vehicle Routing Problems (VRP) are omnipresent in today's industries, ranging from distribution problems to fleet management. They account for a significant portion of the operational cost of many companies. The VRP can be described as follows : given a set of customers C, a set of vehicles V, and a depot d, find a set of routes of minimal length, starting and ending at d, such that each customer in C is visited by exactly one vehicle in V. Each customer having a specific demand, there are usually capacity constraints on the load that can be carried by a vehicle. In addition, there is a maximum amount of time that can be spent on the road. The time window variant of the problem (VRPTW) imposes the additional constraint that each customer c must be visited after time a_c and before time b_c. One can wait in case of early arrival, but late arrival is not permitted.

Column generation is a powerful method used to solve Set Partitioning problems. Introduced by Dantzig and Wolfe [13] to solve linear programs with decomposable structures, it has been applied to many problems with success and has become a leading optimization technique to solve Crew Scheduling Problems [19]. In the first application to the field of Vehicle Routing Problems with Time Windows, presented by Desrochers *et al.* [18], the basic idea was to decompose the problem into sets of customers visited by the same vehicle (routes) and to select the optimal set of routes between all possible ones. Letting r be a feasible route in the original graph (which contains N customers) ; R be the set of all possible r, c_r be the cost of visiting all the customers in r ; $A = (a_{ir})$ be a Boolean matrix expressing the presence of a particular customer (denoted by index $i \in \{1..N\}$) in the route r ; and x_r a Boolean variable specifying whether the route r is chosen ($x_r = 1$) or not ($x_r = 0$), the Set Partitioning Problem is defined as (S) :

$$\begin{aligned} &\min \sum_{r\in R} c_r x_r \\ &s.t \sum_{r\in R} a_{ir} x_r &= 1 \quad \forall i \in \{1..N\} \\ &x \in \{0,1\}^N \end{aligned}$$

This formulation however poses a few problems. Firstly since it is impractical to construct and store the set R because of its very large size, it is usual to work with a partial set R' that is enriched iteratively by solving a subproblem. Secondly, the Set Partitioning formulation is difficult to solve when R' is small and it allows negative dual values which can be problematic for the subproblem. That is why, in general, the following relaxed Set Covering formulation is used instead as a Master Problem (M) :

$$\begin{aligned} &\min \sum_{r \in R'} c_r x_r \\ &s.t \sum_{r \in R'} a_{ir} x_r \quad \geq 1 \quad \forall i \in \{1..N\} \\ &x \in (0,1)^N \end{aligned}$$

To enrich R', it is necessary to find new routes which offer a better way to visit the customers they contain, that is, routes which present a negative reduced cost. The reduced cost of a route is calculated by replacing the cost of an arc (the distances between two customers) d_{ij} by the reduced cost of that arc $c_{ij} = d_{ij} - \lambda_i$ where λ_i is the dual value associated with customer i. The dual value associated with a customer can be interpreted as the marginal cost of visiting that customer in the current optimal solution (given for R'). The objective of the subproblem is then the identification of a negative reduced cost path, that is, a path for which the sum of the travelled distance is inferior to the sum of the marginal costs (dual values). Such a path represents a novel and better way to visit the customers it serves.

The optimal solution of (M) has been identified when there exists no more negative reduced cost path. This solution can however be fractional, since (M) is an linear relaxation of (S), and thus does not represent the optimal solution to (S) but rather a lower bound on it. If this is the case, it is necessary to start a branching scheme in order to identify an integer solution.

Most column generation methods make use of dynamic programming to solve the shortest path subproblem where the elementary constraint (i.e. the constraint that the path does not go through the same node more than once) has been relaxed [20]. This method is very efficient. But since the problem allows negative weight on the arcs, the path produced may contain cycles. However, applications of column generation in Crew Scheduling generally present an acyclic subproblem graph (one dimension of the graph being *time*) which eliminates this problem. Since routing problems are cyclic by nature, most of the methods that address the cyclic cases do so by first rendering the associated graph acyclic. Unfortunately, this transformation requires the different resources (time windows, capacity, etc.) to be discrete and the size of the resulting graph is usually quite impressive (pseudo-polynomial in the resources width).

This paper considers routing application domain and thus concentrates on Cyclic Resource Constrained Shortest Path Problems. These problems are also referred to as Resource Constrained Profitable Tour Problems (PTP) in the literature. The objective is to construct a tour that minimizes the sum of the distance travelled and maximizes the total amount of prize (here, dual values) collected. These objectives are in conflict since more prize collected implies more distance travelled. The combined objective is thus total length of the routes minus the sum of all the dual values collected.

There have been few attempts to combine column generation and constraint programming. Chabrier [9] presented a framework that uses constraint programming search goals to guide the column generation process. Junker *et al.* [48] have proposed a framework they call *constraint programming based column generation* which uses CP to solve constrained shortest path subproblems in column generation. Fahle and Sellmann [23] later enriched the CP based column generation framework with a *Knapsack* constraint for problems which present knapsack subproblems. This framework, which is detailed in the next section, however requires that the underlying graph be acyclic.

The purpose of this paper is to show that constraint programming methods can identify elementary negative reduced cost paths by working on the smaller original cyclic graph. The use of CP also allows the use of any form of constraints on the original problem (which is not the case with the dynamic programming approach). It is thus possible to deal with multiple time windows, precedence constraints amongst visits or any logical implication satisfying special customer demands.

This paper presents the model chosen to describe the Profitable Tour Problem, introduces some redundant constraints and discusses the transformation of a PTP into an Asymmetric Travelling Salesman Problem (ATSP) in order to use known lower bounds. The results section evaluates the impact of the different components and compares the proposed method with two other exact algorithms that solve the same problem.

4.2 CP Based Column Generation

The original motivation to use constraint programming based column generation [48] to solve airline crew assignment problems was that some problems were too complex to be modelled easily by pure Operational Research (OR) methods. Thus, the use of constraint programming to solve the subproblem in a column generation approach provided both the decomposition power of column generation and the modelling flexibility of constraint programming.

To model the constrained shortest path problem, the authors of [48] propose to use a single set variable Y which contains the node to be included in the negative reduced cost path. Since the problem addressed in that framework is by nature acyclic (the underlying network is time directed), it is easy to compute in polynomial time the shortest path covering nodes in Y. A special constraint is also introduced to improve pruning and efficiency of the overall method. This constraint, which ensures that the nodes in Y are part of a feasible path, also enforces bound consistency by solving a shortest path problem on both the required and possible sets of Y. An incremental implementation of the Shortest Path algorithm ensures that the filtering is done efficiently.

4.3 Model

Since the problem considered is cyclic, simple set variables cannot be used to record solutions (as proposed in [48]) because the construction of a complete solution from the set of included visits would require solving a TSP. Thus the need for a new model. Let $N = 0..n$ be the set of all customers. The depot is copied into node $n + 1$, so that a path starts and ends at a different depot. Let $N' = 1..n + 1$ be the set of all nodes except the initial depot.

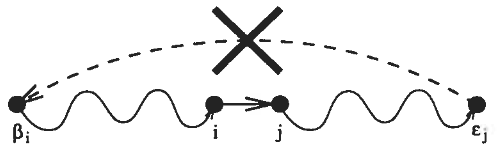


FIG. 4.1 – NoSubTour constraint

4.3.1 Parameters

d_{ij}	Distance from node i to node j.
t_{ij}	Travel time from node i to node j.
a_i, b_i	Bounds of node i's time window.
l_i	Load to take at node i.
λ_i	Dual value associated with node i.
C	Capacity of the vehicle.
$c_{ij} = d_{ij} - \lambda_i$	Reduced cost to go from node i to node j.

4.3.2 Variables

$S_i \in N'$	$\forall i \in N$	Successor of node i.
$T_i \in [a_i, b_i]$	$\forall i \in N \cup n+1$	Time of visit of node i.
$L_i \in [0, C]$	$\forall i \in N \cup n+1$	Truck load before visit of node i.

4.3.3 Objective

$minimize \sum_{i \in N} d_{iS_i}$	Total travelled distance.

4.3.4 Constraints

$AllDiff(S)$		Conservation of flow.
$NoSubTour(S)$		SubTour elimination constraint.
$T_i + t_{iS_i} \leq T_{S_i}$	$\forall i \in N$	Time window constraints.
$L_i + l_i = L_{S_i}$	$\forall i \in N$	Capacity constraints.

The nodes which are left out of the chosen path have their S_i and P_i value fixed to the value i. The *NoSubTour* constraint, illustrated in figure 4.3.4, is taken from the work of Pesant *et al.* [62]. For each chain of customers, we store the name of the first and last visit. When two chains are joined together (when a variable is fixed and a new arc is introduced), we take two actions. First, we update the information concerning the first and last visits of the new (larger) chain, and then, we remove the value of the first customer from the domain of the Successor variable of the last customer.

4.4 Additional Constraints

In order to improve solution time, we introduce redundant constraints, which do not modify the solution set but allow improved pruning and filtering. The constraints introduced in [62], which perform filtering based on time window narrowing, are included in the present method. These constraints maintain for each node (say i) the latest possible departure time and the node (say j) associated with this time. When the domain of S_i is modified the constraint first verifies that j is still in the domain of S_i and if so performs no filtering. This implementations allows the computation of time feasibility to be very efficient since it prevents redundant checking of known feasible solution. For this project, we have implemented similar constraints dealing with the capacity dimension of the VRPTW, which was not present in the TSPTW context of [62]. We also propose a new family of redundant constraints.

4.4.1 Arc Elimination Constraints

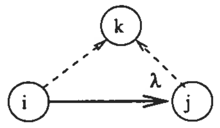
We introduce a new family of redundant constraints that can reduce the number of explored nodes of the search tree by reducing the number of arcs of the subproblem graph. The idea is to eliminate arcs which we know will not be present in the PTP optimal solution. Such a practice is known as cost-based filtering or optimization constraints (introduced by [27]) since it filters out feasible solutions but not optimal ones.

The idea to eliminate arcs that cannot be present in the optimal solution has already been used by Mingozzi *et al.* in [58]. However the two routines they proposed cannot be applied to our method. The first one, which heavily relies on the dynamic programming approach used to identify negative reduced paths, is too expensive in terms of computation (pseudo-polynomial on the resources width) and the second is trivially enforced by the $S-P$ consistency constraint.

We propose three arc eliminating constraints that can significantly reduce the size of the original graph based on the following idea : if the dual value associated with a customer is not sufficiently large, it may then not be worth the trip to visit this customer. Again, these constraints are valid only if the triangular inequality holds for the resources. Otherwise the visit of an intermediate customer could yield savings in some resources and thus allow the visit of extra customers.

Arc Elimination of Type 1

The Arc Elimination constraint of type one is defined as follows : given an arc (i, j), if for all other customers k that are elements of the domain of the successor variable of j (S_j), it is always cheaper to go directly from i to k (d_{ik}) than to travel through j $(d_{ij} + d_{jk} - \lambda_j)$, then the arc $i - j$ can be eliminated from the subproblem graph since it will never be part of an optimal solution.

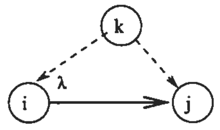


$\forall i \in \{0..N\}$, $\forall j \in S_i : j \neq i$ impose that

$$(\forall k \in S_j : k \neq i \neq j \ \ (d_{ij} + d_{jk} - \lambda_j > d_{ik})) \Rightarrow S_i \neq j$$

Arc Elimination of Type 2

The Arc Elimination constraint of type two is defined as follows : given an arc (i, j), if for all other customers k that are elements of the domain of the predecessor variable of i (P_i), it is always cheaper to go directly from k to j (d_{kj}) than to travel through i $(d_{ki} + d_{ij} - \lambda_i)$, then the arc $i - j$ can be eliminated from the subproblem graph since it will never be part of an optimal solution.

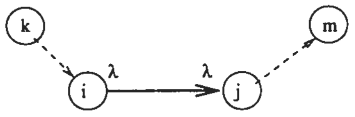


$\forall i \in \{1..N\}$, $\forall j \in S_i : j \neq i$ impose that

$$(\forall k \in P_i : k \neq i \neq j \ \ (d_{ki} + d_{ij} - \lambda_i > d_{kj})) \Rightarrow S_i \neq j$$

Arc Elimination of Type 3

The Arc Elimination constraint of type three is defined as follows : given an arc (i, j), if for all other customers $k \in P_i$ and $m \in S_j$ it is always cheaper to go directly from k to m (d_{km}) than to travel through $i - j$ $(d_{ki} + d_{ij} + d_{jm} - \lambda_i - \lambda_j)$, then the arc $i - j$ can be eliminated from the subproblem graph since it will never be part of an optimal solution.



$\forall i \in \{1..N\}$, $\forall j \in S_i : j \neq i$ impose that

$$(\forall k \in P_i, \forall m \in S_j : k \neq i \neq j \neq m \ \ (d_{ki}+d_{ij}+d_{jm}-\lambda_i-\lambda_j > d_{km})) \Rightarrow S_i \neq j$$

These constraints can be applied before the search to identify a negative reduced cost path is undertaken, and thus could be used in conjunction with any method addressing the PTP (even the dynamic programming approach which solves a relaxation of the PTP). However since we are in the constraint programming paradigm we can obtain further pruning by applying these constraints during search.

To run efficiently these constraints must be implemented incrementally. We detail here the type 1 constraints while the other types have similar implementations. For each arc (i, j) the value of k, a successor of j, for which $(d_{ij}+d_{jk}-\lambda_j \leq d_{ik})$ is kept. When the domain of S_j is modified, the constraint first validates two conditions before performing any filtering. Firstly, the arc (i, j) must still be a possible arc in the solution, that is value j must still be present in the domain of S_i. Secondly, if the value k is still in the domain of S_j then no filtering can occur and the filtering algorithm returns. Only when arc (i, j) is still present and k is no longer a possible successor of j does the constraint look for a new value of k; when none is found the arc is removed.

4.5 Search Strategies

To construct the solution we need to define variable et value selection heuristics. Since the model used is very similar to the TSPTW model presented in [62], the selection strategies presented in that paper could be used to guide the search in the present framework. This has been tried with limited success but, since most successful applications have been achieved through the use of dynamic programming, another search strategy is proposed.

The selection heuristics are based on what has been called Constraint Programming Based dynamic programming [30] and it uses the notion of conditional constrained variables.

4.5.1 Conditional Variables

Conditional variables have been introduced in [30] to model dynamic programming methods (e.g. DP state space relaxation) in Constraint Programming.

Definition

A Conditional Variable V^c is a variable depending on a constraint; it is defined by a the pair (D^c, C^c) where D^c is a domain of possible values, and C^c is a constraint. The following describes the behavior of such variable :

$$C^c \Leftrightarrow D^c \neq \emptyset$$

A conditional variable $V^c = (D^c, C^c)$ is said to be *true* if its definition constraint C^c is true and false otherwise. The constraint C^c of V^c is identified by $C^c(V^c)$.

Operators

Several constraints can be defined using conditional variables. Let us first consider the equality constraint defined between a variable and a conditional variable : let $V^c = (D^c, C^c)$ be a conditional variable, and V be a traditional constrained variable associated to domain D. The constraint $V^c = V$ holds if C^c is false or if C^c is true and the two variables take on the same value. The equality constraint could thus propagate as follows :

$$i \notin D \Rightarrow V^c \neq i, \ \forall i \in D^c$$
$$C^c \Rightarrow (i \notin D^c \Rightarrow V \neq i, \ \forall i \in D)$$

Similarly, let $V_1^c = (D_1^c, C_1^c)$, and $V_2^c = (D_2^c, C_2^c)$ be two conditional variables. The constraint $V_1^c = V_2^c$ holds if $C_1^c \wedge C_2^c$ is false or if $C_1^c \wedge C_2^c$ is true and the two variables assume the same value.

Conditional variables can also be combined via arithmetic operators : for instance, two conditional variables $V_1^c = (D_1^c, C_1^c)$, and $V_2^c = (D_2^c, C_2^c)$ can be combined as $V_3^c = V_1^c + V_2^c$, where V_3^c is defined by $V_3^c = (D_3^c, C_1^c \wedge C_2^c)$ and where D_3^c is defined by the usual propagation rule of the sum operator.

The concept of conditional variables has been used in constraint programming mostly to implement constructive disjunctions and in particular exclusive disjunction. A global constraint used to implement constructive disjunction can be defined as follows : $xunion(V^c_{[1..k]}, y, x)$ where y, x are regular variables, and $V^c_{[1..k]}$ is an array of k conditional variables.

The constraint ensures that exactly one out of the k conditional variables $V^c_{[1..k]}$ are equal to the normal variable y and its index is defined by x. Formally, the constraint $xunion(V^c_{[1..k]}, y, x)$ holds iff :

$$XOR_{i=1..k} C^c(V^c_i)$$
$$x = i | C^c(V^c_i)$$
$$y = V^c_x$$

From this definition, it is easy to see that $C^c(V^c_i)$ is equivalent to $x = i \; \forall \, i : 1..k$

4.5.2 Model Using Conditional Variables

In addition to the model described in section 4.3, new domain variables are introduced to implement a state space relaxation graph of the original problem. The first variables P identify the *position* at which each node is visited. P_i thus indicates the position of node i in the solution and it is set to 0 when node i is not visited in the optimal path except for the source node, which is said to be visited at position 0.

For each node i, new conditional variables $S_{ip} = (N', P_i = p) \, \forall p \in N$ are introduced and represent the node directly succeeding node i in the optimal path if i is visited at position p. Similar variables (T_{ip}, L_{ip}) are also introduced with respect to the time and capacity dimension of the problem. The original variables (S_i, T_i, L_i) is equal to the exclusive disjunction of the n conditional variables S_{ip}, T_{ip}, L_{ip}. These new variables and constraints, once introduced in the model presented in section 4.3, allow the search to proceed in a dynamic programming fashion.

Variables

$$\begin{array}{ll} P_i \in N & \forall i \in N \\ S^c_{ip} \in (N', P_i = p) & \forall i, p \in N \\ T^c_{ip} \in ([a_i, b_i], P_i = p) & \forall i, p \in N \\ L^c_{ip} \in ([0, C], P_i = p) & \forall i, p \in N \end{array}$$

Constraints

$$\begin{array}{ll} P_0 = 0 & \\ S^c_{i0} = i & \forall i \in N \backslash \{0\} \\ S^c_{ip} \neq i & \forall i \in N, p \in N \backslash \{0\} \\ xunion(S^c_{i[0..n]}, S_i, P_i) & \forall i \in N \\ xunion(T^c_{i[0..n]}, T_i, P_i) & \forall i \in N \\ xunion(L^c_{i[0..n]}, L_i, P_i) & \forall i \in N \\ (P_j = p) \Rightarrow \exists i \neq j \mid S^c_{i(p-1)} = j & \forall i, p \in N \\ T^c_{ip} + t_{iS^c_{ip}} \leq T^c_{S^c_{ip}p+1} & \forall i \in N \\ L^c_{ip} + l_i = L^c_{S^c_{ip}p+1} & \forall i \in N \end{array}$$

Once these constraints are added to original model, it is dpossible propagate bound information on all the new variables (those indexed by the position value) in a dynamic programming fashion (as described in [30]). This information is then used to guide the branching process on the original (S) variables. The variable selection policy attempts to construct the shortest path from the source node to the sink node, which means it always selects the successor variable of the last node (say i) that has been added to the path. The value selection heuristic is simply to fix S_i to the most promising[1] value of the domain of S_ip where p is the position at which i was inserted.

Since finding the optimal negative reduced cost path is not important in a column generation framework, the problem is treated as a Constraint Satisfaction Problem and only the first k solutions [2] are kept and included into R'.

[1]the value j which minimizes d_{ij}

[2]k is given as a parameter the method

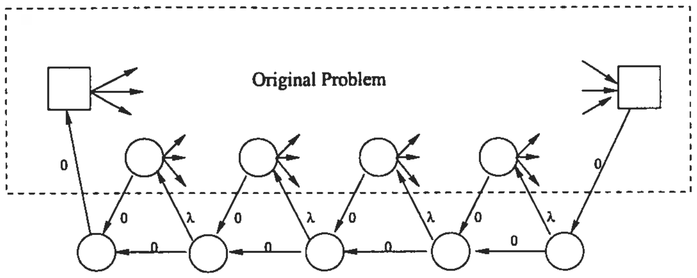


FIG. 4.2 – Transformation of a PTP to an ATSP

4.6 Lower Bounds

In order to prune the search tree efficiently we must be able to compute lower bounds at each node. Unfortunately, even if the literature is prolific in terms of lower bounds for the TSP, none explicitly exists for the PTP. It is fairly simple to transform a PTP into an asymmetric TSP (see [17]) by adding N nodes and 2N arcs (note that in this new graph $c_{ij} = d_{ij}$). This extra portion of the graph constitutes a dummy path that allows the visit of nodes left unvisited by the PTP solution. The cost of visiting a node through this dummy path is set to the cost of its associated dual value. The objective value of the resulting ATSP optimal solution will be superior to the value of the optimal solution to the PTP by a constant that equals the sum of all dual values ($\sum_{i \in \{1..N\}} \lambda_i$).

Well known ATSP lower bounds can then be applied to the transformed graph once the resource constraints have been relaxed. An optimal solution to the Assignment Problem (AP) is a lower bound for the ATSP since it is obtained by relaxing the $NoSubTour$ constraint. Efficient algorithms can be used to solve the AP and since reduced can also be easily computed, some further domain filtering can be achieved. As described in [27], the reduced cost c' can be interpreted as the additional cost to incur if an unused arc is introduced in the solution. Arcs (i, j) whose reduced cost added to lower bound exceed the current upper bound ($LB + c'_{ij} > UB$) can thus be eliminated dynamically during search.

4.7 Branch and Price

The optimal solution to the master problem is obtained once we have proven that no reduced cost path exits. Unfortunately, this solution is not always integral and a branching scheme is thus needed to close the integrality gap. It is not useful to branch on the variables of the master problem because these variables cannot be forced to take the value 0. Even if we fixed x_r to 0 we could not efficiently prevent the CP algorithm from generating again the same route r and adding it to R'.

We therefore choose as branching variables $\{B_i \in N \cup F \mid i \in N \cup I\}$, a set of successor variables similar to those used to describe the subproblem. In the following branching strategy, let f^r_{ij} be a Boolean value indicating whether j is the successor of i in route r.

1. **Node 0 :** Iterate between the master problem and the subproblem until there exists no more negative reduced cost paths.
2. **Upper bounding :** If the current solution to the LP is an integer and its value is better than the best solution found, then update the upper bound, store the current solution and backtrack.
3. **Branching :** Once the optimal solution of the master problem has been found, identify the most fractional variable as the next Branching (B) variable to be fixed. To do so, first calculate the flow that traverses each arc $f_{ij} = \sum_{r \in R'} f^r_{ij} x_r$. Then count for each customer i the number of positive flow outgoing arcs $o_i = \sum_{j \in \{1..N\}} (f_{ij} > 0)$. Finally, select for branching the B_i variable which is associated with the maximum value of o_i and branch on the value j which maximizes f_{ij}.
4. **LP Probing :** In case of a tie in the value selection criteria, when for instance a variable has two outgoing arcs of flow 0.5, a tie breaking strategy must be employed. Since, in the present case, the master problem can be solved very efficiently (in about 0.01 seconds by LP), it can be used to estimate the impact of branching decisions. The results of selecting each value is thus temporarily imposed on the master problem which is then solved. The branching strategy then selects the value which generated the lowest LP value.
5. **Filtering :** Once a branching decision has been made, it is important to enforce it throughout the rest of the algorithm. The first measure to take is to prevent the selection of any columns that violate previously taken decisions. To do so, fix the following variable in the Set Covering Model :

$$x_r = 0 \quad \forall r \in R', i \in N, j \notin B_i : f^r_{ij} = 1$$

It is also crucial to insure that branching decisions are taken into account at the subproblem level. Therefore add the following constraints to the subproblem model :

$$j \notin B_i \rightarrow j \notin S_i \quad \forall i, j \in \{1..N\}$$

6. **Lower bounding** Just as in step one, iterate between the master problem and the subproblem until there exist no more negative reduced cost paths, while taking into consideration the new filtering constraints. If the lower bound obtained is higher than the upper bound then backtrack and cancel the previously taken branching decisions. Otherwise, go back to step two.

4.7.1 Initial Bounds and Columns

In order to accelerate the optimization process, we used known heuristic methods to rapidly obtain an upper bound on the original VRPTW. These heuristics are classic construction heuristics like *insertion*, *savings*, and *sweep* methods and are provided in the ILOG *Dispatcher* [45] library used for this project. We then proceeded with a descent algorithm using the *2-opt*, *or-opt*, *cross*, *exchange* and *relocate* operators to obtain a good solution rapidly. We did not modify the construction heuristics in any way and we used as a descent algorithm the code provided in a *Dispatcher* example file (vrp.cpp). We also took advantage of this preliminary phase to improve the quality of the initial column set (R'). All routes identified during the descent phase are stored in R' and the best solution found is kept as an upper bound.

4.8 Experimental Results

This section compares first evaluates the impact the redundant constraints proposed for the PTP and then reports results on well known vehicle routing benchmarks. The experimental results are compared with two other approaches : a traditional dynamic programming based column generation framework and a pure constraint programming model.

4.8.1 Arc Elimination Constraints

It interesting to study the impact of the Arc Elimination constraints since they can be used to accelerate most constrained Profitable Tour Problem, even when they are solved with dynamic programming. Its behavior is compare on three different PTPs taken from the three problem classes introduced by Solomon. The chosen problems all have a good mixture of large and tight time windows. In the following table we report the total number of backtracks and the CPU time needed to prove the optimality of the best found negative reduced cost path.

Problem	Never		Preprocessing		During Search	
Class	Fails	Time	Fails	Time	Fails	Time
C102	6303	68.99	511	3.65	163	1.82
R102	833	10.54	490	4.83	100	1.81
RC102	5813	79.11	168	1.43	78	0.98

TAB. 4.1 – Arc Elimination Constraints Propagation Strategies : Number of Backtracks and Time to Solve One PTP.

Table 4.1 thus reports the performance of the Arc Elimination constraints with respect to the level of propagation used. The constraint ,when used, is either applied only once before the search starts reducing the domains (preprocessing) or incrementally when the domain of one of the involved variables is modified. Applying the constraints only once before the search starts already reduces significantly both the number of backtracks and the CPU time needed to find the optimal path. However, constraint programming methods can further utilize these constraints by incrementally maintaining consistency during search.

4.8.2 Benchmarking Problems

We have tested the proposed method on the well-known Solomon problems. The geographical data are randomly generated in problem sets R1, clustered in problem sets C1, and a mix of random and clustered structures in problem sets RC1. The customer coordinates are identical for all problems within one type (i.e., R, C and RC). The problems differ with respect to the width of the time windows. Some have very tight time windows, while others have time windows which are hardly constraining. Each problem contains 100 customers but smaller problems are generated by considering only the 25 or 50 first customers. The proposed method was able to solve all of the small size problems and some of the medium and larger ones.

Tables 4.2, 4.3, and 4.4 provide the details of execution on all the problems that could be solved, "—" means that the computation of node 0 was

Problem	Size	UB	LP	nodes	Vehicles	Distance	Time
c101	25	191.81	191.81	0	3	191.81	3.99
c101	50	363.25	363.25	0	5	363.25	58.54
c101	100	891.38	828.94	0	10	828.94	305.13
c102	25	238.65	190.74	0	3	190.74	30.07
c102	50	363.25	362.17	0	5	362.17	127.63
c102	100	973.18	828.94	0	10	828.94	3407.61
c103	25	190.74	190.74	0	3	190.74	175.69
c103	50	434.25	362.17	0	5	362.171	745.26
c104	25	187.45	187.50	0	3	187.50	891.21
c104	50	423.96	358.89	0	5	358.883	1963.43
c105	25	191.81	191.81	0	3	191.81	4.95
c105	50	363.25	363.25	0	5	363.25	66.45
c105	100	896.14	828.94	0	10	828.94	333.57
c106	25	191.81	191.81	0	3	191.81	3.92
c106	50	363.25	363.25	0	5	363.25	67.37
c106	100	936.90	828.94	0	10	828.94	621.58
c107	25	191.81	191.81	0	3	191.81	5.63
c107	50	363.25	363.25	0	5	363.25	79.72
c107	100	854.31	828.94	0	10	828.94	426.70
c108	25	191.81	191.81	0	3	191.81	26.22
c108	50	390.39	363.25	0	5	363.25	150.63
c108	100	949.80	828.94	0	10	828.94	2476.52
c109	25	191.81	191.81	0	3	191.81	100.72
c109	50	363.25	363.25	0	5	363.25	948.75
c109	100	1042.20	828.94	0	10	828.94	2838.02

TAB. 4.2 – Results on the Solomon class C problems

completed but the proof of optimality could not be reached. The LP column gives the value of the possibly fractional solution obtained at the first node of the branch and price tree. Table 4.5 compares the success rates of a pure CP approach, the original Column Generation method and the hybrid proposed in this paper. The proposed method is compared with the OR approach of [18] since it proposes the same decomposition (same master problem and sub-problem).

During this projet the emphasis was put on increasing the flexibility of the method rather than reducing its computational time. The proposed method is much faster than the pure CP approach of [33] and slower than those of [18] when we consider the difference in computer[3] performance over the years.

The results, in terms of the number of solved problems, obtained by our

[3]All experimentations where performed on a SUN computer running at 400 Mhz

Problem	Size	UB	LP	nodes	Vehicles	Distance	Time
r101	25	629.14	618.33	0	8	618.33	4.60
r101	50	1106.90	1046.70	0	12	1046.70	64.55
r101	100	1706.07	1636.39	18	20	1642.88	1668.20
r102	25	561.44	547.40	1	7	548.11	35.10
r102	50	929.29	911.44	0	11	911.44	329.42
r103	25	464.82	455.70	0	5	455.70	46.05
r103	50	808.72	775.65	0	9	775.65	447.41
r104	25	446.13	417.96	0	4	417.96	86.56
r105	25	538.40	531.54	0	6	531.54	7.74
r105	50	982.44	900.94	75	9	914.31	1422.80
r105	100	1473.63	1352.27	—	—	—	—
r106	25	483.13	458.28	6	5	466.48	111.93
r106	50	881.82	794.91	3	8	795.25	641.43
r107	25	457.42	425.27	0	4	425.27	70.58
r107	50	781.53	709.69	24	7	713.50	14394.20
r108	25	428.60	397.74	27	4	398.30	1076.94
r109	25	463.99	442.62	0	5	442.62	8.48
r109	50	888.93	777.82	—	—	—	—
r110	25	448.25	439.69	7	5	445.18	171.33
r110	50	784.36	697.52	—	—	—	—
r111	25	450.38	428.29	3	4	429.70	128.51
r111	50	786.54	698.91	—	—	—	—
r112	25	414.99	388.16	39	4	394.10	1124.26
r112	50	663.12	618.29	—	—	—	—

TAB. 4.3 – Results on the Solomon class R problems

method are comparable to those provided in the literature by similar OR algorithms, but the constraint programming paradigm yields a more flexible approach. For instance, precedence constraints or any kind of logical constraints on customers or vehicles can be easily supported.

It is also important to note that almost all OR column generation methods require the input data of the problem to be discrete. Since the complexity of most dynamic programming approaches is pseudo-polynomial on the resources' (distances, travel times, capacities) width, most authors have truncated all distances to the first decimal point before multiplying them by ten.

This means that some arcs, which are not feasible when distances are computed with real numbers, become feasible. Of course one could argue that this precision is enough to ensure that all solutions found will be feasible w.r.t. real distances, but it is not case. We have found, among the 50 customers problem, that the solution reported for instance R105 is infeasible when true

Problem	Size	UB	LP	nodes	Vehicles	Distance	Time
rc101	25	529.83	409.24	231	4	462.16	669.14
rc101	50	948.58	763.44	—	—	—	—
rc101	100	1778.89	1588.97	—	—	—	—
rc102	25	451.54	352.74	0	3	352.74	50.33
rc102	50	907.44	724.11	—	—	—	—
rc103	25	388.17	333.92	0	3	333.92	145.16
rc103	50	851.10	647.37	—	—	—	—
rc104	25	362.36	307.14	0	3	307.14	161.43
rc104	50	557.04	546.51	0	5	546.51	3025.67
rc105	25	519.98	412.38	0	4	412.38	85.30
rc105	50	953.28	763.44	—	—	—	—
rc106	25	449.94	346.5 0	0	3	346.50	30.03
rc106	50	917.14	668.17	—	—	—	—
rc107	25	363.94	298.95	0	3	298.95	18.10
rc107	50	776.97	604.48	—	—	—	—
rc108	25	295.44	294.99	0	3	294.99	72.47

TAB. 4.4 – Results on the Solomon class RC problems

Problem Size	Pure CP [33]	Column Generation [18]	Hybrid CG-CP
25 Customers	20 %	100 %	100 %
50 Customers	7 %	48 %	55%
100 Customers	0%	24 %	28%

TAB. 4.5 – Percentage of Problem Solved According to Size

travel times are used. This incorrect solution could be computed in only 25% of the normal CPU time by truncating all distances to the first decimal point in our method.

4.9 Conclusion

We have presented a constraint programming Based Column Generation method that addresses vehicle routing problems. The proposed method is flexible since it can handle not only resource based constraints but almost any structure of constraints, while still providing acceptable performance on known benchmark problems.

We also introduced three Arc Elimination algorithms useful in solving any Negative Reduced Cost Shortest Path Problem either in a Column Generation framework or in a Lagrangian Decomposition method [51].

We think that those components have enriched the framework of constraint programming Based Column Generation by enabling the solution of cyclic problems and by proposing tools that will accelerate the execution time of existing methods.

Chapitre 5

CP et méthode en temps réel

Ce chapitre se penche sur le problème de gestion de flotte en temps réel. Lorsque des contraintes additionnelles rendent difficile la résolution avec des méthodes traditionnelles, la programmation par contraintes peut s'avérer fort utile. Ce projet de recherche a mené à la rédaction d'un article qui sera soumis dès que possible à une revue spécialisée en transport.

Rousseau L.-M., Gendreau M. et Pesant G. *The Synchronized Vehicle Dispatching Problem*.

J'ai de nouveau effectué la presque totalité de la recherche et de la rédaction de l'article. J'ai rencontré périodiquement Michel Gendreau et Gilles Pesant, mes directeurs de recherche, dans le but d'assurer le suivi du projet. Leur expérience avec les problèmes complexes de transports et les méthodes de programmation par contraintes a été fort utile à la réalisation de ce projet de recherche.

5.1 Introduction

The synchronized Vehicle Dispatching Problem (SVDP) is a transportation problem which arises in many real world situations. One which illustrates well this problem is a special version of the Dial-a-Ride problem, in which disabled persons require assistance in order to prepare for transportation. In this case, a special team is sent to the residence of the customer a few minutes before the vehicle to insure that the transfer is made safely and efficiently. The help can vary from dressing for winter conditions to providing wheelchair assistance and is not usually required at all times or by all customers. This means the schedule of the special assistance teams needs to be synchronized with the schedules of the rest of the fleet.

Another example is taken from public service companies that offer different levels of service. The case of the cable companies providing internet services illustrates this problem well. When a customer subscribes to a high speed internet connection, some provider will offer a different installation package : a basic cable modem installation or a more complete software configuration package. The latter installation is usually done by two different technicians and their visits must be synchronized. In fact the company has to ensure that the hardware technician will always precede the software technician and that their visit will be as close to one another as possible.

This problem also occurs in large hospitals where patients are transferred between buildings. The dispatcher who plans the transfers for a given period must ensure that the orderlies of both buildings are synchronized with the ambulance.

To our knowledge this problem has not yet been presented in the literature, however there are many variants of the Vehicle Dispatching Problem covering many real life applications. Gendreau and Potvin [38], Qiu and Hsu [68](in the case of Automated Guided Vehicles) and Psaraftis [65] have published detailed surveys describing the different problems and solution approaches. Most efficient algorithms address this problem with local search techniques (like Tabu search for instances). Ichoua [43] has later introduced a taxonomy to further categorize the problems of this class.

The next section presents a model for the SDVP while section 3 describes a solution method to solve this model. In section 4, a transformation is given to produce benchmark problems from known Vehicle Routing instances and results on those problems are given in section 5.

5.2 Problem Description

This section presents a Constraint Programming model for the SVDP. The base problem we have chosen is the Real Time Vehicle Dispatching with Time Windows and Capacity Constraints which is here referred to as the Vehicle Dispatching Problem (VDP). This problem can be described as follows : given a set of customers C, a set of vehicles V, and a depot d, find a set of routes starting and ending at d, such that a maximum number of customers in C are visited by exactly one vehicle. Each customer having a specific demand, there are capacity constraints on the load that can be carried by a vehicle and each customer c must be visited after time a_c and before time b_c. One can wait in case of early arrival, but late arrival is not permitted. When a customer calls for service the dispatcher must be able to tell the customer in a very short amount of time whether he will be serviced or not. Some customers ask for service one or more days in advance and some call during operational hours.

In the synchronized version of this problem customers can ask for special services and will those who do are thus be called *special* customers. Such customers will need to be serviced, in addition to the regular vehicle, by a *special* vehicle. Synchronization requirements such as precedence constraints and restriction on the time delay between regular and special visits are also specified. Capacity constraints are, however, not defined for special vehicles since they usually perform services and not pickups nor deliveries.

The synchronization constraints make this problem quite hard to solve using traditional local search methods. For instance, if the insertion of a special customer delays the route of a special vehicle then all regular visits associated with that route must also be delayed, independently of the vehicle that serviced them. But delaying a regular vehicle means delaying also the visits associated with other special customers and so on. This high number of interconnections means that to insert one customer one might have to recompute the visit time of every customer already inserted. Constraint Programming thus seems well suited for this kind of problem since it not only provides an easy way to express the synchronization constraints but, by representing visit times as domain variables and by propagating the synchronization constraints only when needed, it allows an efficient implementation of those constraints.

5.2.1 Constraint Programming

Before presenting the model, a brief description of the Constraint Programming paradigm is presented here, in case the reader is not familiar with this technology. This approach, which has been very successful in solving hard combinatorial problem in the field of scheduling, planning, and transportation ([21], [80], [82]) has been the subject of a textbook [56] and survey [47].

Traditionally a Constraint Programming model is composed of a set of variables (X), a set of Domains (D), and a set of constraints (C) specifying which assignment of values in D to variable X are legal. The efficiency of the Constraint Programming paradigm lies in powerful constraint propagation algorithms which remove from the domain of the variables the values which will generate infeasible solutions. If propagation does not suffice in finding a feasible solution then a branching process is necessary to further narrow the domains ; a feasible solution has been found when each domain contains only one value.

The branching process is thus useful to search for solutions when propagation is not sufficient, as it is the case in most difficult combinatorial problems. Typically, at each node of the search tree the following four steps are taken : first an unfixed variable is selected, then a value of its domain is chosen, the selected variable is fixed to the chosen value, and constraint propagation occurs. If during propagation the domain of a variable is emptied then the solver has detected an inconsistency in the previously taken decisions and the whole search process backtracks, typically by choosing another value for the variable. When propagation terminates while there are still some unfixed variable, then the solver creates a new search node and goes on with the procedure just detailed. The branching strategy is thus defined by *variable* and *value* selection policies.

It is fairly simple to extend this method to solve combinatorial optimization problems, that is to identify the feasible solution which minimizes (or maximizes) a given objective function. Once a feasible solution as been identified, the set C is extended to contain a new constraint specifying that future feasible solutions should have a strictly better cost than the cost of the solution just identified. The solver will thus keep searching for solutions of improving quality until it can prove that the last one it found was the optimal solution.

5.2.2 Model

Let N be the set of all customers. The depot is copied $2V$ times, where V is the number of vehicles, so that each route starts and ends at a different depot. Let then I and F be respectively the set of initial and final depots. Each special customer c is represented by two nodes, the regular node c and c^* the special node associated to c. The following model (M) is used to represent the problem :

Parameters

d_{ij}	Distance from node i to node j.
t_{ij}	Travel time from node i to node j.
l_i	Load to take at node i.
s_i	Service time at node i.
a_i, b_i	Bounds on node i's time window.
a_i^*, b_i^*	Time window on special visit to node i.
K	Capacity of the vehicles.
C_r	Set of regular customers.
C_s	Set of special customers.
V_r	Set of regular vehicles.
V_s	Set of special vehicles.

Variables

$S_i \in C_r \cup F$	$\forall i \in C_r \cup I$	Successor of regular node i.
$S_i \in C_s \cup F$	$\forall i \in C_s \cup I$	Successor of special node i.
$V_i \in V_r$	$\forall i \in C_r$	Vehicle servicing regular node i.
$V_i \in V_s$	$\forall i \in C_s$	Vehicle servicing special node i.
$T_i \in [a_i, b_i]$	$\forall i \in C_r \cup I \cup F$	Time of visit of regular node i.
$T_i \in [a_i - a_i^*, b_i + b_i^*]$	$\forall i \in C_s$	Time of visit of special node i.
$L_i \in [0, K]$	$\forall i \in C_r \cup I \cup F$	Truck load at regular node i.

Constraints

$AllDifferent(S)$		Conservation of flow.
$NoSubTour(S)$		SubTour elimination constraint.
$S_i = j \Rightarrow L_i + l_i = L_j$	$\forall i \in C_r$	Capacity constraints.
$S_i = j \Rightarrow V_i = V_j$	$\forall i \in N$	Vehicle identification.
$S_i = j \Rightarrow T_i + s_i + t_{ij} \leq T_j$	$\forall i \in N$	Time window constraints.
$S_i = i \Leftrightarrow S_{i^*} = i^*$	$\forall i \in C_r$	Regular-Special synchoronization.
$T_i - a_i^* \leq T_{i^*} \leq T_i + b_i^*$	$\forall i \in C_s$	Time windows synchronization.

Objective

$min \sum_{i \in N} d_{iS_i}$	Minimizing the total distance tavelled

The *AllDifferent* constraint is usually used to enforce conservation of flow in Constraint Programming model. This constraint ensures that all variable take different values by solving a Bipartite Matching Problem and incrementally maintaining consistency of the assignments during search. More details can be found in [71].

The *NoSubTour* constraint is taken from the work of Pesant *et al.* [62]. For each chain of customers, the first and last visits are stored and when two chains are joined together (when a variable is fixed and a new arc is introduced), two actions are taken. First, the information concerning the first and last visits of the new (larger) chain are updated, and then, the value of the first customer is removed from the domain of the S variable of the last customer.

The objective in this problem could be any problem-specific function that represents some goal to achieve. In this model, the objective function is defined as the minimization of travel cost even if the real objective is to maximize the number of the serviced customers. This is, in fact, due to the solution method proposed which is based on the successive insertion of customers as they request service. Since customers are added one at a time an objective function trying to augment the number of visited customers would serve no purpose as it would only affect the one customer being inserted. Whereas the minimization of travel distance generates more space in the routes and thus facilitates future insertions. The next section will describe this insertion process and show how new customer insertion is enforced.

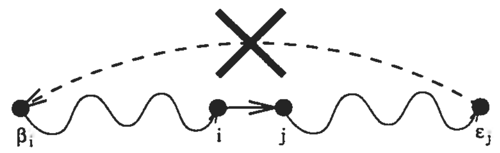


FIG. 5.1 – NoSubTour constraint

5.3 Solution Method

Building an exact solution method for this SVDP is very hard since the dynamic and real time components prevent any solution method from having all the information and decision flexibility at the same time. In the early stages of the dispatching process, the solution method does not possess the list of all the customers who will require service and thus cannot perform optimal planning. Whereas at the end of the process, when most of the information is known, vehicles are already on the road and have serviced a good number of customers. These decisions cannot be undone and directly impact the ability to service new requesting customers. Moreover, calling customers need to be told whether they will be serviced or not in a very short amount of time (typically a few seconds), which further limits the possibility of building optimal solutions.

The solution method proposed in this paper relies on the successive insertion of customers as they request service, while local search methods are applied between requests. This kind of procedure is very straightforward and has been applied in numerous real time dispatching algorithms. The first methods to solve real time dispatching problem were proposed in the 70's by Wilson and his colleague ([85], [86], [84]) in the context of a demand-responsive transportation system in Haddonfield, NJ. These methods used simple insertion to incorporate each new customer into the current solution. This scheme was later applied ([74], [55]) to disabled person transportation systems when some of the requests were known in advance. The metaheuristics proposed in [64], [35], and [34] were developed for the static version of the problem and adapted to deal with the dynamic context. They essentially perform local search until some event (usually a new request) stops the search and triggers the insertion operator.

There are many possibilities for insertion strategies : insertion in the first possible positions and insertion in the best possible positions are two of the most popular ones. Implementing these operators in Constraint Programming allows easy handling of complex constraints such as synchronization. The rest of this section will detail the implementation of the insertion operators using the model previously presented and give insight into the local search used to improve solution quality.

5.3.1 Constraint-Based Insertion

The insertion operator can be modelled using Constraint Programming. Starting from a current solution (visiting some of the customers who have previously requested service) and given that S' is a copy of the values of S of the current solution, that the set of previously inserted (special) customers is defined by P (P^*), and that the customer to insert is given by the index $k(k^*)$, performing the insertion is then only a matter of solving the model (M) with the additional following constraints.

$S_k \in P \cup F$		k must precede an inserted customer.
$S_i \in \{S'_i, k\}$	$\forall i \in C_r \backslash \{k\} \cup I$	Insert no other customer.
$S_{k^*} \in P^* \cup F$		Add only if k needs special visit.
$S_i \in \{S'_i, k^*\}$	$\forall i \in C_s \backslash \{k^*\} \cup I$	Add only if k needs special visit.

When customer k's request becomes known, most vehicles are on the road and some visits have already been performed. Since the previous constraints do not deal with the real time dimension of the SDVP, it is necessary to add two additional sets of constraints. The first set enforces that, at the current time denoted w, every customer that has not already been visited should be visited later than w. The second one forbids insertion after a customer when the vehicle serving it has already left the customer location[1] . By letting the latest departure time[2] of each customer be denoted by d, the following update can be performed before the insertion search is started.

$$d_i \leq w \rightarrow S_i = S'_i \quad \forall i \in N$$
$$d_i > w \rightarrow T_i \geq w \quad \forall i \in N$$

Finding a feasible solution to the problem defined by model (M) enriched of these additional constraints gives a *first insertion* solution servicing customer k, while solving the problem to optimality yields the *best insertion* of k. It is obvious that the best insertion policy should provide better results since it explores a larger space of solutions, although once combined with a local

[1]Ichoua [43] has proposed to diverge enroute vehicles in order to serve a new customer.
[2]This information can be problem dependant, but it is defined here as $min(bi, b_{S_i} - t_{iS_i})$.

search procedure (see section 5.3.3) it is not clear whether this advantage remains. The results section of the paper will compare both insertion strategies. Finally, if no solution to the modified (M) problem exists then the new customer cannot be inserted and his request will be rejected.

5.3.2 Acceleration Techniques

The model previously defined can easily tackle all small and medium size problems, ranging form 100 to 400 customers, but significantly slows downs when the number of customers exceeds 400. In order to accelerate the method so that it can solve problems with up to 1000 requests, further filtering techniques are introduced.

The first one is a redundant constraint taken from the work of Pesant *et al.* [62] that allows to detect inconsistencies in the time windows earlier in the search tree. This constraint, which is checked every time the bounds of one of the T variables is modified, stipulates that :

$$min(T_i) + s_i + d_{ij} > max(T_j) \quad \Rightarrow S_i \neq j \quad \forall\, i, j \in N$$

One can also note that the bounds on the domains of the resource variables (T and L) of the current solution are valid for the solution obtained after a new customer is inserted. For instance, the latest time of visit of each customer cannot be extended after an insertion occurs. This means that the bounds obtain in a given solution can applied directly in all following insertion problems. From this observation another filtering algorithm can be derived. Given that T' and L' are copies of the bounds of the T and L variables of the last solution, the following constraints can be added to the model.

$$\begin{array}{ll} min(T_i') \leq T_i \leq max(T_i') & \forall i \in N \\ min(L_i') \leq L_i \leq max(L_i') & \forall i \in N \end{array}$$

Since capacity is additive in a very straightforward manner (as opposed to the time dimension which can suffer waiting delays), it can also be checked before the insertion process begins. This yields another set of constraints which accelerate the search process, where k is again given as the index of the node to insert.

$$l_k + min(L'_j) > K \quad \Rightarrow V_k \neq j \quad \forall j \in F$$

The variable selection policies traditionally used to solve this kind of problem are based on a *first fail* strategy, which means the solver tries to branch on the most constrained variable first (usually the one with the smallest domain) in order to detect inconsistencies as early as possible in the search tree. However this strategy is not very effective in the context of the proposed insertion strategy because, after all the insertion constraints have been introduced, the domains of all (but one) S variables contain only two values (the previous successor and the customer to be inserted). The only S variable that contains more values is the one associated with the new request. Fixing this variable is equivalent to choosing the insertion position and for this reason S_k is always chosen as the first variable of the branching process. As for values, they are selected in a variable's domain on a most promising basis, meaning the value associated with the minimum cost.

5.3.3 Local Search

The insertion process is a greedy procedure in the sense that it does not reconsider the relative order of the inserted customers nor the vehicle they have been assigned to. To overcome this limitation, the dispatching system can make use of the idle time between requests of service to improve the current planning. This improvement phase is achieved with local search, which are small successive improvements of the current solution.

The local search process is defined using a set of operators and a policy on how to use them (metaheuristic). The set of selected operators are the standard ones used in vehicle routing applications (*2-Opt, Or-Opt, Cross, Exchange* and *Relocate*). These operators are used in conjunction with two different metaheuristics : a simple descent strategy and a more complex Guided Local Search (GLS) [49]. All are taken from the ILOG Dispatcher library [45].

Since the local search procedure can only be used between customer calls for service, the amount of time spent improving the solution is not known *a priori*. Therefore the guided local search metaheuristic, which can run continually until a new request comes, is more appropriate in a real world situation because it uses the maximum amount of time available for improvement. In a simulation context however, a descent strategy which stops after reaching a local minimum has the advantage of having performances which are, to a certain point, independent from simulation time. Again, the results section provides a performance comparison for both approaches.

5.4 Experimental Results

This section presents benchmark problems developed from the Synchronized Vehicle Dispatching Problem which are used to compare different solution methods are parameter settings.

5.5 Benchmarking Problems

The static Vehicle Routing Problem with Time Windows (VRPTW) is probably one of the most benchmarked optimization problems. In 1987 Solomon proposed a set of problems [77] which were to become the reference VRPTW benchmarks.

These problems are grouped in three sets : the geographical data are randomly generated in problem sets R, clustered in problem sets C, and a mix of random and clustered structures in problem sets RC. The customer coordinates are identical for all problems within one type (i.e., R, C and RC). The problems differ with respect to the width of the time windows. Some have very tight time windows, while others have time windows which are hardly constraining. These 100 customer problems have later been extended by Homberger and Gehring [42] to larger instances with up to 1000 customers. This set of problems, even if it is not representative of real life instances, facilitates the comparison between methods that address similar problems.

In the dynamic Vehicle Routing Problem field, most researchers have built problem generators that construct instances according to the probability distributions which best represent the dynamic contexts that their method addressed. This yields test problems similar to those that could be encountered in the real world but makes comparisons between methods much more difficult.

To facilitate further research on the SDVP, a set of dynamic benchmark problems is derived from the well-known Solomon problems. The transformation method which renders the static problems dynamic is first presented followed by the variant which creates the synchronized versions.

The Solomon problems describe a set of customers defined by integer identifier, a set of coordinates in a plane, a quantity of product requested, a time window, and a service time. To generate dynamic problems it is only necessary to determine at which point in time each request becomes known to the dispatcher. Some instances may also show different levels of dynamism, meaning that a certain proportion of the requests are known in advance and that a pre-planning phase is possible before the real-time dispatching process begins. This is namely the case when customers call one or several days prior to their service day. Such instances are generated by selecting a number of customers which constitute a static instance of the VRPTW. This instance can be solved by any method and the best solution found is then used as the starting initial solution for the first customer insertion.

The dynamic problems are generated with the following parameters : customers whose time windows start at time 0 (those who can be serviced right away) are considered to be static customers. All other visits i are to be requested at time of a_i/α. For instance, α set to 2 means customers call for service at a time equivalent to half of their time window lower bound.

Once this is done, building a synchronized problem is just a matter of choosing which customers are going to be special, setting the number of special vehicles and giving values to special visit time windows (a^* and b^*). In the proposed transformation method, the special customers are distributed evenly across the dispatching horizon. The percentage of customers who will be considered special is given by β and the number of special vehicles is equal to β times the number of vehicles usually needed to solve the problem[3]. To select the special customers among those present in the original benchmark files, the following rules are applied :

$$(i \mod (100/\beta) > 0) \quad \Rightarrow \quad i \in C_r \;\; \forall i \in N$$
$$(i \mod (100/\beta) = 0) \quad \Rightarrow \quad i \in C_s \;\; \forall i \in N$$

5.5.1 Simulation

In order to simulate the arrival and dispatching process of a normal day of operation using the transformed benchmark problem, it is necessary to

[3]For the Homberger problems, these number were extrapolated from the solutions to the Solomon problems of size 100. For instance C.2.200.01 was said to need twice as many vehicles as C.2.100.01

Problem	First Insertion			Best Insertion		
Size	No LS	Desc.	GLS	No LS	Desc.	GLS
100	20.4	9.9	10.8	15.1	10.3	10.3
200	8.9	4.8	5.1	7.2	5.0	5.0
400	10.8	7.1	7.5	8.5	6.9	6.8
600	11.8	10.4	10.6	9.8	8.9	8.9
800	14.7	13.6	13.7	12.4	12.2	12.1
1000	17.6	16.6	16.5	18.3	17.9	17.8

TAB. 5.1 – Percentage of rejected customers for the SDVP with $\alpha = 2$, $\beta = 10$ and $\gamma = 10sec$.

generate a mapping between the different time components specified in the test files (like time windows for instance) and the real time during the simulation process. This mapping is achieved by defining γ, the average time interval between requests, and T_{max}, the depot's closing time in the test file. A total runtime R is first computed as the product of γ by the total number of customers ($R = \gamma * |N|$), then a mapping factor $\mu = T_{max}/R$ is defined to link the two time dimensions. Once μ is defined, clock time can be converted to problem time by a simple multiplication.

5.5.2 Results

This section describes the performance of the proposed method on the Synchronized Vehicle Dispatching Problem. A set of synchronized problems was created by setting γ to 10 seconds and β to 10%, meaning one out of 10 customers is considered to be special. The value of (a^*, b^*) was set to (10,0) for each special customer so that the special visit could start up to 10 units of time before the regular visit. Table 5.1 reports the proportion of rejected customers against problem size using the proposed insertion operators and local search methods. There are 56 problems of size 100 and 60 problems of size 200,400,600,800 and 1000 for a total of 356 problems containing 185 600 requests.

Table 5.1 shows the rejection ration yielded by the different solution methods when applied to synchronized problems. As expected, *best insertion* methods are better than *first insertion* but the difference tends to diminish after local search is applied. The main factor of performance thus seems to be local search, whatever form it takes, since simple local search is as effective as Guided Local Search. For the following tables the combination of GLS and

Problem Size	Time interval between requests							
	$\gamma = 5$		$\gamma = 10$		$\gamma = 20$		$\gamma = 40$	
	Reg.	**Spec.**	**Reg.**	**Spec.**	**Reg.**	**Spec.**	**Reg**	**Spec.**
100	8.8	23.9	8.6	24.4	7.9	24.2	8.0	23.5
200	3.3	22.7	3.0	21.3	2.9	21.1	2.8	21.1
400	5.8	25.2	4.8	23.4	3.9	23.1	3.9	22.8
600	7.7	28.5	6.7	28.0	5.6	26.9	5.0	25.4
800	12.0	26.7	10.6	26.4	9.4	25.9	7.9	24.7
1000	16.5	31.0	16.2	32.1	15.1	31.9	12.5	30.0

TAB. 5.2 – Percentage of rejected customers for the SDVP with $\alpha = 2$, $\beta = 10$

Best Insertion will be used to measure the impact of the γ and β parameters, that is the rate of arrival of new requests and ratio of special customers.

But lets first note the significant difference of rejection ratio between the 100 series and the rest of the instances. This discrepancy is probably due to fact that the instances of size 100 were generated by Solomon [77] while the larger problems were later proposed by Homberger [42] and thus present a different structure[4]. For the instances of size 200 to 1000, the rejection ratio increases with the size of the problem. This is explained by the fact that when instances become larger successive insertions become harder and the time available for insertion and local search becomes insufficient.

A key factor that influences the rejection ratio of customers is the average time between requests. Table 5.2 clearly indicates that when more time is available to perform insertion and local search, more customers, both special and regular, can be serviced. It seems however that for smaller problems (100 and 200), a time interval of 20 seconds is sufficient and that increasing the value of γ does not really improve solutions. Problems of size 400 and above would however probably benefit from increased time intervals.

Table 5.3 reports the rejection ratio according to the percentage of special customers. By looking at series 400 to 1000, we note that, although the percentage for regulars customers are quite stable, the percentage of rejected special customers decreases when β increases. This is explained by the flexibility gained with the increased number special vehicles. Thus when the number of special customers rises more of these customers are rejected in absolute value, but in it becomes relatively easier to insert them.

The conflicting results for 100 and 200 series are caused by the discrete nature of the number of special vehicles. In these instances there are too

[4]Although they were built in a similar fashion

Problem Size	Percentage of special customers							
	$\beta = 5\%$		$\beta = 10\%$		$\beta = 25\%$		$\beta = 50\%$	
	Reg.	**Spec.**	**Reg.**	**Spec.**	**Reg.**	**Spec.**	**Reg**	**Spec.**
100	9.1	16.3	8.6	24.4	8.5	33.0	10.6	22.6
200	3.1	14.4	3.0	21.3	3.1	14.9	3.2	11.9
400	5.0	27.5	4.8	23.9	4.7	21.3	5.4	15.4
600	6.7	32.9	6.7	28.0	6.4	24.6	6.9	18.1
800	10.7	30.0	10.6	26.4	10.7	25.3	11.6	20.0
1000	15.8	35.8	16.2	32.1	17.9	28.4	23.1	29.1

TAB. 5.3 – Percentage of rejected customers for the SDVP with $\alpha = 2$, $\gamma = 10$

few regular vehicles to generate the proper ratio of special vehicles. This means that each instance either has too many or too few special vehicles in proportion to regular ones.

5.6 Conclusion

A new variation of dispatching vehicle problems was introduced to meet the specific need of a class of complex real world applications. The synchronization constraints could be applied in the context of any dynamic fleet assignment problem or even in the static case.

Constraint-based insertion operators were derived from the original model for the Synchronized Vehicle Dispatching Problem and were used to define a successive insertion procedure. Local search was also applied between requests to improve the current solution and facilitate future insertions. Results were given on a set of constructed benchmark problems to demonstrate the relative performance of the insertion operators and local search proposed, as well as the sensitivity to the rate of arrival of the requests and the ratio of special customers.

The problem was addressed using Constraint Programming for mainly two reasons. Firstly, the modelling power of this paradigm, which greatly eased the modelling phase, allows for a complete separation between the model and the search procedure. This feature makes it simple to add new constraints without having to modify the insertion operator or the local search method. Secondly but most importantly, the domain variable representation and the propagation techniques of CP eased up the handling of synchronization constraints.

Chapitre 6

Conclusion

Le premier volet du programme de recherche (exposé au chapître 3), présentait l'intégration de la programmation par contraintes à des méthodes de recherche locale. Cette hybridation a pris la forme d'opérateurs utilisant ce paradigme pour effectuer des améliorations locales à une solution réalisable. Ils ont été utilisés à l'intérieure d'une méthode complète de résolution incluant notamment des phases de construction, amélioration, diversification et post-optimisation. Les résultats obtenus lors de tests sur des problèmes de référence démontreront clairement l'efficacité de la combinaison proposée.

Le deuxième volet (chapitre 4) abordait la résolution de manière exacte du problème de confection de tournées de véhicules avec fenêtres de temps. L'approche choisie fut celle d'une hybridation de la programmation par contraintes avec une méthode de décomposition de Dantzig-Wolfe et d'une résolution par génération de colonnes. La programmation par contraintes fut utilisée afin de résoudre le sous-problème associé à la génération de colonnes, et ce, de façon à bénéficier de sa flexibilité et de son expressivité.

Le dernier volet (chapitre 5) aborda sur l'utilisation de la programmation par contraintes dans le cadre de problèmes de gestion de flotte en temps réel. Un problème complexe de transport dynamique fut présenté afin de démontrer la puissance de modélisation et de résolution des méthodes hybrides proposées.

6.1 Principales contributions

La première contribution de cette thèse au domaine de l'optimisation en transport est la démonstration qu'un modèle basé sur la programmation par contraintes peut facilement décrire un ensemble de problèmes différents. Le modèle proposé permet de représenter, suite à de légères modifications, tout aussi bien un problème de voyageur de commerce, de confection de tournées de véhicules, de plus courts chemins, qu'un de répartition de flotte en temps réel, et ce, tout en permettant la modélisation de contraintes complexes. De plus, l'effort investi afin de rendre efficace l'implantation de certaines contraintes est bénéfique pour tous les types de problèmes résolus.

L'hybridation de la programmation par contraintes avec des techniques de recherches locales a permis de concevoir un algorithme de résolution pour le problème de confection de tournées de véhicules qui rivalise avec les meilleures méthodes connues. Cette méthode hybride a d'ailleurs permis d'améliorer les solutions à un nombre considérable de problèmes de références déjà scrutés par la recherche depuis une quinzaine d'années.

Si une bonne solution heuristique est généralement satisfaisante dans le cadre des problèmes de transport, il arrive parfois que l'on ait besoin d'une garantie de qualité ou même d'une preuve d'optimalité. Dans ce contexte, il devient intéressant d'étudier la contribution de la programmation par contraintes au domaine de l'optimisation exacte. Le problème étudié, encore la confection de tournées de véhicules avec fenêtres de temps, est trop complexe pour être résolu directement en CP. Toutefois, en utilisant l'approche de la génération de colonnes on peut bâtir une méthode hybride permettant à la fois de résoudre exactement le VRPTW tout en supportant un grand nombre de contraintes complexes. Ces recherches ont de plus permis de définir un ensemble de composantes qui pourront être utiles dans d'autres domaines d'application. C'est notamment le cas des méthodes de filtrage pour problèmes de plus courts chemins avec coût réduits (*ArcElmination*).

Dans le contexte actuel où les technologies de l'information nous permettent de plus en plus d'effectuer la gestion et la répartition de flotte en temps réel, il devient nécessaire de résoudre des problèmes de plus en plus complexes dans des laps de temps de plus en plus courts. C'est pourquoi l'application de la programmation par contraintes au contexte du temps réel est intéressante ; ses mécanismes puissants de propagation permettent de modéliser et de résoudre des problèmes extrêmement contraints. Une solution au problème de répartition de flotte en temps réel avec contraintes de synchronisation a été réalisée afin de démontrer, non seulement l'expressivité de la programmation par contraintes, mais également la facilité avec laquelle cette méthode peut résoudre des problèmes difficilement résolubles avec des méthodes plus traditionnelles.

6.2 Futures avenues de recherche

L'hybridation de méthodes issues de la programmation par contraintes et de la recherche opérationnelle étant un domaine de recherche relativement jeune, les avenues de recherche sont nombreuses. Celles que nous identifions ici touchent en particulier le domaine des méthodes exactes mais les résultats de ces recherches pourraient aussi être utilisés dans le cadre de méthodes heuristiques.

Dans le contexte de la méthode hybride utilisant la programmation par contraintes et la génération de colonnes, une attention particulière devrait être apportée au calcul des bornes inférieures. En effet, l'analyse des problèmes non résolus démontre que les bornes inférieures obtenues sont de piètre qualité (souvent à plus de 1000% de la solution optimale). La performance des méthodes utilisant la programmation par contraintes dépend grandement de la qualité des bornes disponibles. Une étude approfondie des méthodes de bornes additives proposées par Fischetti et Toth ([25], [24],[26]) permettrait peut-être de développer des bornes de meilleure qualité.

De façon plus générale, une intégration où une coopération entre les méthodes efficaces de la recherche opérationnelle et de la programmation par contraintes permettrait d'améliorer les méthodes de résolutions. Une des formes présentes d'intégration [69], [70] propose de résoudre un modèle à la fois en programmation linéaire et en programmation par contraintes, et ce, de manière à ce que l'information obtenue des divers algorithmes soit communiquée à travers les variables de décision du modèle unique. Concevoir un modèle efficace pour plus d'un paradigme d'optimisation est toutefois un défi considérable. C'est pourquoi cette forme d'intégration pourrait passer par une double modélisation du problème et la collaboration entre les méthodes de résolution s'effectuerait à travers des liens établis entre les modèles. Si la génération automatique d'un deuxième modèle efficace reste un objectif difficile à réaliser à court terme, des études pilotes pourraient être entreprises dans des domaines d'application bien précis afin de valider cette approche. Des domaines où des modèles efficaces existent à la fois en RO et CP seraient tout désignés pour mettre à l'essai ces nouvelles approches.

Bibliographie

[1] J. Allen and S. Minton. Selecting the Right Heuristic Algorithm : Runtime Performance Predictors. In *the Canadian AI Conference*, 1996.

[2] P. Baptiste, C. Le Pape, and W. Nuijten. Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling. In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*, Timberline Lodge, Oregon, 1995.

[3] O. Braysy. *Local Search and Variable Neighborhood Search Algorithms for the Vehicle Routing Problem with Time Windows.* PhD thesis, Universitas Wasaensis, 2001.

[4] J.L. Bresina. Heuristic-Biased Stochastic Sampling. In *American Association for Artificial Intelligence – AAAI*, pages 271–278, Portland,Oregon,USA, 1996.

[5] Y. Caseau and F. Laburthe. Solving Small TSPs with Constraints. In *Proceedings of the 14th International Conference on Logic Programming*, pages 316–330. MIT Press, Cambridge MA, 1997.

[6] Y. Caseau and F. Laburthe. SALSA : A Language for Search Algorithms. In *Principle and Practice of Constraint Programming – CP98*, Pisa, Italy, October 1998.

[7] Y. Caseau and F. Laburthe. Heuristics for Large Constrained Vehicle Routing Problems. *Journal of Heuristics*, 5(3) :281–303, 1999.

[8] Y. Caseau, F. Laburthe, and G. Silverstein. A Meta-Heuristic Factory for Vehicle Routing Problems. In *Principles and Practice of Constraint Programming – CP99*, Lecture Notes in Computer Science. Springer-Verlag, 1994.

[9] A. Chabrier. Using Constraint Programming Search Goals to Define Column Generation Search Procedures. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, pages 19–27, 2000.

[10] A. Chabrier, E. Danna, and C. Le Pape. Cooperation Between Column Generation with Elementary Routes and Local Search for the Vehicle Routing Problem with Time Windows. In *Actes des Journées Francophones de Programmation Logique par Contraintes*, 2002.

[11] P. Chen. Heuristic Sampling : a Method for Predicting the Performance of Tree Searching Programs. *SIAM Journal on Computing*, 21(2) :295–315, 1992.

[12] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, Solomon M. M., and Soumis F. The VRP with Time Windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem, Chapter 7.* SIAM Monographs on Discrete Mathematics and Applications, 1993.

[13] G.B. Dantzig and P. Wolfe. Decomposition principles for linear programs. *Operations Research*, 8 :101–111, 1960.

[14] B. De Backer and V. Furnon. Meta-heuristics in Constraint Programming Experiments with Tabu Search on the Vehicle Routing Problem. In *Proceedings of the Second International Conference on Metaheuristics (MIC'97)*, Sophia Antipolis, France, July 1997.

[15] B. De Backer, V. Furnon, P. Kilby, P. Prosser, and P. Shaw. Local Search in Constraint Programming : Application to the Vehicle Routing Problem. In *CP97 Workshop on Industrial Constraint-Directed Scheduling*, November 1997.

[16] S. de Givry, P. Savéant, and J. Jourdan. Optimisation combinatoire en temps limité :Depth First Branch and Bound adaptatif. In *Actes des huitième journées francophones de programation logique et programmation par contraintes*, pages 161–176. Hermes, 1999.

[17] M. Dell'Amico, F. Maffioli, and P. Varbrand. On prize-collecting tours and the asymmetric travelling salesman problem. Technical report, Dipartimento di Elettronica e Informazione Politecnico di Milano, Milano, Italie, 1994.

[18] M. Desrochers, J. Desrosiers, and M.M. Solomon. A New Optimisation Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, 40 :342–354, 1992.

[19] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time Constrained Routing and Scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, and Nemhauser G.L., editors, *Network Routing*, volume 8 of *Handbooks*

in Operations Research and Management Science, pages 35–139. North-Holland, Amsterdam, 1995.

[20] J. Desrosiers, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. *Handbooks of Operations Research and Management Science*, 8 :35–139, 1993.

[21] M. Dincbas, P. Van Hentenryck, and H. Simonis. Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic Programming*, 8 :75–93, 1990.

[22] M. Dincbas, P. Van Hentenryck, and H. Simonis. Solving the Car Sequencing Problem in Constraint Logic Programming. In *European Conference on Artificial Intelligence ECAI88*, pages 290–295, Munich, Germany, 1998.

[23] T. Fahle and M. Sellmann. Constraint Programming Based Column Generation with Knapsack Subproblems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, pages 33–43, 2000.

[24] M. Fischetti and P. Toth. An Additive Approach for the Optimal Solution of the Prize-Collecting Traveling Salesman Problem. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing : Methods and Studies*. North-Holland, 1988.

[25] M. Fischetti and P. Toth. An Additive Bounding Procedure for Combinatorial Optimization Problems. *Operations Research*, 37 :319–328, 1989.

[26] M. Fischetti and P. Toth. An Additive Bounding Procedure for the Asymmetric Travelling Salesman Problem. *Mathematical Programming*, 53 :173–197, 1992.

[27] F. Focacci, A. Lodi, and M. Milano. "Solving TSP through the Integration of OR and CP Techniques. *Electronic Notes in Discrete Mathematics*, 1, 1999.

[28] F. Focacci, A. Lodi, and M. Milano. "Solving TSP with Time Windows with Constraints . In *International Conference on Logic Programming*, pages 515–529, Cambridge, Massachusetts, November 1999. MIT-press.

[29] F. Focacci and M. Milano. Cutting Planes in Constraint Programming : a Hybrid Approach. In *Principles and Practice of Constraint Programming –CP2000*, Lecture Notes in Computer Science 1894, pages 187–201. Springer-Verlag, 2000.

[30] F. Focacci and M. Milano. Connections and Integrations of Dynamic Programming and Constraint Programming. In *Proceedings of the Third Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 125–138, Ashford, Kent, UK, April 2001.

[31] F. Focacci and M. Milano. Global Cut Framework for Removing Symmetries. In *Principles and Practice of Constraint Programming -CP2001*, Lecture Notes in Computer Science, pages 77–92. Springer-Verlag, 2001.

[32] F. Focacci and P. Shaw. Pruning Sub-Optimal Search Branches Using Local Search. In *Proceedings of the Forth Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 181–189, Le Croisic, France, April 2002.

[33] E. Gaudin. *Contribution de la programmation par contraintes au transport : définition et résolution d'un modèle complexe de gestion de flotte.* PhD thesis, Université Paris 7, 1997.

[34] M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Séguin. Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-Ups and Deliveries. Publication CRT-98-10, Centre de recherche sur les transports, Université de Montréal, Montréal, 1998.

[35] M. Gendreau, F. Guertin, J.Y. Potvin, and E. Taillard. Tabu Search for Real-Time Vehicle Routing and Dispatching. Technical Report CRT-96-47, Centre de Recherche sur les Transports, Université de Montréal, 1996.

[36] M. Gendreau, A. Hertz, and G. Laporte. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research*, 40 :1086–1094, 1992.

[37] M. Gendreau, A. Hertz, G. Laporte, and M. Stan. A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. *Operations Research*, 43 :330–335, 1995.

[38] M. Gendreau and J.-Y. Potvin. Dynamic Vehicle Routing and Dispatching. *Fleet Management and Logistics*, pages 115–126, 1998.

[39] M. Ginsberg and W. Harvey. Iterative Broadning. *Artificial Intelligence*, 55 :367–383, 1992.

[40] F. Glover. Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems. *Discrete Applied Mathmatics*, 65 :223–253, 1996.

[41] W. Harvey and M. Ginsberg. Limited Discrepancy Search. In *Proc. of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 607–615, Montréal, Canada, 1995. Morgan Kaufmann.

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

[43] S. Ichoua. *Problème de gestion de flottes de véhicules en temps réel.* PhD thesis, Université de Montréal, 2001.

[44] ILOG S.A., 12, Avenue Raspail, BP7, 94251 Gentilly Cedex, France. *ILOG SOLVER : Object-Oriented Constraint Programming*, 1995.

[45] ILOG S.A., 12, Avenue Raspail, BP7, 94251 Gentilly Cedex, France. *ILOG DISPATCHER*, 2001.

[46] J. Jaffar and J-L. Lassez. Constraint Logic Programming. In *Proceedings of 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, Germany, January 1987.

[47] J. Jaffar and M. J. Maher. Constraint Logic Programming : A Survey. *Journal of Logic Programming*, 19/20 :503–581, 1994.

[48] U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, and M. Sellmann. A Framework for Constraint Programming Based Column Generation. In *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 261–274, 1999.

[49] P. Kilby, P. Prosser, and P. Shaw. Guided Local Search for the Vehicle Routing Problem with Time Windows. In *META-HEURISTICS Advances and Trends in Local Search Paradigms for Optimization*, pages 473–486. Kluwer Academic Publishers, 1999.

[50] D. Knuth. Estimating the Efficiency of Backtrack Programs. *Mathematics of Computations*, 29(129) :121–136, 1975.

[51] N. Kohl and O.B.G. Madsen. An Optimization Algorithm for the Vehicle Routing Problem with Time Windows Based on Lagrangian Relaxation. *Operation Research*, 45(3) :395–407, 1997.

[52] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, second edition, 1987.

[53] L. Lobjois and M. Lemaître. Branch and Bound Algorithm Selection by Performance Prediction. In *American Association for Artificial Intelligence – AAAI*, pages 353–358, Madison,USA, 1998.

[54] L. Lobjois and M. Lemaîtres. Adaptation de l'estimateur de Knuth aux problèmes d'optimisation combinatoires. In *Rencontres Jeunes Chercheurs en Intelligence Artificielle -RJCIA*, pages 171–179, Toulouse, France, 1998.

[55] O.B.G. Madsen, H.F. Ravn, and J.M. Rygaard. A Heuristic Algorihtm for a Dial-a-Ride Problem with Time Windows, Multiple Capacities and Mulptiple Objectives. *Annals of Operations Research*, 61 :213–226, 1995.

[56] K. Mariott and P.J. Stuckey. *Programming with Constraints.* The MIT Press, 1998.

[57] P. Meseguer and T. Walsh. Interleaved and Discrepancy Based Search. In *ECAI 98*, pages 239–243, Brighton, UK, 1998.

[58] A. Mingozzi, M.A. Boschetti, S. Ricciardelli, and L. Bianco. A Set Partioning Approach to the Crew Scheduling Problem. *Operations Research*, 47(6) :873–888, 1999.

[59] N. Mladenovic and P. Hansen. Variable Neighbourhood Search. *Computer and Operations Research*, 24 :1097–1100, 1997.

[60] G. Pesant and M. Gendreau. A View of Local Search in Constraint Programming. In *Principles and Practice of Constraint Programming – CP96 : Proceedings of the Second International Conference*, pages 353–366. Springer-Verlag LNCS 1118, 1996.

[61] G. Pesant and M. Gendreau. A Constraint Programming Framework for Local Search Methods. *Journal of Heuristics*, 5 :255–279, 1999.

[62] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows. *Transportation Science*, 32 :12–29, 1998.

[63] G. Pesant, M. Gendreau, and J.-M. Rousseau. GENIUS-CP : a Generic Vehicle Routing Algorithm. In *Principles and Practice of Constraint Programming – CP97 : Proceedings of the Third International Conference*, pages 420–434. Springer-Verlag LNCS 1330, 1997.

[64] H.N. Psaraftis. Dynamic Vehicle Routing Problems. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing : Methods and Studies*, pages 223–248. North-Holland, 1980.

[65] H.N. Psaraftis. Dynamic vehicle routing : Status and prospects. *Annals of Operations Research*, (61) :143–164, 1995.

[66] J.-F. Pujet. A C++ Implementation of CLP. Technical Report 94-01, ILOG S.A., 12, Avenue Raspail, BP7, 94251 Gentilly Cedex, France, 1994.

[67] P. Purdom. Tree Size by Partial Backtracking. *SIAM Journal on Computing*, 7(4) :481–491, 1978.

[68] L. Qiu and W. Hsu. Scheduling and routing algorithms for agvs : a survey, 1999.

[69] F. Refalo. Tight Cooperation and its Application in Piecewise Linear Optimization. In *Principles and Practice of Constraint Programming -CP99*, Lecture Notes in Computer Science, pages 375–389. Springer-Verlag, 1999.

[70] F. Refalo. Linear Formulation of Constraint Programming Models and Hybrid Solver. In *Principles and Practice of Constraint Programming -CP2000*, Lecture Notes in Computer Science, pages 369–383. Springer-Verlag, 2000.

[71] J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, 1994.

[72] C. Rego and C. Roucairol. A Parallel Tabu Search Algorithm Using Ejection Chain for the Vehicle Routing Problem. *Metaheuristics : Theory and Applications*, pages 661–675, 1996.

[73] Y. Rochat and É. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1 :147–167, 1995.

[74] S. Roy, J.M. Rousseau, G. Lapalme, and J. Ferland. Routing and Scheduling of Transportation Services for the Disabled : Summary Report. Technical Report CRT-473A, Centre de Recherche sur les Transports, Université de Montréal, 1984.

[75] M.W.P. Savelsbergh. Local Search in Routing Problems with Time Windows. *Annals of Operations Research*, 4 :285–305, 1985.

[76] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Principles and Practice of Constraint Programming - CP98*, Lecture Notes in Computer Science 1520, pages 417–431, Pisa, Italy, October 1998. Springer-Verlag.

[77] M. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35 :254–265, 1987.

[78] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31 :170–186, 1997.

[79] P. Thompson and Psaraftis H.N. Cyclic Transfer Algorithms for Multi-Vehicle Routing and Scheduling Problems . *Operations Research*, 41 :935–946, 1993.

[80] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, Mass., 1989.

[81] P. Van Hentenryck. *The OPL Optimization Programming Language.* MIT Press, 1999.

[82] M. Wallace. Practical Applications of Constraint Programming. *Constraints*, 1 :139–168, 1996.

[83] T Walsh. Depth-Bounded Discrepancy Search. In *International Joint Conferences on Artificial Intelligence – IJCAI*, pages 1388–1395, Nagoya,Japan, 1997.

[84] N.H.M. Wilson and N.H. Colvin. Computer Control of Rochester Dial-a-Ride System. Technical Report R-77-30, Departement of Civil Engineering, Massachusetts Institute of Technology, 1977.

[85] N.H.M. Wilson, J.M. Sussman, H.K. Wang, and B.T. Higonnet. Scheduling Algorithms for Dial-a-Ride Systems. Technical Report USL TR-70-13, Urban Sustems Laboratory, Massachusetts Institute of Technology, 1971.

[86] N.H.M. Wilson and H. Weissberg. Adavance Dial-a-Ride Algorithms Research Project : Final Report. Technical Report TR-76-20, Departement of Civil Engineering, Massachusetts Institute of Technology, 1976.