Université de Montréal

Étude d'un problème de tournées de véhicules sur les arcs avec contraintes de capacité et coûts de service dépendants du temps

par

Mariam Tagmouti

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 option recherche opérationnelle

Décembre 2008

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée:

Étude d'un problème de tournées de véhicules sur les arcs avec contraintes de capacité et coûts de service dépendants du temps

présentée par:

Mariam Tagmouti

a été évaluée par un jury composé des personnes suivantes:

Jacques Ferland

(président-rapporteur)

Jean-Yves Potvin

(directeur de recherche)

Michel Gendreau

(co-directeur)

Bernard Gendron

(membre du jury)

Richard W. Eglese

(examinateur externe)

Thèse acceptée le:

Résumé

Dans le cadre de cette thèse, nous nous intéressons au problème de tournées de véhicules sur les arcs avec contraintes de capacité et coûts de service variables. La fonction de coût de service associée à chaque arc requis est une fonction linéaire par morceaux ayant un intervalle optimal où le service est le moins coûteux. Ce problème est motivé par les opérations de déglaçage où le moment de l'intervention est crucial. En effet, un épandage de matière abrasive qui a lieu trop tôt ou trop tard entraîne des coûts additionnels en matériel et en temps.

Pour résoudre ce problème, nous faisons appel en premier lieu à une méthode exacte de génération de colonnes qui est appliquée à une transformation du problème original en un problème équivalent de tournées sur les sommets. Dans un second travail, nous travaillons directement sur le problème de tournées sur les arcs à l'aide d'une heuristique de descente à voisinage variable. Les mouvements, dans le voisinage d'une solution, sont des échanges d'arcs ou de blocs d'arcs entre les tournées. En dernier lieu, nous nous intéressons à l'aspect dynamique du problème de déglaçage où de nouveaux rapports météorologiques peuvent modifier les données du problème en temps réel. Nous avons donc introduit, pour la première fois, un problème dynamique de tournées sur les arcs. Pour le résoudre, nous proposons une méthodologie où nous réoptimisons une suite de problèmes statiques de plus petite taille. Ici, un nouveau problème statique est généré chaque fois qu'un nouveau rapport météorologique est reçu.

Mots clés. Problème de tournées sur les arcs, coût variable, génération de colonnes, descente à voisinage variable, problèmes dynamiques.

Abstract

In this thesis, we introduce a capacitated arc routing problem with time dependent service costs. This problem is motivated from winter gritting applications, where the timing of each intervention is crucial. The cost function on each required arc is thus a piecewise linear function with an optimal time interval for service. That is, the service cost is minimum within that interval.

To solve the problem, we first propose an exact column generation approach based on a transformation of the original problem into a node routing problem. We also directly solve the arc routing problem with a variable neighborhood descent heuristic. The latter is based on neighborhood structures that move arcs or sequences of arcs. Finally, we introduce a dynamic arc routing problem with time dependent service costs. The dynamic aspect comes from possible changes in the optimal service time intervals when new weather forecasts are received in real time. The dynamic problem is addressed by solving a series of smaller static problems, where a new static problem is defined each time an updated weather forecast is received.

Keywords. Arc routing problem, variable service costs, column generation, variable neighborhood descent, dynamic problems.

Table des matières

Liste des tableaux

Liste des figures

Remerciements

Mes remerciements vont, en premier lieu, à mes directeurs de recherche, les professeurs Jean-Yves Potvin et Michel Gendreau, pour m'avoir fait confiance en acceptant de diriger cette thèse. Leur grande disponibilité, leur enthousiasme et leurs judicieux conseils m'ont permis de travailler dans les meilleures conditions et de me surpasser pour accomplir ce travail. Mes remerciements vont aussi à tous les membres du jury pour avoir accepté d'évaluer ce travail.

Je tiens également à remercier tous les membres de l'équipe du CIRRELT (*Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport*) et du DIRO (*Département d'informatique et de recherche opérationnelle*), les étudiants, les employés, ainsi que les professeurs pour leur accueil, leur soutien, leur bonne humeur et leurs encouragements durant toutes ces années. C'est un peu grâce à eux aussi que j'ai pu terminer ce travail de longue haleine.

Mes remerciements s'adressent, enfin, à mes parents, qui grâce à leur support et leurs encouragements permanents, m'ont donné la chance de faire mes études dans une des plus grandes universités au monde. Je remercie également ma petite famille et plus particulièrement mon mari Badr et ma petite fille Sara, ainsi que mes soeurs et mon frère qui ont toujours été à mes côtés.

Chapitre 1

Introduction

De nombreuses études en recherche opérationnelle portent sur les problèmes de tournées de véhicules dans lesquels il faut desservir un certain nombre de clients dans un réseau urbain ou rural. Ce réseau est en géneral représenté par un graphe dont les sommets sont les intersections des rues et les arcs sont les portions de rues entre deux intersections. Par ailleurs, l'orientation des rues nous oblige à utiliser des graphes orientés ou mixtes.

Soit un graphe $G = (V, A \cup E)$ où $V = \{v_1, v_2, .., v_n\}$ est l'ensemble des sommets, $A = \{(v_i, v_j), v_i, v_j \in V\}$ est l'ensemble des arcs et $E = \{(v_i, v_j), i > j, v_i, v_j \in V\}$ est l'ensemble des arêtes du graphe. Si A et E sont tous les deux non vides alors le graphe G est mixte. Autrement, le graphe est orienté si $A \neq \emptyset$ et non orienté si $E \neq \emptyset$. On retrouve aussi un coût c_{ij} associé à chaque arête (ou arc) (v_i, v_j), qui peut être tout simplement la distance entre v_i et v_j.

Les problèmes de tournées de véhicules se divisent en deux grandes catégories : les problèmes de tournées de véhicules sur les sommets et les problèmes de tournées de véhicules sur les arcs (ou arêtes). La première catégorie regroupe les problèmes où la demande est localisée sur les sommets du graphe. Parmi les applications pratiques qui se modélisent comme des problèmes de tournées sur les sommets, on retrouve la livraison

par courrier spécial, les soins à domicile, le transport adapté, et bien d'autres. Le problème de base, et celui qui a été le plus étudié dans la littérature, est le problème du voyageur de commerce (PVC) qui consiste à construire un cycle (ou circuit) hamiltonien de coût minimum passant exactement une fois par chaque sommet du graphe. Ici, on distingue deux cas : si $c_{ij} = c_{ji}$ pour tout $v_i, v_j \in V$ alors nous avons un problème symétrique. Dans le cas contraire, le problème est dit asymétrique.

La deuxième catégorie regroupe les problèmes où la demande est située sur les arcs ou arêtes du graphe. Parmi les applications pratiques qui se modélisent comme des problèmes de tournées sur les arcs (PTA), on retrouve la distribution du courrier par les postiers où la demande est répartie sur des segments de rues, le nettoyage des rues, le déglaçage, le déneigement, et bien d'autres. Le problème de base est le problème du postier chinois (PPC) où chaque arc ou arête du graphe doit être desservi au moins une fois. Si le graphe est orienté ou non orienté, le PPC se résout en temps polynomial. Par contre, si le graphe est mixte le PPC est NP-difficile. Nous reparlerons plus en détail de ces problèmes et de leurs variantes au chapitre 2.

1.1 Définition du problème

La motivation pour cette thèse provient des opérations de déglaçage. Dans ce travail, nous supposons que le graphe qui représente le réseau routier sous-jacent est orienté. Si une portion de rue doit être desservie dans les deux sens, ce qui est le cas le plus fréquent, une demande est associée à chacun des deux arcs de sens opposé (et ces deux arcs peuvent être desservis par des véhicules différents). De tels arcs avec une demande associée sont dit requis car ils doivent être desservis. En pratique, le traitement d'un arc peut parfois nécessiter plusieurs passages, mais nous considérons ici que chaque arc requis n'est desservi qu'une seule fois. Toutefois, des passages à vide peuvent avoir lieu sur cet arc.

Le problème de déglaçage, tel qu'on le retrouve en pratique, est très complexe et contient un grand nombre de contraintes, parfois difficilement quantifiables. Le problème que nous considérons est donc simplifié et constitue une abstraction du problème réel.

Une première contrainte qui est tenue en compte correspond à la capacité des véhicules. En effet, si un véhicule s'apprête à desservir un arc avec un produit déglaçant ou abrasif, la demande sur cet arc ne doit pas dépasser la quantité disponible dans le véhicule (e.g., quantité de sel). À tout moment, la quantité disponible doit donc se situer entre 0 et Q, où Q est la capacité du véhicule.

On retrouve également des contraintes de nature temporelle sur chacun des arcs requis. En effet, l'heure du début de service sur chaque arc requis est très importante et affecte beaucoup le coût du service. Un épandage trop hâtif ou trop tardif (par rapport au passage prévu d'une tempête ou d'une chute de température) implique une plus grande quantité de sel à épandre, une circulation plus lente sur les arcs ou la nécessité de desservir l'arc plusieurs fois. Ces considérations sont inclues dans une fonction de coût de service qui dépend de l'heure de début de service sur l'arc. Cette fonction possède en particulier un intervalle de temps optimal où le service est le moins coûteux. En arrivant avant ou après cet intervalle de temps, un coût plus élevé est encouru qui dépend du moment de début de service.

Le moment idéal d'intervention sur un arc peut être spécifié sur la base de rapports météorologiques qui nous donnent des informations sur le déplacement d'une tempête de pluie verglaçante, d'une chute de neige ou encore d'une chute subite de température. Il peut aussi dépendre des priorités associées aux différentes portions de rue. Ainsi, les autoroutes, les artères principales et les abords des écoles et arrêts d'autobus doivent être desservis en premier.

On voit donc que le problème étudié dans le cadre de cette thèse est un problème de tournées de véhicules sur les arcs avec contraintes de capacité et coûts de service

variables selon le moment de début du service.

De façon plus formelle, nous disposons d'un graphe orienté $G = (V, A)$ où V est l'ensemble des sommets et A l'ensemble des arcs. Ce dernier ensemble est divisé en deux sous-ensembles : A_1 , qui est l'ensemble des arcs requis et A_2, l'ensemble des arcs non requis. Chaque arc $e \in A_1$ a une demande d_e, une longueur l_e, un temps de parcours tt_e, un temps de service st_e, un coût de parcours tc_e et un coût de service $sc_e(T_e)$. Ce dernier correspond à une fonction linéaire par morceaux qui dépend de T_e, où T_e est le moment de début de service sur l'arc e. Les arcs de A_2 ont une longueur, un temps de parcours et un coût de parcours seulement.

Nous disposons d'un ensemble de véhicules de même capacité Q. Les véhicules sont localisés à un dépôt central d'où ils commencent et terminent leur route. On suppose aussi que les véhicules commencent le service dès qu'ils arrivent sur un arc (i.e., aucune attente n'est permise).

On veut donc desservir tous les arcs requis à moindre coût. Le coût total d'une route est la somme des coûts de service et des coûts de parcours. Soit la route r_k desservie par le véhicule k composée des arcs requis $(e_1, ..., e_l)$ et des arcs non requis $(e_{l+1}, ..., e_{l+p})$. Le coût total de cette route peut s'exprimer comme une fonction du moment T_0^k où le véhicule k quitte le dépôt pour exécuter sa route (puisque les temps de début de service à chacun des arcs requis se dérivent directement de T_0^k) :

$$C(r_k) = \sum_{i=1}^{l} sc_{e_i}(T_0^k) + \sum_{i=l+1}^{l+p} tc_{e_i} = sc_k(T_0^k) + tc_k. \tag{1.1}$$

La figure 1.1 montre des exemples de fonctions de coût de service pour notre problème. Toutes ces fonctions doivent être linéaires par morceaux. Dans nos tests, nous avons utilisé des fonctions du type illustré en (b).

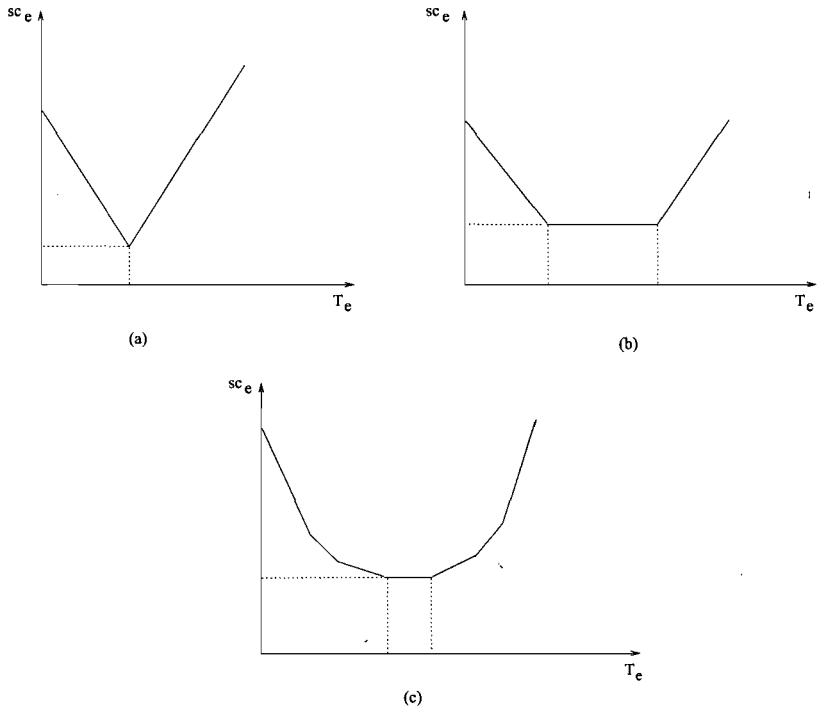

FIG. 1.1 – Différentes fonctions de coût de service

1.2 Méthodes de résolution

Pour résoudre notre problème, nous avons d'abord développé une méthode exacte. Nous avons transformé notre problème en un problème équivalent de même taille, mais où la demande est localisée sur les sommets d'un graphe transformé. Chaque arc requis correspond à un sommet dans le nouveau graphe et les liens entre les sommets correspondent aux plus courts chemins entre l'extrémité terminale du premier arc et l'extrémité initiale du second arc dans le graphe original. La méthode exacte est de type génération de colonnes où le problème maître est un problème de recouvrement et les sous-problèmes sont des problèmes de plus courts chemins élémentaires avec contraintes de ressources. L'article sur ce travail a été publié dans *European Journal of Operational Research* en 2007 et fait l'objet du chapitre 3.

Pour le deuxième travail, nous avons développé une heuristique de type descente à voisinage variable. Cette fois, nous avons travaillé directement sur le problème avec

demandes sur les arcs. Les mouvements dans le voisinage d'une solution sont des échanges d'arcs ou de blocs d'arcs entre les tournées. Nous avons aussi une procédure d'amélioration de tournées, appelée «Shorten». L'heuristique est simple et rapide. Elle a permis de trouver de bons résultats par rapport à une heuristique récente proposée pour un problème similaire de tournées de véhicules avec pénalités temporelles. L'article sur ce travail est présenté au chapitre 4. Cet article a été soumis à *Computers & Industrial Engineering* en janvier 2009.

Dans toutes les instances utilisées pour tester les deux premières méthodes, les intervalles de temps optimaux ont été spécifiés manuellement ou en utilisant des solutions dans la littérature. En réalité, ces intervalles de temps sont souvent déterminés sur la base de rapports météorologiques. Dans le dernier article, nous nous sommes rapprochés du problème réel de déglaçage en exploitant cet aspect dynamique. En effet, en supposant que l'on dispose d'informations en temps réel sur la vitesse et la position courante d'une tempête (une tempête de verglas, par exemple), il est possible d'exploiter ces informations pour faire de nouvelles prévisions quant au moment idéal de visite des arcs requis qui n'ont pas encore été visités. Au chapitre 5, nous présentons en détail l'article portant sur ce travail qui a été soumis à *Transportation Research Part : C* en février 2009.

Chapitre 2

Revue de la littérature

Dans ce chapitre, nous présentons une revue du problème de tournées sur les arcs et de ses variantes, ainsi que certaines de ses applications pratiques.

2.1 Problèmes de tournées sur les arcs (PTA)

Cette classe comporte les problèmes où on doit desservir des arcs dans un graphe $G = (V, A)$ où V est l'ensemble des sommets et A est l'ensemble des arcs. Il existe aussi des versions pour des graphes non orientés ou mixtes où l'on doit desservir des arêtes. Les problèmes les plus simples n'impliquent qu'une tournée qui doit visiter tous les arcs du graphe à coût minimum et sans contraintes additionnelles. La présence ou non d'un circuit (ou cycle) Eulérien dans le graphe est une propriété importante pour la résolution de tels problèmes, comme il est expliqué dans la section suivante.

2.1.1 Graphe Eulérien

Les problèmes de tournées sur les arcs originent du problème des ponts de Konigsberg, formulé par Euler en 1736. En mémoire de ce dernier, un graphe est dit Eulérien s'il contient un circuit (ou cycle) Eulérien qui passe exactement une fois par chacun des

arcs (ou arêtes) du graphe. Les conditions nécessaires et suffisantes pour qu'un graphe G connexe soit Eulérien dépendent du type de graphe [35] :

- Si le graphe est non orienté, chaque sommet doit être de degré pair (i.e., le nombre d'arêtes incidentes à chaque sommet doit être pair).

- Si le graphe est orienté, le nombre d'arcs entrants doit être égal au nombre d'arcs sortants à chaque sommet. De plus, le graphe doit être fortement connexe.

- Si le graphe est mixte, chaque sommet doit être incident à un nombre pair d'arcs et d'arêtes. De plus, pour chaque sous-ensemble S de sommets ($S \subseteq V$), la différence en valeur absolue entre le nombre d'arcs allant de S à $V - S$ et le nombre d'arcs allant de $V - S$ à S doit être inférieur ou égal au nombre d'arêtes entre S et $V - S$.

Plusieurs algorithmes pour trouver un cycle ou circuit Eulérien ont été publiés. On trouve par exemple l'algorithme "end-pairing" proposé par Edmonds et Johnson [26] pour des graphes non orientés. Cet algorithme, qui est décrit ci-dessous, est polynomial.

Algorithme "end-pairing" :

Étape 1 : Tracer un cycle simple (i.e., qui ne passe pas plus d'une fois par un même sommet) en partant d'un sommet quelconque du graphe. Si toutes les arêtes sont visitées, STOP. Sinon, aller à l'étape 2.

Étape 2 : Considérer un nouveau sommet v dans le tour déjà tracé $(...v_1, v, v_2, ...)$ où v doit être incident à une arête non encore visitée. Ce sommet existe car le graphe est connexe. Tracer à partir de ce sommet un deuxième cycle simple $(..., v_3, v, v_4, ...)$ ne contenant que des arêtes non visitées.

Étape 3 : Fusionner les deux tours pour obtenir un tour unique $(..., v_1, v, v_4, ..., v_3, v, v_2, ...)$ et retourner à l'étape 2 tant qu'on n'a pas visité toutes les arêtes.

Pour les graphes orientés, on trouve un algorithme similaire à l'algorithme "end-pairing", sauf qu'ici on respecte l'orientation des arcs. Pour les graphes mixtes, la recherche se fait en deux étapes. Premièrement, une orientation est attribuée aux arêtes afin que le nombre d'arcs sortants soit égal au nombre d'arcs entrants à chaque sommet. Ensuite, un circuit Eulérien est identifié dans le graphe orienté ainsi obtenu.

2.1.2 Le problème du postier chinois (PPC)

Ce problème a été formulé par Mei-Ko Kwan (1962) [70]. L'objectif est de trouver un cycle ou circuit de coût minimum, passant par toutes les arêtes ou arcs d'un graphe non nécessairement Eulérien.

Les algorithmes pour résoudre le problème du postier chinois consistent à augmenter le graphe en ajoutant des arêtes ou arcs à coût minimum afin d'obtenir un graphe Eulérien. Pour ce faire, il faut résoudre un problème d'augmentation minimale qui a pour objectif de minimiser la somme des longueurs des liens qui sont ajoutés. Le problème d'augmentation minimale se résout en temps polynomial si le graphe est orienté ou non orienté. Par contre, si le graphe est mixte, le problème est NP-difficile. Nous distinguons ici le cas orienté, non orienté, et mixte.

Cas non orienté

Le problème d'augmentation minimale est résolu sur un graphe qui ne contient que les sommets de degré impair du graphe original. Ce graphe contient également une arête entre chaque couple de sommets distincts, dont la longueur correspond à la longueur de la chaîne plus courte entre les deux sommets dans le graphe original. Le problème d'augmentation correspond alors à un problème de couplage de coût minimal. Une fois ce problème résolu, les arêtes faisant partie des chaînes plus courtes dans le couplage optimal sont ajoutées au graphe original qui devient Eulérien (et qui peut contenir plusieurs copies d'une même arête). On identifie alors un cycle d'Euler dans ce graphe, qui correspond à la solution du problème. Le problème d'augmentation se formule de la façon suivante. Soit :

c_{ij} = coût (longueur) de l'arête (v_i, v_j).

x_{ij} = nombre d'arêtes ajoutées entre v_i et v_j.

$s(i)$ = ensemble des arêtes incidentes à v_i.

T = ensemble des sommets de degré impair.

Nous avons :

$$\begin{cases} \text{Minimiser} \sum_{(v_i,v_j)\in s(i)} c_{ij}x_{ij} \\ \textit{Sous les contraintes} \\ \sum_{(v_i,v_j)\in E} x_{ij} = \begin{cases} 1\ (mod\ 2) & si\ \ v_i \in T \\ 0\ (mod\ 2) & si\ \ v_i \in V - T \end{cases} \qquad (1) \\ x_{ij} \in \{0,1\} \end{cases}$$

La contrainte (1) signifie que si un sommet v_i est de degré impair alors il faut ajouter un nombre impair d'arêtes incidentes à v_i. Sinon, on doit ajouter un nombre pair d'arêtes incidentes à v_i.

Cas orienté

Pour le cas orienté, le problème d'augmentation est défini sur les sommets dont le nombre d'arcs entrants est différent du nombre d'arcs sortants. Nous distinguons les sommets en déficit où le nombre d'arcs entrants est inférieur au nombre d'arcs sortants, et les sommets en surplus où le nombre d'arcs sortants est inférieur au nombre d'arcs entrants. L'objectif est d'ajouter des arcs à coût minimum entre les sommets en surplus et les sommets en déficit de façon à ce qu'ils soient en équilibre. Le problème d'augmentation correspond ici à un problème de transport où les sommets en surplus sont des sources et les sommets en déficit des puits.

Cas mixte

Pour les graphes mixtes, le problème d'augmentation minimale est NP-difficile. Il existe pour le résoudre des méthodes exactes comme la méthode de Nobert et Picard

[73] qui est de type branch-and-cut, et des heuristiques comme celle de Frederickson [36].

2.1.3 Variantes du problème du postier chinois

Parmi les variantes du PPC, on retrouve le problème du postier chinois incliné (Windy Postman Problem) et le problème du postier chinois hiérarchique. Le premier problème a pour objectif de déterminer un cycle de coût minimum en sachant qu'il y a un coût différent associé à chaque arête qui dépend de la direction selon laquelle l'arête est traversée. Le problème du postier chinois mixte est un cas particulier de ce problème et il se résout en temps polynomial si le graphe est Eulérien [88]. Le deuxième problème consiste à déterminer un cycle dans un graphe où il y a un ordre partiel de visite prédéfini entre les arêtes qui doit être respecté. Ce problème n'a pas été beaucoup étudié dans la littérature, mais il existe néanmoins quelques algorithmes pour le résoudre [25, 37].

Dans certaines variantes, il n'est pas nécessaire de visiter toutes les arêtes ou arcs du graphe. Le problème du postier rural (PPR), par exemple, a pour objectif de trouver un cycle ou circuit de moindre coût desservant un sous-ensemble d'arêtes ou d'arcs du graphe. Parmi les méthodes pour résoudre ce problème, citons l'algorithme de Ghiani et Laporte [40] qui est de type branch-and-cut et l'heuristique de Frederickson [36].

2.1.4 Problèmes avec pénalités sur les virages

Pour certains problèmes, il est important d'attribuer des pénalités sur les virages, comme les virages à gauche et en "U". Ces problèmes apparaissent en particulier pour le nettoyage et le déneigement des rues. Pour l'opération de déneigement par exemple, une déneigeuse doit préférablement tourner à droite plutôt que de tourner à gauche ou de continuer tout droit. Ceci s'explique par le fait qu'en continuant tout droit, la déneigeuse laisse derrière elle des bancs de neige au milieu de la route, ce qui nécessite un ou plusieurs autres passages pour assurer la sécurité des automobilistes. La même

situation se présente si la déneigeuse tourne à gauche. Il est donc important de tenir compte des virages en leur attribuant des pénalités.

Bodin et Kursh [7] ont amélioré une procédure proposée par Beltrami et Bodin [5] pour les tournées de balais mécanisés qui engendrait plusieurs virages à gauche et en «U». Ils ont ajouté des pénalités sur les virages et ont résolu un problème d'affectation visant à minimiser la somme de ces pénalités. Roy et Rousseau [81] ont développé une approche pour les routes des facteurs de la Société Canadienne des Postes en modélisant le problème comme un PPC avec contraintes de capacité et pénalités de virages. Enfin, Clossey et al. [19] ont adapté l'algorithme "end-pairing" pour obtenir une heuristique pour le PPC qui inclut les pénalités sur les virages.

Tous les problèmes précédemment cités sont des problèmes sans contraintes. Dans la réalité, il y a souvent des contraintes d'opérations qui doivent être prises en considération, comme les contraintes de capacité des véhicules, les contraintes de durée de chaque tournée et bien d'autres. Nous nous y attardons dans les sections suivantes.

2.2 Le problème de tournées sur les arcs avec contraintes de capacité (PTAC)

Ce problème, introduit par Golden et Wong [45], est un des problèmes les plus étudiés parce qu'il modélise de nombreuses situations pratiques. Ici, chaque arc ou arête (v_i, v_j) a une demande non négative q_{ij}, et nous disposons d'une flotte de m véhicules, chacun de capacité Q. L'objectif est de déterminer une tournée pour chaque véhicule tel que le coût total soit minimum et que tous les arcs ou arêtes de demande positive soit desservis. Les tournées doivent contenir le dépôt et la demande totale doit être inférieure ou égale à la capacité des véhicules. On note qu'un véhicule peut passer sur un arc ou arête sans le desservir, que seuls ceux ayant une demande positive sont desservis et que chacun d'entre eux est desservi par un seul véhicule.

2.2.1 La formulation du PTAC

Pour le PTAC il y a deux formulations mathématiques pour le cas non orienté. Une formulation avec des variables orientées proposée par Golden et Wong [45] et une autre avec des variables non orientées proposée par Belenguer et Benavent [4]. Dans la suite, nous utilisons la notation suivante :

$$E(S) = \{(v_i, v_j);\ v_i \in S\ et\ v_j \in V - S\ ou\ v_i \in V - S\ et\ v_j \in S\};$$

$$E^+(S) = E(S) \cap \{(v_i, v_j) \in E, q_{ij} > 0\};$$

Ici, $E(S)$ est l'ensemble des arêtes liant un sommet de S à un sommet de $V - S$ tandis que $E^+(S)$ est le sous-ensemble d'arêtes devant être desservies dans $E(S)$.

Chaque arête (v_i, v_j) qui a une demande q_{ij} positive doit être desservie alors que toute arête qui a une demande nulle peut être traversée au besoin. Nous supposons que le dépôt correspond au sommet v_1.

PTAC «formulation non orientée» :

Soient x_{ijk} et y_{ijk} les variables de décision suivantes :

x_{ijk} = nombre de fois qu'on traverse l'arête (v_i, v_j) avec le véhicule k sans la desservir.

$$y_{ijk} = \begin{cases} 1 & \text{si l'arête } (v_i, v_j) \text{ est desservie par le véhicule } k \\ 0 & \text{sinon} \end{cases}$$

La formulation du problème est la suivante :

$$
\begin{cases}
\text{Minimiser} \displaystyle\sum_{k=1}^{m} \sum_{(v_i,v_j)\in E} c_{ij}(x_{ijk} + y_{ijk}) & & \\
\textit{Sous les contraintes} & & \\
\displaystyle\sum_{k=1}^{m} y_{ijk} = 1 & ((v_i, v_j) \in E \ \ et \ \ q_{ij} > 0) & (2) \\
\displaystyle\sum_{(v_i,v_j)\in E} q_{ij} y_{ijk} \leq Q & (k = 1, ..., m) & (3) \\
\displaystyle\sum_{(v_i,v_j)\in E(S)} x_{ijk} + \sum_{(v_i,v_j)\in E^+(S)} y_{ijk} \geq 2y_{hlk} & (S \subseteq V - \{v_1\}; \ \ S \neq \emptyset; \ \ k = 1, ..., m & (4) \\
 & v_h, v_l \in S \ \ et \ \ q_{hl} > 0) & \\
\displaystyle\sum_{(v_i,v_j)\in E(S)} x_{ijk} + \sum_{(v_i,v_j)\in E^+(S)} y_{ijk} = 2z_k^S & (S \subseteq V - \{v_1\}; \ \ S \neq \emptyset; \ \ k = 1, ..., m) & (5) \\
z_k^S \geq 0 \ \ entier & (S \subseteq V - \{v_1\}; \ \ S \neq \emptyset; \ \ k = 1, ..., m) & \\
x_{ijk} \geq 0 \ \ entier & ((v_i, v_j) \in E; \ \ k = 1, ..., m) & (6) \\
y_{ijk} \in \{0, 1\} & ((v_i, v_j) \in E; \ \ k = 1, ..., m) &
\end{cases}
$$

La contrainte (2) assure que seules les arêtes avec une demande positive sont desservies. Les contraintes sur les capacités des véhicules sont données par (3). La contrainte (4) signifie que si (v_h, v_l) est une arête desservie par le véhicule k telle que v_h, $v_l \in S$, alors S doit être connecté aux autres arêtes desservies par le véhicule k. La contrainte (5) assure le retour au dépôt si on visite un sous-ensemble de sommets S qui ne le contient pas en introduisant les variables entières z_k^S, $k = 1, ..., m$, $S \subseteq V - \{v_1\}$. Ceci s'explique par le fait que S doit être connecté un nombre pair de fois à $V - S$.

PTAC «formulation orientée» :

Ici les arêtes sont représentées par deux arcs et on introduit donc deux variables de décision x_{ijk} et x_{jik} pour chaque arête (v_i, v_j). Soient x_{ijk} et y_{ijk} des variables de décision binaires définies comme suit :

$$
x_{ijk} = \begin{cases} 1 & \text{si l'arc } (v_i, v_j) \text{ est traversé par le véhicule } k \\ 0 & \text{sinon} \end{cases}
$$

$$y_{ijk} = \begin{cases} 1 & \text{si l'arc } (v_i, v_j) \text{ est desservi par le véhicule } k \\ 0 & \text{sinon} \end{cases}$$

La formulation du problème est la suivante :

$$\begin{cases}
\text{Minimiser} \sum_{k=1}^{m} \sum_{(v_i,v_j)\in A} c_{ij} x_{ijk} & \\
\textit{Sous les contraintes} & \\
\sum_{(v_j,v_i)\in A} x_{jik} - \sum_{(v_i,v_l)\in A} x_{ilk} = 0 & ((v_i \in V;\ \ k = 1,...,m) \quad (7) \\
\sum_{k=1}^{m} (y_{ijk} + y_{jik}) = \begin{cases} 0 & si\ \ q_{ij} = 0 \\ 1 & si\ \ q_{ij} > 0 \end{cases} & ((v_i, v_j) \in A) \quad (8) \\
x_{ijk} \geq y_{ijk} & ((v_i, v_j) \in A;\ \ k = 1,...,m) \quad (9) \\
\sum_{(v_i,v_j)\in A} q_{ij} y_{ijk} \leq Q & (k = 1,...,m) \quad (10) \\
\sum_{v_i,v_j \in S} x_{ijk} \leq card(S) - 1 + n^2 u_k^S & (S \subseteq V - \{v_1\};\ \ S \neq \emptyset;\ \ k = 1,...,m) \quad (11) \\
\sum_{v_i \in S} \sum_{v_j \notin S} x_{ijk} \geq 1 - w_k^S & (S \subseteq V - \{v_1\};\ \ S \neq \emptyset;\ \ k = 1,...,m) \quad (12) \\
u_k^S + w_k^S \leq 1 & (S \subseteq V - \{v_1\};\ \ S \neq \emptyset;\ \ k = 1,...,m) \quad (13) \\
u_k^S, w_k^S \in \{0,1\} & (S \subseteq V - \{v_1\};\ \ S \neq \emptyset;\ \ k = 1,...,m) \quad (14) \\
x_{ijk}, y_{ijk} \in \{0,1\} & ((v_i, v_j) \in A;\ \ k = 1,...,m) \quad (15)
\end{cases}$$

La contrainte (7) est une contrainte de conservation de flot pour chaque sommet $v_i \in V$, qui assure la continuité des routes. La contrainte (8) assure un service unique pour les arcs qui ont une demande positive. La contrainte (9) signifie que si une route est desservie par un véhicule k ($y_{ijk} = 1$) alors elle est traversée par ce véhicule ($x_{ijk} = 1$). La contrainte sur la capacité des véhicules est donnée par (10). Les inégalités (11), (12), (13) et (14) obligent la solution à ne contenir que des tours connectés au dépôt en ajoutant les variables binaires u_k^S et w_k^S, $k = 1,...,m$, $S \subseteq V - \{v_1\}$, . En effet, tout cycle contenant un sous-ensemble $S \subseteq V - \{v_1\}$ de sommets visités par le véhicule k

doit être connecté à $V - S$ (et donc au dépôt) puisque : $\sum_{v_i,v_j \in S} x_{ijk} > card(S) - 1 \Rightarrow$ $u_k^S = 1 \Rightarrow w_k^S = 0 \Rightarrow \sum_{v_i \in S} \sum_{v_j \notin S} x_{ijk} \geq 1$. Dans les autres cas ces contraintes ne sont pas liantes et les variables u_k^S et w_k^S peuvent prendre des valeurs quelconques.

Contrairement à la formulation non orientée, cette formulation utilise des variables binaires seulement. Belenguer et Benavent [4] ont cependant utilisé leur formulation (non orientée) pour générer plusieurs inégalités valides. Ces inégalités ont été utilisées pour développer une méthode exacte de type branch-and-cut pour résoudre le PTAC.

Le problème de tournées sur les arcs avec contraintes de capacité sur un graphe orienté peut facilement être formulé en le transformant en un problème de tournées sur les sommets. Une telle transformation produit un problème de même taille que le problème original car chaque arc est représenté par un seul sommet dans le nouveau graphe. Il faut noter qu'il existe plusieurs formulations du problème de tournées sur les sommets avec contraintes de capacité dans la littérature [61].

2.2.2 Méthodes de résolution pour le PTAC

La plupart des algorithmes qui ont été proposés pour la résolution du PTAC sont des heuristiques. En effet, les méthodes exactes ne peuvent résoudre que des problèmes de petite taille. Parmi les méthodes exactes, citons l'algorithme proposé par Belenguer et Benavent [4] qui fait appel à une formulation relaxée du problème et qui a permis d'obtenir des bornes inférieures de qualité à l'aide d'une approche de type branch-and-cut. D'autres algorithmes exacts utilisent des approches de type branch-and-bound [53], mais elles ne peuvent résoudre que des problèmes sur des graphes de 30 arcs requis et moins.

La plupart des méthodes de construction pour le PTAC sont des heuristiques gloutonnes ou des heuristiques à deux phases. Les heuristiques à deux phases appartiennent à l'une des deux catégories suivantes :

Cluster First, Route Second

Ici les arcs sont partitionnés en m sous-ensembles, où m est le nombre de véhicules. Chaque sous-ensemble contient un nombre d'arcs dont la demande totale est inférieure ou égale à la capacité d'un véhicule. Ensuite, un problème de tournées sur les arcs pour un seul véhicule est résolu sur chaque sous-ensemble. Ceci revient à déterminer un cycle pour chaque sous-ensemble qui passe par tous ses arcs et qui est de coût minimum. Ceci peut se faire avec un algorithme pour le PPC ou le PPR.

Route First, Cluster Second

Ici, nous construisons un tour qui passe par tous les arcs qui ont une demande positive, soit par la résolution du PPC si le sous-graphe contenant ces arcs est connexe, ou par la résolution du PPR sinon. Le grand tour obtenu est ensuite divisé en sous-tours réalisables tenant compte des contraintes de capacité des véhicules.

Les heuristiques gloutonnes pour le PTAC sont efficaces pour construire (ou améliorer) des solutions. Parmi ces heuristiques, mentionnons les suivantes.

Construct-Strike

Cette procédure a été proposée par Christofides [17]. Elle consiste à construire des tours réalisables et à les retirer graduellement du graphe. L'algorithme s'exécute en répétant les étapes suivantes :

Étape 1 : Tant que le graphe est connexe, faire : (1) construire un tour réalisable (qui satisfait la contrainte de capacité) desservant des arêtes qu'on peut enlever du graphe sans le rendre non connexe ; (2) enlever du graphe les arêtes avec demande positive qui sont desservies dans le tour construit.

Étape 2 : Résoudre un problème de couplage entre les sommets de degré impair et ajouter les arêtes correspondantes au graphe.

Étape 3 : Retourner à l'étape 1 tant qu'il existe des arêtes non encore desservies.

Christofides n'a pas proposé de méthode pour construire les tours réalisables. En effet, il s'est basé sur une inspection visuelle dans l'exemple qu'il a utilisé pour ses tests. Cependant, l'approche utilisée dans la procédure Path-Scanning (décrite plus bas) pourrait être utilisée.

Pearn [75] a proposé une modification de la procédure Construct-Strike où il n'est pas nécessaire que le graphe obtenu, suite au retrait d'un tour, soit connexe. En effet, si le graphe induit G' n'est pas connexe, il suffit de considérer chaque composante connexe comme un sommet d'un graphe G'' et appliquer un algorithme d'arbre de recouvrement minimal pour obtenir des arêtes que nous ajoutons à G' afin de relier ses composantes.

Path-Scanning

Cet algorithme de construction pour le PTAC a été proposé par Golden et al. [45]. Pour former un tour, nous considérons une chaîne qui part du dépôt et à laquelle on ajoute successivement des arêtes de demande positive, selon un critère donné, jusqu'à ce que la capacité du véhicule soit atteinte. Le tour est complété avec la chaîne la plus courte menant du dernier sommet atteint au dépôt.

Différents critères ont été proposés pour le choix de la prochaine arête (v_i, v_j) :

1. Le rapport du coût sur la demande est minimal.
2. Le rapport du coût sur la demande est maximal.
3. La distance entre v_j et le dépôt est minimale.
4. La distance entre v_j et le dépôt est maximale.
5. Si le véhicule n'est pas à moitié rempli, la distance entre v_j et le dépôt est maximale. Sinon, cette distance doit être minimale.

Cet algorithme est facile à implanter et donne de bons résultats dans des temps de calcul raisonnables.

Augment-Merge

Cette heuristique a été proposée pour la première fois par Golden et Wong [45]. Elle a ensuite été améliorée par Golden et al. [46]. Elle est inspirée de la méthode des gains de Clarke et Wright [18] pour les problèmes de tournées de véhicules (PTV). La procédure se résume comme suit :

Étape 1 : Construire des tours individuels ne desservant chacun qu'une seule arête. Le tour pour une arête (v_i, v_j) est construit en identifiant la chaîne plus courte entre le dépôt et v_i, et ensuite entre v_j et le dépôt.

Étape 2 : Ordonner les tours du plus long au moins long et commencer à insérer les arêtes des plus petits tours dans les tours les plus longs, sans dépasser la capacité des véhicules.

Étape 3 : Fusionner deux tours i et j sujet aux contraintes de capacité de façon à ce que le gain S_{ij} soit maximal. Ici, $S_{ij} = l_i + l_j - l_{ij}$, où l_i (l_j) est la longueur du tour i (j), et l_{ij} est la longueur du tour résultant de la fusion de i et j.

Augment-Insert :

Proposé par Pearn [76], cet algorithme est une fusion des deux procédures Augment-Merge et Parallel-Insert [16]. Il s'exécute en deux phases :

Phase 1 (Augment) :

1. Choisir une arête (v_i, v_j) de demande positive, et construire un tour qui dessert seulement cette arête. Si ce n'est pas possible, aller à la phase 2.
2. Augmenter le tour construit en ajoutant d'autres arêtes à desservir en ordre décroissant des $D_{ij} = d_{1i} + d_{1j}$ sans dépasser la capacité des véhicules, où d_{kl} est la chaîne de coût minimal entre v_k et v_l.
3. Enlever le tour du graphe et reprendre (1) avec le graphe réduit.

Phase 2 (Insert) : cette phase s'applique au graphe original

4. Choisir une arête (v_i, v_j) de demande positive non encore desservie telle que $D_{ij} = max_{(v_k,v_l)\in E} D_{kl}$. Insérer cette arête dans un tour existant du graphe original si le coût de l'insertion est inférieur à une certaine constante C et si l'insertion ne cause pas un dépassement de la capacité. Sinon, créer un tour pour cette arête.
5. Répéter (4) jusqu'à ce que toutes les arêtes soient desservies.

Il n'existe pas de règle générale pour déterminer la valeur de la constante C, mais on précise qu'elle doit être dans l'intervalle $[0, max_{v_i\in V}(d_{1i})]$.

Hertz et al. [52] ont développé différentes procédures de base qui peuvent être combinées ensemble pour construire des heuristiques pour résoudre le PTAC, dans le cas non orienté. La meilleure de ces heuristiques est une recherche tabou. Ces procédures de base sont les suivantes.

Procédure PASTE

Elle consiste à fusionner les tours dans une solution du PTAC pour obtenir un grand tour probablement non réalisable pour le PTAC. Étant donné, une solution faite de l tours $s_1 = (v_1, ..., v_{i_1} = v_1)$,..., $s_l = (v_1, ..., v_{i_l} = v_1)$, cette procédure rassemble ces tours en un tour unique $s' = (v_1, ..., v_{i_1}, ..., v_{i_2}, ..., v_{i_l} = v_1)$. Ensuite, elle fait appel à une autre procédure appelée SHORTEN qui identifie un tour s de moindre coût qui passe par toutes les arêtes requises dans s', mais pas nécessairement dans le même ordre (voir [51] pour une description de SHORTEN).

Procédure CUT

Cette procédure consiste à transformer une solution du PPR, qui est un grand tour $s = (v_1 = v_{i_1}, ..., v_{i_r}, ..., v_{i_l} = v_1)$ passant par toutes les arêtes qui ont une demande positive, en une solution pour le PTAC. Ceci se fait en commençant par le dépôt v_1. Nous choisissons un sommet v incident à une arête desservie dans s, tel que :
- La demande totale de la chaîne $(v_1, ..., v)$ ne dépasse pas la capacité des véhicules Q.

- La demande totale de la chaîne restante $(v, ..., v_1)$ ne dépasse pas $Q([d/Q] - 1)$.
Ici, d est la demande totale de s. Si nous avons plusieurs choix pour v, on le choisit afin que $L(v)$ soit minimal, où L est la somme de la longueur de la chaîne la plus courte entre v et v_1, et de la chaîne la plus courte entre v_1 et $v^{'}$, avec $v^{'}$ le premier sommet après v incident à une arête de demande positive dans s. Le sommet v est ensuite relié directement à v_1 par la chaîne la plus courte afin de compléter le tour. Cette procédure est reprise jusqu'à ce que le grand tour ait été épuisé.

Procédure SWITCH

Soit un tour $s = (v_1 = v_{i_1}, ..., v_{i_r}, ..., v_{i_l} = v_1)$ qui contient un sommet v plusieurs fois. Cette procédure crée un tour équivalent $s^{'}$ où tous les sous-tours qui commencent et finissent en v sont inversés. Par exemple, si $s = (v_1, v_2, v, v_3, v_4, v, v_5, v_6, v_7, v, v_8, v_1)$ alors $s^{'} = (v_1, v_2, v, v_4, v_3, v, v_7, v_6, v_5, v, v_7, v_1)$. Cette procédure permet de produire des solutions variées pour le PTAC et facilite l'utilisation de la procédure SHORTEN.

Procédure POSTOPT

Cette procédure est utilisée pour améliorer une solution du PTAC. Elle fait appel aux procédures PASTE, SWITCH, CUT et SHORTEN.

Parmi les heuristiques proposées par Hertz et al. dans [52], à partir des procédures décrites plus haut, mentionnons les suivantes.

Heuristique de construction

Il s'agit d'une heuristique de type Route First, Cluster Second qui construit un tour unique en résolvant le PPR avec l'heuristique de Frederickson [36]. Ensuite, elle fait appel à la procédure CUT pour obtenir une solution au PTAC. Enfin, la procédure POSTOPT est appliquée.

Recherche Tabou

Cette heuristique d'amélioration, appelée CARPET, s'inspire de l'heuristique TABUROUTE [38]. À chaque itération, le voisinage de la solution courante est exploré afin d'identifier la meilleure solution possible. Ce voisinage est défini comme suit. Étant donné une route s et une arête (v_j, v_k) dans s, une autre route s' est identifiée qui est soit une route vide ou une route qui contient une arête dont l'une des extrémités est suffisamment proche de v_j ou v_k. Une nouvelle solution est alors obtenue en déplaçant l'arête (v_j, v_k) de la route s à la route s'.

Greistorfer [47] a développé une autre recherche tabou pour le PTAC, mais défini sur un graphe orienté. La méthode proposée s'inspire des algorithmes génétiques. En effet, une population de solutions initiales est d'abord générée. Chaque solution dans le voisinage de ces solutions initiales est ensuite ajoutée à la population si elle est suffisamment différente et si son coût diffère du coût des meilleures solutions dans la population par un certain pourcentage. Le voisinage d'une solution est défini en échangeant des blocs d'arcs entre les routes ou en les déplaçant de leur route originale. Ici, un bloc contient au plus deux arcs requis.

Algorithme génétique pour le PTAC

Un algorithme génétique pour résoudre le PTAC est rapporté par Lacomme et al. [58]. Cette heuristique se compare à celle de Hertz et al. [52] au niveau des meilleurs résultats obtenus. Les solutions sont représentées par des chromosomes de la forme $(s_1, ..., s_m)$ où chaque s_i, $1 \leq i \leq m$, est un tour pour un véhicule. Dans la population initiale, certains individus sont générés par l'heuristique Augment-Merge, et d'autres aléatoirement. Différents types d'opérateurs de croisement ont été testés, en choisissant aléatoirement une ou deux positions de croisement. Par exemple, soient les deux solutions suivantes avec 3 tours chacune, pour un problème avec 8 arêtes à desservir, numérotées de 1 à 8.

Parent 1 : $6, 7, 8 - 1, 2, 3 - 4, 5$

$$\textit{Parent 2 : } 7, 6 - 1, 8, 2 - 5, 4, 3$$

Si on fait un croisement à la position 5, nous obtenons les deux solutions enfants suivantes :

$$\textit{Enfant 1 : } 6, 7, 8 - 1, 2, 5 - 4, 3$$
$$\textit{Enfant 2 : } 7, 6, 1 - 8, 2, 3 - 4, 5$$

Parmi les types de mutation testés, citons :
Swap : On sélectionne deux positions p et q et on échange les arêtes situées à ces positions. Chacune de ces deux arêtes peut ensuite être inversée avec une probabilité 0,5.
Move : On choisit deux positions p et q tel que $p \neq q$ et $q \neq p - 1$, et l'arête en position p est déplacée après celle en position q. Chacune de ces deux arêtes peut ensuite être inversée avec une probabilité 0,5.

Les auteurs ont publié d'autres articles sur des variantes du PTAC qu'ils ont résolues en modifiant l'algorithme génétique décrit ci-dessus. Ils ont entre autres considéré le PTAC périodique [60] où chaque arc requis e doit être desservi $f(e)$ fois sur un horizon de l journées avec, $1 \leq f(e) \leq l$. Le problème consiste à sélectionner $f(e)$ jours distincts pour chaque arc e et à résoudre ensuite le PTAC associé à chaque journée.

2.2.3 Le problème de tournées sur les arcs avec contraintes de capacité et fenêtres de temps (PTACFT)

Ce problème est identique au problème de tournées sur les arcs avec contraintes de capacité, avec en plus une contrainte sur le temps de début de service sur chaque arc requis. En effet, chaque arc requis e a une fenêtre de temps $[a_e, b_e]$. Le service doit débuter au plus tard au temps b_e. Si le véhicule arrive avant le temps a_e il doit attendre avant de commencer le service. Ce problème a été peu étudié dans la littérature. Citons toutefois Eglese [28] qui s'est intéressé au problème de déglaçage où certaines rues doivent être traitées 2 ou 3 heures après la précipitation.

Mullaseril [72] a traité le PTACFT sur un graphe orienté, tandis que Gueguen l'a considéré pour un graphe non orienté [48]. Dans les deux cas, le problème a été modélisé en le transformant d'abord en un problème de tournées sur les sommets. Dans sa thèse de doctorat, Wohlk [89] a travaillé directement sur le PTACFT. Elle a proposé différentes heuristiques simples et un algorithme de programmation dynamique pour résoudre ce problème. Elle a aussi proposé une méthode exacte de génération de colonnes, ce qui lui a permis entre autres d'identifier des bornes inférieures de qualité.

2.2.4 Domaines d'application

Parmi les applications réelles qui se modélisent comme des problèmes de tournées sur les arcs avec ou sans contraintes de capacité, on peut citer par exemple :

- **La collecte des ordures :** Il s'agit d'un problème de tournées sur les arcs avec contraintes de capacité [5, 67].

- **Le déneigement :** Nous incluons ici le déglaçage et le déblaiement des rues et des trottoirs. Contrairement au problème de déglaçage, le déblaiement des rues et des trottoirs se modélise comme un problème de tournées sur les arcs, mais sans contraintes de capacité. Bien que ces problèmes aient été étudiés dans la littérature en tant que problèmes statiques [13, 28, 49, 53], où toutes les données sur le problème sont supposées connues à l'avance, les aspects dynamiques de ces problèmes sont très importants en pratique.

- **Le balayage des rues et la lecture des panneaux d'électricité :** ce sont deux problèmes de tournées sur les arcs sans contraintes de capacité [7, 30, 83]

Les problèmes de tournées sur les arcs décrits jusqu'à présent n'ont qu'un seul dépôt bien qu'on note souvent la présence de plusieurs dépôts dans les problèmes réels. Dans la section suivante, nous présentons en particulier certains problèmes mixtes de localisation et de tournées de véhicules.

2.3 Problèmes mixtes de localisation et de tournées de véhicules

Le problème mixte de localisation et de tournées de véhicules est habituellement divisé en trois sous-problèmes distincts qui sont résolus de façon séparée :

1. Un problème de localisation qui consiste à localiser les dépôts dans le graphe sous-jacent.
2. Un problème d'affectation, qui consiste à affecter chaque demande dans le graphe à un dépôt.
3. Un problème de routage qui consiste à construire des tournées pour les véhicules ; dépendamment des variantes considérées, on peut exiger ou non que la route d'un véhicule commence et se termine au même dépôt.

Le problème mixte de localisation et de tournées de véhicules sur les arcs (PLTA) n'a pas été très étudié dans la littérature comparativement au problème équivalent sur les sommets. Ici, on pose $D \subseteq A$ l'ensemble des arcs potentiels où on peut localiser les dépôts. Chaque localisation sur un arc $e \in D$ entraîne un coût fixe f_e. L'objectif est de localiser un nombre fixe de dépôts dans D et de déterminer un ensemble de routes pour chaque dépôt de façon à minimiser la somme des coûts fixes de localisation et des coûts de routage.

Les méthodes proposées pour résoudre ce problème sont en majorité des heuristiques. Toutefois, dans [41], on résout le PLTA avec une méthode exacte en le transformant en un PPR. La plupart des heuristiques décomposent le problème en sous-problèmes, soit : (1) la localisation des dépôts, (2) l'affectation de chaque arc requis à un dépôt et (3) la construction des routes. Ces heuristiques se divisent ainsi en deux classes.

Localisation-Affectation-Routage (L-A-R)

Ici, les dépôts sont localisés en premier. Ensuite, chaque arc requis est affecté au dépôt le plus proche. Finalement, un problème de tournées sur les arcs est résolu à chaque dépôt.

Affectation-Routage-Localisation (A-R-L)

Premièrement, on regroupe les arêtes qui seront desservies par un même véhicule. Ensuite, on détermine la séquence de visite des arêtes dans chaque route. Enfin, on localise les dépôts et on affecte les routes aux dépôts.

Parmi les travaux dans lesquels on a proposé des heuristiques pour le problème mixte de localisation et de tournées sur les arcs, mentionnons Ghiani et al. [42], qui ont introduit le PTAC avec dépôts intermédiaires (PTACDI). L'objectif est de déterminer une tournée pour chaque véhicule, où la demande totale peut dépasser la capacité du véhicule si cette tournée passe par un ou plusieurs dépôts intermédiaires. En effet, le véhicule peut être rempli (ou vidé) au niveau de ces dépôts intermédiaires.
Les auteurs ont proposé deux heuristiques pour résoudre le PTACDI : la première détermine une solution pour le PPR et dérive ensuite une solution pour le PTACDI en introduisant les dépôts intermédiaires. La seconde applique une heuristique pour le PTAC sur un graphe induit de G qui permet d'obtenir un solution réalisable pour le PTACDI. Finalement, une heuristique d'amélioration est appliquée.

Bouliane et Laporte [10] ont proposé une autre approche pour résoudre un problème de livraison du courrier. L'objectif est de localiser des boîtes à lettres dans les routes de facteurs afin de permettre à ces derniers de remplir leur sac sans revenir au dépôt principal. Ce problème est aussi un PTACDI. Une fois les routes définies, la localisation des boîtes se fait par un algorithme de type «set covering».

2.4 Problèmes dynamiques

Les problèmes dynamiques sont des problèmes où les données ne sont pas toutes connues à l'avance. Dans ce type de problèmes, une nouvelle donnée peut modifier la solution courante qui est en cours d'exécution, ce qui demande des temps de réponse très courts. Une grande rapidité de calcul est donc exigée [79, 80]. À notre connaissance, aucun travail sur les problèmes dynamiques de tournées sur les arcs n'a été rapporté dans la littérature. Cependant, plusieurs travaux se sont intéressés aux problèmes dynamiques de tournées sur les sommets, par exemple le travail de Gendreau et al. [39] où une heuristique de recherche tabou a été développée.

Comme cette thèse est motivée par les problèmes de déglaçage, nous présentons dans ce qui suit les différentes opérations hivernales liées au déglaçage, au déblaiement des rues et au transport de la neige vers les sites de déversement.

2.5 Les principales opérations de déneigement

2.5.1 L'épandage d'abrasifs et de fondants (déglaçage)

Quand les routes sont susceptibles de devenir dangereuses à cause de la pluie verglaçante, de la neige ou de la formation de glace, les autorités locales les traitent en y épandant un déglaçant comme le sel ou des abrasifs (sable, petites pierres ...). Dans le cas de la neige, cet épandage se fait en général pour des précipitations de moins de 2,5 cm. Quand les prévisions météo annoncent de la pluie verglaçante ou de la formation de glace due à des chutes subites de température, l'épandage devrait idéalement avoir lieu un peu avant la formation de la glace.

Les deux côtés d'une route peuvent parfois être traités suite à un seul passage de l'épandeuse. Mais pour les routes principales, un passage dans les deux sens est souvent nécessaire. Pour les trottoirs, de petites épandeuses-déneigeuses sont utilisées. Les interventions sont généralement réalisées par ordre de priorité : artères principales

d'abord, artères situées aux abords des écoles et des arrêts d'autobus, trottoirs, etc.

Les principaux travaux rapportés dans la littérature traitent du problème de génération de tournées pour des épandeuses dans les rues d'une ville en supposant que ces dernières partent et reviennent à un même dépôt. Ce problème est donc très semblable au PTAC. Cependant, si l'on tient compte de toutes les contraintes liées aux opérations de déglaçage, le problème peut devenir très complexe.

Les principaux aspects que l'on doit aborder dans ce contexte sont les suivants.

Localisation des dépôts

Les épandeuses partent d'un dépôt de sel et y retournent après avoir effectué une tournée. Il est clair qu'une bonne localisation des dépôts aidera à réduire les coûts de déglaçage qui sont proportionnels à la distance parcourue par les épandeuses.

Moment de l'intervention

En fonction des prévisions météo, le traitement des routes doit commencer à un moment bien précis de la journée.

Capacité variable des véhicules

Chaque épandeuse dispose d'une capacité particulière, dépendant du type de matériau qui est déversé. Il faut affecter à chaque épandeuse une route qui tienne compte de sa capacité et du taux d'épandage.

Différentes heuristiques ont été proposées pour résoudre ce type de problèmes. Ainsi, Eglese [28] a proposé une heuristique où il tient compte de la capacité des épandeuses et de la priorité des routes. Avant de déterminer une solution, il s'intéresse également à la localisation des dépôts de sel. La fonction objectif à minimiser est $C_G.G + C_D.D.N$, où :

C_G : coût annuel d'une épandeuse.

G : nombre d'épandeuses utilisées.

C_D : coût de l'épandage par kilomètre.

D : nombre total de kilomètre parcouru.

N : nombre de jours par année où on fait l'épandage.

Ainsi, le premier terme correspond au coût annuel des épandeuses et dépend du nombre d'épandeuses utilisées. Le second terme représente le coût annuel du déglaçage et dépend de la distance totale parcourue.

Pour résoudre ce problème, l'heuristique suivante est proposée :

1. Résoudre le PPR sur le graphe considéré de façon à générer un tour unique.
2. Générer des sous-tours pour les épandeuses.
3. Localiser les dépôts à l'aide d'un algorithme glouton dans un graphe réduit où chaque sommet représente un sous-tour dans le graphe initial.
4. Affecter les sous-tours aux dépôts à l'aide d'une heuristique basée sur les gains.
5. Améliorer la solution par la méthode de recuit simulé en tenant compte des contraintes de capacité.

Cook et Alpin [20] ont proposé une heuristique qui a pour objectif de minimiser le temps total requis pour couvrir toutes les arêtes avec un nombre donné d'épandeuses. Le principe de l'heuristique est le suivant : quand une épandeuse est chargée, elle est affectée à une arête non encore desservie qui a une demande n'entraînant pas le dépassement de la capacité de l'épandeuse, et qui est située le plus près du dépôt. La procédure est répétée jusqu'à ce toutes les arêtes requises soient desservies.

Lotan et al. [66] ont analysé l'interdépendance entre le problème de localisation de dépôts et le problème de génération de tournées pour les épandeuses. Ils ont proposé un modèle pour une application dans la province d'Antwerp qui, selon eux, est un cas représentatif des problèmes de déglaçage en Belgique. Dans cette étude, ils ont considéré deux types de dépôts : (1) des dépôts principaux, qui correspondent au point de départ et de retour des épandeuses, et qui sont localisés en premier et (2) des dépôts supplémentaires où les véhicules peuvent s'arrêter au besoin pendant une tournée, si la

demande totale sur la route dépasse leur capacité.

Il faut noter que les gens s'intéressent de plus en plus au déglaçage préventif, qui est beaucoup moins coûteux que le déglaçage correctif. Le déglaçage préventif se fait en se basant sur les prévisions météorologiques et sur les prévisions provenant de stations météo routières. D'après une étude faite en 1992 [9], les stations de météo routières permettent de réduire considérablement les coûts du déglaçage, en permettant de mesurer à la fois la température de la chaussée et la température de l'air ambiant.

2.5.2 Le déblaiement des rues et des trottoirs

Le déblaiement des rues et des trottoirs se fait pour des accumulations de neige de plus de 2,5 cm. Dans chaque secteur, les artères principales sont dégagées avant les rues secondaires. La neige est poussée sur les côtés des rues et des trottoirs. Une fois que les routes sont dégagées, l'épandage d'abrasifs et de sel peut être entrepris.

Les principaux travaux se sont intéressés au problème de génération de tournées pour des déneigeuses dans les rues d'une ville [53]. Ce problème ressemble au problème de collecte des ordures, mais on doit tenir compte des particularités suivantes :

1. La pose de panneaux interdisant le stationnement.
2. La planification du début du déblaiement des rues et des trottoirs en fonction de l'évolution prévue de la tempête, en particulier l'heure de fin de la chute de neige.
3. Les heures de travail des ouvriers.
4. Les délais de déblaiement (e.g., pour 20 cm et moins, on essaie d'assurer un délai de 96 heures après la fin de la chute de neige)

Ce problème est un problème de tournées sur les arcs sans contraintes de capacité. Parmi les travaux qui ont été réalisés pour déterminer les tournées des déneigeuses, mentionnons les travaux de Gilbert [44], qui ont porté sur le problème d'enlèvement de la neige dans un contexte urbain. Le problème a été modélisé comme un problème de

tournées de véhicules sur les sommets, en considérant chaque segment de rue à déneiger comme un sommet à visiter. Pour la résolution, l'heuristique proposée fait appel à deux variantes d'une méthode d'insertion.

Wang et al. [87] ont développé un logiciel nommé CASPER qui fait appel à une recherche tabou pour améliorer les tournées selon des objectifs et des contraintes qui peuvent être spécifiées par l'utilisateur. Le modèle pour le déneigement considère les objectifs suivants : (1) minimiser la déviation par rapport au temps de service désiré, (2) minimiser le nombre d'arêtes parcourues à vide et (3) maximiser l'homogénéité des routes.

Haslan et Wright [49] ont proposé une heuristique de construction et amélioration pour le problème de déneigement dans l'État de l'Indiana sur un réseau rural orienté. L'heuristique tient compte de deux objectifs : minimiser le coût de déneigement et maximiser la sécurité des routes. Pour le second objectif, on demande une bonne condition des routes et une fluidité de la circulation pendant l'enlèvement de la neige.

2.5.3 Chargement de neige et transport vers les sites de déversement

Le chargement se fait à la fin de la précipitation, pour des accumulations de plus de 5 cm, provenant d'une ou de plusieurs précipitations. Une fois le chargement complété, la neige ramassée est transportée vers les sites de déversement. L'ensemble des considérations stratégiques, tactiques et opérationnelles liées à ce problème sont les suivantes :

1. Division de la ville en secteurs de déneigement : Pour faciliter les opérations de déneigement dans une ville, cette dernière est divisée en secteurs.
2. Localisation des sites de déversement : Ce problème consiste à localiser les sites de déversement le plus près possible des différents secteurs de déneigement de la ville pour réduire les coûts.
3. Affectation des secteurs aux sites de déversement : Une fois que les secteurs sont déterminés et que les sites de déversements sont localisés, la neige chargée dans

chaque secteur doit être transportée vers un site de déversement approprié afin de réduire les coûts de déneigement, d'où le problème d'affectation des secteurs aux sites de déversement. Ce problème peut se formuler à l'aide d'un modèle mathématique où l'objectif est de minimiser le coût de déneigement de chaque secteur, ce coût étant proportionnel à la distance entre chaque secteur et le site de déversement qui lui est affecté.

Clairement, les problèmes de division de la ville en secteurs, de localisation des sites de déversement et d'affectation des secteurs aux sites de déversement sont des problèmes interdépendants, ce qui rend leur étude très complexe. Les principaux travaux dans ce domaine ont été réalisés par Campbell et Langevin [13], qui ont étudié le problème de division de la ville de Montréal en secteurs de déneigement et l'affectation de chaque secteur à un site de déversement. Les auteurs identifient d'abord des zones ayant des frontières naturelles bien définies au sein de la ville. Chacune des zones est ensuite associée à un site de déversement en résolvant un problème d'affectation. Les zones voisines sont enfin regroupées pour former des secteurs.

Campbell et Langevin [14] ont par la suite étudié le problème d'affectation des secteurs de déneigement aux sites de déversement. Le problème a été résolu avec une heuristique basée sur celle de Martello et Toth dans [69], ainsi qu'avec une heuristique d'échange d'arcs.

Chapter 3

Arc Routing Problems with Time-Dependent Service Costs

Résumé. Cet article étudie un problème de tournées sur les arcs avec contraintes de capacité et des coûts dépendants du temps de début de service. Ce problème s'inspire du problème de déglaçage des routes où le moment de chaque intervention est crucial. La méthode exacte de résolution présentée ici commence par transformer le problème de tournées sur les arcs en un problème de tournées sur les noeuds. Ensuite, une approche de génération de colonnes est utilisée pour résoudre ce dernier. Le problème maître est un problème classique de recouvrement, alors que les sous-problèmes sont des problèmes de plus courts chemins élémentaires, dépendants du temps, avec contraintes de ressources. Ces sous-problèmes sont résolus en utilisant une extension d'un algorithme précédemment développé. Des résultats numériques sont présentés sur des problèmes dérivés d'un ensemble d'instances classiques pour le problème de tournées de véhicules avec fenêtres de temps.

Arc Routing Problems with Time-Dependent Service Costs

Mariam Tagmouti

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Michel Gendreau

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Jean-Yves Potvin

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Article publié dans *European Journal of Operational Research*
181, pp.30–39, 2007

Abstract. This paper studies an arc routing problem with capacity constraints and time-dependent service costs. This problem is motivated by winter gritting applications where the "timing" of each intervention is crucial. The exact problem-solving approach reported here first transforms the arc routing problem into an equivalent node routing problem. Then, a column generation scheme is used to solve the latter. The master problem is a classical set covering problem, while the subproblems are time-dependent shortest path problems with resource constraints. These subproblems are solved using an extension of a previously developed algorithm. Computational results are reported on problems derived from a set of classical instances of the vehicle routing problem with time windows.

Keywords: arc routing, column generation, elementary shortest paths, resource constraints.

3.1 Introduction

We study a variant of the Capacitated Arc Routing Problem (CARP), introduced in [45], where a subset of arcs must be serviced at a cost that depends on the time of beginning of service. In this paper, the cost is a piecewise linear function of time. This problem is motivated from winter gritting operations, where the timing of an intervention is of prime importance [12, 28, 29, 64, 66]. That is, if the intervention is too early or too late, the cost in material and time sharply increases.

The CARP is NP-hard and was first addressed with relatively simple heuristics, like the construct-strike [17], path-scanning [46] and augment-merge [45] heuristics. They have been improved over time, mainly through the use of metaheuristics, like tabu search [52] or genetic algorithms [58]. An interesting variant of the CARP is presented in [42], where intermediate facilities are available to reload the vehicle. Although we do not address this variant here, these intermediate facilities could correspond to salt boxes, for example, in winter gritting applications. Surveys on the CARP and some of its variants can be found in [2, 21, 24, 31, 32]. We are not aware of any variant of the CARP with time-dependent service costs, although a particular case would be the CARP with soft time windows, when the service cost is "flat" within a given time interval and then increases linearly on both sides of the interval.

The paper is organized as follows. The problem is first introduced in Section 2. Then, its transformation into an equivalent node routing problem is described and the resulting mathematical formulation is presented in Section 3. The column generation approach for solving this problem is reported in Sections 4 and 5, where time-dependent shortest path subproblems with resource constraints are addressed by extending a previously reported algorithm. In Section 6, numerical results on problems derived from instances of the vehicle routing problem with time windows (VRPTWs) are reported [82]. Finally, the conclusion follows.

3.2 Problem Description

Let $G = (V, A)$ be a directed graph where V is the vertex set and A is the arc set. We assume that A is partitioned into a subset of required arcs A_1, which must be serviced, and a complementary subset of arcs A_2. With each required arc $e \in A_1$ is associated a demand d_e, a length l_e, a travel time tt_e, a service time st_e, a travel cost tc_e and a time-dependent piecewise linear service cost function $sc_e(T_e)$, where T_e is the time of beginning of service on arc e. The other arcs in subset A_2 have a length, a travel time and a travel cost only. Note that the service time is typically larger than the travel time because it takes more time to service an arc than to simply travel along the arc.

A set $K = \{1, ..., m\}$ of identical vehicles with capacity Q is available to service the required arcs. These vehicles are located at a central depot node from which each vehicle services a single route that starts and ends at the depot. The vehicles are not allowed to wait along their route and must be back at the depot by a given deadline. The objective is to service all required arcs in the graph at least cost with feasible routes, where the cost is related to the number of vehicles used, the travel cost and the service cost.

Figure 1 shows typical piecewise linear service cost functions for the required arcs. Note that the fuction in Figure 1a is a degenerate form of the one shown in Figure 1b, where the "optimal" time interval for service reduces to a single point. The function shown in Figure 1b was used in our computational experiments, due to its similarity with classical "soft" time windows, but other piecewise linear forms could have been used as well, like the one shown in Figure 1c.

3.3 Problem Formulation

The first step is to transform the arc routing problem in graph $G = (V, A)$ into an equivalent node routing problem in a transformed graph $G' = (V', A')$. This type of

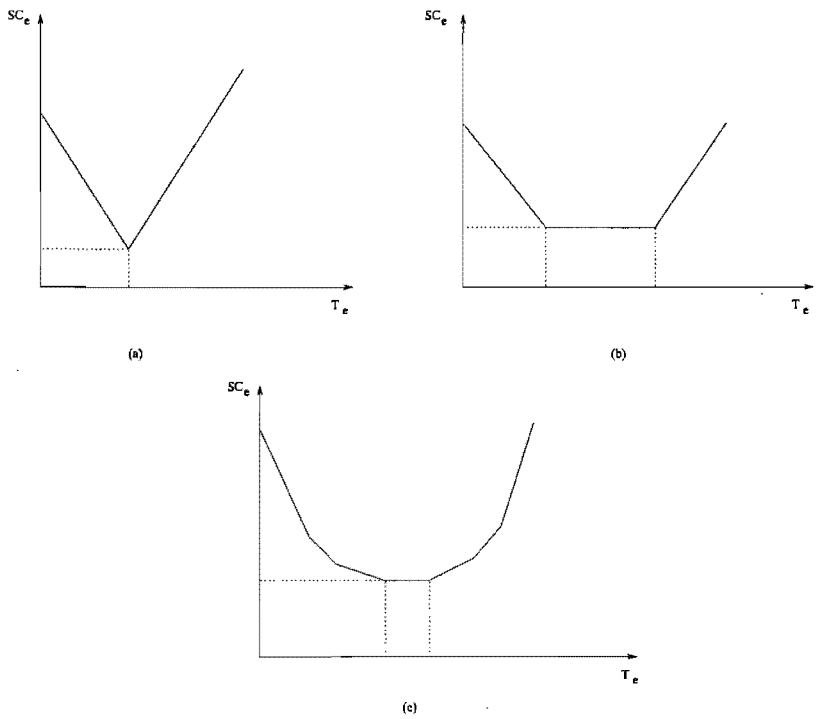

FIG. 3.1 – Different service cost functions

transformation is well known and was first proposed in [77] for an undirected graph. The drawback in the undirected case is that multiple nodes must be associated with each required edge. Basically, if $e_i = (i_1, i_2)$ and $e_j = (j_1, j_2)$ are two required edges, there are four possible ways to link them together (i.e., either i_1 with j_1, i_1 with j_2, i_2 with j_1, or i_2 with j_2). To keep this information when solving the node routing problem, the transformation proposed in [77] creates three nodes for each required edge. Recently, the authors in [3, 65] have proposed new transformations where only two nodes are associated with each required edge. In our case, since we are working on a directed graph, there is only one way to link two arcs and a single node can thus be associated with each required arc (a benefit also mentioned in [68]). More precisely, the transformation is the following.

Each required arc $e_i \in A_1$ corresponds to a node i in graph G' with demand d_i, service time st_i and time-dependent service cost $sc_i(T_i)$. Each pair of distinct nodes i and j in G' is connected by an arc $(i, j) \in A'$ with length l_{ij}, travel time tt_{ij} and travel

cost tc_{ij}. The latter values are those of the shortest path between the two corresponding required arcs in graph G. These shortest paths are calculated from the end node of the first required arc to the start node of the second required arc and include the second required arc. Finally, the central depot node is added and connected to all other nodes in G'. The arc values in the latter case are given by the shortest paths from the depot node to the start node of all required arcs in G (including the required arc) or from the end node of all required arcs to the depot node, depending on the orientation of the arc.

Figure 2 illustrates this transformation. In the figure, the solid arcs in G are required arcs while dotted arcs are non required ones. Nodes i and j in G' correspond to arcs e_i and e_j in G, respectively. Each arc $e \in G$ is labeled with the vector $(d_e, st_e, l_e, tt_e, tc_e)$. Nodes $i, j \in G'$ are labeled with vectors (d_{e_i}, st_{e_i}) and (d_{e_j}, st_{e_j}), respectively, while arc $(i, j) \in G'$ is labeled with the vector $(l_{ij}, tt_{ij}, tc_{ij})$.

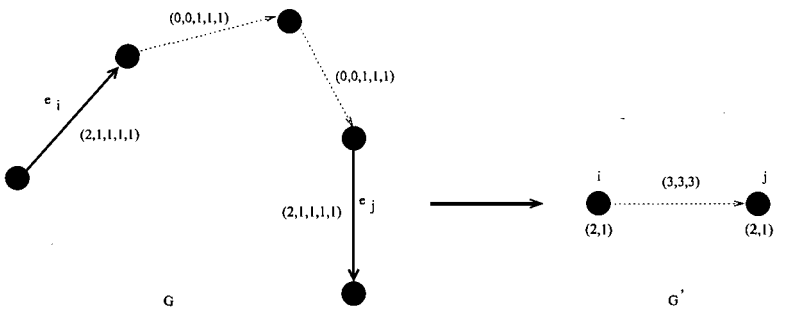

FIG. 3.2 – Graph transformation

After the transformation, we obtain a vehicle routing problem with time-dependent service costs. This type of problem has never been addressed in the literature, although some variants of the VRPTW can be related to it (see [54, 56, 86], for example, where missing a time window or a desired service time within a time window is penalized). A good review of different time-constrained vehicle routing problems can also be found in [22].

In the problem formulation below, N' stands for the set of nodes V' minus the

depot (i.e., the nodes that must be served). Also, the depot is duplicated into an origin depot o and a destination depot d in V'. The decision variables are: (1) the binary flow variables on the arcs $x_{ij}^k, (i,j) \in A', k \in K$, which are equal to 1 if vehicle k travels on arc (i,j) to service node j, 0 otherwise, (2) the non negative load variables $Q_i^k, i \in V'$ which specify the load of vehicle k just after serving node i and (3) the non negative time variables $T_i^k, i \in V'$, which specify the time of beginning of service of vehicle k at node i. Note that $Q_o^k = Q,\ k \in K,\ d_o = d_d = 0$ and that T_o^k and T_d^k stand for the departure and return time of vehicle k at the depot.

$$\text{Min} \sum_{k\in K} \left(\sum_{(i,j)\in A'} tc_{ij} x_{ij}^k + \sum_{i\in N'} sc_i(T_i^k) \sum_{j\in N'\cup\{o\}} x_{ji}^k \right)$$

Subject to

$$\sum_{k\in K} \sum_{i\in N'\cup\{o\}} x_{ij}^k = 1 \qquad j \in N' \qquad (1)$$

$$\sum_{k\in K} \sum_{j\in N'} x_{oj}^k \leq m \qquad (2)$$

$$\sum_{j\in N'\cup\{d\}} x_{oj}^k = 1 \qquad k \in K \qquad (3)$$

$$\sum_{j\in N'\cup\{d\}} x_{ij}^k - \sum_{j\in N'\cup\{o\}} x_{ji}^k = 0 \qquad k \in K \quad j \in N' \qquad (4)$$

$$\sum_{i\in N'\cup\{o\}} x_{id}^k = 1 \qquad k \in K \qquad (5)$$

$$x_{ij}^k (T_i^k + st_i + tt_{ij} - T_j^k) \leq 0 \qquad k \in K \quad (i,j) \in A' \qquad (6)$$

$$x_{ij}^k (Q_i^k - d_j - Q_j^k) \leq 0 \qquad k \in K \quad (i,j) \in A' \qquad (7)$$

$$0 \leq T_i^k \leq T \qquad k \in K \quad i \in V' \qquad (8)$$

$$0 \leq Q_i^k \leq Q \qquad k \in K \quad i \in V' \qquad (9)$$

$$0 \leq x_{ij}^k \leq 1 \qquad k \in K \quad (i,j) \in A' \qquad (10)$$

$$x_{ij}^k \in \{0,1\} \qquad k \in K \quad (i,j) \in A' \qquad (11)$$

The objective is to minimize the sum of travel costs and time-dependent service costs. A fixed charge can also be added to the travel costs $tc_{oi}, i \in N'$, if one wants to penalize the use of an additional vehicle. Constraints (1) require that each node in N' be serviced once. Constraints (2) impose an upper bound m on the number of vehicles. Constraints (3), (4) and (5) are the flow conservation constraints. Constraints (6) and (7) ensure the feasibility of the time schedule and loads, respectively. Constraints (8) impose that the time of beginning of service at each node (including the departure and return time at the depot) be a non negative value that does not exceed the deadline

T. Constraints (9), (10) and (11) ask for non negative load values that do not exceed vehicle capacity Q and binary values for the flow variables. Note that contraints (6) and (7) can be linearized, due to the presence of binary flow variables (see [22], for details). In the next section, a Dantzig-Wolfe decomposition, or column generation, scheme is thus proposed to solve this problem.

3.4 Column Generation

The column generation scheme proposed here is well documented in the literature. Consequently, we will only introduce the master problem and the shortest path subproblems. The interested reader will find more details about column generation in [22].

The master problem corresponds to constraints (1) and (2) in the original formulation and can be expressed as follows:

$$
\begin{array}{ll}
\text{Min} \sum_{p\in\Omega} C_p u_p & \\
\text{Subject to} & \\
\sum_{p\in\Omega} a_{ip} u_p = 1 & i \in N' \\
\sum_{p\in\Omega} u_p \leq m & \\
u_p \geq 0 & p \in \Omega
\end{array}
$$

where decision variable u_p is 1 if path p is selected, 0 otherwise. In this formulation, Ω is the set of all feasible paths from the origin depot o to the destination depot d, C_p is the total cost of path p (sum of travel costs and service costs on all arcs and nodes along the path), and a_{ip} is 1 if node i is in path p. It is thus a linear relaxation of a set covering problem with an additional constraint on the total number of vehicles.

The subproblem, for each vehicle k, is an elementary shortest (least cost) path problem with resource constraints and corresponds to constraints (3) to (11) in the original formulation.

The resource constraints are the capacity constraint and the time deadline for the return of the vehicle at the depot. Denoting the reduced travel cost on arc $(i,j) \in A'$ by $\bar{tc}_{ij}$, the subproblem for a given vehicle k is expressed as follows:

$$\text{Min} \sum_{(i,j)\in A'} \bar{tc}_{ij} x_{ij}^k + \sum_{i\in N'} sc_i(T_i^k) \sum_{j\in N'\cup\{o\}} x_{ji}^k$$
Subject to
(3) to (11).

We start with an initial set of columns (paths) in the master problem, using a solution construction approach based on the savings heuristic of Clarke and Wright [18]. This simple heuristic works as follows. At the start, it is assumed that each node is serviced by a single route. Then, at each iteration, a pair of routes is selected and merged together on the basis of the best cost saving that can be achieved. This is repeated until a single route is obtained or no feasible merge exits. It is worth noting that the evaluation of the savings is based on the true time-dependent costs, as it is explained in Section 3.5.

The dual variables associated with this initial set of columns are then used to calculate the reduced travel costs in the corresponding subproblem. The latter is an elementary shortest path problem with resource constraints, which is solved with the algorithm of Feillet et al. [33]. All non dominated paths of negative cost are then added to the master problem and the latter is solved with CPLEX to obtain new dual variables. This procedure is repeated until no more paths with negative cost are found. Note that we incorporate as many paths as possible into the master problem, since the latter is relatively easy to solve when compared with the shortest path subproblems. Different filtering schemes have been tried to reduce the number of columns in the master problem, but these have invariably led to less efficient algorithms.

To obtain an integer solution, the column generation scheme is embedded within a previously reported branch-and-bound algorithm, where branching takes place on the arcs of the graph (i.e., an arc is forced into or excluded from a solution). More details about this algorithm can be found in [33].

The main difficulty when solving the shortest path subproblems stems from the time-dependent service costs that must be taken into account when evaluating a path. The algorithm of Feillet et al. was thus extended to address this issue. The original algorithm and the new extension are described in the following.

3.5 Elementary Shortest Path with Resource Constraints

The algorithm of Feillet et al. is a label correcting algorithm that solves the elementary shortest path problem with resource constraints on graphs with, possibly, negative cycles. In this context, a path is characterized by the consumption of each resource, in addition to its cost. When different paths lead to the same node, it might well be that no path dominates, or is better than the others, over all criteria. As a consequence, many different labels are typically maintained at each node (i.e., all non-dominated paths leading to that node).

Since elementary paths must be generated, cycles are detected by keeping a trace of previously visited nodes. More precisely, a path p from some origin node o to some node j is labeled with $R_p = (C_p, t_p^1, \ldots, t_p^l, s_p, V_p^1, \ldots, V_p^n)$, where $L = \{1, ..., l\}$ is the set of resources, C_p is the cost of path p, t_p^k is the consumption of resource $k = 1, ..., l$, s_p is the number of unreachable nodes (either because they have already been visited or because their inclusion would violate one or more resource constraints) and $V_p^i = 1$ if node i is unreachable, 0 otherwise. The following dominance relation is then defined:

Dominance relation. If p and p' are two different paths from origin o to node j with labels R_p and R'_p, respectively, then path p dominates p' if and only if $C_p \leq C_{p'}$, $s_p \leq s_{p'}$, $t_p^k \leq t_{p'}^k$, $k = 1, ..., l$, $V_p^i \leq V_{p'}^i$, $i = 1, ..., n$.

That is, path p dominates p' if (1) it is not more costly, (2) it does not consume more resources for every resource considered and (3) every unreachable node is also unreachable for path p'. Note that s_p, the number of unreachable nodes, is included in the label only to speed up the computations. As stated in [33], by eliminating paths through this dominance relation, only labels corresponding to non-dominated

elementary paths are kept and a solution to the problem is obtained at the end.

In our application, there are $l = 2$ resource constraints: the capacity constraint (where the resources consumed are load units) and the deadline constraint for the return of the vehicle at the depot (where the resource consumed is time). As the algorithm is executed, we maintain and update at each node all non dominated shortest paths leading from the origin depot to that node. In particular, with each path p is associated its time-dependent total cost $C_p(T_0^p)$ *which is expressed as a function of departure time variable* T_0^p. Given some path p from the origin depot to node i with cost function $C_p(T_0^p)$, the cost function of the new path p' obtained by extending path p from node i to node j is simply:

$$C_{p'}(T_0^p) = C_p(T_0^p) + sc_j(T_0^p + TT_i + tt_{ij}) + \bar{tc}_{ij},$$

where TT_i is the total travel time from the origin depot to node i.

An example is provided for a complete path in the Euclidean plane from the origin depot o to the destination depot d (see Figure 3). It is assumed that the travel time and the travel cost are the same and correspond to the Euclidean distance. The latter values are 2.8, 2.2, 3.6, 1.4, and 2.0 for arcs (o,1), (1,2), (2,3), (3,4) and (4,d), respectively. The time-dependent service cost function for each vertex is also shown in Figure 4. The total cost function associated with the destination depot d is given in Figure 5 (where only the portion in the positive quadrant is considered). The minimum cost of this path is thus 24.8 for any T_0 between 0.4 and 2.

As we associate a cost function (rather than a cost value) with each path, the dominance relation between two partial paths that lead to the same node holds only when one function is "under" the other in the positive quadrant, as illustrated in Figure 6.

It is worth noting that these cost functions can be easily handled within the Clarke and Wright heuristic, which is used to generate the initial set of columns. This is due to the merging process which consists in appending one path at the end of another.

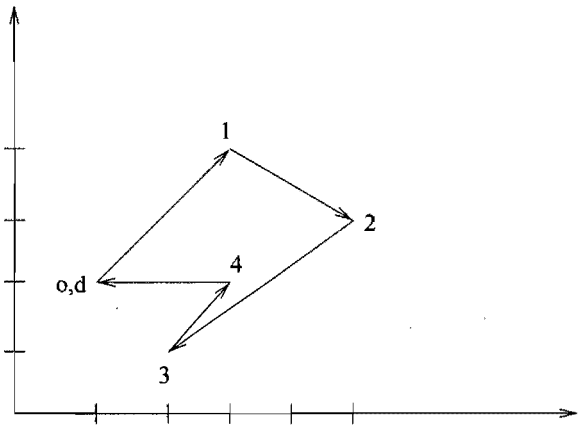

FIG. 3.3 – A complete path

Let $p1$ and $p2$ be two paths from the origin depot o to the destination depot d with cost functions $C_{p1}\left(T_0^{p1}\right)$ and $C_{p2}\left(T_0^{p2}\right)$, where T_0^{p1} and T_0^{p2} denote the departure time variables associated with paths $p1$ and $p2$, respectively. Let also l and f be the last node in path $p1$ (just before destination depot d) and the first node in path $p2$ (just after the origin depot o), respectively. Then, if we merge $p1$ and $p2$ to form the new path p, we can easily express its cost C_p in function of a single departure time variable. Namely, we have:

$$T_0^{p2} = T_0^{p1} + TT_l + tt_{lf} - tt_{of}$$

and we then make the replacement in $C_{p2}\left(T_0^{p2}\right)$ to obtain C_{p2} in function of T_0^{p1}. Then, the total cost of path p can be expressed in function of the single departure time variable T_0^{p1} as follows:

$$C_p\left(T_0^{p1}\right) = C_{p1}\left(T_0^{p1}\right) - tc_{ld} + C_{p2}\left(T_0^{p1}\right) - tc_{of} + tc_{lf}.$$

It is thus an easy matter to update the cost functions as the Clarke and Wright heuristic unfolds.

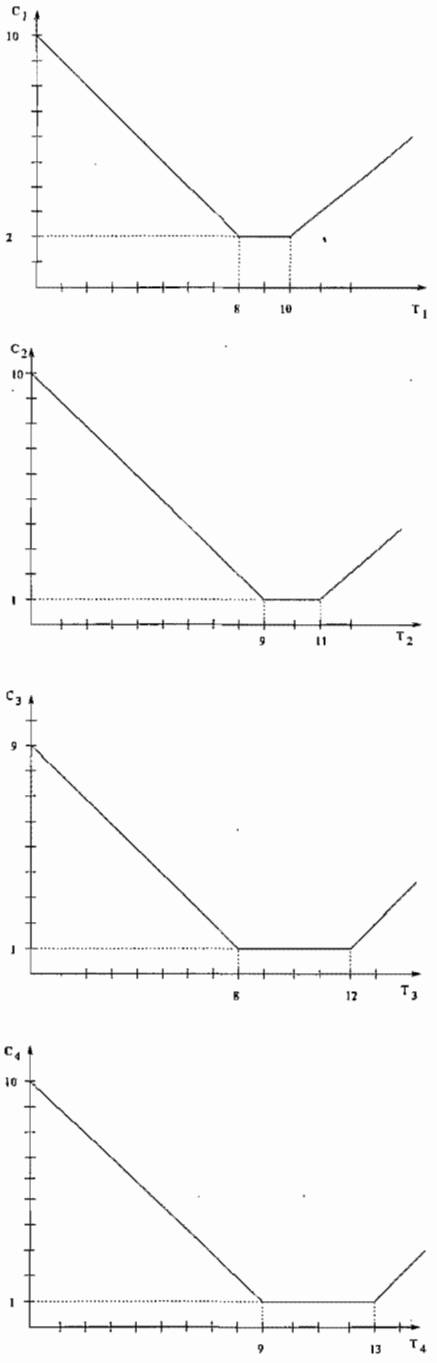

FIG. 3.4 – Service cost for each vertex i = 1,2,3,4

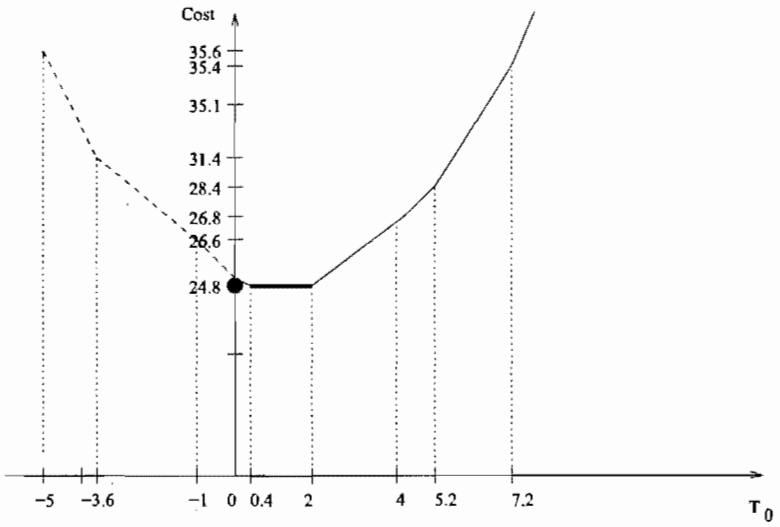

FIG. 3.5 – Total cost

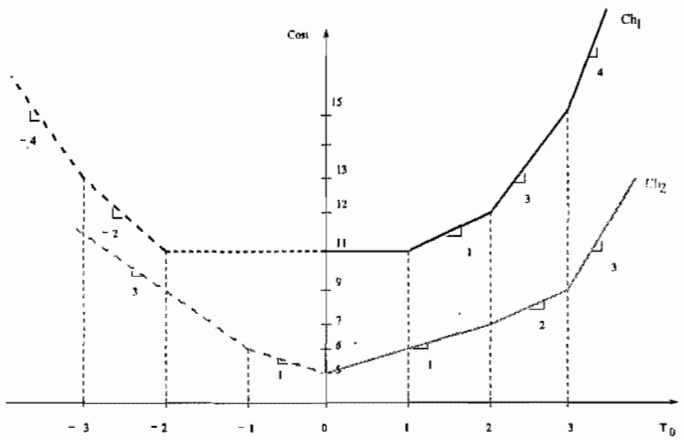

FIG. 3.6 – Example of dominance relation between two cost functions

3.6 Numerical Results

For testing our exact column generation algorithm, Solomon's VRPTW benchmark problems were used [82]. We have focused on problem instances of type 1 which are easier to solve due to a restrictive time deadline that leads to shorter routes. As these problems are Euclidean, the length, travel time and travel cost between two nodes are identical and correspond to the Euclidean distance. The service cost function at each node was specified as follows. First, the "flat" or minimum cost interval of the curve corresponds to the original time window in Solomon's instances. Second, the minimum cost was set to the service time at the node, as defined in Solomon's instances. Finally, a slope parameter was added to specify a linear cost increase on both sides of the window. The shape of the time-dependent service cost at each node is thus similar to the one shown in Figure 1b.

3.6.1 Parameter study

Our first results were obtained on the 25-customer instances in class R1 (random customer locations) and RC1 (mix of random and clustered customer locations). These results are shown in Table 3.1, with slopes of +1 and -1 on both sides of the time window and no penalty for the use of vehicles (recall that this penalty can be included in the travel cost from the depot to the first node on a route). In this Table, the first six columns provide the instance number, initial solution cost provided by the Clarke and Wright heuristic, optimal solution cost, number of vehicles used, solution length (i.e., total travel distance) and computation time in seconds. In the last two columns, we indicate the total number of columns or paths generated by our algorithm and the total number of calls to the shortest path algorithm, respectively. These tests were run on a 400 MHz UltraSparc II processor.

Table 3.2 shows the results obtained when the slope is set to -10000 and +10000 on both sides of the time window. In this case, the algorithm is "virtually" forced to visit each node within its window, otherwise a huge penalty is incurred. We thus obtain a

Instance	Initial Cost	Optimal Cost	#Vehicles	Length	CPU Time	#Columns (sec)	#Iterations
R101	949.4	893.7	5	532.2	1485.6	1037	16
R102	861.2	806.0	5	479.2	2357.7	2565	41
R103	799.5	726.1	4	439.2	462.9	2339	11
R104	713.5	690.0	4	415.3	3283.8	5463	50
R105	959.3	780.4	5	487.3	679.8	1366	17
R106	767.8	721.4	4	454.2	1362.0	2847	45
R107	710.4	679.5	4	427.4	792.6	4062	28
R108	687.7	647.7	3	379.6	1338.9	8610	50
R109	714.5	691.3	5	441.3	412.4	1598	13
R110	696.7	668.8	4	407.6	724.0	3237	14
R111	712.9	676.3	4	416.8	907.6	2974	34
R112	703.7	643.0	4	393.0	2047.4	7673	50
RC101	844.1	623.8	3	362.4	358.8	1945	14
RC102	773.4	598.3	3	344.0	1034.1	2846	16
RC103	711.3	585.1	3	335.1	5413.4	5554	23
RC104	728.6	572.4	3	322.4	18302.8	8999	19
RC105	816.8	623.1	3	359.4	4276.3	3770	16
RC106	760	588.7	3	327.6	2142.3	2557	14
RC107	735.9	548.3	3	298.3	4742.4	2771	18
RC108	709.5	544.5	3	294.5	19878.1	5986	19

TAB. 3.1 – Results on 25-customer instances with slopes +1 and -1

solution to a vehicle routing problem with hard time windows, where "hard" applies to both the lower and upper bound of the window. This feasibility/infeasibility issue leads to the larger variance observed in the CPU times. For some instances, a feasible solution is quickly found, which allows the algorithm to prune the search space early. Conversely, for some other instances, it is much more difficult to identify a feasible solution. Also, it is clear that a larger number of vehicles is needed to meet the time windows in this case.

In Table 3.3 the slopes are reset to +1 and -1, but a penalty of 50 Euclidean units is associated with the use of each vehicle. As expected, the number of vehicles is reduced when compared with the numbers shown in Table 1.

3.6.2 Problem size

As our algorithm is an exact one, this section shows how "far" we can go with regard to problem size. To this end, we used a more powerful machine with a 2.4 GHz ADM Opteron(tm) 250 processor. In all those tests, the slopes were set to +1 and -1 and there was no penalty for the use of a vehicle. Also, the tests were restricted to the R1 problem class. Tables 3.4 and 3.5 show that the limit is reached with only a slightly larger number of customers, even if a more powerful machine was used.

In particular, a sharp increase in computation times is observed from the instances with 25 customers to the instances with 35 customers (e.g., one day of computation for R110). When the number of customers grows to 40, only 6 instances out of 12 were solved to completion. For the remaining instances, the algorithm stopped due to insufficient memory.

Instance	Initial Cost	Optimal Cost	#Vehicles	Length	CPU Time (sec)	#Columns	#Iterations
R101	1083.0	1068.6	11	818.6	72.1	83	7
R102	910.6	887	7	637.0	429.5	1241	50
R103	811.6	755	5	505.0	293.3	1356	28
R104	744.1	694.4	4	444.4	242.2	2267	15
R105	914.6	847.5	6	597.5	489.1	478	21
R106	800	752.2	6	502.2	611.1	1961	29
R107	754.4	683.9	4	433.9	203.9	2415	11
R108	723.1	660.3	4	410.3	1336.3	4086	50
R109	744.2	691.3	5	441.3	871.3	1423	12
R110	735.7	694.1	5	444.1	42437.5	2848	50
R111	734.4	684.7	4	434.7	836.3	1913	30
R112	691.1	643.0	4	393.0	2094.0	6365	50
RC101	945.7	724	4	474.0	291.5	702	50
RC102	866.3	601.8	3	351.8	269.7	1853	15
RC103	738.5	585.1	3	335.1	11418.4	3109	15
RC104	765.9	574.8	3	324.8	3737.7	3894	19
RC105	952	687.2	4	437.2	428.5	3770	16
RC106	747.2	595.5	3	345.5	1148.5	1528	17
RC107	717.2	548.3	3	298.3	1477.5	2766	16
RC108	650.6	544.5	3	294.5	4793.8	5803	15

TAB. 3.2 – Results on 25-customer instances with slopes +10000 and -10000

Instance	Initial Cost	Optimal Cost	#Vehicles	Length	CPU Time (sec)	#Columns	#Iterations
R101	1249.4	1140.3	4	510.3	4153.2	3958	37
R102	1161.2	1016.5	4	483.2	28149.8	6839	50
R103	1249.4	926.1	4	439.2	70031.9	8956	50
R104	979.8	835.9	3	408.3	58008.3	13625	25
R105	1275.2	991.1	4	488.7	1683.7	3416	12
R106	1017.8	905.2	3	397.5	20287.7	5606	21
R107	963.3	874.6	3	410.3	109674.0	14741	50
R108	869.9	797.7	3	379.6	32309.9	14598	28
R109	955.1	904.3	4	442.4	5505.0	4488	20
R110	931.6	868.8	4	407.6	29219.3	7983	50
R111	952.4	875.7	3	426.1	61181.5	9558	50
R112	941.1	803.1	3	389.4	60551.4	19266	49
RC101	1166.3	773.8	3	362.4	1551.2	4560	17
RC102	1002.5	748.3	3	344.0	21090.2	7870	21
RC103	911.3	735.1	3	335.1	60319.4	10398	16
RC104	928.6	724.8	3	324.8	89256.0	12687	20
RC105	1110.3	773.1	3	359.4	7201.9	7846	15
RC106	1013.6	745.5	3	345.5	1748.2	2127	17
RC107	937.5	697.3	3	296.3	56334.7	9174	18
RC108	913.5	694.5	3	294.5	253741.0	12830	20

TAB. 3.3 – Results on 25-customer instances with slopes +1 and -1 and vehicle penalty

Instance	Initial Cost	Cost	#Vehicles	Length	CPU Time (sec)	#Columns	#Iterations
R101	1216.9	1168.3	7	675.9	23953.6	3659	50
R102	1121.7	1048	7	608.1	37801.3	9978	50
R103	991.8	929.2	5	527.1	29496.8	13770	40
R104	885.8	837.8	4	469.8	35712.4	41532	70
R105	1045.1	1017.1	6	611.7	16825.6	5228	12
R106	1003.8	930.8	5	562.2	12921.1	14921	14
R107	928.5	874.8	5	514.6	51504.8	31736	70
R108	861.7	805.7	4	446.6	84516.5	96249	68
R109	953.9	914.8	6	559.7	46254.0	7708	48
R110	935.4	893.6	5	531.5	93493.0	11215	50
R111	942.5	864.4	5	493.9	57256.3	18455	40
R112	848.2	834.3	5	476.4	59672.9	28430	50

TAB. 3.4 – Results on 35-customer instances

Instance	Initial Cost	Cost	#Vehicles	Length	CPU Time (sec)	#Columns	#Iterations
R101	1363.7	1318.4	8	778.7	25687.0	5953	28
R102	1280.4	1194.8	9	733.4	107191.0	15764	70
R103	—	—	–	—	—	—	–
R104	—	—	–	—	—	—	–
R105	1328.3	1171.2	7	721.7	21735.9	7127	22
R106	—	—	–	—	—	—	–
R107	1082.3	995.3	6	591.3	29696.3	34771	15
R108	—	—	–	—	—	—	–
R109	1318.6	1045.7	6	626.4	76125.3	12944	40
R110	—	—	–	—	—	—	–
R111	—	—	–	—	—	—	–
R112	999.9	937.1	5	529	165221.0	58859	50

TAB. 3.5 – Results on 40-customer instances

3.7 Conclusion

In this paper, we have considered an arc routing problem with time-dependent service costs. To address this problem, we first transformed it into an equivalent node routing problem. The latter was solved exactly with a column generation approach. In the process, a previously reported elementary shortest path algorithm with resource constraints was extended to solve the time-dependent subproblems. Future research will now focus on problem formulations that are closer to the winter gritting application that motivated this work. Heuristic problem-solving approaches will then be considered.

Acknowledgments

This work was partially supported by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Québec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche. This support is gratefully acknowledged.

Chapter 4

A Variable Neighborhood Descent Heuristic for Arc Routing Problems with Time-Dependent Service Costs

Résumé. Cet article étudie un problème de tournées sur les arcs avec contraintes de capacité et des coûts dépendants du temps de début de service. Ce problème s'inspire du problème de déglaçage des routes où le moment de chaque intervention est crucial. L'heuristique de résolution proposée est une descente à voisinage variable. Les voisinages de la solution courante sont obtenus en déplaçant un arc d'une route à une autre et en échangeant des blocs d'arcs de différentes longueurs entre les routes. Des résultats numériques sont présentés sur le problème classique de tournées sur les arcs avec contraintes de capacité auxquels on a ajouté, pour chaque arc requis, une fonction de coûts de service dependants du temps où le coût de service est minimal sur un certain intervalle. Une comparaison avec une heuristique récente proposée pour un problème de tournées de véhicules avec pénalités temporelles est rapportée.

A Variable Neighborhood Descent Heuristic for Arc Routing Problems with Time-Dependent Service

Mariam Tagmouti

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Michel Gendreau

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Jean-Yves Potvin

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Article soumis à *Computers & Industrial Engineering, 2009.*

Abstract. This paper proposes a variable neighborhood descent heuristic for solving a capacitated arc routing problem with time-dependent service costs. The problem is motivated by winter gritting applications where the timing of each intervention is crucial. The variable neighborhood descent is based on neighborhood structures that manipulate arcs or sequences of arcs. Computational results are reported on problems derived from classical capacitated arc routing problem instances. A comparison is also provided with an alternative approach where the arc routing problem is solved after being transformed into an equivalent node routing problem.

Keywords. capacitated arc routing, time-dependent service costs, variable neighborhood descent

4.1 Introduction

The origins of arc routing can be traced back to the 1730's when Euler studied the Königsberg bridge problem (from the name of a city in Prussia) which led to the notion of Euler cycles and Euler graphs. In 1962, the Chinese mathematician Kwan Mei-Ko proposed a problem whose name was later derived from his origins: the classical Chinese Postman Problem (CPP) [70]. Later on, various extensions of the CPP were reported in the literature, in particular the capacitated arc routing problem (CARP). In this problem, a subset of required arcs, with some demand on each arc, must be served at least cost by a fleet of capacity constrained vehicles initially located at a central depot.

As opposed to the CPP, the CARP is NP-hard and was first addressed with relatively simple heuristics [16, 17, 45, 75, 76]. Then, several metaheuristics were adapted to this problem like simulated annealing [27], tabu search [1, 11, 52], genetic algorithms [59], ant colony optimization [23] and variable neighborhood search [50].

The main variants of the CARP are:

- *CARP with multiple depots (M-CARP).* Here, the vehicle routes can start from and end at several depots. This problem is addressed in [15] by first dividing the service area into smaller areas and by locating a depot within each area. Then, routes are designed with the augment-insert CARP heuristic [76] to serve all required arcs. In [1], the problem is transformed into a multiple center capacitated minimum spanning tree problem with arc constraints, which is then solved with tabu search. Ghiani et al. [42] consider a variant of the problem where intermediate facilities (salt boxes) are available to replenish the vehicles along their routes.

- *CARP with time windows (CARPTW).* In this variant, each required arc has a time window for the beginning of service. This problem has not been studied much in literature, although some early work can be found in [7, 28]. Wohlk presented a node routing and an arc routing formulation for the CARPTW in her thesis [89].

She also proposed a column generation approach to find good lower bounds. Recently, the authors in [57] proposed a greedy randomized adaptive procedure with path relinking for the CARPTW. It is worth noting that in both [57] and [89] the time windows are hard and vehicles are allowed to wait before beginning the service.

- *Periodic CARP.* Here, the required arcs must be served a number of times over a given time horizon. This problem is found in [60] where a genetic algorithm is proposed to solve it.

- *Stochastic CARP.* In this variant, the demand on the required arcs is a random variable [34].

In this paper, a new variant is considered where a time-dependent piecewise linear service cost is associated with each required arc in a directed graph. This problem is motivated by winter gritting applications, where the timing of each intervention is crucial. That is, if the intervention is too early or too late, the cost in time and de-icing material (like salt) increases. Also, as it is usually the case in these applications, the vehicle is not allowed to wait along its route. The problem considered here is still an abstraction of real-world problems which typically exhibit some specific characteristics. For example, it might be possible in some cases to treat both sides of the road by serving it only once in either direction. In other cases, a first service on a road is followed later by a second service, but with less material. That is, a large part of the road is treated on the first service, but there is still a need for some additional material on the other side of the road. Such issues have not been addressed in this work.

The problem reported here was first studied in [84] where small instances were solved with an exact algorithm. We are not aware of any other work on this problem, although a special case would be the CARP with (soft) time windows when the service cost is flat and minimal within a given time interval and linearly increases on both sides of this interval. In [62], a rural postman problem is studied where the upper bound of the time window at each required arc depends on the arc type. This way of defining time windows has been exploited in some of the computational results reported

in Section 4.4.

In the next section, the problem is first precisely defined. Then, a heuristic approach to this problem is introduced in Section 3, based on a variable neighborhood descent heuristic. Computational results are reported in Section 4 and a conclusion follows.

4.2 Problem Definition

Let $G = (V, A)$ be a directed graph where V is the vertex set and A is the arc set. It is assumed that A is partitioned into a subset of required arcs A_1, which must be served, and another subset of arcs A_2 which is used for traveling purposes only. With each required arc $e \in A_1$ is associated a demand d_e, a length l_e, a travel time tt_e, a service time st_e, a travel cost tc_e and a time-dependent piecewise linear service cost function $sc_e(T_e)$, where T_e is the time of beginning of service on arc e. The arcs in A_2 have a length, a travel time and a travel cost only. Note that the service time is typically larger than the travel time because it takes more time to serve an arc than to travel along it.

A set of K identical vehicles with capacity Q is available to serve the required arcs. These vehicles are located at a central depot from which each vehicle serves a single route that starts from and ends at the depot. Also, the vehicles are not allowed to wait along their route. The objective is to serve all required arcs in the graph with least-cost feasible routes, where the cost is the sum of service cost and travel cost (or deadhead cost).

Figure 1 shows typical piecewise linear service cost functions for the required arcs. Note that the function in Figure 1(a) is a degenerate form of the one shown in Figure 1(b), where the optimal service time reduces to a single point. The function shown in Figure 1(b) was used in our computational experiments, due to its similarity with soft time windows, but other piecewise linear forms could be used as well, like the one shown in Figure 1(c).

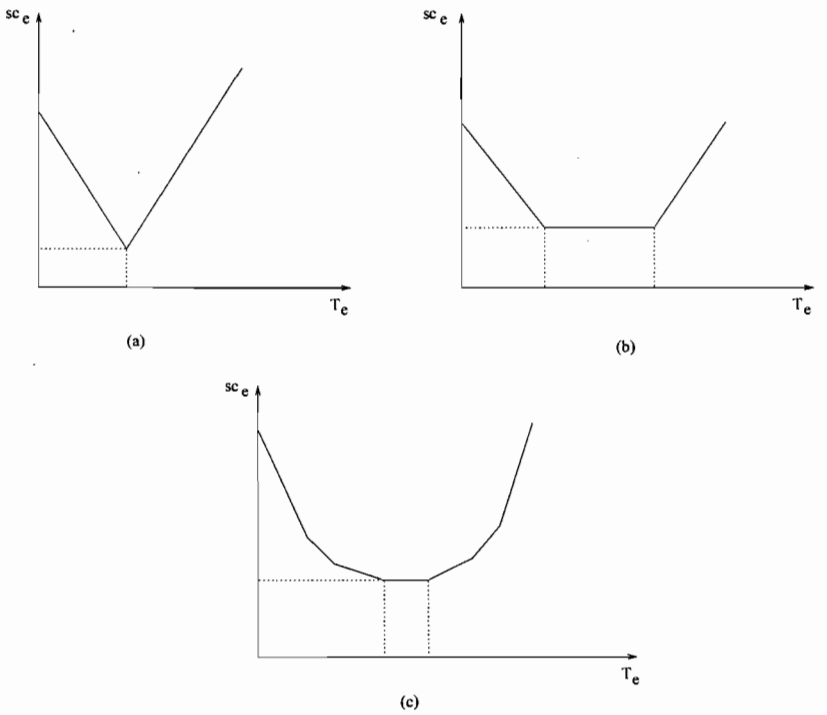

FIG. 4.1 – Different arc service cost functions

Let R_k be the route traveled by vehicle k, which is made of required arcs $(e_1, ..., e_l)$ and non-required (or deadhead) arcs $(e_{l+1}, ..., e_{l+p})$. These deadhead arcs come from the shortest paths to travel from one required arc to the next. The route cost is then

$$C(R_k) = \sum_{i=1}^{l} sc_{e_i}(T_{e_i}^k) + \sum_{i=l+1}^{l+p} tc_{e_i}, \tag{4.1}$$

where $T_{e_i}^k$ is the time of beginning of service on each required arc e_i, $i = 1, ..., l$, served by vehicle k. Since there is no waiting time along the route, the time of beginning of service on each required arc can be derived from the starting time T_0^k of vehicle k at the depot [84]. The route cost can thus be written as:

$$C(R_k) = \sum_{i=1}^{l} sc_{e_i}(T_0^k) + \sum_{i=l+1}^{l+p} tc_{e_i} = sc_k(T_0^k) + tc_k. \tag{4.2}$$

Figure 2 illustrates such a cost function. In this example, the minimum route cost

is obtained for any starting time between 5 and 6 time units.

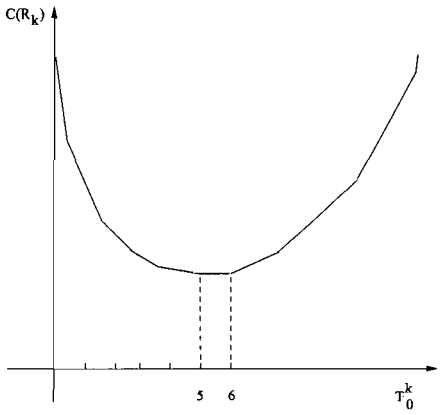

FIG. 4.2 – Route cost function

4.3 Problem-Solving Methodology

In the first phase, an initial solution is obtained with either an insertion heuristic or an adaptation of the Clarke and Wright heuristic [18]. In the second phase, a variable neighborhood descent (VND), a streamlined variant of variable neighborhood search (VNS) [71], is applied to the initial solution to improve it. The basic idea is to perform a local descent based on a number of different neighborhoods. That is, when a local optimum is reached with the current neighborhood, the search resumes with a different neighborhood to escape from the current local optimum. This is repeated until the current solution cannot be improved anymore, that is, when the solution is a local optimum over every neighborhood considered. This approach was applied, for example, in [50] to solve the CARP using neighborhood structures previously proposed in [52].

4.3.1 Initial Solution

Savings heuristic

This heuristic is based on the Clarke and Wright savings heuristic [18]. Initially, each required arc is served by a single route that starts and ends at the depot. Then,

at each iteration, the pair of routes associated with the largest savings are merged together. This is repeated until no further merging is feasible. Given two routes R_1 and R_2, the savings S_{12} is equal to $C(R_1) + C(R_2) - C(R_{12})$, where R_{12} is the route obtained when R_2 is appended to R_1.

Insertion heuristic

This is a classical sequential insertion heuristic, where the routes are constructed one by one. First, a route is created for serving the closest (unserved) required arc from the depot. Then, at each iteration a new unserved required arc is inserted into the current route, until no additional insertion is feasible. At this point, the procedure is repeated with a newly created route. The next arc to be added to the current route during the insertion procedure is the (feasible) arc that incurs the least additional cost to the route.

4.3.2 Neighborhood Structures

Each neighborhood is explored using a first-improvement local descent. These neighborhoods are described in the following in increasing order of complexity. It is worth noting that the last procedure, called shorten, does not really generate a neighborhood structure. It is aimed at reducing as much as possible the travel cost of the current routes.

Arc move

Here, a required arc is removed from one route and inserted between two other required arcs in the same route or in another route. All such moves are systematically considered until an improvement is found. The procedure is then repeated with the improved solution until a local optimum is reached. An arc move is illustrated in Figure 3. In this figure, the arcs shown are required arcs while the dotted arcs stand for one or more deadhead arcs on the shortest path from one required arc to the next. Clearly, after moving arc (v_{1j}, v_{1k}), the shortest paths between v_{1j} and v_{1l}, v_{2j} and v_{1j}, and v_{1k}

and v_{2k} are introduced into the solution to reconnect the two routes.

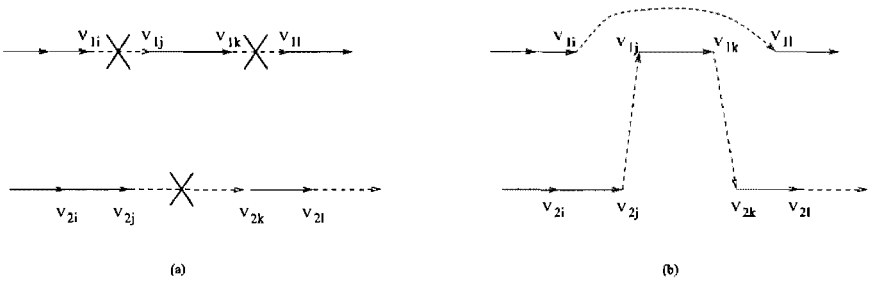

FIG. 4.3 – Arc move

Cross exchanges

Given a pair of routes in the current solution, two sequences of arcs are exchanged. The two sequences contain exactly m required arcs, $1 \leq m \leq M$, where M is a parameter set to 5 in our computational experiments. This is illustrated in Figure 4 for $m = 2$. Basically, a sequence of arcs in the first route, made of the two required arcs v_{1j} and v_{1k} plus the arcs on the shortest path between them, is exchanged with a sequence in the second route made of the two required arcs v_{2j} and v_{2k}. In the case of the second route, v_{2k} is visited immediately after v_{2j}, so there are no deadhead arcs. For a given m, all possible exchanges on every pair of routes are systematically considered until an improvement is found. The procedure is then repeated with the improved solution until a local optimum is reached.

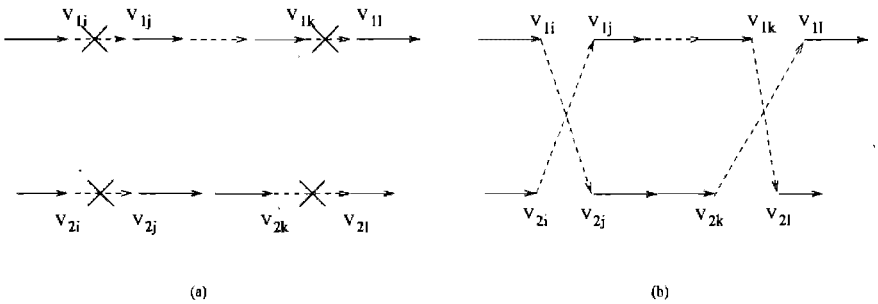

FIG. 4.4 – Cross exchange

Block exchanges

Here, sequences made of consecutive required arcs (i.e., sequences with no deadhead arcs in-between), called blocks, are identified and exchanged between two routes. The number of required arcs in a block is not limited and two blocks can be exchanged even if they do not contain the same number of required arcs. Note that exchanges involving two blocks with the same number of required arcs m, for $1 \leq m \leq M$, are discarded because they also belong to the previous cross exchange neighborhoods. For every pair of routes, all possible block exchanges are considered until an improvement is found. The procedure is then repeated with the improved solution until a local optimum is reached.

Shorten

In [52], a shorten procedure for the undirected CARP is reported to reduce the travel cost of a route. This procedure has been adapted for the directed case as follows. When an arc is crossed twice, once for serving it and once for traveling on the shortest path between two other required arcs, it might be possible to improve the route by inverting the order of the two activities on this arc. This is illustrated in Figures 5 and 6. In these figures, a broken arc is used when the vehicle travels along the arc, while a full arc is used when the vehicle serves the arc. In Figure 5(a), the vehicle first serves arc (v_i, v_j) and, later on, travels on this arc. The travel and service on (v_i, v_j) are switched in 5(b), thus providing an opportunity to travel directly from v_k to v_l in 5(c), instead of going through v_i and v_j. Figure 6 illustrates the case when the vehicle travels on the arc before serving it. In our context, even if an improvement in travel cost is observed, there is no guarantee that the solution is better overall, due to the service costs. Basically, the procedure is applied sequentially to every required arc which is crossed twice. After processing a required arc, the new solution is kept if it is better overall than the current solution. The shorten procedure then proceeds with the next required arc with this new solution.

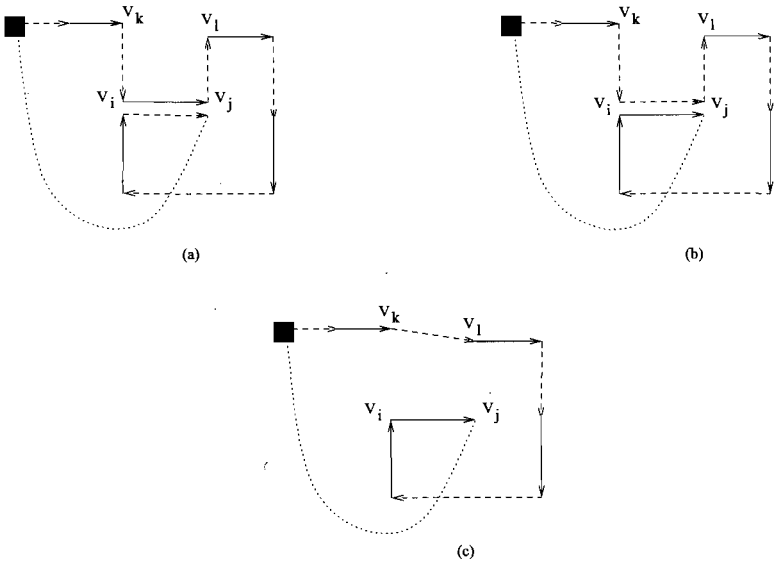

FIG. 4.5 – Shorten procedure (service before travel)

4.3.3 Variable Neighborhood Descent

The complete algorithm is described in pseudo-code in the following, where:

- N_1 is the arc move neighborhood.
- N_2 to N_{M+1} are the M cross exchange neighborhoods.
- N_{M+2} is the block exchange neighborhood.
- s^{N_j} is a local optimum solution based on neighborhood N_j, j = 1,..., M+2.

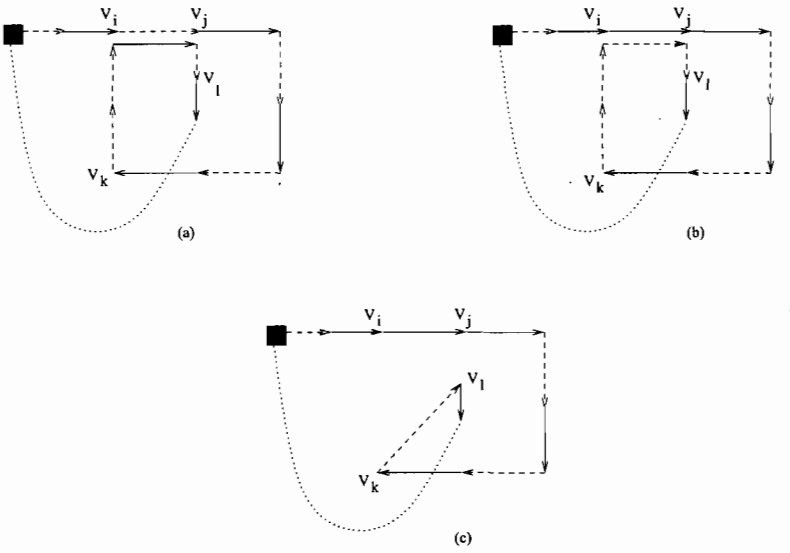

FIG. 4.6 – Shorten procedure (travel before service)

VND

Step 0: Initialization

Create an initial solution s^{N_0}.

Step 1: Loop

For $j = 1, ..., M + 2$ do:

starting from $s^{N_{j-1}}$, perform a local descent based on neighborhood N_j

and let s^{N_j} be the local optimum obtained.

if s^{N_j} is different from $s^{N_{j-1}}$ then $s^{N_0} \leftarrow s^{N_j}$ and go to Step 1.

Step 2 : Shorten

Apply shorten to $s^{N_{M+2}}$ to obtain s^{short}

If s^{short} is better than $s^{N_{M+2}}$ then $s^{N_0} \leftarrow s^{\text{short}}$ and go to Step 1; else return $s^{N_{M+2}}$.

In this procedure, Step 2 is reached and the shorten procedure is applied only when the $M + 2$ neighborhoods are explored without any improvement to the starting solution. If the shorten procedure improves the solution, the loop in Step 1 is restarted with the improved solution; otherwise the best solution found is returned. In the computational results, this variable neighborhood descent algorithm is applied twice, using the savings and the insertion heuristics to generate an initial solution. The best of the two runs is then the final result of VND.

4.4 Computational Results

The undirected CARP instances of Golden et al. [46] and Li and Eglese [63, 64], found at http://www.uv.es/~belengue/carp.html, have been used to test our VND. In these instances, the length, travel time and travel cost are the same. To use these classical instances, edges were first replaced by arcs, that is, every edge (v_i, v_j) was replaced by two arcs (v_i, v_j) and (v_j, v_i). Then, assuming that (v_i, v_j) is a required edge in the CARP instance, the best known routes for this instance were used to choose an appropriate orientation for the corresponding required arc (i.e., to select either (v_i, v_j) or (v_i, v_j) as the required arc). Then, we applied Dijkstra's algorithm to compute the shortest path between all pairs of required arcs. Finally, a service cost function was defined on each required arc. This function is similar to the one shown in Figure 1(b). It has an optimal time interval where the service cost corresponds to the travel cost in the original CARP instance. Then, on both sides of this optimal interval, the service cost increases linearly based on slope values of -1 and 1.

4.4.1 Preliminary Study

Our first results were obtained on Golden et al. instances [46] in which the graphs contain up to 27 nodes and 51 required arcs. The results were obtained using an objective function with an equal weight on service and travel costs. In these instances, the optimal time interval on each required arc was determined by considering the best

known solutions for the corresponding CARP instance. That is, these intervals were set in such a way that the best routes for the CARP also incur the minimum service costs. Thus, every required arc can be visited within its optimal time interval.

For each instance, our algorithm has generated the best known solution where each required arc is served within its optimal time interval. This is a strong indication that our algorithm behaves appropriately. Note that the Savings heuristic gives better solutions in most cases. Over the 23 instances, the savings heuristic found the best solution in 12 cases, the insertion heuristic in 4 cases and there was a tie in 7 cases. Even if the saving heuristic is superior to the insertion heuristic, the later still allow a number of best solutions to be found, so we decided to keep it.

4.4.2 Results with Service Cost Functions of Type 1

In this subsection, we report results obtained again on Golden et al. instances, but with another way of setting the optimal time interval for each required arc. Here, the time horizon is divided into three parts (early horizon, intermediate horizon, late horizon) and the optimal service time interval for each required arc belongs to exactly one of them. The required arcs are assigned to each horizon based on the the best known solution for the corresponding CARP instance. That is, the first required arcs in the routes are assigned to the early horizon, the last required arcs in the routes are assigned to the late horizon, and the remaining arcs are assigned to the intermediate horizon. The optimal service time interval for each required arc, as determined in subsection 4.4.1, is slid along the time line so as to fall within its corresponding horizon.

We also provide a comparison with the recent adaptive multi-start local search algorithm of Ibaraki et al. [54] for solving vehicle (node) routing problems with soft time window constraints, since it can handle our service cost functions. This algorithm exploits an adaptive memory that contains the best local minima encountered during the search. The routes in these elite solutions are then used to generate a new starting solution at each restart of the local search. The latter is based on classical moves for node routing problems with time windows, namely, cross-exchanges [86], 2-opt* [78]

and Or-opt [74].

To compare our VND method with the algorithm of Ibaraki et al., each arc routing problem was first transformed into an equivalent node routing problem, using the technique reported in [84]. In the latter, since there is only one way to connect two required arcs in a directed graph, a single node is created for each required arc. Thus, the size of the problem does not increase after the transformation.

As opposed to our arc routing problem, waiting times along the routes are allowed in [54]. The optimal service time intervals have thus been set to prevent any waiting time to be beneficial. That is, it is never advantageous for the vehicle to wait at its current location in order to reach the next location within its optimal service time interval. Also, two objective functions are considered in [54] and the second one, which is closer to ours, was selected. It involves a weighted sum of service cost, travel cost and overcapacity. The latter component penalizes capacity constraint violations, as the algorithm is allowed to go into the infeasible domain. We observed that the algorithm failed to find a feasible solution on some instances with the default value of 1 for the weighting coefficients. The weight associated with overcapacity was thus increased to force feasible solutions to emerge.

Each instance was first solved with our VND approach. Then, each instance was solved with the algorithm of Ibaraki et al., based on the number of vehicles obtained with VND (the number of vehicles is an input to the algorithm of Ibaraki et al.). For each instance, Table 4.1 shows the number of vehicles, total route cost (service cost plus travel cost), and CPU time in seconds to reach the best solution with VND. These CPU times have been obtained on a 3 GHz Intel Pentium IV processor. The last three columns display the same numbers for the code of Ibaraki et al. The CPU time to the best solution was preferred over the total CPU time, because the algorithm of Ibaraki et al. often finds its best solution early, even when it runs for a long time. In the case of our algorithm, the two values are similar because the VND is basically a descent approach where the best solution is generated toward the end of a run (only one additional pass through the neighborhoods is needed to confirm that no further

improvement is possible). The time to get to the best solution using VND is faster on average when compared with the algorithm of Ibaraki et al. (although this figure mostly comes from the large gaps observed for gdb9 and gdb23).

The results in Table 4.1 show that the algorithm of Ibaraki et al. ended up with a solution with fewer vehicles than the number provided in input on instances gdb8 and gdb13. Although our algorithm is not specifically designed to minimize the number of vehicles, we introduced this consideration into the objective by adding a penalty whenever an arc that leaves the depot is used. As expected, the new solutions obtained often use fewer vehicles, but at the expense of additional route costs, as shown in Table 4.2. The number of vehicles, total route cost and CPU time in seconds to reach the best solution with the algorithm of Ibaraki et al., using the reduced number of vehicles in input, are also shown in the last three columns of Table 4.2. Now, the algorithm of Ibaraki et al. could not save any additional vehicles.

The results in Table 4.2 indicate that, for the same number of vehicles, an average reduction of 1.6% on the total cost is obtained. The penalty cost for visiting required arcs outside of their optimal time interval accounts for 6.1% and 7.3% of the total cost for VND and the algorithm of Ibaraki et al., respectively. Finally, the time to get to the best solution with VND is six times faster on average when compared with the algorithm of Ibaraki et al. (although this figure mostly comes from the large gaps observed for gdb22 and gdb23).

4.4.3 Results with Service Cost Functions of Type 2

In these new experiments, the optimal time interval for service is set to $[0, x]$, where x corresponds to the upper bound of the early, intermediate and late horizons (see subsection 4.4.2). The optimal service time interval for each required arc is assigned to exactly one of these three values, based on the best known solution for the corresponding CARP instance. That is, the first required arcs in the routes are assigned to the x value associated with the early horizon, the last required arcs in the routes are assigned to the x value associated with the late horizon, and the remaining arcs are assigned to

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	CPU Time (s)	#Veh.	Route Cost	CPU Time (s)
gdb1	5	322	0.03	5	326	0.02
gdb2	6	343	0.14	6	346	0.12
gdb3	6	303	0.02	6	310	0.02
gdb4	5	321	0.02	5	323	0.02
gdb5	6	423	0.06	6	437	0.11
gdb6	5	350	0.16	5	351	0.02
gdb7	5	348	0.18	5	351	0.02
gdb8	13	368	0.63	11	375	1.43
gdb9	12	398	1.35	12	398	11.15
gdb10	5	309	0.06	5	312	0.05
gdb11	7	435	0.87	7	436	0.45
gdb12	7	464	0.03	7	480	0.04
gdb13	9	591	0.07	8	593	0.06
gdb14	5	105	0.07	5	105	0.02
gdb15	4	60	0.05	4	61	0.08
gdb16	6	131	0.05	6	133	0.10
gdb17	8	93	0.04	8	101	0.44
gdb18	5	165	0.56	5	169	0.15
gdb19	4	63	0.01	4	63	0.01
gdb20	6	125	0.02	6	125	0.02
gdb21	7	164	0.12	7	164	0.21
gdb22	15	219	0.31	15	220	0.22
gdb23	12	241	0.57	12	241	13.20
avg.	7.1	275.7	0.23	7.0	279.1	1.22

TAB. 4.1 – Results on Golden et al. instances with service cost functions of type 1

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	CPU time (s)	#Veh.	Route Cost	CPU time (s)
gdb1	5	322	0.03	5	326	0.02
gdb2	6	343	0.14	6	358	0.12
gdb3	5	324	0.03	5	326	0.03
gdb4	5	321	0.02	5	323	0.02
gdb5	6	423	0.07	6	437	0.11
gdb6	5	350	0.13	5	351	0.02
gdb7	5	348	0.18	5	351	0.02
gdb8	11	369	0.66	11	375	1.24
gdb9	11	384	1.45	11	398	3.32
gdb10	5	309	0.06	5	312	0.05
gdb11	6	424	0.87	6	438	0.22
gdb12	7	464	0.03	7	480	0.04
gdb13	7	600	0.26	7	605	0.15
gdb14	5	105	0.08	5	105	0.02
gdb15	4	60	0.06	4	60	0.12
gdb16	5	134	0.06	5	137	2.20
gdb17	5	115	0.20	5	106	0.29
gdb18	5	165	0.60	5	165	0.15
gdb19	3	69	0.01	3	71	0.01
gdb20	5	131	0.05	5	133	0.23
gdb21	7	164	0.13	7	164	0.21
gdb22	9	255	0.41	9	258	14.08
gdb23	12	241	0.60	12	241	13.20
avg.	6.3	279.1	0.26	6.3	283.5	1.55

TAB. 4.2 – Results on Golden et al. instances with service cost functions of type 1 and with penalty on number of vehicles

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	CPU Time (s)	#Veh.	Route Cost	CPU Time (s)
gdb1	6	370	0.08	6	387	0.15
gdb2	6	415	0.20	6	415	0.26
gdb3	5	331	0.10	5	352	0.17
gdb4	5	352	0.03	5	356	0.02
gdb5	6	430	0.20	6	437	0.06
gdb6	7	375	0.07	6	396	0.23
gdb7	5	336	0.04	5	357	0.13
gdb8	12	399	0.82	11	421	1.71
gdb9	10	379	1.73	10	396	6.78
gdb10	7	321	0.09	6	329	0.17
gdb11	9	467	1.14	8	478	13.49
gdb12	7	515	0.13	7	530	0.13
gdb13	9	584	0.11	9	601	0.24
gdb14	6	105	0.03	6	105	0.17
gdb15	4	60	0.07	4	60	0.18
gdb16	8	135	0.06	8	135	0.10
gdb17	9	95	0.09	7	97	0.59
gdb18	5	194	0.71	5	204	2.82
gdb19	4	63	0.01	4	66	0.02
gdb20	5	127	0.03	5	127	0.07
gdb21	6	225	0.45	6	225	6.03
gdb22	8	297	0.75	8	321	6.04
gdb23	10	327	1.22	10	341	17.87
avg.	6.9	300.1	0.35	6.7	310.3	2.48

TAB. 4.3 – Results on Golden et al. instances with service cost functions of type 2

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	Time to best (s)	#Veh.	Route Cost	Time to best (s)
gdb1	5	379	0.12	5	406	0.07
gdb2	7	385	0.14	7	407	0.13
gdb3	5	331	0.14	5	352	0.17
gdb4	4	360	0.02	4	372	0.03
gdb5	6	430	0.13	6	437	0.06
gdb6	6	373	0.07	6	396	0.18
gdb7	5	336	0.04	5	357	0.13
gdb8	11	404	0.81	11	421	0.86
gdb9	12	357	2.01	12	385	9.60
gdb10	5	311	0.24	5	331	0.09
gdb11	8	455	1.20	8	478	2.09
gdb12	7	515	0.13	7	530	0.13
gdb13	8	595	0.08	8	601	0.46
gdb14	6	105	0.04	6	105	0.17
gdb15	4	60	0.09	4	60	0.18
gdb16	8	135	0.04	8	135	0.10
gdb17	5	109	0.15	5	107	2.88
gdb18	5	194	0.69	5	204	2.82
gdb19	3	64	0.01	3	68	0.02
gdb20	6	125	0.08	6	126	0.07
gdb21	6	225	0.45	6	225	6.03
gdb22	10	256	0.52	10	262	1.64
gdb23	12	277	1.92	12	280	11.50
avg.	6.7	294.8	0.39	6.7	306.3	1.71

TAB. 4.4 – Results on Golden et al. instances with service cost functions of type 2 and with penalty on number of vehicles

the x value of the intermediate horizon. This approach is directly motivated by winter gritting operations where different priorities are associated with highways, roads close to schools, etc. The results are shown in Tables 4.3 and 4.4, using the format of Tables 4.1 and 4.2. Once again, when the VND heuristic was applied without explicitly considering the number of vehicles, the algorithm of Ibaraki et al. ended up with fewer vehicles on 5 different instances (see Table 4.3). The new results in Table 4.4 have been obtained by introducing a penalty in the objective of VND for the use of any arc that leaves the depot, as it was done in Table 4.2. In this case, the algorithm of Ibaraki et al. could not save any additional vehicle over the solutions produced by VND.

The results in Table 4.4 indicate that, for the same number of vehicles, an average reduction of 3.8% on the total cost is obtained. The penalty cost for visiting required arcs outside of their optimal time interval accounts for 7.7% and 11.3% of the total cost for VND and the algorithm of Ibaraki et al., respectively.

In both cases, this is more than what is observed for the service cost functions of type 1, although a larger increase is incurred by the algorithm of Ibaraki et al. Once again, due to the large gaps observed on a few instances, the time to obtain the best solutions with VND is about four times faster on average when compared with the algorithm of Ibaraki et al.

The service cost functions of type 2, which are closer to real applications, were also integrated into the instances of Li and Eglese [63, 64]. After applying the procedure described at the beginning of this section to generate optimal service time intervals, their width was further reduced by 20% to 25% to get more difficult instances. The instances of type *egl-e* have 77 nodes and up to 98 required edges while those of type *egl-s* have 140 nodes and up to 190 required edges. Due to the size of these instances, a more powerful Dual Core AMD Opteron 285 processor was used to run these tests. The results are summarized in Table 4.5 in the usual format, with the introduction of a penalty on any arc that leaves the depot. We note that the algorithm of Ibaraki et al. has generated one fewer route on instance *egl-s2-C*. The number of vehicles on this instance could not be reduced further with VND, even through a further increase in the

Instance	VND			Ibaraki et al.		
	#Veh.	Route Cost	CPU Time (s)	#Veh.	Route Cost	CPU Time (s)
egl-e1-A	5	3705	5.18	5	3775	13.59
egl-e1-B	7	4635	3.76	7	4878	34.71
egl-e1-C	10	5926	2.98	10	6448	16.75
egl-e2-A	8	5521	12.63	8	6301	11.12
egl-e2-B	11	6702	10.2	11	8511	5.37
egl-e2-C	15	8764	7.4	15	9025	8.4
egl-e3-A	9	6757	19.25	9	8038	20.38
egl-e3-B	13	8528	16.16	13	8711	6.69
egl-e3-C	18	10939	13.31	18	11175	37.58
egl-e4-A	11	7518	30.25	11	9160	51.15
egl-e4-B	15	9661	22.82	15	9750	22.08
egl-e4-C	21	12553	19.63	21	13834	59.79
egl-s1-A	8	5677	25.58	8	7225	27.01
egl-s1-B	10	6872	18.54	10	7517	11.85
egl-s1-C	15	8874	14.02	15	9715	96.82
egl-s2-A	14	10599	150.43	14	12827	193.21
egl-s2-B	21	14128	134.88	21	16389	38.96
egl-s2-C	28	17391	112.62	27	20630	225.83
egl-s3-A	15	11400	111.54	15	14561	72.74
egl-s3-B	22	15103	161.09	22	16122	254.49
egl-s3-C	30	18953	141.32	30	20888	121.79
egl-s4-A	20	13323	306.41	20	16760	211.58
egl-s4-B	27	16411	251.18	27	19964	114.48
egl-s4-C	37	20449	253.01	37	22536	173.89
avg.	16.25	10432.87	76.84	16.25	11864.16	76.26

TAB. 4.5 – Results on Li and Eglese instances with service cost functions of type 2 and with penalty on number of vehicles

penalty (but note, once again, that our algorithm was not really designed to minimize the number of vehicles). Overall, on 23 instances out of 24, VND has produced a solution with the same number of vehicles than the algorithm of Ibaraki et al., but with a smaller total route cost (a substantial reduction of 12%, on average). As opposed to the data set of Golden et al., the average computation time of both methods is similar.

Here, no instance looks particularly difficult to solve with the algorithm of Ibaraki et al. (or, conversely, particularly easy to solve with VND).

4.5 Conclusion

In this paper, we have considered a CARP with time-dependent service costs. To solve this problem, a heuristic method based on a variable neighborhood descent was proposed. Using instances derived from two benchmark data sets, the method was shown to be fast and competitive when compared with a recently reported algorithm. Future work will now be aimed at introducing perturbation mechanisms into the heuristic search to get a more global exploration of the solution space (at a computational cost, though). We also want to consider a dynamic variant of this problem where the service cost functions are regularly updated due to the arrival of new information, like meteorological forecasts in the case of winter gritting applications.

Acknowledgments. We would like to thank Prof. Toshihide Ibaraki and his research team for providing us with the computer code of their algorithm. Also, financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

Chapter 5

A Dynamic Capacitated Arc Routing Problem with Time-Dependent Service Costs

Résumé. Cet article étudie un problème dynamique de tournées sur les arcs avec contraintes de capacité et coûts de service dépendants du temps. À chaque arc qui doit être desservi est associée une fonction de coûts linéaire par morceaux, où le coût de service est minimal sur un certain intervalle de temps. Ce problème est motivé par le problème de déglaçage où l'évolution temporelle d'une tempête peut induire des changements dans les intervalles optimaux de service. En effet, en disposant de rapports météorologiques en temps réel, nous pouvons faire une mise à jour des fonctions de coûts et réoptimiser la solution courante. Des tests ont étés réalisés sur des instances Euclidiennes générées aléatoirement.

A Dynamic Capacitated Arc Routing Problem with Time-Dependent Service Costs

Mariam Tagmouti

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Michel Gendreau

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Jean-Yves Potvin

CIRRELT and Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Article soumis à *Transportation Research Part: C, 2009.*

Abstract. This paper describes a dynamic capacitated arc routing problem motivated from winter gritting applications. In this problem, the service cost on each arc is a piecewise linear function of the time of beginning of service. This function also exhibits an optimal time interval where the service cost is minimal. Since the timing of an intervention is crucial, the dynamic aspect considered in this work stems from changes to these optimal service time intervals due to weather report updates. A variable neighborhood descent heuristic, initially developed for the static version of the problem, where all service cost functions are known in advance and do not change thereafter, is adapted to this dynamic variant.

Keywords. capacitated arc routing, time-dependent service costs, dynamic, weather forecasts, variable neighborhood descent.

5.1 Introduction

In the capacitated arc routing problem (CARP), a set of required arcs, each with an associated demand, is served at minimum cost with a fleet of vehicles of finite capacity based at a given depot. The CARP occurs in real world applications when street segments are served, like winter gritting and street sweeping, or when the demand is aggregated over a number of locations along a street segment, like postman problems. In the classical version of the problem, there is a fixed cost associated with each arc for traveling on this arc or for serving it. In the time-dependent variant of the problem, the service cost on a required arc is also a function of the time of beginning of service.

Winter gritting was first modeled as a CARP by Eglese et Li [29]. The authors discuss routing efficiency issues based on different network characteristics. For example, the presence of T-junctions in rural networks and one-way streets and highways in urban networks are shown to significantly impact routing efficiency measures. In [28], a winter gritting problem is reported where a number of salt depots are located on the network for vehicle replenishment. The problem is solved with simulated annealing, using a savings-based heuristic [18] to generate the starting solution. In [63], the authors propose a two-phase construction heuristic. After selecting an arc to initialize a route, new arcs are inserted in the first phase by going backward from the node at one end of the selected arc back to the depot. In the second phase, arcs are inserted by going forward from the node at the other end of the selected arc to the depot. In [15], the authors first divide the service area into districts that are served from the same depot. Then, routes are constructed in each district with the Augment-Insert heuristic [76]. In [66], a combined depot location-arc routing problem is reported.

Apart from winter gritting, other real-life problems modeled as CARP are:

- Street sweeping: this problem was first studied in [6, 7]. In this work, each arc has to be serviced within a given time window.
- Electric meter reading: the "capacity" constraint here comes from the fact that each employee has a work time limit, see [83].

- Refuse collection: this problem, first considered in [8], is a CARP with two-dimensional capacity constraints due to a true capacity constraint for each vehicle and a work time limit for each driver.

- Snow removal: this problem was addressed in a rural context in [49]. In this work, the authors both consider the minimization of operating costs and the maximization of road safety while the snow is being removed.

These real-life problems have all been studied in a static context, where it is assumed that all data about the problem are known in advance. However, this is not necessarily the case in real-word applications, where some information might not be readily available when the vehicles start their routes. When new information is unveiled as the routes are executed, the problem becomes dynamic. Although there is fair number of papers on dynamic node routing problems, particularly with regard to the integration of new customer requests into vehicle routes [43, 55, 80], dynamic CARPs have not attracted yet the attention of the research community. In this work, weather report updates lead to modifications to the optimal time of beginning of service on each required arc and thus, to dynamic modifications to the current routes. The service cost function considered on each required arc is a piecewise linear function of the time of beginning of service. This function also exhibits an optimal time interval where the service cost is minimal [84].

The paper is organized as follows. In Section 2, the static version of the problem is formally introduced and the variable neighborhood descent (VND) heuristic proposed in [85] to solve it is briefly described. In Section 3, the dynamic variant is introduced, followed by a description of the corresponding adaptation of the VND. In Section 4, the problem generator to run the tests is detailed and numerical results are reported.

5.2 The Static Problem

5.2.1 Problem Definition

Let $G = (V, A)$ be a directed graph where V is the vertex set and A is the arc set. A is partitioned into a subset of required arcs A_1, and a subset of non required arcs A_2. With each required arc $e \in A_1$ is associated a demand d_e, a length l_e, a travel time tt_e, a service time st_e, a travel cost tc_e and a time-dependent piecewise linear service cost function $sc_e(T_e)$, where T_e is the time of beginning of service on arc e. The arcs in A_2 have a length, a travel time and a travel cost only. A set of K identical vehicles, each with capacity Q, is available to serve the required arcs.

The objective is to serve all required arcs in the graph with least-cost feasible routes, where the cost is the sum of service costs and travel costs. More precisely, let r_k be the route traveled by vehicle k, which is made of required arcs $(e_1, ..., e_l)$ and non-required (or deadhead) arcs $(e_{l+1}, ..., e_{l+p})$. The route cost is then:

$$C(r_k) = \sum_{i=1}^{l} sc_{e_i}(T_0^k) + \sum_{i=l+1}^{l+p} tc_{e_i} = sc_k(T_0^k) + tc_k, \tag{5.1}$$

where T_0^k is the start time from the depot node of the route served by vehicle k. Given that no waiting time is allowed along a route, the time of beginning of service on any required arc T_{e_i} can be easily derived from the start time of the route T_0^k [84].

Figure 1 shows typical piecewise linear service cost functions. The type illustrated in Figure 1(b), where the flat portion corresponds to the optimal service time interval, is the one used in our computational experiments.

5.2.2 Problem-Solving Methodology

The problem-solving methodology for solving the static version of the problem is reported in [85]. We briefly restate it here for the sake of completeness.

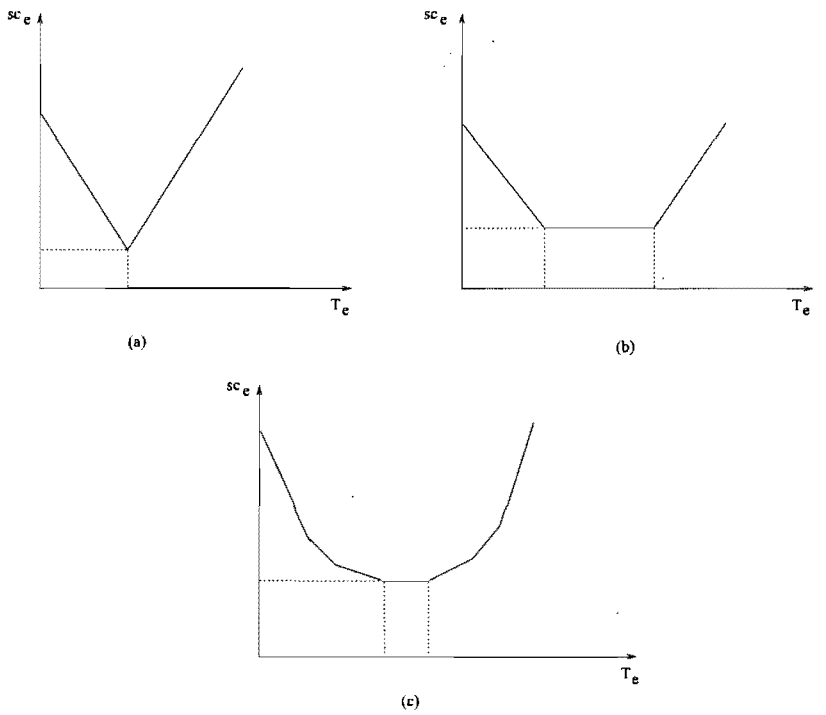

FIG. 5.1 – Different types of service cost functions

Initial solution

In the first phase, an initial solution is obtained with either an insertion heuristic or an adaptation of the Clarke and Wright's savings heuristic [18].

- *Savings heuristic :*

  Initially, each required arc is served by a single route that starts and ends at the depot. Then, at each iteration, the pair of routes associated with the largest savings are merged together. This is repeated until no further route merging is feasible.

- *Insertion heuristic*

  First, a route is created for serving the closest required arc from the depot. Then, at each iteration a new unserved required arc is inserted into the route, until no additional insertion is feasible. Additional routes are constructed sequentially in this way until all required arcs are served.

Neighborhood structures

Each neighborhood is explored using a first-improvement local descent. The last procedure presented below, called shorten, does not really generate a neighborhood structure. It is simply aimed at reducing as much as possible the travel cost of the routes by inverting the service and travel on a given arc, when this arc is crossed twice.

- *Arc move*

  Here, a required arc is removed from one route and inserted between two other required arcs in the same route or in another route.

- *Cross exchanges*

  Given a pair of routes in the current solution, two sequences of arcs are exchanged. The two sequences contain exactly m required arcs, $1 \leq m \leq M$, where M is a parameter set to 5 in our computational experiments.

- *Block exchanges*

  Here, sequences made of consecutive required arcs with no deadhead arcs in-between, called blocks, are identified and exchanged between two routes. The number of required arcs in a block is not limited and two blocks can be exchanged even if they do not contain the same number of required arcs.

- *Shorten*

  When a required arc is crossed twice, once for serving it and once for traveling on the shortest path between two other required arcs, it might be possible to improve the route by inverting the order of the two activities on this arc. This is illustrated in Figures 2 and 3, using full arcs for service and broken arcs for travel. In the first case, the service is postponed after the travel on arc (v_i, v_j) while, in the second case, the travel is postponed after the service. In our context, even if an improvement in travel cost is observed, there is no guarantee that the solution is better overall, due to the time-dependent service costs.

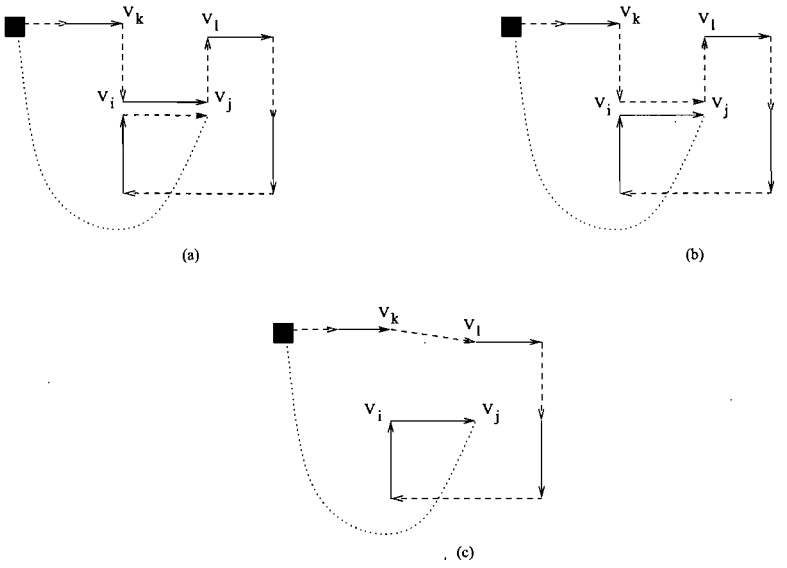

FIG. 5.2 – Shorten procedure (service before travel).

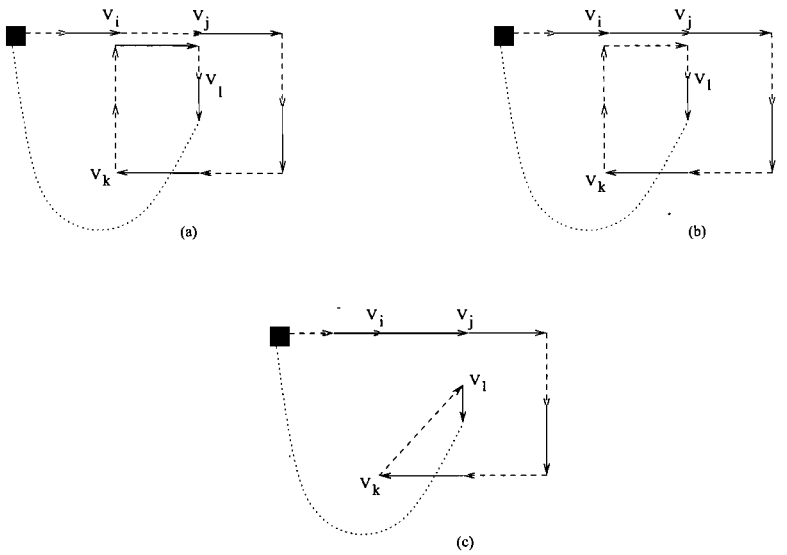

FIG. 5.3 – Shorten procedure (travel before service).

Basically, the procedure is applied sequentially to every required arc which is crossed twice. After processing a required arc, the new solution is kept if it is better than the current solution. The shorten procedure then proceeds with the next required arc.

Variable neighborhood descent

The complete VND algorithm is described in pseudo-code in the following, where:

- N_1 is the arc move neighborhood.
- N_2 to N_{M+1} are the M cross exchange neighborhoods.
- N_{M+2} is the block exchange neighborhood.
- s^{N_j} is a local optimum solution based on neighborhood N_j, $j = 1, ..., M + 2$.

Step 0: Initialization

Create an initial solution s^{N_0}.

Step 1: Loop

For $j = 1, ..., M + 2$ do:

starting from $s^{N_{j-1}}$, perform a local descent based on neighborhood N_j

and let s^{N_j} be the local optimum obtained.

if s^{N_j} is different from $s^{N_{j-1}}$ then $s^{N_0} \leftarrow s^{N_j}$ and go to Step 1.

Step 2 : Shorten

Apply shorten to $s^{N_{M+2}}$ to obtain s^{short}

If s^{short} is better than $s^{N_{M+2}}$ then $s^{N_0} \leftarrow s^{\text{short}}$ and go to Step 1; else return $s^{N_{M+2}}$.

This algorithm is executed twice, once with each initialization heuristic, and the best solution is returned at the end.

5.3 Dynamic Variant

In the dynamic variant of the problem, a starting solution is first computed with VND using service time cost functions based on some initial forecast. As vehicles execute their routes, regular weather report updates lead to modifications to the optimal service time interval associated with each required arc. This is explained in the following.

5.3.1 Context

We assume that a storm (e.g., freezing rain) goes through a city represented as a square in the euclidean plane, where one unit of time is equivalent to one unit of euclidean distance. A first solution is available and computed beforehand based on an initial storm position $(x_0, y_0) = (0, 0)$ at time $t = 0$ and some initial storm speed forecast along the $x-$ and $y-$axis $(sp_0, 0)$. For simplicity purposes, it is thus assumed that the storm moves along the x-axis only. At time h, $2h$, $3h$, ..., where h is a fixed time step, weather reports are received that update the storm speed. This is illustrated in Figure 4, where the storm front is represented as a vertical bar that moves across the euclidean plane (assuming that one unit of time corresponds to one unit of euclidean distance and that the speed of the storm is fixed at one).

In our simulations, the speed varies over time. Namely, if the speed is sp_{t-h} at time $t - h$, then the speed at the current time t is :

$$sp_t = sp_{t-h}(1 + \xi_t),$$

where ξ_t is uniformly randomly chosen in $[-\alpha, \alpha]$, with α a parameter.

5.3.2 Definition of New Static Problems

Basically, the VND is applied on a new static problem each time an information update (weather report) is received. Each static problem is defined as follows:

- all required arcs that have been serviced are removed from the problem.
- all required arcs that are currently served are assumed to be served by the same vehicle up to their endpoint, using their current service time function.
- all other required arcs have their service time functions updated.

The next section explains how the update is performed on the third category of required arcs.

5.3.3 Service Cost Function Updates

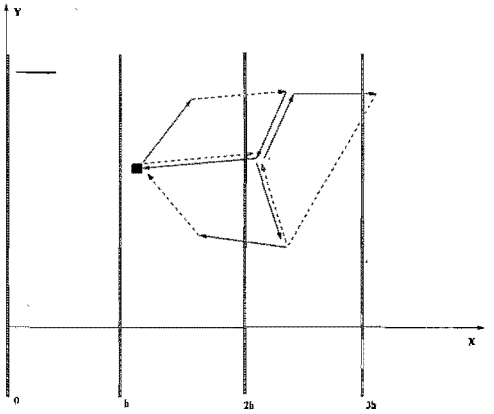

FIG. 5.4 – Example of storm movement.

When the speed of the storm varies, the optimal service time interval of each required arc must be shifted along the time line. The time $mt_e(t)$ at which the storm will reach the middle of a required arc $e = (v_i, v_j)$, based on the current time t, is first determined using its new speed:

$$mt_e(t) = t + \frac{mx_e - x_t}{sp_t},$$

where t is the current time, x_t is the location of the storm along the x axis at time t, sp_t is the speed of the storm at time t and $mx_e = \frac{x_{v_i} + x_{v_j}}{2}$ is the middle of arc e.

We chose $mt_e(t)$ to be the middle of the optimal service time interval for arc e. This interval is thus defined as $[mt_e(t) - \gamma, mt_e(t) + \gamma]$, where γ is a parameter set to 3 time units in our computational experiments.

5.3.4 Time Projection

The computation of a new solution with VND takes some time. Accordingly, when a new weather report is received, the current solution is followed for an additional Δt time units using the updated service time functions. During that time, the solution is optimized with VND based on the projected state of the system at time $t + \Delta t$. Due to this time projection, the optimized solution can then be applied as soon as it is available.

5.3.5 Overall Procedure

The evolution of the system over time can be summarized in pseudo-code as follows, using an initial solution S_0 at time $t = 0$ based on some initial storm position x_0 and speed sp_0. As previously mentioned, the storm moves only along the x-axis.

Step 0: Initialization

$t \leftarrow h$;

Step 1: Loop

While $(t \leq t_{max})$

define new static problem with updated service cost functions for

unserved required arcs, based on new storm location x_t and speed sp_t;

set projected time $t_{proj} \leftarrow t + \Delta t$;

follow solution S_{t-h} up to t_{proj} while optimizing the solution with VND

from time t_{proj} onward;

implement new solution S_t obtained with VND from time t_{proj} onward;

$t \leftarrow t + h$.

5.3.6 A Small Example

Figure 5 illustrates a route r with two required arcs e_1 and e_2 with both service time and optimal service cost equal to 1. The travel times and travel costs on non required arcs are also equal to 1. Here, the route cost is recomputed at time $t = 3$, based on new data. As usual, the storm is initially located at $x_0 = 0$ and its speed is 1 euclidean distance unit (du) by time unit (tu), that is $sp_0 = 1$ du/tu, where the distance and time units are the same.

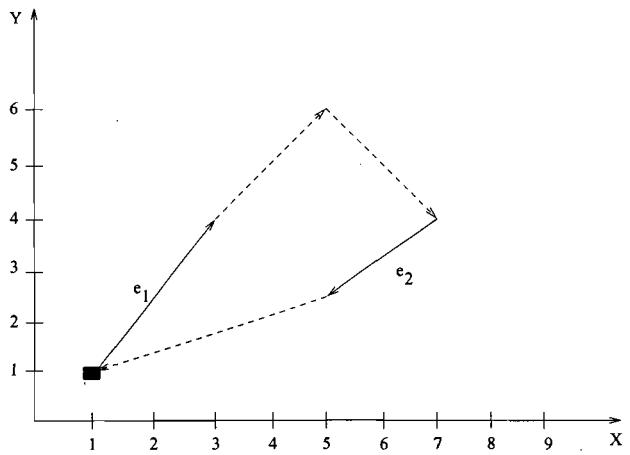

FIG. 5.5 – A small example

Since the middle of arcs e_1 and e_2 is located at $x = 2$ and 6, respectively, their optimal service time interval corresponds to $[1, 3]$ and $[5, 7]$, respectively, assuming that $\gamma = 1$. If the vehicle starts its route at $t = 2$, then it serves e_1 within its optimal time interval and the cost is 1. Then, after traveling on the two non required arcs at a cost of 2, it reaches e_2 at time $t = 5$, thus within its optimal time interval, for a cost of 1. Finally, the last arc is traveled at a cost of 1, for a total route cost $C_r = 5$. Now, suppose that a new weather report is received at time $t = 3$, when the storm is located

at $x = 3$, and where the speed of the storm is modified to 0.8 du/tu. At this point, a new problem is defined. Since the vehicle has just finished its service on arc e_1, this arc is removed from the problem and a new optimal service time interval is computed for e_2. With its reduced speed, the storm will now reach the middle of arc e_2 at time 6.75 and the optimal time interval becomes $[5.75, 7.75]$. As the new problem contains only required arc e_2, there is nothing to optimize. Thus, the vehicle will arrive at e_2 at time $t = 5$ and an additional cost of 0.75 is incurred (if we assume that the service cost function increases linearly outside of the optimal interval with a slope of one) for a total route cost $C_r = 5.75$. It should be noted that if we knew everything in advance, a better solution could have been obtained. A solution of cost $C_r = 5$ is produced simply by moving the start time of the route from $t = 2$ to $t = 3$, as the vehicle will now reach both required arcs within their optimal time interval. The difference 5.75 - 5 = 0.75 between the two solutions is the value of advanced (or perfect) information.

5.4 Computational Results

In this section, we first describe the test instances. Then, we report computational results obtained with our VND on these instances.

5.4.1 Test Problems

A simulator was developed to test our algorithm. In this simulator, the vertices are generated within a square in the euclidean plane. The horizontal and vertical sides of the square are divided into n_{max} segments to obtain a grid with n_{max}^2 smaller squares. In each one of them, a vertex is randomly generated, and the depot is located in the bottom left square. Each vertex is connected to the vertices in the North, South, East and West squares with two arcs, one in each direction. Then, 10% of those arcs are eliminated to break the symmetry, while keeping the graph strongly connected. Among the remaining arcs, 50% of them are randomly selected and defined as required arcs. Their demand is randomly set between 1 and 20 while the vehicle capacity is set

between 30 and 130.

We generated 3 types of problems with $5 \times 5 = 25$, $7 \times 7 = 49$ and $10 \times 10 = 100$ vertices and with 36, 76, and 162 required arcs, respectively. The first two types of problems are defined in a 10 km $\times$ 10 km squared area which is represented as a 100 $\times$ 100 euclidean square. Accordingly, 1 *du* corresponds to 0.1 km. The vehicle speed is set at 30 km/h or 1 du/tu (where the distance and time units are the same). Thus, 1 *tu* corresponds to 12 seconds. The storm is initially located at $x_0 = 0$ with an initial speed sp_0 of 6 km/h or 0.2 du/tu. New weather reports are received every 5 minutes or 25 *tu*.

In the last type of problems with 100 vertices, a squared area of 25 km $\times$ 25 km is used. Accordingly, 1 *du* is 0.25 km and 1 *tu* corresponds to 30 seconds, assuming again that the vehicle speed is 30 km/h. As in the previous case, the storm is initially located at $x_0 = 0$ with an initial speed of 6 km/h or 0.2 du/tu. New weather reports are received every 5 minutes or 10 *tu*.

5.4.2 Results

Tables 6.1 to 5.3 show the average results obtained on each type of problems based on 20 different instances for each type, using three different values for parameter α. We recall that this parameter is used to define the bounds of the storm speed updates, with larger values of α leading to larger speed modifications. In the tables, the first column corresponds to parameter α while the second column contains the number of vehicles in the solutions. The third column contains the cost of the *a priori* solutions, the ratio of their cost to the *a posteriori* solutions (see below) and the computation time in seconds, which includes the time to construct the initial solutions plus the postoptimization time with VND. The *a priori* solutions are the solutions obtained with the initial service cost functions (based on storm speed sp_0), but evaluated in the dynamic setting. The fourth column contains the same values, but for the dynamic solutions. The computation time reported for these solutions corresponds to the largest reoptimization time with VND over all static problems defined during the course of the

dynamic process. These times are negligible on the 25 and 50-vertex instances and Δt was set to zero during the simulations. In the case of the 100-vertex instances, Δt was set of $1tu$ or 30 seconds. The last column is the cost of the *a posteriori* solutions, which serve as a basis to quantify the benefits of advanced information, because they are computed with the true service cost functions, namely those obtained at the end of the dynamic process when everything is known. The computation times are not shown for these solutions, because they are very similar to those of the *a priori* solutions.

The cost of the dynamic solutions lies between the cost of the *a priori* and *a posteriori* solutions. The *a priori* solutions are not very good, because they are generated under the initial conditions, but evaluated in the dynamic setting. This explains, in particular, the fast degradation of these solutions with increasing α values. Clearly, when α is large, the initial conditions change a lot and constitute a poor approximation of the true dynamic conditions. On the instances with 25 vertices, the dynamic solutions are quite good for $\alpha = 0.05$ and 0.1, and lie well within 10% of the *a posteriori* solutions, which are based on perfect information. However, when $\alpha = 0.4$, the gap jumps to 26.7%. A substantial increase of this gap is also observed on the 50-vertex instances. However, this gap stabilizes on the largest 100-vertex instances, due to a smaller time step between two updates on these instances (i.e., 10 tu versus 25 tu), which leads to more frequent calls to the reoptimization procedure.

α	#Veh.	A priori			Dynamic			A posteriori
		cost	ratio	CPU	cost	ratio	CPU	
0.05	7.7	3156.3	1.149	1.03	2850.1	1.042	0.05	2736.7
0.1	7.7	3574.9	1.306	1.04	2919.1	1.070	0.05	2735.0
0.4	7.7	5429.2	1.957	1.03	3522.5	1.267	0.09	2786.3

TAB. 5.1 – 25-vertex instances

α	#Veh.	A priori			Dynamic			A posteriori
		cost	ratio	CPU	cost	ratio	CPU	
0.05	16.8	5766.1	1.162	9.39	5409.9	1.089	0.18	4968.6
0.1	16.8	6668.3	1.351	9.17	5669.7	1.146	0.32	4950.3
0.4	16.8	10445.5	2.126	10.38	7404.1	1.502	0.81	4933.6

TAB. 5.2 – 49-vertex instances

α	#Veh.	A priori			Dynamic			A posteriori
		cost	ratio	CPU	cost	ratio	CPU	
0.05	22.2	9785.5	1.300	242.18	8188.0	1.086	4.74	7552.3
0.1	22.2	14085.5	1.760	214.48	9240.5	1.164	8.31	7880.0
0.4	22.2	23430.9	2.920	247.75	11951.7	1.485	12.17	8067.3

TAB. 5.3 – 100-vertex instances

5.5 Conclusion

In this paper, we have tackled for the first time a dynamic CARP with time-dependent service costs, using a variable neighborhood descent heuristic. We intend to pursue this line of research by improving the proposed problem-solving methodology and by getting closer to real-world applications. With regard to the first issue, it would be interesting to consider an approach that would better utilize the time available between two weather reports. The VND is very fast and allows the system to quickly use the new solution. However, it could be beneficial to use more time for reoptimization, even if it means that the new solution will be made available a little bit later. This trade-off could certainly be the subject of a further study.

Acknowledgments. Financial support for this work was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

Chapter 6

Conclusion

Cette thèse a présenté différents algorithmes pour la résolution du problème de tournées de véhicules sur les arcs avec contraintes de capacité et coûts de service dépendants du moment de début du service. Pour résoudre ce problème, nous avons d'abord utilisé une méthode exacte basée sur la génération de colonnes. Pour ce faire, nous avons commencé par transformer le problème de tournées sur les arcs en un problème équivalent de tournées sur les sommets. La difficulté de cette approche résidait dans la résolution des sous-problèmes qui, dans notre cas, correspondaient à des problèmes de plus courts chemins éleméntaires avec contraintes de ressources.

Dans un second temps, nous avons travaillé directement sur le problème de tournées sur les arcs et avons développé une heuristique de descente à voisinage variable. Les mouvements proposés pour transformer la solution courante sont des échanges d'arcs ou de blocs d'arcs entre les tournées. La méthode s'est avérée rapide et efficace suite à une comparaison avec un algorithme récent qui résout des problèmes de tournées sur les sommets avec pénalités temporelles.

Dans le dernier travail, nous nous sommes penchés sur une version dynamique du problème. Dans le cas considéré, de nouveaux rapports météo sur l'évolution d'une tempête sont reçus à intervalle régulier, entraînant une révision des fonctions de coûts de service sur les arcs. Ce problème dynamique a été abordé en résolvant une suite

de problèmes statiques au fil du temps, où un nouveau problème statique est défini à chaque fois que de nouvelles informations sont reçues.

Les problèmes pratiques reliés au déglaçage des routes, qui est à la source de cette thèse, sont encore peu étudiés et comportent de nombreuses pistes de recherche pour le futur. Par exemple:

1. En pratique, le temps de service dépend aussi du moment du début de l'intervention sur un arc. Ceci s'explique par les conditions plus difficiles (e.g., chaussée plus glissante) à l'extérieur de la fenêtre optimale de service qui forcent ainsi le véhicule à rouler plus lentement lors de l'épandage. Il faudrait donc pouvoir traiter le problème de tournées sur les arcs avec temps et coûts de service dépendants du moment de l'intervention.

2. L'extension à la version multi-dépôts, incluant des dépôts intermédiaires ou les véhicules peuvent se recharger, serait également intéressante à aborder en nous permettant de nous rapprocher des applications réelles.

3. Enfin, nous pourrions aborder d'autres problématiques liées à la nature dynamique des problèmes rencontrés en pratique (e.g., bris d'un véhicule forçant une révision en temps réel des tournées courantes).

Bibliography

[1] A Amberg, W Domschke, and S Voss. Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees. *European Journal of Operational Research*, 124:360–376, 2000.

[2] AA Assad and BL Golden. Arc routing methods and applications. *MO Ball et al. (eds), Network Routing, Elsevier*, pages 375–483, 1995.

[3] R Baldacci and V Maniezzo. Exact methods based on node routing formulations for undirected arc routing problems. *Networks*, 47:52–60, 2006.

[4] JM Belenguer and E Benavent. A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 30:705–728, 2003.

[5] E Beltrami and L Bodin. Networks and vehicle routing for municipal waste collection. *Networks*, 4:65–94, 1974.

[6] LD Bodin and SJ Kursh. A detailed description of a computer-assisted system for the routing and scheduling of street sweepers. *Computers & Operations Research*, 6:181–198, 1979.

[7] LD Bodin and SJ Kursh. A computer-assisted system for the routing and scheduling of street sweepers. *Operations Research*, 26:525–537, 1987.

[8] LD Bodin, G Fagin, J Greenberg, and R Welebny. The design of a computerized sanitation vehicle routing and scheduling system for the town of oyster bay, new york. *Computers & Operations Research*, 16:45–54, 1989.

[9] SE Boselly. Benefit-cost assessment of the utility of road weather information systems for snow and ice control. *Transportation Research Record*, 1352:75–82, 1992.

[10] J Bouliane and G Laporte. Locating postal relay boxes using a set covering algorithm. *American Journal of Mathematical and Management Sciences*, 12:65–74, 2002.

[11] J Brandao and RW Eglese. A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35:1112–1126, 2008.

[12] JF Campbell and A Langevin. Roadway snow and ice control. *M Dror (ed.), Arc Routing: Theory, Solutions and Applications*, pages 389–418, 2000.

[13] JF Campbell and A Langevin. Sector design for snow removal and disposal in urban areas. *Socio-Economic Planning Sciences*, 36:183–202, 2002.

[14] JF Campbell and A Langevin. The snow disposal assignment problem. *Journal of Operational Research Society*, 46:219–229, 1995.

[15] D Cattrysse, T Lotan, L Muyldermans, and DV Oudheusden. Districting for salt spreading operations. *European Journal of Operational Research*, 139:521–532, 2002.

[16] L Chapleau, JA Ferland, G Lapalme, and JM Rousseau. A parallel insert method for the capacitated arc routing problem. *Operations Research Letters*, 3:95–99, 1984.

[17] N Christofides. The optimum traversal of a graph. *OMEGA, The International Journal of Management Science*, 1:719–732, 1973.

[18] G Clarke and J Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[19] J Clossey, G Laporte, and P Soriano. Solving arc routing problems with turn penalities. *Journal of the Operational Research Society*, 52:433–439, 2001.

[20] TM Cook and BS Alprin. Snow and ice removal in urban environment. *Management Science*, 23:227–234, 1976.

[21] JF Cordeau and G Laporte. Modeling and optimization of vehicle routing and arc routing problems. *Les Cahiers du GERAD*, G-2002-30, 2002.

[22] J Desrosiers, Y Dumas, MM Solomon, and F Soumis. Time constrained routing and scheduling. *M.O. Ball et al. (eds), Network Routing, Elsevier*, pages 35–139, 1995.

[23] KF Doerner, RF Hartl, V Maniezzo, and M Reimann. An ant system metaheuristic for the capacitated arc routing problem. In *5th Metaheuristics International Conference*. Tokyo, Japan, 2003.

[24] M Dror. Arc routing: Theory, solutions and applications. *Kluwer*, 2000.

[25] M Dror, H Stern, and P Trudeau. Postman tour on a graph with precedence relation. *Networks*, 17:283–294, 1987.

[26] J Edmonds and EL Johnson. Euler tours and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[27] RW Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46:271–281, 1990.

[28] RW Eglese. Routeing winter gritting vehicles. *Discrete Applied Mathematics*, 48: 231–244, 1994.

[29] RW Eglese and LYO Li. Efficient routing for winter gritting. *Operational Research Society*, 43:1031–1034, 1992.

[30] RW Eglese and H Mordock. Routing road sweepers in a rural area. *Journal of Operational Research Society*, 42:281–288, 1991.

[31] HA Eiselt, M Gendreau, and G Laporte. Arc routing problems, part I : The chinese postman problem. *Operations Research*, 43:231–242, 1995.

[32] HA Eiselt, M Gendreau, and G Laporte. Arc routing problems, part II : The rural postman problem. *Operations Research*, 43:399–414, 1995.

[33] D Feillet, P Dejax, M Gendreau, and C Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints : Application to some vehicle routing problems. *Networks*, 44:216–229, 2004.

[34] G Fleury, P Lacomme, and C Prins. Evolutionary algorithms for stochastic arc routing problems. *Evo Workshop 2004, Lecture Notes in Computer Science*, 3005: 501–512, 2004.

[35] LR Ford and DR Fulkerson. Flows in networks. *Princeton University Press, Princeton. NJ*, pages 70–71, 1962.

[36] GN Frederickson. Approximation algorithms for some routing problems. *Journal of the Association for Computing Matchinery*, 26:538–554, 1979.

[37] E Gélinas. Le problème du postier chinois avec contraintes générales de préséance. *Mémoire de Maitrise, École Polytechnique de Montréal, Canada*, 1992.

[38] M Gendreau, A Hertz, and G Laporte. Tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[39] M Gendreau, F Guertin, JY Potvin, and E Taillard. Parallel tabu search for real time vehicle routing and dispatching. *Transportation Science*, 33:381–390, 1999.

[40] G Ghiàni and G Laporte. A branch-and-cut algorithm for the undirected rural postman problem. *Mathematical Programming Series A*, 87:467–481, 2000.

[41] G Ghiani and G Laporte. Eulerian location problems. *Networks*, 34:291–302, 1999.

[42] G Ghiani, G Improta, and G Laporte. The capacitated arc routing problem with intermediate facilities. *Networks*, 37:134–143, 2001.

[43] G Ghiani, F Guerriero, G Laporte, and R Musmanno. Real-time vehicle routing: solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 141:1–11, 2003.

[44] J Gilbert. Générateur d'itinéraires d'enlèvement de la neige. *Rapport technique CRT-648, Centre de Recherche sur les Transports, Université de Montréal*, 1989.

[45] BL Golden and RT Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[46] BL Golden, JS DeArmon, and EK Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10: 47–59, 1983.

[47] P Greistorfer. A tabu scatter search metaheuristic for the arc routing problem. *Computers & Industrial Engineering*, 44:249–266, 2003.

[48] C Gueguen. Méthodes de résolution exacte pour les problèmes de tournées de véhicules. *Thèse de doctorat, Laboratoire production logistique, École centrale de Paris*, 1999.

[49] E Haslan and JR Wright. Application of routing technologies to rural snow and ice control. *Transportation Research Record*, 1307:202–211, 1991.

[50] A Hertz and M Mittaz. A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. *Transportation Science*, 35:425–434, 2001.

[51] A Hertz, G Laporte, and P Nanchen-Hugo. Improvement procedures for the undirected rural postman problem. *INFORMS Journal on Computing*, 11:53–62, 1999.

[52] A Hertz, G Laporte, and M Mittaz. A tabu search heuristic for the capacitated arc routing problem. *Operations Research*, 48:129–135, 2000.

[53] R Hirabayashi, Y Saruwatira, and N Nishida. Tour construction algorithm for the capacitated arc routing problem. *Asia Pacific Journal of Opererational Research*, 9:155–175, 1992.

[54] T Ibaraki, S Imahori, M Kudo, T Masuda, T Uno, and M Yagiura. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, 39:206–232, 2005.

[55] S Ichoua, M Gendreau, and JY Potvin. Diversion issues in real-time vehicle dispatching. *Transportation Science*, 34:426–438, 2000.

[56] I Ioachim, S Gélinas, J Desrosiers, and F Soumis. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31:193–204, 1998.

[57] N Labadi, C Prins, and M Reghioui. GRASP with path relinking for the capacitated arc routing problem with time windows. *Computational Intelligence in Transport Logistics and Supply Chain Management, A Fink and F Rothlauf (eds.), Studies in Computational Intelligence*, 144:111–135, 2008.

[58] P Lacomme, C Prins, and W Ramdane-Chérif. A GA for the CARP and its extensions. *E.J.W. Boers et al. (eds), Applications of Evolutionary Computing, Lecture Notes in Computer Science 2037, Springer*, pages 473–483, 2001.

[59] P Lacomme, C Prins, and W Ramdane-Chérif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131:159–185, 2004.

[60] P Lacomme, C Prins, and W Ramdane-Chérif. Evolutionary algorithms for period arc routing problems. *Europen Journal of Operational Research*, 165:535–553, 2005.

[61] G Laporte. The vehicle routing problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[62] AN Letchford and RW Eglese. The rural postman problem with deadline classes. *European Journal of Operational Research*, 105:390–400, 1998.

[63] LYO Li. Vehicle routeing for winter gritting. *Ph.D. Thesis, Dept. of Management Science, Lancaster University, UK*, 1992.

[64] LYO Li and RW Eglese. An iterative algorithm for vehicle routeing for winter-gritting. *Journal of the Operational Research Society*, 47:217–228, 1996.

[65] H Longo, MP de Aragão, and E Uchoa. Solving capacitated arc routing problems using a transformation to the CVRP. *Computers & Operations Research*, 33: 1823–1837, 2006.

[66] T Lotan, D Cattrysse, and D Van Oudheusden. Winter gritting in the province of Antwerp: a combined location and routing problem. *Technical Report IS-MG 96/4, Institut de Statistique, Université Libre de Bruxelles*, 1996.

[67] JW Male and JC Liebman. Districting and routing for solid waste collection. *Journal of the Environmental Engineering Division*, 104(1):1–14, 1978.

[68] V Maniezzo. Algorithms for large directed CARP instances: Urban solid waste collection operational support. *Technical Report UBLCS-2004-16, Department of Computer Science, University of Bologna, Italy*, 2004.

[69] S Martello and P Toth. Knapsack problems : Algorithmic and computer implementations. *Wiley, Chichester*, 1990.

[70] K Mei-Ko. Graphic programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.

[71] N Mladenović and P Hansen. Variable neighborhood search. *Computers & Operations Researchs*, 24:1097–1100, 1997.

[72] PA Mullaseril. Capacitated rural postman problem with time windows and split delivery. *Ph.D Thesis, MIS Departement, University of Arizona*, 1997.

[73] Y Nobert and JC Picard. An optimal algorithm for the mixed Chinese postman problem. *Networks*, 27:95–108, 1996.

[74] I Or. Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking. *PhD Thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University, USA*, 1976.

[75] WL Pearn. Approximate solutions for the capacitated arc routing problem. *Computers & Operations Research*, 16:589–600, 1989.

[76] WL Pearn. Augment-insert algorithms for the capacitated arc routing problem. *Computers & Operations Research*, 18:189–198, 1991.

[77] WL Pearn, AA Assad, and BL Golden. Transforming arc routing into node routing problems. *Computers & Operations Research*, 14:285–288, 1987.

[78] JY Potvin and JM Rousseau. An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society*, 46:1433–1446, 1995.

[79] HN Psaraftis. Dynamic vehicle routing problem. *Vehicle Routing: Methods and Studies, B Golden and AA Assad (ed.) North-Holland, Amsterdam*, pages 223–248, 1988.

[80] HN Psaraftis. Dynamic vehicle routing : Statuts and prospects. *Annals of Operations Research*, 61:143–164, 1995.

[81] S Roy and JM Rousseau. The capacitated canadian postman problem. *Rapport technique CRT-587, Centre de Recherche sur les Transports, Université de Montréal*, 1988.

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

[83] HI Stern and M Dror. Routing electric letters readers. *Computers & Operations Research*, 6:209–223, 1979.

[84] M Tagmouti, M Gendreau, and JY Potvin. Arc routing problem with time-dependent service costs. *European Journal of Operational Research*, 181:30–39, 2007.

[85] M Tagmouti, M Gendreau, and J.Y Potvin. A variable neighborhood descent heuristic for arc routing problems with time-dependent service costs. *Soumis à Computers & Industrial Engineering*, 2009.

[86] 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.

[87] J.Y Wang, P Kandula, and J.R Wright. Evaluation of computer- generated routing for improved snow and ice control. *Transportation Research Record*, 1509:15–21, 1995.

[88] Z Win. On the windy postman problem on Eulerien graphs. *Mathematical Programming*, 44:97–112, 1989.

[89] S Wohlk. Contributions for arc routing. *PhD thesis, Faculty of Social Sciences, University of Southern Denmark, Denmark*, 2005.