N° d'ordre : 86/2018-C/MT

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté de Mathématiques


**THÈSE DE DOCTORAT**

**Présentée pour l'obtention du grade de Docteur**
**En : MATHÉMATIQUES**
**Spécialité : Recherche Opérationnelle et Mathématiques Discrètes**

**Par : Hayet CHENTLI**

**Sujet**

# MODÈLES ET MÉTHODES POUR UNE CLASSE DE PROBLÈMES DE RAMASSAGE ET LIVRAISON

Soutenue publiquement le : 20/12/18 à 09h30, devant le jury composé de :

| | | | |
|---|---|---|---|
| M. Meziane AIDER | Professeur | à l'USTHB | Président |
| M. Rachid OUAFI | Professeur | à l'USTHB | Directeur de thèse |
| M. Mohamed AIDENE | Professeur | à l'UMMTO | Examinateur |
| M. Bachir SAADI | Professeur | à l'UMMTO | Examinateur |
| M. Ahmed SEMRI | Professeur | à l'USTHB | Examinateur |

# Remerciements

Je remercie mon directeur de thèse M. le Professeur Rachid OUAFI d'avoir cru en moi depuis le début. Je le remercie aussi pour sa disponibilité et ses conseils. Je tiens à remercier également Mme la Professeure Wahiba RAMDANE CHERIF-KHETTAF pour sa patience, pour ses encouragements et pour ses qualités de relecture.

Je tiens à remercier M. le Professeur Noureddine HANNOUN pour ces précieux conseils concernant la langue anglaise.

Merci également à M. le Professeur Meziane AIDER d'avoir bien voulu présider le jury de cette thèse. Merci à MM. les Professeurs Mohamed AIDENE, Bachir SAADI et Ahmed SEMRI d'avoir accepté d'être examinateurs de ce modeste travail.

Je voudrais aussi exprimer ma profonde gratitude envers toute ma famille et mes amies pour leur soutien incommensurable, ainsi qu'à toute personne ayant contribué de prés ou de loin à l'aboutissement de cette thèse.

# Table des matières

# Liste des figures

# Liste des tables

# Introduction

La thèse s'inscrit dans le domaine de l'optimisation combinatoire. Nous nous intéressons particulièrement à l'application de ce domaine au développement du transport durable. Plus précisément, nous portons notre attention sur les entreprises de production et/ou de transport qui cherchent à assurer la distribution ou la collecte de certains produits aux consommateurs, tout en veillant au respect de l'environnement. Les applications ciblées sont du domaine de la gestion des schémas de transport incluant la distribution de produits finis, et/ou la collecte des déchets engendrés par ces produits à la fin de leurs cycles de vie, pour des fins de recyclage. Il est aussi question de réduire les émissions des gaz à effet de serre engendrés par le déplacement inapproprié des transporteurs.

Afin d'être menée à bien, cette initiative de protection de l'environnement via le transport durable doit rester profitable pour l'entreprise. Cela s'avère très difficile étant donné que les opérations de collectes de déchets peuvent entraîner des coûts supplémentaires. Il est donc primordial de développer des outils d'aide à la décision permettant de tracer un schéma optimal de distribution et de collecte.

Formulé ainsi, le problème étudié peut être identifié comme un problème d'optimisation combinatoire complexe nécessitant des techniques de recherche opérationnelle. Ces dernières permettent de fournir des solutions de bonne qualité réduisant les coûts de transport et contribuant, ainsi, et de façon significative à l'optimisation du coût de revient global pour l'entreprise.

L'objet de cette thèse est de développer des algorithmes d'optimisation pour résoudre une classe de problèmes de transport durable dans laquelle les flux de livraison de produits finis et de collecte de déchets peuvent être combinés. Les problèmes étudiés font partie de la famille des *problèmes de tournées de véhicules* (Vehicle Routing Problems en anglais ou VRP). Dans un premier temps, nous nous intéressons à un problème qui considère soit des opérations de collecte ou des opérations de livraison. Ce problème vise à optimiser l'utilisation des capacités des véhicules afin de réduire le nombre de déplacements. Après cela, nous traitons une extension de ce problème, dans laquelle des opérations de collecte et de distribution sont combinées.

Les résultats attendus consistent en l'élaboration de méthodes permettant l'optimisation de schémas de transport durable. Il sera question de réduire les coûts de revient du transport pour l'entreprise et particulièrement, ceux engendrés par la considération des contraintes environnementales. Ainsi, l'entreprise pourra agir en tant qu'acteur responsable dans la protection de l'environnement.

La thèse regroupe quatre Chapitres répartis sur deux parties. La première partie a pour but de familiariser le lecteur avec les définitions et notions de bases, utiles pour une bonne compréhension de la thèse. La seconde partie vise, quant à elle, à

proposer des méthodes de résolution pour deux cas particuliers du *problème de tournées de véhicules* avec ramassages et/ou livraisons. Les Chapitres sont organisés comme suit :

Le premier Chapitre traite les *problèmes de tournées de véhicules* d'une façon générale. On y retrouve, tout d'abord, des concepts et notions de base de *théorie des graphes* et de *programmation linéaire*. On y explique comment un *problème de tournées de véhicules* peut être modélisé grâce aux graphes et aux programmes linéaires. Un peu plus loin, on y décrit le *problème de voyageur de commerce* et le *problème de tournées de véhicules* ainsi que certaines de leurs variantes.

Le second Chapitre aborde les méthodes de résolution les plus utilisées pour le *problème de tournées de véhicules* et ses variantes. Ces méthodes sont présentées afin de permettre au lecteur de comprendre la revue de la littérature et les méthodes utilisées dans les Chapitres suivants.

Le troisième Chapitre traite une variante du *problème de tournées de véhicules* avec opérations de ramassage ou opérations de livraison, appelée *problème de la tournée la plus profitable avec contrainte de capacité* (Capacitated Profitable Tour Problem en anglais ou CPTP). Une revue de la littérature et plusieurs heuristiques sont proposées pour le CPTP. Les heuristiques proposées sont hybridées de différentes façons et une étude expérimentale est effectuée afin d'évaluer la performance de chaque méthode. La meilleure méthode implémentée est comparée à d'autres méthodes de la littérature du CPTP.

Le quatrième Chapitre présente une nouvelle variante du *problème de tournées de véhicules avec ramassages et livraisons*. Cette variante est appelée *problème de la tournée la plus profitable avec ramassages et livraisons simultanés* (Profitable Tour Problem with Simultaneous Pickup and Delivery services en anglais ou PTPSPD). Le PTPSPD est une extension du CPTP qui prend en charge des opérations de ramassage et de livraison devant être effectuées simultanément. Tout d'abord, une revue de la littérature est présentée pour les problèmes plus ou moins similaires au PTPSPD. Un modèle mathématique est ensuite proposé suivi d'une version modifiée de la *recherche adaptative à grand voisinage* (Adaptive Large Neighborhood Search en anglais ou ALNS). Cette nouvelle version d'ALNS est notée sALNS. Les composantes d'sALNS sont évaluées et comparées avec celles proposées dans la littérature. En outre, les résultats d'sALNS sont comparés avec ceux du solveur CPLEX utilisant le modèle mathématique proposé et avec ceux de l'ALNS classique. Enfin, sALNS est testée sur certaines instances de la littérature du *problème de tournées de véhicules avec ramassages et livraisons simultanés* (Vehicle Routing Problem with Simultaneous Pickup and Delivery services en anglais ou VRPSPD) et du CPTP.

Cette thèse a fait l'objet de quelques papiers publiés :

— H. Chentli, R. Ouafi, and W. Ramdane Cherif-Khettaf. A selective adaptive large neighborhood search heuristic for the profitable tour problem with simultaneous pickup and delivery services. RAIRO-Operations Research. DOI : https ://doi.org/10.1051/ro/2018024.

— H. Chentli, R. Ouafi, and W. Ramdane Cherif-Khettaf. Behaviour of a hybrid ils heuristic on the capacitated profitable tour problem. In Proceedings of

the 7th International Conference on Operations Research and Enterprise Systems - Volume 1 : ICORES, pages 115–123. INSTICC, SciTePress, 2018. DOI : 10.5220/0006630401150123.

— H. Chentli, R. Ouafi, and W. Ramdane Cherif-Khettaf. Impact of Iterated Local Search heuristic hybridization on Vehicle Routing Problems : Application to the Capacitated Profitable Tour Problem. Communications in Computer and Information Science. Springer. Accepted, 2018.

**Note :** Étant donné que la plupart des travaux de la littérature sont en anglais, nous avons choisi de garder les acronymes anglais (par exemple, VRP pour *Vehicle Routing Problem* au lieu de PTV pour *Problème de Tournées de Véhicules*). Nous pensons que cela rendra la compréhension de la thèse plus rapide, notamment pour les lecteurs habituées aux acronymes anglais.

# Première partie

# Le problème de tournées de véhicules et sa résolution dans la littérature

# Problèmes de tournées de véhicules : Description et variantes

1

Sommaire



Nous traiterons dans ce Chapitre les concepts et notions de bases nécessaires pour la modélisation des *problèmes de tournées de véhicules*. Ces concepts et notions sont également utiles pour la compréhension des Chapitres suivants. Nous parlerons aussi du *problème de voyageur de commerce* et celui de *tournées de véhicules* qui constituent l'assise même des problèmes traités dans cette thèse. Quelques variantes du problème de tournées de véhicules sont également présentées.

## Introduction

Selon Irnich et al. [57], la famille des *problèmes de tournées de véhicules* (VRPs) se caractérise par la donnée d'un ensemble de demandes de transport et d'une flotte de véhicules, de sorte que l'on cherche à déterminer, moyennant un coût minimum, un ensemble de routes empruntées par les véhicules disponibles et satisfaisant la totalité (ou une partie) des demandes. On veut donc décider quel véhicule affecter à quelle demande et dans quel ordre de façon à respecter les contraintes imposées sur les routes.

On emploie souvent, dans les *problèmes de tournées de véhicules*, le terme *client* pour désigner l'emplacement d'une demande. On prend également en considération un ou plusieurs emplacement(s) additionnel(s) représentant les points de départ et d'arrivée des véhicules. Ces emplacements sont communément appelés *dépôts*. De même, le terme *tournées* fait allusion aux routes empruntées par les véhicules.

## 1.1 Concepts et notions de base

Afin de donner des descriptions plus formelles aux *problèmes de tournées de véhicules*, les chercheurs utilisent deux outils principaux, à savoir la théorie des graphes et la programmation linéaire.

L'utilisation de la théorie des graphes en optimisation combinatoire permet une représentation simplifiée de situations à priori compliquées. En effet, dans le concept de graphes, on emploie des notions faciles à comprendre, utilisant des relations assez intuitives entre les éléments du problème étudié. On parlera alors de sommets, d'arêtes (ou arcs) et de relations entre les éléments du graphe.

La programmation linéaire est, quant à elle, un outil de modélisation fréquemment utilisé pour formuler le problème étudié sous forme d'un ensemble d'inéquations avec un objectif de minimisation ou de maximisation.

Nous consacrons les sections suivantes aux rappels des définitions de la théorie des graphes et de la programmation linéaire ainsi qu'aux notations dont nous aurons besoin par la suite.

### 1.1.1 Définitions et notations de la théorie des graphes

Les définitions suivantes sont issues du livre de Claude Berge [11].

Un **graphe orienté** est un schéma constitué d'un ensemble (que l'on suppose fini ici) de points $v_1, v_2, \ldots, v_n$ ainsi que d'un ensemble de flèches $1, 2, \ldots, m$ de sorte que chaque flèche relie deux points. Chaque point est dit **sommet** du graphe alors que chaque flèche est appelée **arc** du graphe.

L'ensemble des sommets du graphe $G$ est noté $V$. L'ensemble des arcs de $G$ est noté $U$.

Un arc $i$ où $i \in \{1, 2, \ldots, m\}$ allant d'un sommet $v_1$ à un sommet $v_2$ est dit de la forme $(v_1, v_2)$, et l'on écrit par convention $i = (v_1, v_2)$. Cette convention permet de faire la distinction entre deux arcs différents allant tous deux d'un sommet $v_1$ à un sommet

$v_2$.

Le graphe est donc complètement déterminé par l'ensemble de ses sommets $V$ et par la famille $U$ constituée par les différentes formes de ses arcs.

Formellement, un graphe $G = (V, U)$ est un couple constitué d'un ensemble $V = \{v_1, v_2, \ldots, v_n\}$ de sommets et d'une famille $U = (u_1, u_2, \ldots, u_m)$ d'éléments du produit cartésien $V \times V = \{(v_1, v_2) \backslash\ v_1 \in V, v_2 \in V\}$ ; où un élément $(v_1, v_2)$ peut apparaître plusieurs fois dans la famille $U$.

Tout arc de $G$ de la forme $(v_1, v_1)$ est dit **boucle**. Pour tout arc $i = (v_1, v_2)$, le sommet $v_1$ est appelé **extrémité initiale** et le sommet $v_2$ est appelé **extrémité terminale**.

Un arc ayant pour extrémité initiale le sommet $v_1$ est un arc **sortant** de $v_1$. Par ailleurs, un arc ayant pour extrémité terminale le sommet $v_1$ est un arc **entrant** dans $v_1$.

Un sommet $v_2$ est dit **successeur** d'un sommet $v_1$ s'il existe un arc dont l'extrémité initiale est $v_1$ et l'extrémité terminale est $v_2$. L'ensemble des successeurs de $v_1$ est noté $\Gamma_G^+(v_1)$.

Par ailleurs, un sommet $v_2$ est dit **prédécesseur** d'un sommet $v_1$ s'il existe un arc dont l'extrémité initiale est $v_2$ et l'extrémité terminale est $v_1$. L'ensemble des prédécesseurs de $v_1$ est noté $\Gamma_G^-(v_1)$.

L'ensemble $\Gamma_G(v_1) = \Gamma_G^+(v_1) \cup \Gamma_G^-(v_1)$ est dit ensemble des **voisins** de $v_1$. Dans le cas où $\Gamma_G(v_1) = \varnothing$, $v_1$ est dit sommet **isolé**.

Pour tout $A \subset V$, on pose $\Gamma_G(A) = \bigcup_{a \in A} \Gamma_G(a)$. Si $v_1 \in \Gamma_G(A)$ avec $v_1 \notin A$, $v_1$ est dit **adjacent** à l'ensemble $A$.

Dans certains cas, la direction des flèches d'un graphe $G$ importe peu. Dans ces cas là, on ne considère plus l'arc $u_i = (v_1, v_2)$ mais plutôt l'ensemble formé par les sommets $v_1$ et $v_2$ et noté $e_i = [v_1, v_2]$ ou encore $e_i = \{v_1, v_2\}$ si $v_1 \neq v_2$. $e_i$ est dite $i$-ième **arête** de $G$. Dans ces cas, le graphe $G$ est dit **non orienté**. La famille $(e_1, e_2, \ldots, e_m)$ des $m$ arêtes de $G$ est notée $E$.

Notons que tous les graphes sont orientés, il est seulement parfois peu commode de considérer les graphes avec des lignes orientées si le problème traité est non orienté.

Dans certains cas, on associe des valeurs (en général réelles) aux arcs du graphe appelées **longueurs** ou **poids**. Le graphe devient un triplé $G = (V, U, c)$ constitué de l'ensemble de ses sommets $V$, de la famille $U$ constituée par les différentes formes de ses arcs et d'une fonction $c : U \to \mathbb{R}$ qui associe à chaque élément $u_i \in U$ un poids $c_{u_i} \in \mathbb{R}$. On dit alors que le graphe $G$ est **valué**.

La partie gauche de la Figure 1.1 donne un exemple de graphe valué sans sommets isolés et non orienté à quatre sommets (1, 2, 3 et 4) et six arêtes ($\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}$ et $\{3,4\}$). Alors que dans la partie droite, une version orientée de ce même graphe est présentée (les poids des arcs y sont modifiés).

FIGURE 1.1 – *Illustration de graphes selon l'orientation.*


Un graphe $G$ est dit **simple** s'il est sans boucles et si chaque paire de sommets dans $G$ n'est reliée au plus que par un seul arc.

On appelle **multiplicité** d'une paire de sommets $v_1$ et $v_2$ le nombre d'arcs du graphe $G$ ayant l'extrémité initiale $v_1$ et l'extrémité terminale $v_2$. Ce nombre est noté $m_G^+(v_1, v_2)$. On pose $m_G^-(v_1, v_2) = m_G^+(v_2, v_1)$ et $m_G(v_1, v_2) = m_G^+(v_1, v_2) + m_G^-(v_1, v_2)$.
Lorsque $v_1 \neq v_2$, $m_G(v_1, v_2)$ représentera le nombre d'arcs avec les deux extrémités $v_1$ et $v_2$.

On dit qu'un graphe $G$ est **complet** si pour chaque paire de sommets $v_1$ et $v_2$ de $V$ avec $v_1 \neq v_2$, $m_G(v_1, v_2) = m_G^+(v_1, v_2) + m_G^-(v_1, v_2) \geq 1$.

Soit $G_A$ le graphe dont les sommets sont les éléments de $A \subset V$ et dont les arcs sont tous les arcs de $G$ ayant leurs deux extrémités dans $A$. Alors, $G_A$ est dit **sous-graphe de** $G$ **engendré par** $A \subset V$.

Un graphe partiel de $G$ engendré par $B \subset U$ est un graphe $(V, B)$ dont les sommets sont les éléments de $V$ et les arcs sont les éléments de $B$.

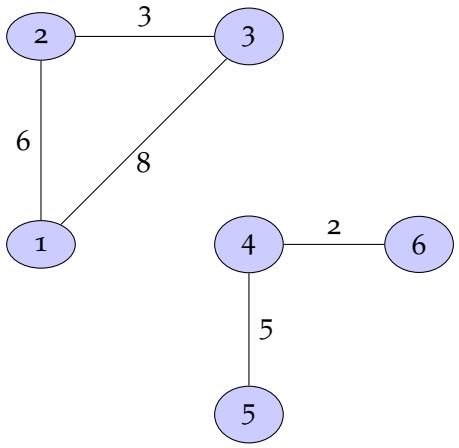
Dans la Figure 1.1, les deux graphes sont simples et complets. Dans chaque partie de la Figure, le graphe formé par les sommets 1, 2 et 4 et les arcs (arêtes) qui les relient est considéré comme un sous-graphe du graphe présenté engendré par le sous-ensemble de sommets $\{1, 2, 4\}$. Dans cette même Figure, à droite, le graphe engendré par les arêtes $\{2, 4\}$, $\{3, 1\}$ et $\{1, 4\}$ et l'ensemble des sommets du graphe initial est un graphe partiel du graphe initial. Dans la partie gauche, le graphe engendré par les arcs $(1, 2)$, $(4, 1)$, $(3, 1)$ et $(2, 4)$ et l'ensemble des sommets du graphe initial est un graphe partiel du graphe initial.

Une séquence $\mu = (u_1, u_2, \ldots, u_q)$ d'arcs de $G$ où chaque arc possède une extrémité en commun avec l'arc qui le précède et une autre extrémité en commun avec l'arc qui le succède est appelée **chaîne de longueur** $q \geq 0$.

Lorsque dans une chaîne $\mu = (u_1, u_2, \ldots, u_q)$, on a, pour chaque arc $u_i$ où $i < q$, l'extrémité terminale de $u_i$ est l'extrémité initiale de $u_{i+1}$, $\mu$ est dite **chemin de longueur** $q > 0$.

On appelle **cycle** toute chaîne $\mu = (u_1, u_2, \ldots, u_q)$ dans laquelle un même arc ne figure pas deux fois dans une séquence, et où les deux sommets aux extrémités de la

FIGURE 1.2 – *Illustration d'un graphe non connexe avec deux composantes connexes.*




chaîne coïncident.

Un cycle $\mu = (u_1, u_2, \ldots, u_q)$ où pour chaque $i < q$, l'extrémité terminale de $u_i$ coïncide avec l'extrémité initiale de $u_{i+1}$ est dit **circuit**.

Un graphe est dit **connexe** si pour toute paire de sommets distincts $v_1$ et $v_2$, il existe une chaîne reliant $v_1$ et $v_2$.

La relation « $v_1 = v_2$, ou $v_1 \neq v_2$ et il existe dans $G$ une chaîne reliant $v_1$ et $v_2$ » est une relation d'équivalence (noté $v_1 \equiv v_2$) vu que :

$$v_1 \equiv v_1 \text{ réflexivité} \tag{1.1}$$

$$v_1 \equiv v_2 \Rightarrow v_2 \equiv v_1 \text{ symétrie} \tag{1.2}$$

$$v_1 \equiv v_2, v_2 \equiv v_3 \Rightarrow v_1 \equiv v_3 \text{ transitivité} \tag{1.3}$$

.

Les classes de cette équivalence forment une partition de l'ensemble $V$ en sous-graphes connexes du graphe $G$. Ces derniers sont appelés **composantes connexes** de $G$.

La Figure 1.2 représente un graphe simple, non complet et non connexe, ayant deux composantes connexes.

Un **arbre** est un graphe connexe sans cycles.

Dans la Figure 1.2, la composante connexe dont les sommets sont 4, 5 et 6 est un arbre, alors que la composante connexe dont les sommets sont 1, 2 et 3 n'est pas un arbre.

Dans un graphe $G$ valué, un **arbre couvrant de poids minimal** est un graphe partiel $G'$ de $G$ tel que $G'$ est un arbre ayant la plus petite somme de poids des arcs.

**Modélisation des *problèmes de tournées de véhicules* à l'aide de graphes**

La modélisation d'un *problème de tournées de véhicules* au moyen d'un graphe implique habituellement la représentation du/des dépôt(s) et des clients (plus précisément leurs emplacements) par des sommets et la représentation des tronçons de routes reliant deux sommets par une arête (ou un arc). Les poids des arêtes (respectivement des arcs) sont quant à eux utilisés pour représenter les distances (ou temps de traversée ou encore coûts de transport) entre les différents emplacements.

Lors de la modélisation d'un *problème de tournées de véhicules* au moyen d'un graphe, l'orientation ou non du graphe utilisé dépend du problème étudié. En effet, si par exemple, la distance notée $c_{ij}$ entre le client $i$ et le client $j$ diffère de celle entre $j$ et $i$ ($c_{ji}$), comme dans le cas de routes à sens unique, on sera amené à utiliser un graphe orienté. Si par contre, la distance entre $i$ et $j$ reste la même quel que soit le point de départ choisi, il sera préférable d'employer un graphe non orienté.

En ce qui concerne les solutions des *problèmes de tournées de véhicules*, elles sont généralement représentées par un certains nombres de circuits (quelle que soit l'orientation du graphe), commençant et se terminant par le dépôt, et faisant référence aux routes empruntées par les véhicules. L'utilisation de circuits (et non de cycles) est due au fait que dans certaines variantes, l'ordre de visite des clients est important. On ne peut donc pas commencer la route par le dernier client et terminer par le premier. C'est le cas par exemple lorsque chaque clients impose un intervalle de temps (fenêtre de temps) où il doit être desservi. Le fait d'inverser le sens d'une route pourrait entraîner alors une solution irréalisable.

Notons que, pour certaines variantes un peu plus complexes du *problème de tournées de véhicules*, on peut aussi trouver des poids sur les sommets (désignant par exemple les demandes ou le temps de service des clients ...).

### 1.1.2 Définitions et notations de la programmation linéaire

Un *Programme Linéaire* (PL) est un problème d'optimisation sous contraintes visant à minimiser (ou à maximiser) une fonction linéaire sous certaines contraintes qui sont, elles aussi, linéaires.

Sans perte de généralités, nous considérerons dans cette Section les problèmes avec objectifs de minimisation. Un PL peut donc s'écrire sous la forme suivante :

$$(P): \quad Z = \min f(x) \tag{1.4}$$

sous contraintes :

$$g_i(x) \geq 0 \quad \forall i = 1, \ldots, m \tag{1.5}$$

$$x \in X \tag{1.6}$$

où l'équation (1.4) est appelée *fonction objectif*, le vecteur $x \in X$ est composé de $n$ éléments $x_1, x_2, \ldots, x_n$ nommés *variables de décision* et (1.6) ainsi que les $m$ inéquations (1.5) représentent les *contraintes* du problème.

Un vecteur donné $x$ avec des valeurs fixées de $x_1, x_2, \ldots, x_n$ constitue une solution *réalisable* pour le PL $(P)$ si et seulement si $x$ vérifie les contraintes (1.5) et (1.6).

Une solution $x$ est évaluée grâce à la fonction objectif.

Une solution *optimale* pour le problème est une solution réalisable offrant la plus petite valeur de la fonction objectif parmi l'ensemble des solutions réalisables. Notons qu'il peut exister plusieurs solutions optimales.

Lorsque toutes les variables de décision sont définies sur l'ensemble des entiers positifs, le PL est alors appelé *Programme Linéaire en Nombres Entiers* (PLNE). De même, lorsqu'une partie des variables de décision sont entières et que le reste des variables sont réelles, on parle de *Programme Linéaire Mixte* (PLM).

**Modélisation des *problèmes de tournées de véhicules* à l'aide de programmes linéaires**

Afin de modéliser un *problème de tournées de véhicules* sous forme de PL, il faudrait définir les variables de décision. Celles-ci peuvent représenter par exemple, l'appartenance ou non d'une paire de clients à une tournée telle que, cette paire représente deux clients qui se succèdent. Avec l'exemple donné, nous pouvons définir des variables de décision $x_{ij}^k$ de la façon suivante :

$$x_{ij}^k = \begin{cases} 1 & \text{si le véhicule } k \text{ visite l'emplacement } i \text{ juste avant l'emplacement } j, \\ 0 & \text{sinon.} \end{cases} \tag{1.7}$$

où les emplacements $i$ et $j$ appartiennent à l'ensemble des emplacements du problème (emplacements des clients et emplacement du dépôt). $k$ prend ses valeurs dans $\{1, \ldots, m\}$, où $m$ représente le nombre de véhicules du problème étudié.

Pour modéliser un *problème de tournées de véhicules* sous forme de PL, il faudrait aussi déterminer la fonction objectif selon les buts du problème étudié et ce, en faisant usage des variables de décision préalablement définies et de certaines données du problème. Par exemple, on pourrait chercher à minimiser les coûts de transport engendrés par les tournées. Nous aurons donc la fonction objectif suivante :

$$Z' \quad = \quad min \quad \sum_{i,j,k} \; c_{ij} \cdot x_{ij}^k \tag{1.8}$$

où $c_{ij}$ représente le coût de transport entre les emplacements $i$ et $j$ et $x_{ij}^k$ sont les variables de décision telles que définies plus haut (voir l'Expression (1.7)).

Enfin, pour modéliser un *problème de tournées de véhicules* sous forme de PL, il faudrait définir les contraintes imposées sous forme d'inéquations linéaires. Ainsi, pour empêcher la capacité des véhicules d'être excédée, on pourrait, par exemple, définir les contraintes suivantes :

$$\sum_i \sum_j \; d_i \cdot x_{ij}^k \;\; \leq \;\; Q \quad \forall k \in \{1, \ldots, m\} \tag{1.9}$$

où $i$ est un client, $j$ peut être l'emplacement d'un client ou bien celui du dépôt, $k$ est l'indice d'un véhicule, $d_i$ est la demande du client $i$ et $Q$ est la capacité du véhicule. $x_{ij}^k$ sont les variables de décision telles que définies plus haut (voir l'Expression (1.7)). Autrement dit, pour un véhicule $k$ fixé, nous calculons la somme des demandes des clients visités par $k$ et vérifions que cette somme est inférieure à la capacité du véhicule. Bien entendu, un client $i'$ visité par un véhicule $k'$ est un client pour lequel il existe un successeur $j'$ qui est également visité par $k'$. Ainsi $x_{i'j'}^{k'} = 1$.

## 1.2 Le problème du voyageur de commerce

Le *problème du voyageur de commerce* (Traveling Salesman Problem en anglais ou TSP) est l'un des problèmes de transport les plus étudiés en optimisation combinatoire. Il a été présenté pour la première fois sous forme de jeu par William Rowan Hamilton en 1859. L'objectif était alors de trouver le meilleur parcours possible en termes de durée, que devait emprunter un voyageur de commerce pour visiter un nombre fini de villes une et une seule fois, en retournant au point de départ à la fin du parcours.

Dès lors, le problème commença à s'étendre et eut de plus en plus d'applications notamment dans le domaine du transport d'individus et de marchandises, dans la planification (choisir l'ordre ou les trajectoires des opérations à effectuer pour terminer une tâche le plus rapidement possible ; comme dans le perçage de points dans des cartes électroniques), etc.

Tout comme la plupart des problèmes d'optimisation combinatoire, le *problème du voyageur de commerce* possède la particularité d'avoir un énoncé simple et facile à comprendre. Il n'en demeure pas moins qu'il soit très difficile à résoudre lorsque le nombre de villes augmente. D'ailleurs, dans l'un de leurs papiers, Garey et Johnson [40] mettent en évidence la NP-difficulté du problème.

Afin d'illustrer la difficulté du *problème du voyageur de commerce*, nous proposons d'abord un exemple simple contenant quatre villes. Pour déterminer la solution optimale du problème, il suffira donc d'énumérer toutes les combinaisons possibles de visite de ces villes et d'en retenir la combinaison de moindre coût. Nous aurons donc : 4 choix de positionnement pour la première ville dans le parcours (soit en 1re, en 2e , en 3e ou en 4e position), 3 choix pour la seconde (étant donné que la position de la première ville soit fixée), 2 choix pour la troisième et enfin, 1 choix pour la quatrième. Ce qui nous donne un total de $4! = 24$ choix de parcours possibles. Lorsque nous passons à un exemple avec dix villes, le nombre de combinaisons possibles atteint $10! = 3628800$. Plus généralement, avec un problème à $n$ villes, nous aurons $n!$ choix de parcours. Le problème devient très difficile à résoudre lorsque $n$ est supérieur à 30.

Notons que l'on peut réduire, pour le *problème du voyageur de commerce* classique, le nombre de combinaisons à évaluer, en ne considérant que les chemins « non équivalents ». Par exemple, si nos quatre villes étaient $a$, $b$, $c$ et $d$, les parcours équivalents obtenus par rotation comme $abcd$, $bcda$, $cdab$ et $dabc$ ainsi que ceux obtenus en inversant les parcours précédents (qui ont aussi la même durée) : $dcba$, $adcb$, $badc$ et $cbad$ seraient

considérés comme identiques. On n'aurait donc pas à inclure de tels chemins dans les combinaisons à évaluer. On obtiendrait alors $\frac{(n-1)!}{2}$ combinaisons possibles. Cependant, la résolution du problème resterait encore hors de portée en termes de temps de calcul. En effet, même avec les avancées technologiques cela prendrait plusieurs milliards d'années à un ordinateur pour déterminer la solution optimale d'un tel problème.

Vu les différents cas pratiques rencontrés en entreprise, les chercheurs se sont vus dans l'obligation d'élargir le champ de recherche du *problème du voyageur de commerce* en ajoutant certaines contraintes réalistes telles :

1. Les contraintes relatives à capacité du véhicule : dans le cas où l'on doit satisfaire une demande donnée (comme une demande de livraison) dans chaque ville et que le véhicule possède une capacité finie.
2. Les contraintes relatives aux fenêtres de temps : chaque ville possède une fenêtre de temps telle que le véhicule doit visiter une ville donnée durant l'intervalle de sa fenêtre de temps.
3. L'utilisation de plusieurs véhicules.
4. ...

## 1.3 Les problèmes de tournées de véhicules

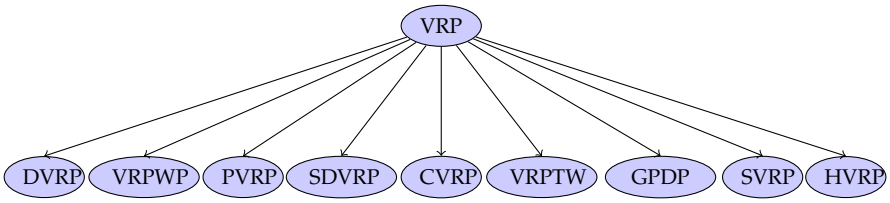
Dans le cas d'utilisation de plusieurs véhicules, on ne parle plus de *problème du voyageur de commerce* mais de *problème de tournées de véhicules* (Vehicle Routing Problem en anglais ou VRP). Notons que pour le VRP, les emplacements de départ et d'arrivée sont considérés comme des *dépôts* (souvent comme un seul et unique dépôt). Les villes intermédiaires sont, quant à elles, considérées comme des *clients*. Chaque véhicule devra donc démarrer du dépôt, visiter un ensemble de clients puis, revenir vers le dépôt de sorte à ce que chaque client soit visité, une et une seule fois, par un unique véhicule. Le parcours de chaque véhicules est alors appelé *route* ou encore *tournée*. Le VRP inclut généralement une ou plusieurs contraintes et le type des données des VRPs peuvent varier d'une version à l'autre.

### 1.3.1 Variantes du problème de tournées de véhicules

De même que pour le *problème du voyageur de commerce*, plusieurs variantes du VRP ont été étudiées pour faire face aux problèmes rencontrés en pratique. Une étude taxonomique complète du sujet peut être trouvée dans le papier d'Eksioglu et al. [32]. Parmi les contraintes considérées dans la littérature du VRP, on peut citer celles relatives au :

1. Nombre de clients par route.
2. À la divisibilité des demandes des clients.
3. Aux types de demandes des clients (livraisons, ramassages, ...).
4. À l'obligation ou non de passer par tous les clients.
5. Aux temps de service et d'attente.
6. Aux contraintes de fenêtres de temps (pour les clients, le dépôt...).
7. Au type de fenêtres de temps (par exemple, certaines fenêtres de temps sont dites souples ou flexibles telles que leur violation devient possible en ajoutant un certain coût à la valeur de la fonction objectif).

FIGURE 1.3 – *Principales variantes du problème de tournées de véhicules.*




8. À la périodicité (le VRP peut être étudié sur plusieurs périodes).
9. Aux contraintes sur la simultanéité des services (certains services doivent être effectués simultanément).
10. À l'orientation du graphe (le graphe représentatif du problème peut être orienté dans le cas d'existence de routes à sens unique et dans ce cas, la distance entre deux points $a$ et $b$ sera différente de celle entre $b$ et $a$).
11. Au nombre de véhicules utilisés (ce nombre peut être fini ou bien supposé infini).
12. À la représentation des clients sur le graphe (les clients peuvent être représentés sous forme de nœuds ou bien d'arcs dans le cas d'existence de nombreux clients sur un même segment de route).
13. Aux contraintes sur les nœuds/arcs (comme les contraintes de précédence qui font que les véhicules soient contraints de visiter un nœud $i$ donné juste après la visite d'un nœud $j$).
14. Aux emplacements géographiques (ville, montagne, etc.) L'utilisation d'un camion avec remorque dans ce cas mènera à une étude particulière. En effet, lorsque le camion dessert un client se trouvant sur une montagne, il doit se diriger vers un parking spécifique pour détacher la remorque. Le chauffeur reviendra chercher la remorque plus tard pour continuer sa tournée. Il faudra donc gérer ces déplacements additionnels.
15. Au nombre de points d'origine (on peut avoir un ou plusieurs dépôts).
16. À la capacité des véhicules. La capacité peut être considérée comme infinie (comme lorsque les demandes des clients représentent des demandes de réparation) ou bien finie.
17. À l'homogénéité des véhicules (ces derniers peuvent avoir différentes capacité, différentes fonctionnalités...).
18. À la nature du temps de traversée (qui peut être stochastique, déterministe ...).
19. Aux dépendances du coût de transport (le coût de transport peut dépendre du temps de traversée, des véhicules utilisés, des opérations à effectuer ...).
20. Aux types de données (statiques, inconnues, stochastiques ...).

La Figure 1.3 présente les variantes du VRP les plus étudiées dans la littérature. Ces variantes sont de gauche à droite : le *VRP dynamique* (Dynamic Vehicle Routing Problem en anglais ou DVRP), le *VRP avec profits* (Vehicle Routing Problem With Profits en anglais ou VRPWP), le *VRP périodique* (Periodic Vehicle Routing Problem en anglais ou PVRP), le *VRP avec demandes divisibles* (Split Delivery Vehicle Routing Problem en anglais ou SDVRP), le *VRP avec contrainte de capacité* (Capacitated Vehicle Routing Problem en anglais ou CVRP), le *VRP avec fenêtres de temps* (Vehicle Routing Problem with Time Windows en anglais ou VRPTW), le *problème général de ramassages et livraisons* (General Pickup and Delivery Problem en anglais ou GPDP), le *VRP stochastique*

(Stochastic Vehicle Routing Problem en anglais ou SVRP) et enfin, le *VRP hétérogène* (Heterogeneous Vehicle Routing Problem en anglais ou HVRP).

Une description détaillée de ces variantes peut être trouvée dans le livre de Toth et Vigo [98]. Plus de détails concernant les GPDPs peuvent être trouvés dans les ouvrages de Parragh et al. [76] et [77].

Dans ce qui suit, nous nous limiterons à la description des variantes dont nous aurons besoin dans les Chapitres suivants ; à savoir, le CVRP, le GPDP et le VRPWP, ainsi qu'à quelques variantes du VRP qui prennent en compte la protection de l'environnement.

**Le problème de tournées de véhicules avec contraintes de capacité** Le *VRP avec contraintes de capacité* (Capacitated Vehicle Routing Problem en anglais ou CVRP) est l'un des VRPs les plus étudiés dans la littérature et sur lequel se basent la plupart des variantes de VRP. De nombreux travaux ont été proposés dans le but de trouver les meilleures solutions pour cette variante en un temps record. Le CVRP peut inclure soit des demandes de ramassage ou bien des demandes de livraison mais pas les deux.

Comme son nom l'indique, le CVRP est une extension du VRP classique où chaque véhicule possède une capacité limitée. Cette capacité est supposée être identique pour tous les véhicules disponibles.

Le CVRP a pour objectif de minimiser les coûts de transport (ou distances) engendrés lors de la visite des clients.

**Le problème général de ramassages et livraisons** Le *problème général de ramassages et livraisons* (General Pickup and Delivery Problem en anglais ou GPDP) est aussi une variante très étudiée. Sa particularité est que les clients possèdent des demandes de livraison et de ramassage.

On peut distinguer deux catégories de GPDP. Dans la première catégorie, appelée *problème de tournées de véhicules avec retours* (Vehicle Routing Problem with Backhauls en anglais ou VRPB), les biens à livrer doivent être chargés depuis un ou plusieurs dépôts, et les biens collectés doivent être transportés à un ou plusieurs dépôts. Dans la seconde catégorie, appelée *problèmes de tournées de véhicules avec ramassages et livraisons* (Vehicle Routing Problem with Pickup and Delivery en anglais ou VRPPD), les biens sont transportés entre les différents clients (un véhicule peut collecter un bien depuis un client donné et le livrer à un autre client). Notons que dans les deux catégories, il peut s'agir de transport de passagers au lieu de transport de biens.

Dans la thèse, nous nous intéressons particulièrement à la catégorie VRPB. Cette dernière peut à son tour être divisée en quatre classes selon le type de demandes des clients et les contraintes imposées lors des visites. Ainsi, nous avons :

— Le *VRP avec ramassages et livraisons regroupés* (Vehicle Routing Problem with Clustered Backhauls en anglais ou VRPCB) dans lequel chaque client possède soit une demande de ramassage ou bien une demande de livraison de sorte que,

les clients ayant des demandes de livraison doivent être visités en premier.

— Le *VRP avec ramassages et livraisons mixtes* (Vehicle Routing Problem with Mixed linehauls and Backhauls en anglais ou VRPMB), où l'ordre des visites n'est pas fixé.

— Le *VRP avec ramassages et livraisons divisibles* (Vehicle Routing Problem with Divisible Delivery and Pickup en anglais ou VRPDDP), où un client peut avoir une demande de livraison et une autre demande de ramassage pouvant être satisfaites à différents moments : une visite pour la livraison et une autre pour le ramassage.

— Le *VRP avec ramassages et livraisons simultanés* (Vehicle Routing Problem with Simultaneous Pickup and Delivery services en anglais ou VRPSPD), où les clients possèdent les deux types de demandes et doivent être visités une et une seule fois pour satisfaire les deux demandes simultanément.

Pour plus de détails concernant les GPDPs et leur classification, le lecteur est prié de se référer aux ouvrages de Parragh et al. [76] et [77].

**Le problème de tournées de véhicules avec profits** Relativement aux autres variantes du VRP, les *problèmes de tournées de véhicules avec profits* (Vehicle Routing Problem With Profits en anglais ou VRPWP) ont été peu étudiés dans la littérature.

En 2014, Archetti et al. [3] ont passé en revue différents travaux ayant traité les VRPWPs. Nous pouvons distinguer trois grandes catégories de VRPWPs conformément aux objectifs étudiés. Dans un premier temps, les chercheurs ont abordé le sujet avec seulement un véhicule et par la suite, leurs études ont été étendues à plusieurs véhicules en ajoutant parfois d'autres contraintes.

Dans la première catégorie du VRPWP nommée *problème de course d'orientation* (Orienteering Problems en anglais ou OP), l'objectif est la maximisation du profit collecté sous contrainte de distance maximum. Cela signifie qu'en effectuant sa tournée, le véhicule ne peut dépasser une certaine distance fixée au préalable.

Dans la seconde catégorie du VRPWP appelée *problème de la tournée la plus profitable* (Profitable Tour Problem en anglais ou PTP), le but est de maximiser la différence entre la somme des profits collectés et la distance totale parcourue (ou coûts de transport) dans la solution.

Finalement, la troisième catégorie du VRPWP vise à minimiser la durée de la tournée sous contrainte de collecte d'un profit total minimum. Ce type de problèmes est appelé *problème de voyageur de commerce avec collecte de récompenses* (Prize Collecting Traveling Salesman Problem en anglais ou PCTSP).

Parmi les VRPWPs décrits plus haut, les OPs sont ceux qui ont été les plus étudiés. On peut citer par exemple l'*OP avec fenêtres de temps* (OP with time windows en anglais ou OPTW), le *problème de tournées de bus* (One Period Bus Touring Problem en anglais) où un profit est aussi associé aux arcs (les clients représentent les sommets et les trajets

entre les clients sont les arcs) et la maximisation des profits des arcs est incluse dans la fonction objectif, l'OP avec un ensemble de clients obligatoires, l'OP avec profits stochastiques et enfin le *problème de planification d'un séjour de touriste* (Tourist Trip Design Problem en anglais) qui consiste à créer un planning réalisable pour un touriste voulant visiter le plus d'attractions possibles en une période de longueur fixée.

Dans la littérature, le PCTSP a été traité en utilisant des contraintes de capacités ainsi que des pénalités pour les clients non visités (voir l'article [5] à titre d'exemple).

Une variante bi-objectif du VRPWP a également été étudiée dans la littérature (voir le travail de Bérubé et al. [12]). Cette variante consiste en la visite d'un sous-ensemble de clients avec les objectifs de maximisation des profits collectés et de minimisation des coûts de transport.

En utilisant plusieurs véhicules, les chercheurs ont transformé l'OP en *problème de tournées sélectives* (Team Orienteering Problem en anglais ou TOP ou encore Multiple Tour Maximum Collection Problem). En incluant des contraintes relatives aux capacités des véhicules, le problème devient alors le *problème de tournées sélectives avec contraintes de capacité* (Capacitated TOP en anglais ou CTOP). De même, avec plusieurs véhicules et une capacité fixée, le PTP devient *PTP avec contraintes de capacité* (Capacitated Profitable Tour Problem en anglais ou CPTP).

Plusieurs modifications ont été apportées au TOP. Il existe une version qui autorise les services partiels. Une autre autorise les demandes divisibles à condition que chaque client visité ait la totalité de sa demande dans la solution finale s'il est pris et rien sinon. Cette dernière condition a été enlevée dans une autre version. Le TOP a aussi été étudié avec des contraintes de fenêtres de temps dans une variante nommée TOPTW. Enfin, il existe une autre version incluant des contraintes de capacités, de durée maximum ainsi que de fenêtres de temps. Cette dernière version est nommée *problème de tournées sélectives avec fenêtres de temps* (Selective Vehicle Routing Problem with Time Windows en anglais ou SVRPTW).

Le lecteur intéressé par les VRPWPs est prié de consulter l'ouvrage d'Archetti et al. [3].

**Problèmes de tournées de véhicules et protection de l'environnement** Dans certaines variantes du VRP, une importance particulière est accordée aux problèmes environnementaux. Dans ces variantes là, on peut trouver une ou plusieurs contraintes caractérisant les variantes classiques du VRP citée plus haut (CVRP, VRPB ...).

Il existe trois grandes catégories de VRPs dans lesquelles les contraintes environnementales sont prises en compte. Ces catégories diffèrent les unes des autres de par les stratégies de protection de l'environnement utilisées.

Dans la première catégorie de VRPs avec contraintes environnementales, appelée *problème de tournées de véhicules vert* (Green Vehicle Routing Problem en anglais ou GVRP), il est question d'optimiser la consommation d'énergie durant les opérations de transport. Selon le rapport du Département de l'Énergie des États-Unis (United States Department of Energy en anglais ou DoE) [74], les facteurs principaux affectant

la consommation de carburants lors d'opérations de transport sont :

i) la vitesse à laquelle le véhicule se déplace,

ii) le poids du chargement du véhicule,

iii) la distance parcourue par le véhicule lors de l'opération de transport.

Certains auteurs de la littérature se sont intéressés au GVRP. Xiao et al. [103], par exemple, ont étudié une variante de VRPs avec un objectif de minimisation de la consommation de carburants. Afin de tenter de matérialiser le taux de consommation de carburants par les véhicules, Xiao et al. prennent en compte les distances traversées et le chargement des véhicules.

Kuo [62] utilise, quant à lui, les trois facteurs définis par le DoE cités plus haut.

D'autres auteurs tels qu'ErdoÄŸan et Miller-Hooks [34] ont considéré l'utilisation de véhicules avec des sources d'énergie autres que les carburants dérivés du pétrole. Les tournées étudiées prennent en considération le rechargement/réalimentation des véhicules utilisés lorsque cela s'avère nécessaire.

La seconde catégorie de VRPs avec des contraintes environnementales est appelée *problème de tournées de véhicules avec contraintes de pollution* (Pollution Routing Problem en anglais ou PRP). Dans cette catégorie, il est question de réduire le $CO_2$ émis par les véhicules lors des tournées.

Certains auteurs, tels que Pronello et André [79], ont pris en compte la durée de transport lorsque les véhicules sont encore froids. D'autres auteurs tels que Sbihi et Eglese [85] ont cherché à construire des tournées évitant les points de congestion même si cela allongeait les distances parcourues par les véhicules. L'idée est que moins de pollution est produite lorsque les véhicules circulent à la « bonne » vitesse.

Bauer et al. [6] se sont intéressés à la minimisation des émissions de gaz à effet de serre dans le cadre d'un transport intermodal de marchandises. Ils se sont également intéressés à l'impact de ce type de transport dans la réduction de gaz à effet de serre.

Certains auteurs tels que Ubeda et al. [99] ont combiné les réduction des coûts de transport et des coûts environnementaux dans une même fonction objectif. Ces auteurs sont arrivés à la conclusion que la logistique inverse semblait plus efficace dans le contrôle des émissions de gaz à effet de serre.

La troisième catégorie de VRPs avec contraintes environnementales est appelée *VRP avec logistique inverse* (Vehicle Routing Problem with Reverse Logistics en anglais ou VRPRL). La stratégie adoptée dans cette catégorie est de protéger l'environnement en incluant des opérations de ramassages dans les tournées des véhicules.

Les opérations de ramassages peuvent être effectuées indépendamment comme dans le cas du *problème de collecte de déchet* (Waste Collection en anglais) (voir le travail de Beltrami et Bodin [8] par exemple) et le *problème de collecte de produits à la fin de leurs*

*cycles de vie* (End-of-life Goods Collection en anglais) (voir le travail de Schultmann et al. [86] par exemple).

Les ramassages et les livraisons peuvent également être effectués au sein d'une même tournée, comme c'est le cas pour certaines variantes du *problème de tournées de véhicules avec profits* (VRPWP). On peut citer, à titre d'exemple, le *problème de tournées de véhicules avec ramassages sélectifs et livraisons* (Single Vehicle Routing Problem with Deliveries and Selective Pickups en anglais ou SVRPDSP) [46] où les livraisons sont obligatoires et les ramassages sont optionnels. C'est également le cas pour le VRPB et le VRPPD.

## Conclusion

Dans ce premier Chapitre, nous avons rappelé les définitions de base de théorie des graphes et de programmation linéaire. Ces définitions seront utiles lors de la modélisation mathématiques des problèmes étudiés aux Chapitres 3 et 4. Nous utiliserons également ces définitions pour faciliter la compréhension de certaines méthodes décrites dans les Chapitres suivants.

Nous avons également décrit le *problème du voyageur de commerce*, le *problème de tournées de véhicules* ainsi que les variantes du *problème de tournées de véhicules* les plus étudiées dans la littérature. L'accent a été mis sur les variantes dont nous aurons besoin dans les Chapitres suivants.

# Méthodes de résolution pour les problèmes de tournées de véhicules

2

Sommaire



Après avoir donné les concepts de base et défini le *problème de tournées de véhicules* ainsi que ses variantes les plus connues dans le Chapitre 1, nous passons maintenant à la présentation de quelques méthodes de résolution proposées dans la littérature. Ces méthodes tentent de venir à bout des problèmes d'optimisation combinatoire en général et des problèmes de tournées de véhicules en particulier.

Dans le présent Chapitre, nous commencerons donc par présenter les deux principales familles de méthodes de résolution. Puis, nous soulignerons leurs différences. Nous décrirons, par la suite, les méthodes dont nous aurons besoin dans les Chapitres suivants.

## Introduction

La difficulté de résolution des problèmes d'optimisation combinatoire constitue, jusqu'à présent, un challenge pour les chercheurs du domaine. En effet, pour la plupart de ces problèmes, et notamment pour le VRP, on a, à priori, cette impression de s'attaquer à des sujets faciles du fait de la simplicité de leurs énoncés. Cependant, la réalité est tout autre. Comme montré dans la Section 1.2, plus l'instance étudiée est grande et plus le temps de recherche de la solution optimale par des méthodes intuitives (telles l'énumération exhaustive des solutions réalisables du problème) accroît. Même les ordinateurs les plus performants sont incapables de fournir un résultat en un temps raisonnable.

La difficulté des problèmes d'optimisation combinatoire a poussé les chercheurs du domaine à concevoir d'autres méthodes de résolution plus rapides que la simple énumération. Parmi ces méthodes nous distinguons deux grandes catégories, à savoir les méthodes exactes et les méthodes approchées (ou heuristiques).

Les méthodes exactes utilisent, en général, un principe d'énumération implicite évitant ainsi de parcourir toutes les solutions du problème. Cela mène à une convergence beaucoup plus rapide de l'algorithme. Néanmoins, pour certains problèmes très difficiles, et en général pour les problèmes de grande taille, le temps de calcul peut être très grand. Les méthodes exactes doivent alors être arrêtées prématurément, ne fournissant ainsi, dans le meilleur des cas, qu'une borne supérieure (dans le cas de minimisation) ou inférieure (dans le cas de maximisation) au problème étudié.

Les heuristiques ont, quant à elles, une tout autre approche de recherche qui consiste en une multitude de petites modifications opérant sur une solution donnée. Cet ensemble de modifications permet d'obtenir rapidement des solutions de bonne qualité mais pas forcément optimales. Le point fort des méthodes approchées est qu'elles assurent une convergence rapide vers des solutions de bonne qualité, même pour des problèmes de grande taille. Par contre, il n'y a aucune garantie relative à l'optimalité de la solution.

Les méthodes exactes et approchées sont complémentaires. En effet, les bornes fournies par les méthodes exactes peuvent servir à l'évaluation des méthodes approchées. D'autre part, les méthodes approchées peuvent être utilisées au sein des méthodes exactes pour accélérer la recherche.

Dans les sections suivantes, nous décrirons d'abord les méthodes exactes les plus utilisées pour les VRPs. Puis, nous passerons à la présentation des différents types de méthodes approchées. Nous enchaînerons, ensuite, avec quelques exemples d'heuristiques connues.

## 2.1 Méthodes exactes

Pour résoudre les problèmes où l'énumération exhaustive devient inadéquate, les chercheurs se tournent vers des algorithmes plus élaborés qui permettent de diminuer le temps de calcul. Parmi ces algorithmes, nous citons ceux utilisant le principe de séparation et évaluation, les algorithmes de génération de colonnes et la programmation

dynamique. Ces trois familles d'algorithmes constituent les principaux outils de résolution exacte des problèmes de tournées de véhicules.

### 2.1.1 Séparation et évaluation

Connues sous le nom de *Branch and Bound (B&B)* et introduites par Land et Doig [63], ces méthodes utilisent une structure en forme d'arbre appelée **arbre de recherche**. Un arbre de recherche a pour but de structurer le parcours des solutions grâce à la division (ou séparation) du problème principal en un ensemble de sous-problèmes. La résolution de ces sous-problèmes donnera, par la suite, les solutions du problème principal. La racine de l'arbre de recherche constitue le point de départ de la recherche. Les nœuds de l'arbre représentent les sous-problèmes générés par l'attribution de valeurs à certaines variables. Enfin, les feuilles de l'arbre sont les solutions du problème. Dans chaque solution, une valeur est affectée à chaque variable. Pour passer d'un sous-problème à un autre, il suffit de changer de branche dans l'arbre de recherche. De même, dans un même sous-problème, on passe d'une solution à une autre en parcourant les feuilles de la branche étudiée.

La seconde caractéristique des méthodes B&B est l'évaluation de la valeur de la fonction objectif, qui évite de parcourir la totalité des solutions existantes. Ceci est effectué en esquivant, au sein de l'arbre de recherche, l'ajout des nœuds ne pouvant mener à une solution optimale (de part leurs qualités ou leurs réalisabilités).

Nous recommandons les ouvrages de Desaulniers et al. [28] et de Vanderbeck et Wolsey [100] pour plus de détails concernant la méthode de *génération de colonnes*.

### 2.1.2 Programmation dynamique

La *programmation dynamique* est une méthode exacte proposée par Bellman [7]. Cette méthode divise le problème étudié en un ensemble de sous-problèmes que l'on traite séquentiellement. Les sous-problèmes sont traités du plus simple au plus complexe, l'idée étant qu'un sous-problème complexe soit composé de sous-problèmes simples.

À la fin de la résolution de chaque sous-problème, l'ensemble des **états** (configurations des solutions obtenues) possibles est considéré. L'objectif est de décider pour chaque sous-problème quels états conserver afin que ceux-ci mènent à une solution optimale (selon certains critères) à la fin de la procédure de recherche. Ainsi, les états de « mauvaise qualité », qui ne peuvent influer positivement sur la qualité finale de la solution, ne seront pas pris en compte. Plus précisément, on appelle **politique** un ensemble de décisions séquentielles (conservation ou non de certains états) et on cherche la politique optimale (qui domine toutes les autres ou qui est au pire équivalente selon les critères étudiés) allant de l'état initial (début de la recherche) à l'état final (fin de la recherche).

Pour davantage de détails concernant cette méthode, nous orientons le lecteur intéressé vers l'ouvrage d'Escoffier et Spanjaard [35].

## 2.2 Méthodes approchées

Comme nous l'avons souligné plus haut, les méthodes de résolution des problèmes d'optimisation combinatoire se divisent en deux grandes catégories : les méthodes approchées et les méthodes exactes. Nous nous intéressons ici aux différentes méthodes approchées adaptées aux VRPs mais avant cela, nous examinons les différents types et les différentes caractéristiques de ces méthodes.

Les méthodes approchées peuvent à leur tour être divisées en trois types : les heuristiques de construction, les opérateurs de voisinage et les métaheuristiques.

### 2.2.1 Heuristiques de construction

Les heuristiques de construction ont pour but de fournir une ou plusieurs solutions constituant le point de départ des autres types de méthodes approchées.

Les heuristiques de construction partagent toutes plus ou moins le même principe. Ce dernier consiste en la sélection séquentielle de sommets (clients) ou d'arêtes et leur insertion dans une solution partielle jusqu'à l'obtention d'une solution réalisable. Cette sélection est basée sur certains critères (comme la minimisation des coûts de transport par exemple) et généralement sur la satisfaction des contraintes du problème.

Il existe deux types de méthodes de construction : *i)* les *méthodes séquentielles* où l'on construit une route à la fois et *ii)* les *méthodes parallèles* où les routes sont construites en même temps.

Les heuristiques de construction ont l'avantage de donner rapidement une solution au problème étudié.

Une description des heuristique de construction les plus utilisées peut être trouvée dans l'article de Bräysy et Gendreau [15].

Parmi les heuristiques de construction, nous pouvons citer :

#### L'heuristique du plus proche voisin (Nearest-Neighbor en anglais)

Dans cette heuristique, on commence la construction d'une route en rattachant le dépôt à son plus proche voisin et à chaque itération, on tente de rattacher le dernier client de la route avec son plus proche voisin non encore visité. Si à une étape donnée on n'arrive plus à insérer de clients sans violer des contraintes du problème étudié alors, une nouvelle route est créée et le processus est répété jusqu'à ce que tous les clients soient desservis.

L'heuristique du *plus proche voisin* peut sembler mener à une solution optimale. Cependant, ce n'est généralement pas le cas. Cela est dû au fait que l'heuristique laisse les arêtes (ou arcs) avec des coûts de transport (ou distances) relativement grand(e)s à la fin. Ce qui fait qu'en général, les derniers clients de la tournée soient très éloignés les uns des autres et du dépôt.

La Figure 2.1 montre que l'heuristique du *plus proche voisin* ne mène pas automatiquement à une solution optimale.

En haut de la Figure 2.1 nous proposons un exemple d'une instance de CVRP classique avec 5 clients : 1, 2, 3, 4 et 5 ainsi qu'un dépôt 0. Afin de ne pas trop encombrer la Figure 2.1, seules les arêtes pertinentes sont représentées. Nous supposons que la capacité d'un seul véhicule permet de contenir les demandes de tous les clients.

Au milieu de la Figure 2.1, la solution obtenue par l'heuristique du *plus proche voisin* est donnée. La valeur de la fonction objectif ($Z$) de cette solution est calculée à partir de la somme des arêtes utilisées. $Z$ est égale à 22.

En bas de la Figure 2.1, une autre solution du problème est donnée. Nous remarquons que cette solution possède une meilleure valeur de la fonction objectif ($Z' = 19$) que la solution obtenue par l'heuristique du *plus proche voisin*.

#### L'heuristique de Clarke et Wright

Cette heuristique a été proposée par Clarke et Wright [22]. Initialement, chaque client est dans une route à part. Puis, on calcule pour chaque paire de routes $r_i$ et $r_j$ contenant respectivement les clients $i$ et $j$, l'**économie** $S_{ij}$ de combinaison de ses deux routes en reliant les clients $i$ et $j$. L'économie est calculée suivant l'Expression $S_{ij} = c_{i0} + c_{0j} - c_{ij}$ où $c_{ij}$ est le poids (coût) de l'arête $\{i, j\}$ et 0 désigne le dépôt.

Le principe de l'heuristique de Clarke et Wright est de combiner à chaque fois des routes $r_i$ et $r_j$ en reliant leurs clients respectifs $i$ et $j$ de sorte à maximiser $S_{ij}$ et engendrer une solution réalisable. Ce processus s'arrête lorsqu'il n'y a plus d'améliorations possibles.

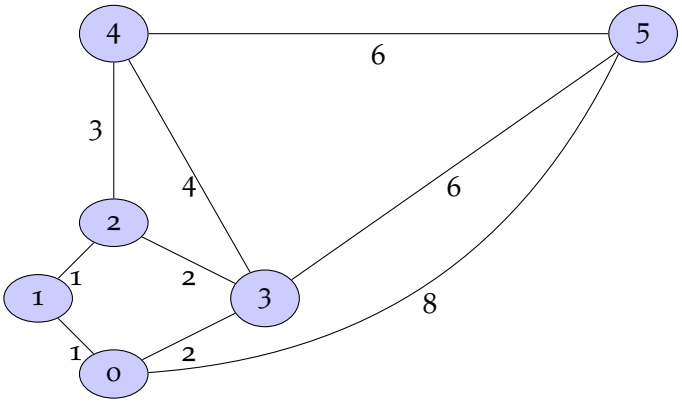
Dans son papier, Paessens [75] propose une variante de cette méthode incluant des paramètres $f$ et $g$ dans la fonction d'économie $S_{ij}$. La nouvelle fonction d'économie devient $S'_{ij} = c_{i0} + c_{0j} - g \cdot c_{ij} + f \cdot |c_{0i} + c_{0j}|$, telle que $f$ et $g$ sont tirés aléatoirement des intervalles $[0, 1]$ et $[0, 3]$ respectivement.

#### L'heuristique de l'insertion la moins coûteuse (Cheapest Insertion en anglais)

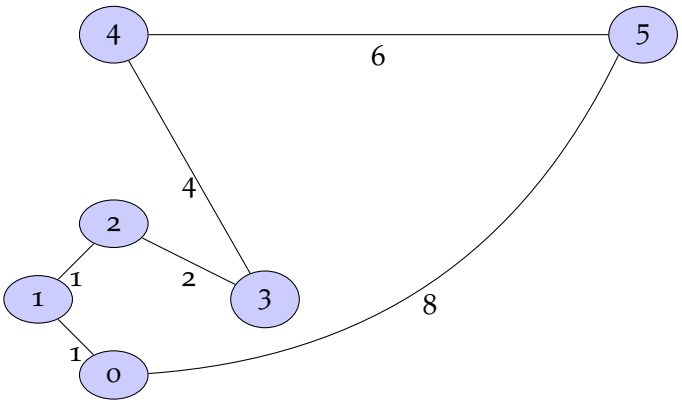
Dans cette heuristique, un client dit « **client graine** » (seed customer en anglais) est relié au dépôt pour construire une route partielle. Puis, on tente d'insérer les clients non encore visités dans la route selon certains critères.

Pour choisir les clients à insérer, on cherche d'abord, pour chaque client, la meilleure position d'insertion au sein de la route selon un certain critère et selon les contraintes du problème (un client est fixé et les positions d'insertion sont comparées). Après cela, on sélectionne le client dont l'insertion au sein de sa meilleure position optimise le critère étudié (les clients sont comparés entre eux). Le client sélectionné est ensuite inséré dans la route à la position adéquate.

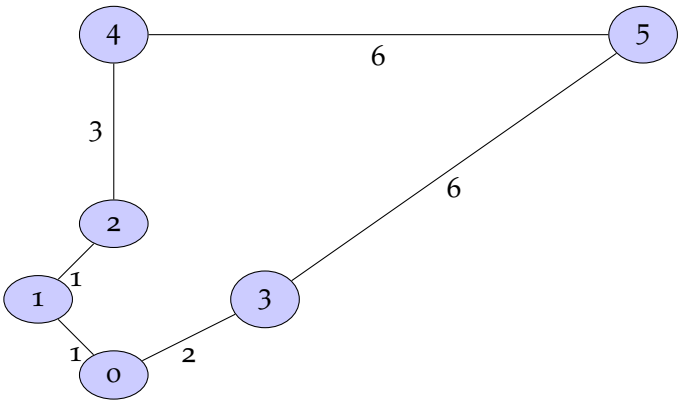
FIGURE 2.1 – *Illustration de l'utilisation de l'heuristique du plus proche voisin.*




Instance étudiée




Solution de l'heurisique du plus proche voisin ($Z = 22$)




Une meilleure solution ($Z' = 19$)

La procédure est répétée jusqu'à ce qu'il n'y ait plus d'insertions réalisables. On dit alors que la route est saturée.

Lorsque la route étudiée est saturée, une nouvelle route est créée de la même façon que la précédente. Ce processus est réitéré jusqu'à ce qu'il n'y ait plus de clients non visités.

Cette méthode peut aussi être implémentée de façon parallèle, en considérant les insertions sur toutes les routes disponibles au lieu d'une seule route à la fois.

L'heuristique de l'*insertion la moins coûteuse* a été modifiée par Solomon [90] pour fournir une meilleure solution au VRPTW. L'auteur a ajouté d'autres critères de sélection des clients basés sur les fenêtres de temps. Solomon a également ajouté quelques paramètres permettant de privilégier certains critères par rapport à d'autres.

#### L'heuristique aléatoire

Cette méthode permet d'insérer les clients dans les routes de façon aléatoire, en tenant compte des contraintes du problème traité.

#### L'heuristique gloutonne

L'heuristique *gloutonne* sélectionne les clients dans un ordre arbitraire et les insert dans les positions de coûts minima (dans le cas des VRPs dont l'objectif est la minimisation des temps de traversée). Voir par exemple l'heuristique $H_{insert}$ de Hifi et Wu [51].

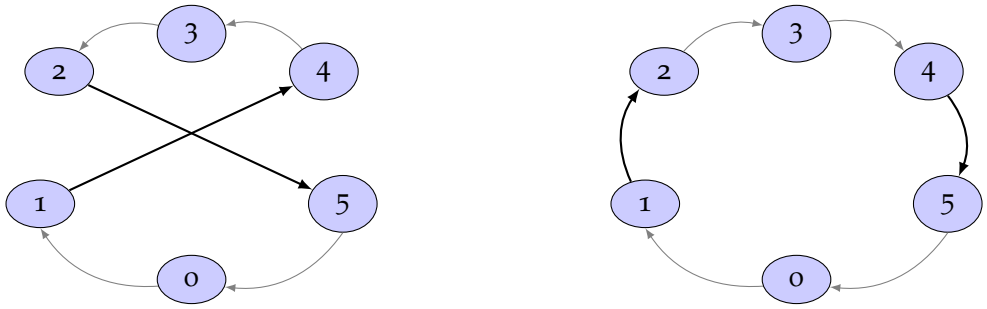
### 2.2.2 Opérateurs de voisinage

Les opérateurs de voisinage (ou heuristiques d'amélioration) sont des procédures qui tentent d'améliorer la qualité d'une solution donnée grâce à un ensemble de modifications élémentaires (appelées également **mouvements** élémentaires). Après chaque mouvement, la nouvelle solution aura encore beaucoup de similitudes avec la solution initiale. On dira alors que ces deux solutions sont **voisines** ou encore, que la nouvelle solution fait partie du **voisinage** de la première.

En balayant l'ensemble d'un voisinage, les heuristiques d'amélioration simulent une modification de la solution initiale, évaluent la nouvelle solution obtenue avec cette modification, retournent à la solution de départ et poursuivent ce processus jusqu'à ce que toutes les modifications possibles soient évaluées. La meilleure solution rencontrée, selon les *critères d'acceptation* des solutions est alors sauvegardée. En général, les critères d'acceptation sont basés sur l'amélioration de la valeur de la fonction objectif. Une fois le balayage terminé, si la solution retenue est meilleure que la solution actuelle alors, la procédure est répétée en considérant la solution sauvegardée comme solution initiale. Sinon, la procédure se termine.

L'avantage de ce type de méthodes est qu'elles donnent des solutions de bonne qualité en un temps record. Par contre, elles sont connues pour être rapidement bloquées dans des *optima locaux*. Notons qu'un **optimum local** représente la meilleure solution que l'on puisse trouver en utilisant un opérateur donné. Cette dernière est en général

FIGURE 2.2 – *Illustration d'un mouvement de l'opérateur 2-Opt.*




différente de la solution optimale du problème étudié qui est dite **optimum global**.

Dans les paragraphes suivants, nous donnons quelques exemples d'opérateurs de voisinage. Ceux-ci constituent l'ensemble des opérateurs que nous avons étudié tout au long de notre parcours de recherche. Quelques exemples supplémentaires pourront être trouvés dans l'article de Bräysy et Gendreau [15].

#### 2-Opt

Cet opérateur remplace deux arcs non-adjacents appartenant à une route donnée par deux arcs qui n'appartiennent pas à cette dernière, de façon à ce que la connectivité de la route soit rétablie. Le processus est répété jusqu'à ce qu'il n'y ait plus d'améliorations possibles. La Figure 2.2 représente un exemple de mouvement de l'opérateur *2-Opt*. À gauche, nous avons une route composée de cinq clients : $1, \ldots, 5$ et d'un dépôt 0. À droite, nous illustrons une nouvelle solution obtenue après l'application du mouvement. Ce dernier supprime les arcs $(1, 4)$ et $(2, 5)$ et les remplace par $(1, 2)$ et $(4, 5)$ en inversant le sens des arcs intermédiaires.

#### 2-Opt*

*2-Opt** consiste en la division d'une paire de routes en quatre sections en supprimant un arc de chaque route. La procédure insert ensuite deux nouveaux arcs qui connectent la section initiale de la première route avec la section terminale de la seconde et vice versa. La Figure 2.3 donne un exemple d'un mouvement de l'opérateur.

Dans cet exemple, nous avons, à gauche, deux routes contenant chacune trois clients. Les arcs de la première route sont de couleur grise et ceux de la seconde route sont en noir. Le dépôt est représenté par le 0. La première route passe par les clients 1, 2 et 3 alors que la seconde passe par 4, 5 et 6. On cherche ici à remplacer le segment $2, 3, 0$ de la première route par le segment $6, 0$ de la seconde et inversement. La nouvelle solution est donnée dans la partie droite de la Figure 2.3, où la première route devient $0, 1, 6, 0$ et la seconde $0, 4, 5, 2, 3, 0$.

#### relocate-inter

C'est une procédure inter-routes qui supprime un client de l'une des routes de la solution et l'insert dans une autre. Comme on peut le voir dans la Figure 2.4, nous avons à gauche, deux routes : $0, 1, 2, 3, 0$ et $0, 4, 5, 6, 0$. Le client 3 de la première route

FIGURE 2.3 – *Illustration d'un mouvement de l'opérateur 2-Opt*.*

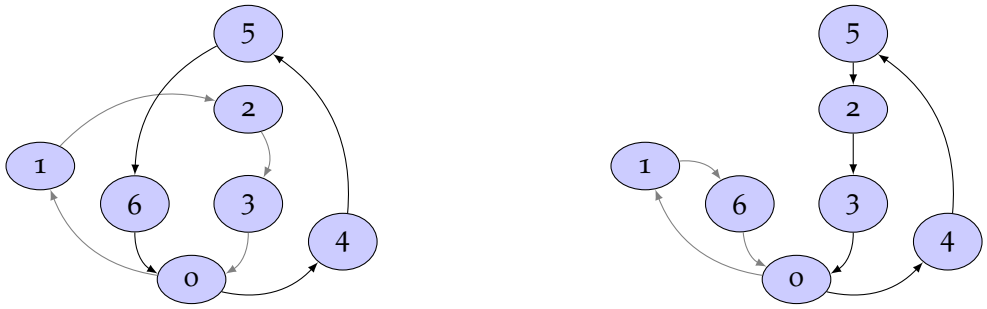
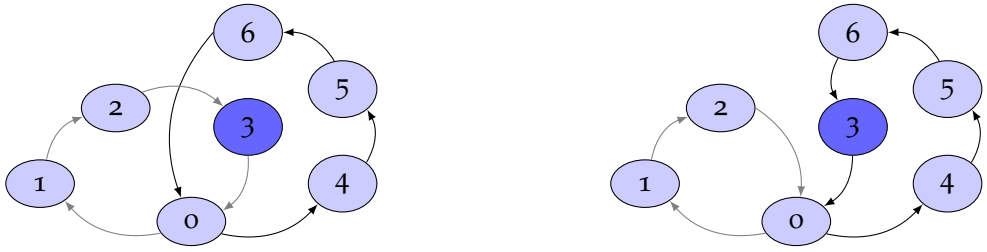



FIGURE 2.4 – *Illustration d'un mouvement de l'opérateur relocate-inter.*




est déplacé dans la seconde route (voir partie droite) et l'on obtient les deux nouvelles routes $0, 1, 2, 0$ et $0, 4, 5, 6, 3, 0$.

**relocate-intra**

Cette procédure supprime un client de l'une des routes de la solution et l'insert à une autre position dans la même route. Un exemple illustratif peut être trouvé dans la Figure 2.5 où le client 2, situé entre le client 5 et le dépôt (partie gauche), est déplacé (partie droite) entre les clients 1 et 3.

**swap-inter**

Cet opérateur échange les emplacements de deux clients dans deux routes différentes. Dans la Figure 2.6, à gauche, le client 3 de la première route est sélectionné pour être échangé avec le client 6 de la seconde route (les arcs de même couleur font partie de la même route). La solution obtenue est illustrée dans la partie droite.

FIGURE 2.5 – *Illustration d'un mouvement de l'opérateur relocate-intra.*

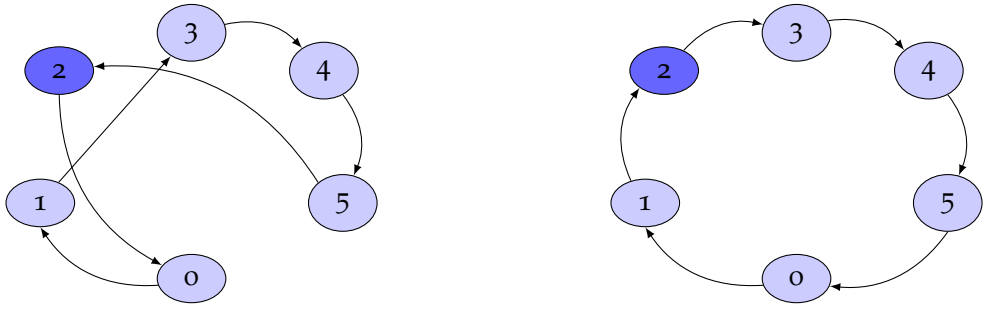


FIGURE 2.6 – *Illustration d'un mouvement de l'opérateur swap-inter.*


FIGURE 2.7 – *Illustration d'un mouvement de l'opérateur swap-intra.*


#### swap-intra

L'opérateur *swap-intra* échange les emplacements de deux clients dans une même route. Ainsi, dans l'exemple de la Figure 2.7, nous avons, à gauche, une route dont les clients 2 et 5 sont sélectionnés pour être permutés. À droite, la nouvelle solution (après l'application du mouvement) est représentée.

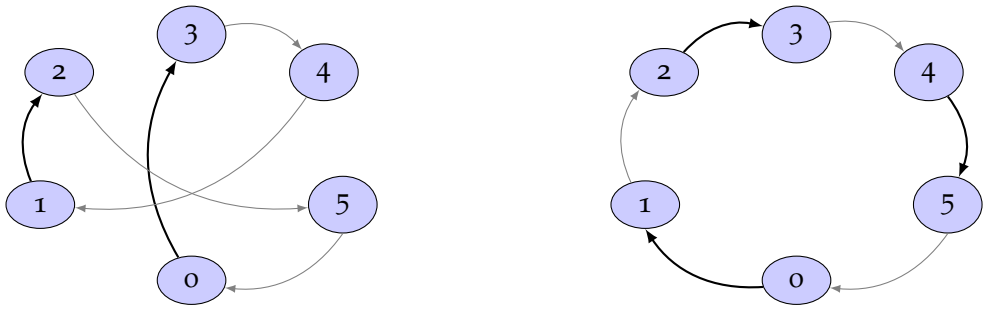
#### Or_Opt

La procédure *Or_Opt* supprime une séquence de deux ou plusieurs clients (2 clients seulement sont considérés dans ce document) d'une route donnée puis, insert cette séquence dans un autre emplacement de la même route. La Figure 2.8 fournit un exemple illustratif où, une route $0, 3, 4, 1, 2, 5, 0$ est donnée (à gauche) et la séquence de clients $(1, 2)$ est sélectionnée pour être placée entre le dépôt et le client 3. Le résultat est montré dans la partie droite.

### 2.2.3 Métaheuristiques

Le fait que les opérateurs de voisinage soient rapidement bloqués dans des optima locaux a incité les chercheurs à développer un autre genre de méthodes approchées,

FIGURE 2.8 – *Illustration d'un mouvement de l'opérateur Or_Opt.*



capables de s'extraire elles-mêmes de ce genre de « pièges ». Ces méthodes s'intitulent *les métaheuristiques*.

Les métaheuristiques sont elles aussi partagées en deux catégories conformément à leurs stratégies de recherche. On peut trouver des métaheuristiques qui, à chaque itération, tentent d'améliorer une solution unique et qui portent le nom de *méthodes de trajectoire* ou encore *méthodes de recherche locale*. Comme on peut trouver des métaheuristiques qui manipulent plusieurs solutions simultanément et qui sont connues sous le nom de *méthodes à population de solutions*.

Parmi les méthodes de recherche locale, on peut citer le *recuit simulé*, la *recherche taboue* ou encore la *recherche adaptative à grand voisinage*. D'autre part, l'algorithme génétique représente l'une des méthodes à population de solutions les plus connues notamment pour les VRPs. Notons que les méthodes citées dans ce paragraphe seront décrites plus bas.

Une autre classification des métaheuristiques peut être basée sur le fait que la méthode utilise ou non une mémoire tout au long de la recherche (comme la sauvegarde de la meilleure solution rencontrée).

En outre, la stratégie de recherche des métaheuristiques est guidée par deux critères appelés respectivement **intensification** et **diversification**. Une métaheuristique pourra ainsi être qualifiée d'**intensificatrice** si elle privilégie le premier critère et de **diversificatrice** si elle privilégie le second. Plus précisément, l'intensification consiste en l'exploitation de la (ou des) meilleure(s) solution(s) afin de l'améliorer (ou de les améliorer) davantage. Alors que la diversification a pour principe d'explorer d'autres parties de l'espace de recherche même si cela implique une dégradation de la qualité de la solution. Le but de la diversification est de sortir de l'optimum local.

Chaque métaheuristique possède des points forts et des points faibles. On peut donc penser à combiner deux ou plusieurs métaheuristiques complémentaires. On obtiendra alors une métaheuristique dite *hybride*. Ce genre de combinaisons mène en général à des méthodes beaucoup plus efficaces.

Dans ce qui suit, nous décrivons les métaheuristiques que nous avons soit implémentées ou bien citées tout au long de cette thèse. Pour davantage de détails et d'exemples concernant les métaheuristiques, nous prions le lecteur intéressé de consulter l'ouvrage de Talbi [95].

#### Recherche locale réitérée

La *recherche locale réitérée* (Iterative Local Search en anglais ou ILS) est une métaheuristique qui fait appel à un opérateur de voisinage et à une procédure dite de **perturbation** permettant de changer la structure de la solution. Plus précisément, l'opérateur de voisinage est exécuté et fournit un optimum local. Un ensemble de modifications sont ensuite effectuées à l'aide de la procédure de perturbation afin de diversifier la recherche. Les mouvements de la procédure de perturbation sont généralement aléatoires et la qualité de l'optimum local y est souvent altérée. Après cela, l'opérateur de voisinage est de nouveau exécuté. Le tout est répété jusqu'à l'atteinte d'un nombre prédéfini d'itérations.

**Recuit simulé**

Le *recuit simulé* (Simulated Annealing en anglais ou SA) est une métaheuristique qui se base sur le principe de recuit utilisé en métallurgie. Concrètement, le recuit d'un matériau est un procédé consistant à faire monter la température, puis, à la faire baisser de façon graduelle, afin de placer les atomes du matériau dans une configuration plus solide que celle obtenue par refroidissement naturel.

La métaheuristique du recuit simulé a été proposée par Kirkpatrick et al. [60] et par Černỳ [17]. Elle consiste à fixer une température initiale (matérialisée par le nombre d'itérations de l'algorithme) à un niveau assez haut et à faire baisser cette dernière graduellement (à l'aide d'un coefficient de décroissance). La méthode utilise des mouvements d'opérateurs de voisinage pour passer d'une solution à une autre mais elle ne considère pas les mêmes critères d'acceptation. En effet, SA suit un principe, tantôt aléatoire (acceptation des solutions de façon aléatoire) et tantôt déterministe (acceptation des solutions si elles améliorent la valeur de la fonction objectif), qui favorise la diversification (et donc l'acceptation de mauvaises solutions) lorsque la température est élevée et qui intensifie la recherche (acceptation de moins en moins de mauvaises solutions) lorsque la température décroît. Formellement, après chaque mouvement, si la solution est de meilleure qualité que la solution courante alors elle est retenue. Autrement, on tire un nombre aléatoirement dans l'intervalle $[0,1]$ et si ce nombre s'avère inférieur à

$$\exp^{-(f(x^{'})-f(x))/T} \tag{2.1}$$

(pour un critère de minimisation), où $f(x^{'})$ est la valeur de la fonction objectif de la nouvelle solution $x^{'}$, $f(x)$ est celle de la solution courante $x$ et $T$ est la température actuelle, alors la solution est quand même retenue. On dira que la règle de Métropolis est vérifiée.

L'algorithme s'arrête lorsqu'un certain seuil de température est atteint.

**Recherche taboue**

La *recherche taboue* ou *recherche avec tabous* (Tabu Search en anglais ou TS) a été introduite par Glover [42]. C'est une métaheuristique dont le principal objectif est d'utiliser une mémoire qui sauvegarde des informations relatives au processus de recherche, afin de parvenir à trouver des solutions de bonne qualité. TS consiste à visiter un ou plusieurs voisinages de sorte à éviter les mouvements pouvant mener à des solutions déjà rencontrées.

Contrairement au SA, la recherche taboue est une méthode déterministe. Plus précisément, tout au long du processus de recherche, la méthode parcourt des solutions en utilisant des opérateurs de voisinage de façon déterministe. Lorsque l'optimum local est atteint, la méthode poursuit sa recherche en sélectionnant les meilleures solutions rencontrées même si celles-ci sont de moins bonne qualité que la solution courante. A chaque fois qu'un mouvement est effectué, le mouvement inverse est déclaré **tabou** et il est sauvegardé dans ce qu'on appelle une **liste taboue**. Cela signifie que la méthode ne pourra plus effectuer un tel mouvement pendant un certain nombre d'itérations, afin d'éviter de retomber sur les solutions déjà visitées. La méthode utilise aussi un autre principe appelé **critère d'aspiration**, qui consiste à lever le statut tabou d'un mouve-

ment si ce dernier permet d'avoir une solution de meilleure qualité que la meilleure solution rencontrée tout au long de la recherche.

#### Recherche locale à grand voisinage

Introduite par Shaw [87], la *recherche locale à grand voisinage* (Large Neighborhood Search en anglais ou LNS) est une méthode de recherche locale qui explore de grands voisinages de la solution courante en supprimant puis en ré-insérant un certain nombre (pouvant être variable) de clients.

Une itération d'LNS correspond donc à l'application d'un opérateur de suppression suivi d'un opérateur d'insertion, et à l'acceptation ou non de la solution obtenue.

#### Recherche adaptative à grand voisinage

En s'appuyant sur les travaux de Shaw [87], Ropke et Pisinger [82] ont développé la *recherche adaptative à grand voisinage* (Adaptive Large Neighborhood Search en anglais ou ALNS). Tout en gardant le même principe de suppression et de ré-insertion, ALNS emploie une stratégie de sélection d'opérateurs se basant sur les précédents **scores** (performances) de ces derniers.

Concrètement, l'heuristique démarre avec des scores $\pi_i$ identiques pour chaque opérateur $i$ et à chaque fois qu'une opération de suppression/insertion est effectuée, les scores des opérateurs concernés sont incrémentés.

Un score peut être incrémenté par :

— $\sigma_1$ si une nouvelle meilleure solution globale est trouvée,
— $\sigma_2$ si une solution non encore acceptée est trouvée et que cette dernière est de meilleure qualité que la solution courante,
— $\sigma_3$ si la solution trouvée est moins bonne que la solution courante et qu'elle est quand même acceptée sachant qu'elle ne l'a jamais été auparavant.

A chaque fois qu'un certain nombre d'itérations préalablement fixé est atteint, on dira qu'ALNS est passée à un nouveau **segment**. Dès lors, la méthode calcule le poids $w_{i,j+1}$ de chaque opérateur $i$ pour le nouveau segment $j+1$ en fonction du poids $w_{i,j}$ de cet opérateur durant le segment précédent $j$. On emploie la Formule : $w_{i,j+1} = w_{i,j}\,(1-r) + r\,\frac{\pi_i}{\theta_i}$ où $r$ est un paramètre contrôlant de la vitesse de réaction de l'algorithme aux changements de scores, $\pi_i$ et $\theta_i$ représentent, respectivement, le score et le nombre d'appels de l'opérateur $i$ durant le segment $j$.

La sélection des opérateurs se fait ensuite, à chaque itération du segment $j+1$, en utilisant le principe de la *roulette* (Roulette-Wheel Selection en anglais). Ainsi, plus un opérateur $i$ possède un grand poids durant le segment précédent, et plus il aura de chances d'être sélectionné. Plus exactement, la chance de sélection d'un tel opérateur est de $\frac{w_i}{\sum_{a \in K} w_a}$ où $K$ est l'ensemble des opérateurs d'un même type (suppression ou insertion).

Les critères d'acceptation d'ALNS reposent sur la règle de Métropolis (voir la mé-

thode du Recuit Simulé dans la Section 2.2.3).

Ropke et Pisinger ont utilisé trois opérateurs de suppression et deux d'insertion afin que la méthode s'adapte par elle-même aux instances étudiées. Dans un autre papier, Pisinger et Ropke [78] ont utilisé ALNS avec sept opérateurs de suppression et deux d'insertion pour donner plus de robustesse à la méthode. Dans ce qui suit, nous présentons les opérateurs de suppression utilisés par Pisinger et Ropke [78]. Comme les noms de ces opérateurs en français peuvent être très longs, nous avons choisi de garder l'appellation anglaise.

**Random removal** L'opérateur *random removal* supprime des clients de façon aléatoire.

**Worst removal** Cet opérateur supprime les clients dont la suppression engendre la meilleure valeur de la fonction objectif (par rapport aux autres suppressions possibles). Un paramètre de randomisation est introduit afin de ne pas supprimer les mêmes clients à chaque fois.

**Related removal** Le principe de *related removal* est de supprimer les clients semblables (selon un critère appelé **similarité** (relatedness en anglais)) dans le but d'échanger leurs positions lors de l'appel des opérateurs d'insertion. Comme les auteurs ont traité des variantes de VRP avec ramassages et livraisons, ils ont employé la similarité $r_{ij} = \frac{1}{D}\left(d'(i,j) + d'(i,j+n) + d'(i+n,j) + d'(i+n,j+n)\right)$ entre les clients $i$ et $j$ où, $d'(i,j) = c_{ij}$ (distance/coût de transport entre les sommets $i$ et $j$) si $i$ et $j$ ne sont pas situés au dépôt et 0 sinon. Les clients $i$ et $j$ sont représentés chacun par deux sommets pour les opérations de ramassage et de livraison. On a donc $i$, $i+n$ pour représenter respectivement le ramassage et la livraison de $i$, et $j$, $j+n$ pour représenter respectivement le ramassage et la livraison de $j$, où $n$ est le nombre de clients du problème. Enfin, $D$ représente le nombre de termes différents de zéro de l'équation. Dans l'opérateur de suppression *related removal*, un client $i$ est d'abord tiré aléatoirement puis, l'opérateur cherche le client $j$ qui donne la plus grande valeur de $r_{ij}$. Le client $j$ est retenu pour la suppression et le processus est répété en commençant par $j$. Comme dans *worst removal*, un paramètre de randomisation est utilisé afin de ne pas supprimer les mêmes clients à chaque appel de la procédure.

**Historical node-pair removal** *Historical node-pair removal* sauvegarde les arcs qui font partie des meilleures solutions rencontrées tout au long de la recherche et leur donne la valeur des fonctions objectifs de ces solutions là. Les arcs les plus coûteux et plus précisément, les clients dont les nœuds constituent les extrémités de ces arcs, seront supprimés de la solution.

**Historical request-pair removal** Dans cet opérateur, les $B$ meilleures solutions rencontrées sont sauvegardées. Pour chaque paire de clients, un poids est affecté, représentant le nombre de fois où ces clients ont été desservis par le même véhicule dans les solutions sauvegardées. Ainsi, plus le poids d'une paire de clients est grand, et plus les deux clients sont considérés comme similaires. Ils seront alors supprimés de la même façon que dans *related removal*. Notons que l'ensemble des solutions sauvegardées est mis à jours à chaque fois qu'une bonne solution (meilleure que la moins bonne solution

sauvegardée) est trouvée. La plus mauvaise solution est supprimée de sorte à garder toujours le même nombre de solutions en mémoire.

**Cluster removal** Cet opérateur utilise l'algorithme de Kruskal (voir 2.2.4) pour séparer les clients d'une route donnée en deux blocs. De la même façon que pour *related removal*, le premier client est choisi aléatoirement. Puis, en partant de ce client, la procédure tente de diviser la route à laquelle il appartient en deux blocs en utilisant l'algorithme de Kruskal. Après cela, l'un des blocs est retenu pour la suppression (choix aléatoire) et l'un de ses clients est utilisé pour sélectionner la prochaine route à étudier. Plus exactement, le client est d'abord choisi de façon aléatoire parmi les clients du même bloc. Ensuite, la similarité des autres clients non encore supprimés est calculée par rapport à ce client, en privilégiant les clients des autres routes. Le client dont la similarité est la plus grande est considéré comme le nouveau point de départ. La procédure poursuit la suppression de la même façon. Notons que dans *cluster removal*, si la suppression du dernier bloc engendre la suppression d'un plus grand nombre total de clients que celui fixé au départ, l'opérateur supprime quand même ce nombre là.

**Time-oriented removal** Comme des contraintes relatives au temps sont également utilisées dans les variantes étudiées par Pisinger et Ropke [78], un opérateur se basant sur les temps d'arrivée des clients dans la solution courante est implémenté. Ce dernier est une variante de *related removal* où le calcul de la similarité est légèrement différent. En effet, cette dernière est évaluée entre deux clients en comparant leurs temps d'arrivée pour les opérations de ramassages et de livraisons.

Quant aux opérateurs d'insertion, les auteurs ont utilisé :

**Basic greedy** Cet opérateur calcule pour chaque client non inséré sa meilleure position d'insertion dans toutes les routes disponibles. Il retient ensuite, le meilleur client (selon le coût d'insertion) et l'insert à sa meilleure position. Ceci est répété jusqu'à insertion de tous les clients dans la solution.

**Regret** La stratégie de l'opérateur de *regret* consiste à évaluer le regret d'insertion d'un client à une position différente de sa meilleure position. Le but étant d'éviter de laisser les clients « difficiles à insérer » en dernier (ce qui est le cas dans *basic greedy*). Plus précisément, l'opérateur de *regret* tente de placer dans la solution, le client que l'on regrettera le plus s'il n'est pas inséré à l'itération actuelle. En effet, l'opérateur calcule pour chaque client les coûts de ses insertions dans les différentes positions des différentes routes disponibles. L'opérateur de *regret* évalue ensuite l'écart entre les $k$ meilleurs coûts d'insertion du client étudié et le coût de sa meilleure insertion. On dira que l'on utilise un opérateur de *regret* de niveau $k$. Le client choisi sera celui dont l'écart est le plus grand. Il sera donc inséré à sa meilleure position. Ceci est répété jusqu'à ce qu'il ne reste plus de clients non insérés.

#### Recherche à voisinages variables

La *recherche à voisinages variables* (Variable Neighborhood Search en anglais ou VNS) a été proposée par Mladenović et Hansen [68]. Il s'agit d'un algorithme qui utilise une stratégie de recherche efficace, combinant les points forts de plusieurs opérateurs de voisinage, dans le but de s'extraire des optima locaux. Plus précisément, la méthode

consiste à visiter séquentiellement un ensemble de voisinages de sorte que, la sélection de ces derniers se fait soit de façon aléatoire ou bien dans un ordre prédéfini.

**Descente à voisinages variables** La *descente à voisinages variables* (Variable Neighborhood Descent en anglais ou VND) est une version déterministe de VNS. Elle a pour principe de parcourir un ensemble de voisinages $V_i$ avec $i = 1, \ldots, i_{max}$ où $max$ désigne le nombre de voisinages étudiés, tels que le premier voisinage $V_1$ soit visité entièrement jusqu'à ce qu'il n'y ait plus d'améliorations possibles. C'est alors que la méthode passe au second voisinage $V_2$ et là, deux cas se présentent : $i$) si une amélioration est trouvée, VND retourne à $V_1$ et cherche de nouveau l'optimum local en ce voisinage ; $ii$) sinon, VND parcourt l'intégralité du voisinage $V_2$ et passe au voisinage suivant $V_3$. De la même manière, VND peut aller jusqu'au voisinage $V_{i_{max}}$. Si aucune amélioration n'est trouvée avec $V_{i_{max}}$, la recherche s'arrête.

Pour davantage de détails sur VNS et ses variantes, nous prions le lecteur intéressé de consulter l'ouvrage de Talbi [95].

Notons que certains auteurs tels que Subramanian et al. [92] considèrent une autre définition de VND. Celle-ci correspond à la définition de VNS que nous avons présenté plus haut. Étant donné que nous nous sommes basés sur les travaux de Subramanian et al. [92] lors de l'implémentation de nos méthodes, nous allons donc considérer leur définition de VND et pas celle donnée plus haut.

Notons de la même façon que certains auteurs de la littérature considèrent une définition de VNS équivalente à la définition de VND donné plus haut.

#### Procédure de recherche gloutonne aléatoire adaptative

*La procédure de recherche gloutonne aléatoire adaptative* (Greedy Randomized Adaptive Search Procédure en anglais ou GRASP) est une heuristique itérative qui a été proposée par Feo et Resende [37]. À chaque itération, GRASP construit une nouvelle solution réalisable grâce à un algorithme glouton aléatoire puis, applique une procédure de *recherche locale* à la solution obtenue dans le but de l'améliorer. À la fin de la recherche, la meilleure solution est retournée. La procédure s'arrête lorsqu'un certain nombre d'itérations est atteint.

#### Recherche locale guidée

La *Recherche Locale Guidée* (Guided Local Search en anglais ou GLS) est une approche caractérisée par un changement dynamique de la fonction objectif en fonction des optima locaux rencontrés précédemment. Le but de GLS est de tenter de visiter des régions plus prometteuses dans l'espace de recherche. En effet, à chaque fois qu'un optimum local est rencontré, GLS identifie ses attributs (ensemble d'arcs par exemple) et modifie la fonction objectif en ajoutant des pénalités pour ces derniers, de sorte à rendre la visite d'une solution ayant les mêmes attributs plus coûteuse et sortir ainsi de l'optimum local. Voir l'ouvrage de Talbi [95] pour plus de détails concernant cette méthode et l'article de Voudouris et Tsang [101] pour son application au TSP.

**Programmation à mémoire adaptative**

La *programmation à mémoire adaptative* (Adaptive Memory programming en anglais ou AM) a été proposée par Taillard et al. [94]. L'idée principale de cette méthode est d'avoir une approche unificatrice des métaheuristiques les plus performantes. En effet, ces dernières partagent plus ou moins le même principe : elles mémorisent des solutions (ou plutôt, certaines de leurs caractéristiques), elles génèrent de nouvelles solutions à partir de celles mémorisées, puis, elles utilisent une procédure de recherche locale pour améliorer les solutions trouvées. La programmation à mémoire adaptative suit donc ces étapes pour tenter d'obtenir des solutions de bonne qualité.

**Algorithme de colonie de fourmis**

L'*algorithme de colonie de fourmis* (Ant Colony Optimization algorithm en anglais ou ACO) est un algorithme proposé par Dorigo [31]. ACO est inspiré du comportement des individus dans une colonie de fourmis. Ces individus cherchent à rassembler de la nourriture en choisissant le plus court chemin menant à cette dernière.

Plus précisément, les fourmis communiquent entre elles en déposant une substance chimique appelée phéromone tout au long des passages qu'elles empruntent. Elles reconnaissent et suivent cette substance afin de retrouver le plus court chemin menant à la nourriture. D'un autre coté, les phéromones ont la particularité de s'évaporer rapidement. Ainsi, la probabilité qu'un individu choisisse un chemin donné dépend du nombre de fourmis qui y sont passées et qui y ont déposé des phéromones. En début de recherche, les trajectoires sont aléatoires mais à la fin, elles tendent à converger vers le plus court chemin.

Pour les problèmes d'optimisation combinatoire, le comportement décrit ci-dessus est imité par ACO. En effet, ACO manipule un ensemble de solutions. Ces solutions représentent les individus de la colonie qui interagissent entre eux. Dans le cas du VRP par exemple, on peut d'abord définir une trace de phéromones pour chaque arête du problème. Cette trace correspond à l'utilité de l'arête en question. Ainsi, plus une arête appartient à des solutions de bonne qualité et plus la trace de phéromones y sera importante. Au début les valeurs attribuées aux traces des phéromones sont identiques. Chaque solution est construite par l'ajout d'arêtes selon une certaine probabilité dépendant de la trace de phéromones et du coût de l'arête à introduire. La trace de phéromones de chaque arête est ensuite mise à jour en fonction de la qualité des solutions obtenues. Donc de meilleures solutions pourront être construites en suivant les traces des phéromones de bonne qualité. La procédure d'évaporation des phéromones consiste en la réduction de la trace de phéromones dans chaque arête. Ceci peut être fait à l'aide d'un coefficient de réduction.

Talbi [95] donne une présentation complète et illustrée d'exemples du fonctionnement d'ACO. Nous encourageons donc le lecteur intéressé à consulter son ouvrage.

**Optimisation par essaims particulaires**

L'*optimisation par essaims particulaires* (Particles Swarm Optimization en anglais ou PSO) a été présentée pour la première fois en optimisation combinatoire par Kennedy et Eberhart [59]. Tout comme ACO, PSO se base sur le comportement d'espèces à la

recherche de nourriture. Cependant, au lieu d'imiter les fourmis, PSO s'inspire du comportement des oiseaux lors de leur déplacement (migration) à la recherche d'endroits pour se nourrir. L'idée est que, des déplacements individuels très simples entraînent la convergence progressive de l'ensemble du groupe vers l'endroit recherché. En fait, le déplacement de chaque oiseau (ou particule), à l'instant $t$, comprend deux paramètres qui sont : la position actuelle et la vitesse de vol. Un déplacement se base sur la meilleure position rencontrée par l'individu à un instant donné et celle rencontrée par l'ensemble du groupe. Le déplacement d'un individu influencera, par la suite, le comportement des autres individus de la population.

D'un point de vue algorithmique, chaque particule représente une solution. D'une itération à une autre, les solutions sont modifiées (les positions des particules changent). Une modification d'une solution s'effectue en appliquant la nouvelle vitesse de déplacement de la particule à sa position actuelle, dans le but de déterminer sa nouvelle position. Par exemple, la nouvelle vitesse peut être représentée par un croisement entre la solution actuelle, la meilleure solution globale et la meilleure solution obtenue par l'application d'une recherche locale à la solution actuelle. Autrement dit, un croisement entre la position actuelle, la position obtenue par l'influence sociale exercée sur la particule, et la position obtenue par l'influence individuelle de la particule elle-même. Tout cela est régi par un ensemble de coefficients ajustant la probabilité d'effectuer ou non des modifications, et représentant les degrés d'influence. Des détériorations des solutions peuvent aussi être introduites dans les vitesses de déplacement.

#### Algorithme génétique

L'*algorithme génétique* (Genetic Algorithm en anglais ou GA) est une métaheuristique à population de solutions proposée par Holland [55]. Cette métaheuristique s'inspire de phénomènes biologiques et plus exactement de la transmission de gènes des parents à leurs progénitures au fil des générations. Comme cela est bien connu dans le domaine de la génétique, les noyaux des cellules des organismes vivants sont constitués de chromosomes qui sont, à leur tour, constitués de chaînes d'ADN. Ces dernières contiennent les informations génétiques de l'individu et donc ses caractéristiques. Ainsi, lorsque deux parents donnent naissance à un nouvel individu, ce dernier possédera une partie des caractéristiques de chaque père.

Par analogie, dans les problèmes de tournées de véhicules, chaque solution est considérée comme étant un **chromosome** et les caractéristiques de cette dernière représentent les chaînes d'ADN. Le processus de recherche de l'algorithme génétique commence par le regroupement d'un certain nombre de solutions pour former une **population**. À chaque étape de la recherche, on sélectionne des couples de solutions faisant partie de cette population et on leur applique des opérateurs de **croisement** pour déterminer quelle partie de chaque parent sera retenue dans le nouvel individu. Afin d'ajouter de la diversification à la recherche, un opérateur de **mutation** est ajouté. Cet opérateur vise à modifier, sous une certaine probabilité, quelques unes des caractéristiques de l'individu créé (même si cela engendre une nouvelle caractéristique qui n'est propre à aucun parent). Après cette génération d'individus, l'algorithme passe à la **sélection** des solutions qui constitueront la nouvelle population. Le processus est répété jusqu'à ce qu'un certain nombre d'itérations soit atteint.

#### Algorithme mémétique

L'*algorithme mémétique* (Memetic Algorithm en anglais ou MA) est une version du GA utilisant des opérateurs de voisinage après l'application de l'opérateur de croisement. Cette version donne souvent de bien meilleurs résultats que la version d'origine.

### 2.2.4 Heuristiques utiles pour le problème de tournées de véhicules

Bien que certains algorithmes ne soient pas spécialement conçus pour résoudre les VRPs, leur utilisation pour ce type de problèmes n'en demeure pas moins avantageuse. Parmi ces algorithme, nous pouvons citer l'*algorithme de Kruskal* qui est utilisé dans l'une des procédures de suppression d'ALNS, dans le but de déterminer l'ensemble des clients à supprimer de la solution. Cet algorithme peut être décrit comme suit :

#### Algorithme de Kruskal

L'*algorithme de Kruskal* (voir l'article de Kruskal [61]) a pour objectif de trouver l'arbre couvrant de poids minimal dans un graphe connexe valué. Au début, les arêtes du graphes sont triées dans l'ordre croissant de leurs poids. L'algorithme commence par un ensemble vide auquel il ajoute une arête à chaque itération. Cela forme à chaque fois un nouveau sous-graphe (pas forcément connexe) constitué des arêtes choisies. Ce sous-graphe peut être vu comme un ensemble de composantes connexes. Le principe de l'algorithme de Kruskal est de choisir les arêtes de poids minimum, dont l'ajout n'entraîne pas la création d'un cycle au sein du sous-graphe. L'algorithme s'arrête lorsque tous les sommets du graphe initial sont dans le sous-graphe créé et que ce dernier forme une seule composante connexe. Il est clair qu'à cette étape, l'ajout d'une quelconque arête du graphe initial au sous-graphe entraînerait la création d'un cycle.

Lorsqu'il s'agit d'utiliser l'algorithme de Kruskal au sein de la procédure *cluster removal* d'ALNS, l'algorithme est arrêté lorsque tous les sommets du graphe initial sont dans le sous-graphe et que deux composantes connexes sont trouvées. Si lors de l'avant dernière itération, le sous-graphe n'est composé que d'une seule composante connexe, on considère ce sous-graphe comme première composante connexe, le dernier sommet à ajouter représentera la seconde composante connexe.

## Conclusion

Dans ce Chapitre, nous avons parlé des deux principales classes de méthodes de résolution pour le problème de tournées de véhicules : la classe des méthodes exactes et celle des méthodes approchées. Nous avons montré leurs différences et pour chaque classe, nous avons donné quelques exemples des méthodes les plus utilisées dans la littérature. Ceci a pour but de familiariser le lecteur avec ces méthodes et de lui faciliter la compréhension des Chapitres suivants.

## Deuxième partie

# Méthodes approchées pour quelques variantes du problème de tournées de véhicules avec ramassages et livraisons

# Méthodes approchées pour le problème de la tournée la plus profitable avec contrainte de capacité

3

Sommaire



Dans ce Chapitre, nous aborderons le *problème de la tournée la plus profitable avec contrainte de capacité* (Capacitated Profitable Tour Problem en anglais ou CPTP). Le CPTP est une variante du VRP dans laquelle les clients possèdent soit des demandes de ramassage ou bien des demandes de livraison. L'une des principales caractéristiques du CPTP est que la visite de tous les clients n'est plus imposée. Nous commencerons

donc par décrire le CPTP de façon plus détaillée et passerons en revue les travaux de la littérature. Nous présenterons, par la suite, les méthodes que nous avons implémentées pour le problème étudié. Enfin, nous analyserons les résultats obtenus et terminerons par une conclusion.

## Introduction

L'un des rôles des entreprises de transport est d'acheminer ou de collecter certains biens à/depuis un ensemble de clients. De nos jours, le nombre de demandes de livraison (respectivement, de ramassage) accroît, et la concurrence entre les entreprises de transport devient rude. Les transporteurs cherchent à étendre leurs zones de livraison (respectivement, de ramassage) pour attirer plus de clients. Les entreprises de transport sous-traitent en général avec plusieurs entreprises de production qui cherchent soit à livrer des produits finis aux clients ou bien à récupérer, des clients, les produits à la fin de leurs cycles de vie pour des fins de recyclage ou de réparation.

Les capacités des véhicules des entreprises de transport ne sont pas forcément adaptées à toutes les demandes. Cela engendre généralement de longues distances de retour traversées par des camions à moitié vides ainsi que d'importants coûts de transport.

Le retour de ces camions à moitié vides engendre à son tour une augmentation de l'encombrement routier et ainsi, une augmentation de la pollution. De plus, ces retours vides ne sont pas profitables pour les transporteurs. En effet, non-seulement les coûts des longues distances parcourues ne sont pas rentabilisés par les gains des transporteurs, l'utilisation des véhicules n'est pas optimisée, et les demandes d'autres clients se trouvant sur le chemin du retour doivent être satisfaites par d'autres véhicules.

Une des stratégies adoptées par les entreprises de transport pour faire face à cette situation consiste à transformer les trajets où les véhicules sont vides en opérations de transport de même type que celle déjà programmées (ramassage ou livraison) générant des profits. Ces derniers contribuent à la réduction du coût de transport global.

D'un point de vue environnemental, s'acquitter d'une part des demandes sur le marché, moyennant des véhicules déjà en service, empêche l'utilisation d'autres véhicules pour satisfaire ces demandes. En d'autres termes, le fait d'augmenter le rendement des véhicules en service implique moins de consommation de carburant et ainsi moins de pollution.

Dans le *problème de la tournée la plus profitable avec contrainte de capacité*, il est supposé qu'une entreprise de transport sous-traite avec plusieurs entreprises de production moyennant des contrats à long terme. De plus, les transporteurs peuvent choisir d'effectuer des opérations de transport additionnelles lorsque le chargement des véhicules utilisés n'atteint pas la limite de la capacité. Le choix se fait en fonction du gain (ou profit) obtenu en visitant chaque client ainsi qu'en fonction du coût de transport engendré par le détour. D'une façon générale, dans un CPTP, il est question de choisir le sous-ensemble le plus profitable de clients à visiter. Pour assurer la visite des clients des contrats à long terme, un profit assez élevé peut leur être affecté.

Dans ce qui suit et sans perte de généralités, nous supposons que les demandes à satisfaire dans un *problème de la tournée la plus profitable avec contrainte de capacité* sont des demandes de ramassage.

## 3.1 Description du problème de la tournée la plus profitable avec contrainte de capacité

Le *problème de la tournée la plus profitable avec contrainte de capacité* (Capacitated Profitable Tour Problem en anglais ou CPTP) est une variante du VRP dans laquelle la visite de l'ensemble des clients n'est plus obligatoire. C'est aussi une extension du *problème de la tournée la plus profitable* proposé par Dell'Amico et al. [26] dans laquelle une limite de capacité est imposée pour les véhicules utilisés.

Plus précisément, dans un CPTP, $i$) une flotte finie de véhicules identiques et de capacités limitées est disponible dans un dépôt, $ii$) chaque client du problème possède une demande de ramassage devant être entièrement satisfaite si le client est visité, $iii$) un profit spécifique est collecté à chaque fois qu'un client est visité, $iv$) chaque client est visité au plus une fois, $v$) l'objectif du CPTP est de concevoir des tournées réalisables maximisant la différence entre les profits collectés et les coûts de transport.

Le CPTP peut être modélisé à l'aide d'un graphe non orienté et valué $G = (V, E, c)$ dans lequel $V = N \cup \{0\}$ est l'ensemble des sommets où $N = \{1, \ldots, n\}$ est l'ensemble des $n$ clients et 0 est le dépôt, $E$ représente l'ensemble des arêtes de $G$ (ensembles de sections de routes reliant deux clients ou un client et le dépôt), $c : E \rightarrow \mathbb{R}^+$ est une fonction de poids qui associe à chaque arête $\{i, j\} \in E$ un poids $c_{ij} \in \mathbb{R}^+$ où $c_{ij}$ représente le coût de transport entre les éléments $i$ et $j$ de $V$. À chaque sommet $i \in N$, une demande de ramassage $p_i$ et un profit $pr_i$ sont associés. Il faudra alors déterminer dans $G$, un ensemble de cycles dont l'intersection donne le sommet 0, de sorte à maximiser la différence entre les profits des clients inclus dans les cycles et le coût de transport engendré lors de la visite des clients. Le nombre de cycles doit être inférieur ou égal à $m$, où $m$ représente le nombre de véhicules disponibles. Chaque cycle doit donc inclure le sommet 0 et un sous-ensemble de sommets de $N$. La capacité $Q$ de chaque véhicules doit être respectée.

## 3.2 État de l'art

Malgré son importance, le CPTP a été peu étudié dans la littérature. Il a été introduit par Archetti et al. [2] afin d'étendre le PTP à une version incluant des contraintes de capacité limitée des véhicules.

Dans leur papier, Archetti et al. [2] ont implémenté trois méthodes pour résoudre le problème. Il s'agit de la *recherche à voisinages variables* (VNS), la *recherche taboue* à solutions réalisables et la *recherche taboue* à solutions irréalisables.

Comme son nom l'indique, la *recherche taboue* à solutions réalisables n'accepte que des solutions réalisables lors de l'exploration de l'espace de recherche. La seconde *recherche taboue* permet, quant à elle, de visiter des solutions irréalisables en favorisant celles avec de « petites irréalisabilités ».

Dans les deux *recherches taboues*, les mouvements utilisés sont des mouvements inter-routes. Afin de supprimer, ou du moins, de diminuer l'irréalisabilité, les auteurs

ont aussi proposé une heuristique de réparation.

L'algorithme VNS proposé par Archetti et al. [2] utilise la *recherche taboue* à solution réalisables avec un petit nombre d'itérations, ce qui procure l'avantage de visiter un grand nombre de régions de l'espace de recherche.

## 3.3 Approches proposées pour le CPTP

Dans la présente Section, nous décrivons les heuristiques que nous avons implémentées pour le CPTP, il s'agit de la *recherche locale réitérée* (Iterative Local Search heuristic en anglais ou ILS), la *descente à voisinages variables avec ordre aléatoire des voisinages* (Variable Neighborhood Descent with Random neighborhood ordering en anglais ou RVND) et la *recherche locale à grand voisinage* (Large Neighborhood Search en anglais ou LNS).

Dans la littérature, des combinaisons d'ILS et RVND ont été appliquées avec succès à une variante du VRP appelée *VRP avec ramassages et livraisons simultanés* (voir Subramanian et al. [92], Subramanian et al. [93]). Ces combinaisons s'avèrent très compétitives au moment de la publication du présent document, en comparaison avec les autres approches conçues pour le même problème.

Combinées ensemble, les heuristiques ILS et VND parviennent également à donner des solutions de bonne qualité pour certaines variantes de *problèmes de ramassage et livraison* dont le *problème de voyageur de commerce avec services de ramassage et de livraison mixtes* (Traveling Salesman Problem with Mixed Pickup and Delivery en anglais) présenté par Subramanian et Battarra [91], le *problème de voyageur de commerce avec ramassages et livraisons et chargement de type premier-arrivé-premier-servi* (Pickup and Delivery Traveling Salesman Problem with first-in-first-out loading en anglais) étudié par Erdoğan et al. [33], le *problème de voyageur de commerce avec ramassages et livraisons à un seul produit* (One-commodity Pickup and Delivery Traveling Salesman Problem en anglais) proposé par Hernández-Pérez et al. [49] et le *1-à-1 problème de voyageur de commerce avec ramassages et livraisons de plusieurs produits* (Multi-Commodity One-to-One Pickup and Delivery Traveling Salesman Problem en anglais) proposé par Rodríguez-Martín et Salazar-González [81]. De plus, les heuristiques ILS et VND ont donné de bons résultats sur quelques variantes de *problème de ramassage et livraison avec profits* (Pickup and Delivery Problems with Profits en anglais) tels que, le *problème de tournées de véhicules multiobjectif avec livraisons fixées et ramassage optionnels* (Multiobjective vehicle routing problem with fixed delivery and optional collections en anglais) introduit par Assis et al. [4] et le *problème de la tournée la plus profitable multi-véhicules avec ramassages et livraisons* (multi-vehicle profitable pickup and delivery problem en anglais) étudié par Gansterer et al. [39].

Au meilleur de notre connaissance, aucun article de la littérature n'a proposé de combinaison d'ILS avec LNS pour la résolution d'une variante de VRP. Cependant, les deux heuristiques ont été appliquées indépendamment avec succès (voir par exemple les travaux de Cuervo et al. [25], Silva et al. [88], Morais et al. [71] et Li et al. [64] pour ILS, et les travaux de François et al. [38], Grangier et al. [44], Akpinar [1] et Dominguez et al. [30] pour LNS).

Nous rappelons que l'heuristique ILS tente d'améliorer des solutions de recherches locales élémentaires. En effet, après l'exécution d'une recherche locale, un optimum local est atteint, ensuite ILS perturbe l'optimum local trouvé et fait encore appel à la recherche locale pour tenter d'améliorer la qualité de la solution. Ce processus est répété jusqu'à ce que le critère d'arrêt soit atteint.

Dans une heuristique de type RVND, une liste de structures de voisinages est utilisée de sorte que, à chaque itération, une structure de voisinage est aléatoirement choisie et appliquée à la solution actuelle. La nouvelle solution est alors acceptée ou non conformément aux critères d'acceptation établis. Par la suite, la structure de voisinage choisie est supprimée de la liste et le processus continue de la même façon pour les structures restantes. RVND s'arrête lorsque la liste de structures de voisinages est vide.

En comparaison avec les autres heuristiques de recherche locale, LNS permet la visite d'une plus grande zone de l'espace de recherche. Cela est dû au changement dans la structure de la solution étudiée engendré par LNS. En effet, LNS supprime un nombre de clients relativement grand de la solution actuelle, et ré-insert les clients supprimés dans des positions différentes. Cela permet de générer une solution complètement différente de la solution actuelle et aide l'heuristique à sortir des optima locaux. Si le nombre de clients à supprimer est fixé à une petite valeur, LNS peut être considérée comme une simple heuristique de recherche locale.

Afin d'essayer de résoudre le CPTP, nous avons proposé plusieurs combinaisons des heuristiques ILS, RVND et LNS. Dans ce qui suit, nous décrirons l'heuristique de construction implémentée ainsi que chacune des combinaisons proposées.

### 3.3.1 Heuristique de construction

L'heuristique de construction utilisée dans ce travail est une heuristique séquentielle basée sur l'heuristique I1 (voir la travail de Solomon [90] pour davantage de détails).

I1 a été initialement implémentée pour le *problème de tournées de véhicules avec fenêtres de temps* (Vehicle Routing Problem with Time Window en anglais ou VRPTW). Le pseudo-code de l'heuristique de construction est donné dans l'Algorithme 1.

L'heuristique de construction commence à partir d'une route vide. Cette dernière est initialisée en reliant un *client graine* (seed customer en anglais) au dépôt. Ensuite, les insertions des clients restants au sein de la route sont évaluées. L'heuristique sélectionne alors la meilleure position d'insertion pour chaque client non visité $u$ entre deux clients consécutifs $i$ et $j$ selon le critère $cr_1(i,u,j)$. Enfin, l'heuristique de construction sélectionne le client $u^*$ dont la meilleure position d'insertion optimise le critère $cr_1(i,u,j)$ parmi toutes les meilleures positions d'insertion obtenues. Ce processus est répété jusqu'à ce qu'aucun client ne puisse être inséré au sein de la route (en tenant compte des conditions de réalisabilité). Les autres routes de la solutions sont construites de façon similaire. Dans l'heuristique de construction proposée, le critère $cr_1(i,u,j)$ considère les coûts d'insertion et les profits collectés lors du choix des clients à insérer. Le critère basé sur les contraintes de fenêtres de temps de l'heuristique présentée par Solomon [90] est omis.

**Algorithme 1** Heuristique de construction pour le CPTP

**Entrées:**
Une instance de CPTP
Une liste $L_{unr}$ de clients ne faisant pas partie de la solution
Un nombre $nbRoutes = 0$ de routes présentes dans la solution actuelle
**Sorties:**
Une solution réalisable
**tq** $nbRoutes <$ nombre de véhicules **faire**
  Créer une nouvelle route ;
  $nbRoutes++$ ;
  Ajouter un "client graine" à la nouvelle route ;
  **tq** $\exists u \in L_{unr}$ pouvant être inséré sans mener à une solution irréalisable **faire**
    Evaluer l'insertion de chaque client non visité $u \in L_{unr}$ dans la route actuelle ;
    Sélectionner, pour chaque $u$, la meilleure position d'insertion selon le critère $cr_1(i,u,j)$ ;
    Sélectionner le client $u^*$ avec la meilleure valeur de $cr_1(i,u,j)$ ;
    Insérer le client $u^*$ dans sa meilleure position dans la route actuelle ;
    Mettre à jour $L_{unr}$ ;
  **fin tq**
**fin tq**

Soit $(i_0, i_1, \ldots, i_h)$ la route actuelle, où $i_\rho$ représente le client à la $\rho^{ième}$ position de la route si $\rho \notin \{0, h\}$, et $i_\rho$ représente le dépôt sinon ($i_0 = i_h = 0$). L'heuristique calcule pour chaque client $u$ sa meilleure position d'insertion au sein de la route actuelle selon les Expressions (3.1)–(3.4), où (sans perte de généralités) $c_{ij}$ représente le coût de transport entre $i$ et $j$, $pr_u$ est le profit du client $u$, et $\alpha_1, \alpha_2 \geq 0$ sont des paramètres définis par l'utilisateur, avec $\alpha_1 + \alpha_2 = 1$.

$$cr_1(i(u), u, j(u)) = \max\left\{cr_1(i_{\rho-1}, u, i_\rho), \rho = 1, \ldots, h\right\}; \tag{3.1}$$

$$cr_1(i,u,j) = \alpha_1 \cdot cr_{11}(i,u,j) - \alpha_2 \cdot cr_{12}(i,u,j); \tag{3.2}$$

$$cr_{11}(i,u,j) = pr_u; \tag{3.3}$$

$$cr_{12}(i,u,j) = c_{iu} + c_{uj} - c_{ij}. \tag{3.4}$$

Dans notre cas, le "client graine" est choisi de façon aléatoire.

Remarquons que pour différentes valeurs des paramètres $\alpha_1$ et $\alpha_2$, nous pouvons obtenir des solutions différentes pour le CPTP.

### 3.3.2 Heuristique ILS avec différents opérateurs de voisinage

L'heuristique ILS est composée de deux procédures principales qui sont exécutée alternativement jusqu'à l'atteinte d'un certain nombre d'itérations sans améliorations. La première procédure est celle de recherche locale. Cette procédure tente d'améliorer une solution actuelle grâce à un ensemble de mouvements élémentaires jusqu'à l'atteinte d'un optimum local. La seconde procédure est celle de perturbation. La perturbation détruit en quelques sortes la solution produite par la recherche locale afin de diversifier la recherche.

Nous avons proposé sept versions de l'heuristique ILS. Dans chaque version, un opérateur de voisinage différent est utilisé comme procédure de recherche locale.

Chaque heuristique implémentée est notée de la façon suivante : ILS_OpVois où OpVois est l'opérateur de voisinage utilisé.

ILS_OpVois s'arrête lorsqu'un certain nombre d'itérations sans améliorations est atteint.

Le pseudo-code d'une heuristique de type ILS_OpVois est donnée par l'Algorithme 2, où *OpVois* fait référence à l'un des opérateurs de voisinage implémentés.

**Algorithme 2** ILS_OpVois

```
1: Entrées:
       Une instance de CPTP
2: Sorties:
       La meilleure solution rencontrée
3: tq critère d'arrêt non atteint faire
4:     Générer une solution initiale avec l'heuristique de construction ;
5:     tq critère d'arrêt d'ILS non atteint faire
6:         Exécuter OpVois ;
7:         Mettre à jour la meilleure solution ;
8:         Perturber la solution actuelle ;
9:     fin tq
10: fin tq
```

Notons que l'heuristique proposée est une heuristique multi-départs qui génère, à chaque itération, une solution initiale avec une nouvelle valeur des paramètres. Toutes les valeurs possibles des paramètres seront testées à la fin de la recherche et l'algorithme principal devra s'arrêter (critère d'arrêt ligne 3). Ceci est également applicable pour toutes les heuristiques de type ILS décrites dans ce Chapitre.

Les procédures de recherche locale et de perturbation étudiées pour ILS sont données dans ce qui suit. L'impact de l'utilisation d'ILS sur la performance des opérateurs de voisinage est souligné.

#### Recherche locale

Les opérateurs de voisinage implémentés sont ceux décrits dans la Section 2.2.2.

Pour chaque opérateur de voisinage, nous avons une heuristique ILS différente. Nous obtenons alors les heuristiques ILS_2-Opt, ILS_2-Opt*, ILS_relocate-intra, ILS_relocate-inter, ILS_swap-intra, ILS_swap-inter et ILS_OrOpt qui font référence à ILS avec 2-Opt, 2-Opt*, relocate-intra, relocate-inter, swap-intra, swap-inter et OrOpt respectivement.

#### Perturbation

La procédure de perturbation supprime un certain nombre de clients d'une façon aléatoire grâce à l'opérateur *random removal*. Un ensemble de clients est, par la suite, éventuellement inséré dans la solution obtenue grâce à l'opérateur *basic greedy*. Voir la Section 2.2.3 pour plus de détails concernant ces deux opérateurs.

**Impact de l'utilisation d'ILS sur la performance des opérateurs de voisinage**

L'importance de l'utilisation d'ILS est mise en évidence dans la Figure 3.1. Dans cette Figure, nous avons 6 clients 1, 2, 3, 4, 5 et 6 ainsi qu'un dépôt 0. Pour faciliter la compréhension, nous supposons qu'il n'y ait pas de clients non visités dans les solutions présentées. Supposons que les profits des six clients soient égaux à 5 et que les coûts de transport soient donnés par les poids des arcs. Supposons aussi qu'un optimum local est atteint par l'opérateur *swap-intra*. Cet optimum local est donné par le graphe se situant en haut à gauche de la Figure et possède une valeur de la fonction objectif (somme des profits $-$ coûts de transport) égale à $6 \cdot 5 - 18 = 12$.

Dans ILS, une fois que cet optimum local est atteint, la procédure de perturbation est lancée. Celle-ci supprime un certain nombre de clients (dans ce cas 3) et les ré-insert dans la solution. Notons que la procédure de perturbation peut ne pas remettre de clients dans la solution et qu'elle peut remettre juste un sous-ensemble de clients (pas nécessairement les clients supprimés). Par soucis de clarté, nous supposons que les clients supprimés sont tous remis dans la solution.

Les clients 3, 4 et 5 sont supprimés des deux routes et réinsérés dans une seule route. Cela engendre la solution donnée en haut à droite de la Figure. La valeur de la fonction objectif de la solution perturbée est égale à $6 \cdot 5 - 19 = 11$. Cette solution est donc moins profitable que l'optimum local obtenu par l'opérateur swap-intra.

Après cela, ILS fait une fois de plus appel à l'opérateur swap-intra. Ce dernier permute les positions des clients 4 et 5. Ce qui engendre la solution en bas de la Figure. Cette solution possède une valeur de la fonction objectif égale à $6 \cdot 5 - 17 = 13$. La nouvelle solution est donc plus profitable que les deux précédentes.

Notons que cette solution n'aurait pas pu être atteinte par l'opérateur swap-intra sans l'utilisation d'ILS. En effet, étant donné que swap-intra permute des clients d'une même route, le client 3 n'aurait pas pu être placé dans la seconde route.

### 3.3.3 Hybridation d'ILS avec RVND

L'heuristique hybride combinant ILS avec RVND (notée ILS_RVND) utilise le même principe que l'heuristique ILS présentée dans la Section 3.3.2. La seule différence est que dans ILS_RVND, une descente à voisinage variable avec ordre aléatoire des voisinages est utilisée au lieu d'un unique opérateur de voisinage. ILS s'arrête lorsqu'un certain nombre d'itérations sans améliorations est atteint.

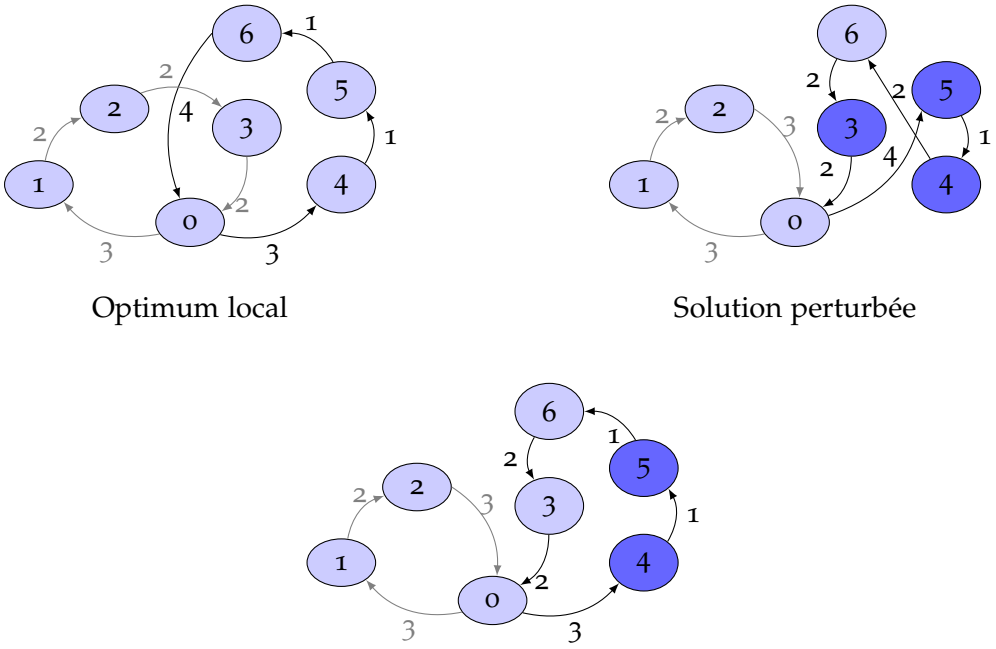
Le pseudo-code de l'heuristique hybride ILS_RVND est donné par l'Algorithme 3.

L'heuristique RVND utilisée et l'impact de l'utilisation d'ILS sur la performance d'RVND sont donnés dans ce qui suit.

**Descente à voisinage variable avec ordre aléatoire des voisinages**

L'heuristique RVND implémentée utilise tous les opérateurs de voisinage décrits dans la Section 2.2.2. Ces derniers sont d'abord mis dans une liste et à chaque itération

FIGURE 3.1 – *Illustration de l'impact de l'utilisation d'ILS sur la qualité de la solution générée par l'opérateur swap-intra.*




**Algorithme 3** ILS_RVND

```
1: Entrées:
       Une instance de CPTP
2: Sorties:
       La meilleure solution rencontrée
3: tq critère d'arrêt non atteint faire
4:     Générer une solution initiale avec l'heuristique de construction ;
5:     tq critère d'arrêt d'ILS non atteint faire
6:         Exécuter RVND ;
7:         Mettre à jour la meilleure solution ;
8:         Perturber la solution actuelle ;
9:     fin tq
10: fin tq
```

d'RVND, l'heuristique sélectionne de façon aléatoire le voisinage à utiliser. Le voisinage sélectionné est par la suite supprimé de la liste et la recherche continue de la même façon jusqu'à ce que la liste des opérateurs de voisinage devienne vide.

Le pseudo-code de l'heuristique RVND étudiée est donné par l'Algorithme 4.

### Impact de l'utilisation d'ILS sur la performance d'RVND

ILS permet à RVND d'avoir accès à une plus grande zone de l'espace de recherche. Pour montrer l'impact que peut avoir ILS sur l'heuristique RVND, supposons qu'on ait une instance avec 6 clients 1, 2, 3, 4, 5 et 6 et un dépôt 0. Supposons aussi que chaque client mis à par le client 6 possède un profit égal à 1, que le profit du client 6 est égal à 5, que les demandes des clients sont égales à 1, et que l'on possède une

**Algorithme 4** RVND

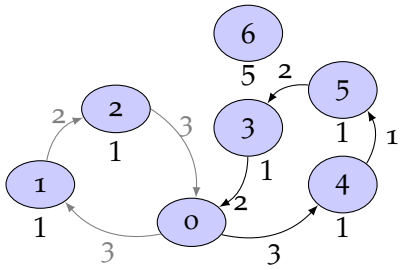
1: **Entrées:**
   Une solution initiale
   Une liste $L$ contenant les 7 opérateurs de voisinage
2: **Sorties:**
   La meilleure solution rencontrée
3: **tq** $L$ non vide **faire**
4: Choisir aléatoirement un opérateur $o$ dans $L$ ;
5: Appliquer $o$ à la solution actuelle ;
6: Mettre à jour la meilleure solution rencontrée ;
7: Supprimer $o$ de la liste $L$ ;
8: **fin tq**

flotte de 2 véhicules avec une capacité égale à 3 pour chaque véhicule. Ainsi, chaque véhicule contient au plus 3 clients. Supposons à présent que la solution initiale contient les clients 1, 2, 3, 4 et 5. Une solution d'RVND peut être une solution de très bonne qualité si l'ensemble des clients visités dans la solution initiale est bien choisi. Dans le cas contraire, RVND ne peut jamais atteindre une solution de bonne qualité. Cela est dû au fait que les opérateurs qui y sont utilisés ne prennent pas en considération les clients qui ne se trouvent pas déjà dans la solution.

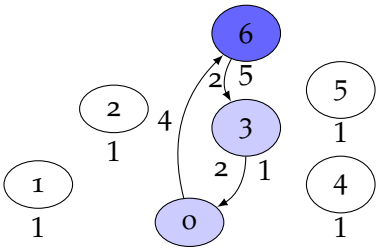
L'exemple donné peut être représenté par la Figure 3.2. À gauche, nous donnons une solution atteinte par RVND contenant les clients 1, 2, 3, 4 et 5. Les poids des arcs représentent les coûts de transport entre les sommets. Les poids en-dessous des sommets représentent les profits. Nous remarquons que le client 6 n'est pas visité dans la solution donnée et que la valeur de la fonction objectif pour cette solution est de $Z = 5 - 16 = -11$.

L'heuristique de perturbation d'ILS peut, par exemple, supprimer tous les clients de la solution sauf le client 3 et insérer le client 6, comme montré à droite de la Figure 3.2. La valeur de la fonction objectif pour la solution de la perturbation est de $Z = 6 - 8 = -2$. Cette solution est donc meilleure que celle d'RVND.

FIGURE 3.2 – *Illustration de l'impact de l'utilisation d'ILS sur la qualité de la solution générée par RVND.*




Optimum local d'RVND
$Z = 5 - 16 = -11$




Solution perturbée
$Z = 6 - 8 = -2$

### 3.3.4 Hybridation d'ILS avec LNS

Nous avons proposé une hybridation des heuristiques ILS et LNS. Pour cela, nous avons d'abord étudié l'heuristique LNS avec différents composants. Les meilleurs com-

posants d'LNS sont retenus lors de son hybridation avec ILS.

L'étude des composants d'LNS et l'hybridation d'LNS avec ILS sont données dans les sections suivantes.

**Recherches locales à grand voisinage**

Dans le présent travail, nous avons implémenté plusieurs heuristiques de *recherche locale à grand voisinage* (LNS). Ces dernières utilisent un seul opérateur de suppression et un seul opérateur d'insertion.

Nous avons implémenté tous les opérateurs présentés par Pisinger et Ropke [78] à l'exception de l'opérateur *time-oriented removal*, étant donné que le CPTP ne contient pas de contraintes de fenêtres de temps.

Les opérateurs ont été implémentés en utilisant la fonction objectif et les contraintes du CPTP (voir la Section 2.2.3 pour plus de détails concernant les opérateurs de Pisinger et Ropke [78]).

Chaque LNS implémentée possède un couple différent d'opérateurs de suppression/insertion. Chaque LNS implémentée commence la recherche depuis une solution générée par l'heuristique de construction. À chaque itération, LNS supprime un certain nombre de clients à l'aide de l'opérateur de suppression sélectionné, puis ré-insert des clients grâce à l'opérateur d'insertion sélectionné. Les clients insérés sont ceux dont l'insertion maximise la différence entre les profits collectés et les coûts de transport. LNS s'arrête lorsqu'un certain nombre d'itérations sans améliorations est atteint. Les heuristiques LNS implémentées peuvent être décrites par l'Algorithme 5.

**Algorithme 5** LNS

1: **Entrées:**
   Une solution initiale (valeurs aléatoires des paramètres)
   Un opérateur de suppression *sup*
   Un opérateur d'insertion *ins*
2: **Sorties:**
   La meilleure solution rencontrée
3: **tq** conditions d'arrêt non atteintes **faire**
4: Choisir le nombre de clients $r$ à supprimer ;
5: Supprimer $r$ clients de la solution actuelle avec l'opérateur *sup* ;
6: Insérer les clients les plus profitables avec l'opérateur *ins* ;
7: Mettre à jour la meilleure solution rencontrée ;
8: **fin tq**

Notons que pour le *related removal*, la similarité ou relatedness $s_{ij}$ entre les clients $i$ et $j$ (comme décrite dans la Section 2.2.3) devient :

$$s_{ij} = \left|pr_i - pr_j\right| + c_{ij} \tag{3.5}$$

où $c_{ij}$ est le coût de transport entre $i$ et $j$, $pr_i$ et $pr_j$ sont les profits de $i$ et $j$ respectivement. Ainsi, plus la valeur de $s_{ij}$ est petite, plus le client $j$ sera similaire au client $i$.

TABLE 3.1 – *Couples d'opérateurs de suppression/insertion.*

| ind | random | worst | related | historical node-pair | historical request-pair | cluster | basic greedy | regret-2 | regret-3 | regret-4 | regret-m |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | X | | | | | | X | | | | |
| 2 | X | | | | | | | X | | | |
| 3 | X | | | | | | | | X | | |
| 4 | X | | | | | | | | | X | |
| 5 | X | | | | | | | | | | X |
| 6 | | X | | | | | X | | | | |
| 7 | | X | | | | | | X | | | |
| 8 | | X | | | | | | | X | | |
| 9 | | X | | | | | | | | X | |
| 10 | | X | | | | | | | | | X |
| 11 | | | X | | | | X | | | | |
| 12 | | | X | | | | | X | | | |
| 13 | | | X | | | | | | X | | |
| 14 | | | X | | | | | | | X | |
| 15 | | | X | | | | | | | | X |
| 16 | | | | X | | | X | | | | |
| 17 | | | | X | | | | X | | | |
| 18 | | | | X | | | | | X | | |
| 19 | | | | X | | | | | | X | |
| 20 | | | | X | | | | | | | X |
| 21 | | | | | X | | X | | | | |
| 22 | | | | | X | | | X | | | |
| 23 | | | | | X | | | | X | | |
| 24 | | | | | X | | | | | X | |
| 25 | | | | | X | | | | | | X |
| 26 | | | | | | X | X | | | | |
| 27 | | | | | | X | | X | | | |
| 28 | | | | | | X | | | X | | |
| 29 | | | | | | X | | | | X | |
| 30 | | | | | | X | | | | | X |

La liste de toutes les combinaisons possibles d'opérateurs de suppression/insertion utilisés dans ce document est donnée dans le Tableau 3.1. Dans ce Tableau, *ind* désigne l'indice du couple d'opérateurs de suppression/insertion. Le symbol « X » indique si un opérateur est utilisé ou non.

### Hybridation des deux méthodes

L'heuristique hybride ILS_LNS est une heuristique multi-départs qui exécute, à chaque itération, une heuristique de type ILS moyennant une heuristique de type LNS dans la phase de recherche locale. Après chaque exécution d'LNS, ILS passe à la phase de perturbation de la solution. Cette perturbation est effectuée de la même façon que pour les heuristiques hybrides précédemment décrites.

ILS s'arrête lorsqu'un certain nombre d'itérations sans améliorations sont atteintes, et l'heuristique multi-départs s'arrête lorsque toutes les combinaisons possibles des valeurs de paramètres $\alpha_1$ et $\alpha_2$ sont testées. L'heuristique hybride ILS_LNS est décrite dans l'Algorithme 6.

**Algorithme 6** ILS_LNS

```
1: Entrées:
       Une instance de CPTP
2: Sorties:
       La meilleure solution rencontrée
3: tq critère d'arrêt non atteint faire
4:     Générer une solution initiale avec l'heuristique de construction;
5:     tq critère d'arrêt d'ILS non atteint faire
6:         Exécuter LNS;
7:         Mettre à jour la meilleure solution;
8:         Perturber la solution actuelle;
9:     fin tq
10: fin tq
```

Afin qu'LNS puisse être considérée comme une procédure de recherche locale, l'opérateur de suppression doit retirer à chaque itération un petit nombre de clients. La suppression d'un trop grand nombre de clients engendrerait un changement important dans la structure de la solution. Les mouvements d'LNS à chaque itérations seraient donc bien différents des mouvements élémentaires supposés être effectués dans une procédure de recherche locale traditionnelle.

Nous supposons que lorsque LNS est combinée à ILS, cela lui permet d'atteindre de meilleures solutions plus rapidement. Si LNS est exécutée seule avec un grand nombre de clients à supprimer, la recherche serait très diversifiée mais pas du tout intensifiée. Si LNS est exécutée seule avec un petit nombre de clients à supprimer, il n'y aurait pas assez de diversification. Par contre, lorsque ILS est combinée à LNS, LNS avec un petit nombre de clients à supprimer joue le rôle de la recherche locale et intensifie la recherche alors que ILS permet de diversifier la recherche en supprimant et/ou insérant un plus grand nombre de client dans la solution et en relançant la recherche locale avec LNS.

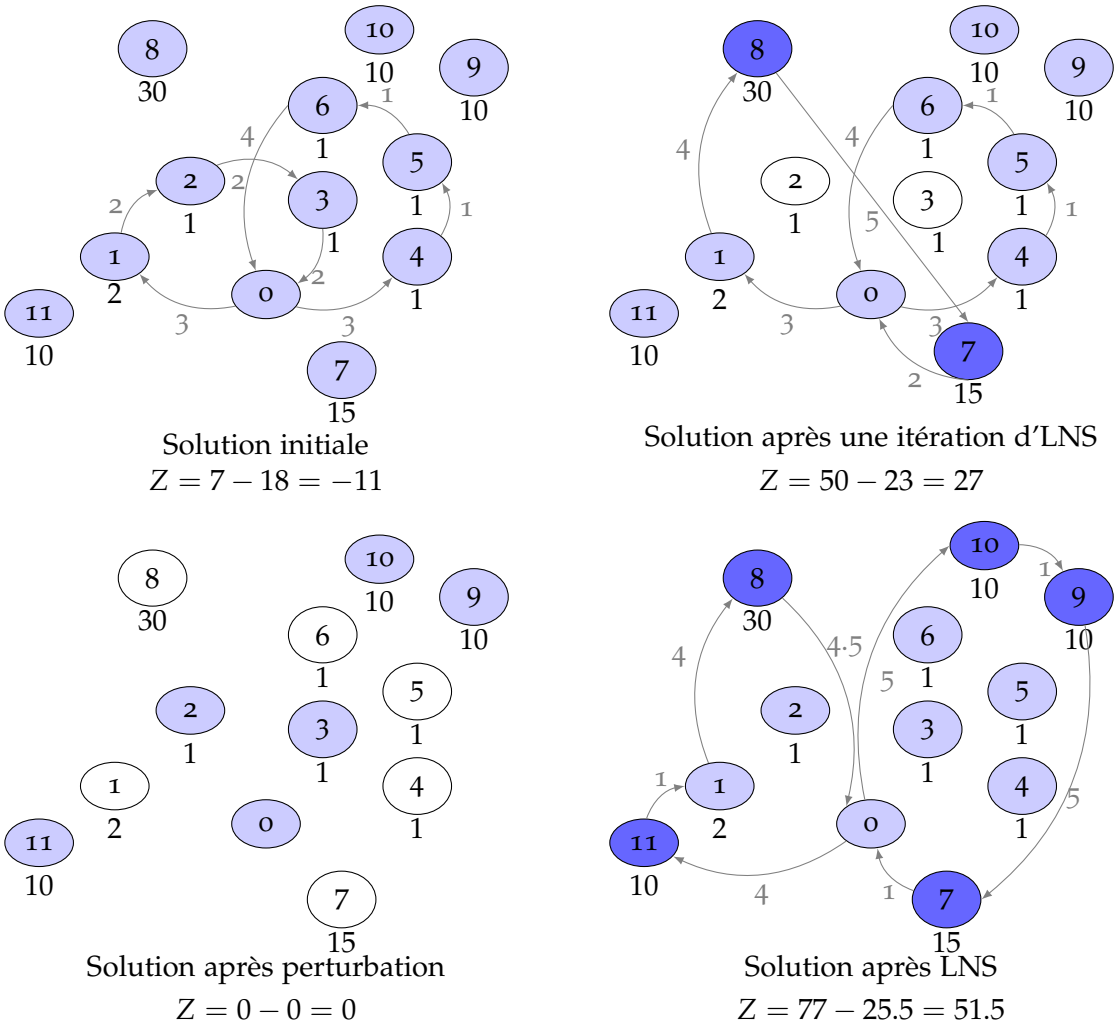
La Figure 3.3 illustre une situation où ILS est combinée à LNS avec un petit nombre

de clients. Dans cette Figure, nous supposons que les demandes des clients sont toutes égales à 1, que la flotte est composée de 2 véhicules et que la capacité de chaque véhicule est égale à 3. Chaque route peut donc au plus contenir 3 clients. Supposons également que le nombre de clients à supprimer dans LNS est égal à 2.

LNS supprime donc deux clients (clients 2 et 3 dans la Figure) et en insert deux autres (clients 7 et 8 dans la Figure). LNS doit effectuer plusieurs itérations afin d'arriver (éventuellement) à la solution en bas à droite de la Figure étant donné que l'heuristique ne peut supprimer plus de deux clients à la fois.

Par contre, lorsque ILS est utilisée, celle-ci permet de supprimer tous les clients de la solution grâce à la procédure de perturbation. Cela facilite l'insertion des clients profitables par LNS, ce qui peut se faire très rapidement.

FIGURE 3.3 – *Illustration de l'impact de l'utilisation d'ILS sur la qualité de la solution générée par LNS.*




### 3.3.5 Hybridation d'ILS avec LNS et un opérateur de voisinage

Dans cette heuristique nous considérons que la recherche locale d'ILS est effectuée à l'aide d'une combinaison d'LNS utilisant le meilleur couple d'opérateurs de suppres-

sion/insertion avec l'opérateur de voisinage 2-Opt*.

Nous avons choisi d'utiliser l'opérateur 2-Opt* car c'est celui qui donne les meilleurs résultats pour le problème traité lorsqu'il est utilisé au sein d'ILS, en comparaison avec les autres opérateurs de voisinage définis dans la Section 3.3.2 (voir la Section 3.4.2 pour plus de détails concernant les résultats).

Le pseudo-code d'ILS_2-Opt*_LNS est donné par l'Algorithme 7.

**Algorithme 7** ILS_2-Opt*_LNS

```
1: Entrées:
       Une instance de CPTP
       Un opérateur de suppression sup
       Un opérateur d'insertion ins
2: Sorties:
       La meilleure solution rencontrée
3: tq critère d'arrêt non atteint faire
4:     Générer une solution initiale avec l'heuristique de construction;
5:     tq conditions d'arrêt d'ILS non atteintes faire
6:         tq conditions d'arrêt de 2-Opt*_LNS non atteintes faire
7:             Exécuter LNS avec le couple d'opérateurs de suppression/insertion;
8:             Exécuter l'opérateur de voisinage 2-Opt*;
9:         fin tq
10:        Mettre à jour la meilleure solution;
11:        Perturber la solution actuelle;
12:    fin tq
13: fin tq
```

### 3.3.6 Hybridation d'ILS avec RVND et LNS

De la même façon que pour ILS_2-Opt*_LNS, nous avons hybridé ILS avec une heuristique combinant RVND et LNS. Cette dernière est définie dans ce qui suit.

#### Hybridation d'LNS avec RVND

L'heuristique hybride proposée ici est notée LNS_RVND. Cette heuristique commence à partir d'une unique solution initiale obtenue à l'aide de l'heuristique de construction avec des valeurs aléatoires de $\alpha_1$ et $\alpha_2$. Après qu'une solution initiale soit construite, l'heuristique LNS est exécutée pour un certain nombre d'itérations. Ensuite, la solution obtenue est améliorée, suivant une certaine probabilité, à l'aide de l'heuristique RVND décrite dans la Section 3.3.3. Par la suite, l'heuristique retourne encore à LNS et le processus est répété jusqu'à atteinte d'un certain nombre d'itérations sans améliorations. Le pseudo-code d'LNS_RVND est donné par l'Algorithme 8.

#### Hybridation d'ILS avec LNS_RVND

L'approche proposée ici combine ILS, LNS et RVND dans une heuristique multi-départs notée ILS_RVND_LNS. Tout d'abord, l'heuristique de construction est exécutée afin de générer différentes solutions initiales à chaque itération. Ensuite, pour chaque solution initiale, une heuristique de type ILS est exécutée jusqu'à l'atteinte d'un certain nombre d'itérations sans améliorations. Dans ILS, une heuristique de type LNS est

**Algorithme 8** LNS_RVND

```
1: Entrées:
       Une solution initiale
       Un opérateur de suppression sup
       Un opérateur d'insertion ins
2: Sorties:
       La meilleure solution rencontrée
3: tq conditions d'arrêt non atteintes faire
4:     Exécuter LNS avec le couple d'opérateurs de suppression/insertion;
5:     Exécuter RVND avec une certaine probabilité;
6: fin tq
```

exécutée suivie d'une heuristique de type RVND. Les deux heuristiques combinées jouent le rôle de procédure de recherche locale dans ILS. Cette combinaison d'LNS et RVND est répétée un certain nombre de fois. Puis, la procédure de perturbation est lancée. Cette dernière est la même que celle décrite dans la Section 3.3.2. Le pseudo-code d'ILS_RVND_LNS est donné par l'Algorithme 9.

**Algorithme 9** ILS_RVND_LNS

```
1: Entrées:
       Une instance de CPTP
2: Sorties:
       La meilleure solution rencontrée
3: tq critère d'arrêt non atteint faire
4:     Générer une solution initiale;
5:     tq critère d'arrêt d'ILS non atteint faire
6:         tq critère d'arrêt d'LNS+RVND combinés non atteint faire
7:             Lancer LNS;
8:             Lancer RVND;
9:         fin tq
10:        Mettre à jour la meilleure solution;
11:        Perturber la solution actuelle;
12:    fin tq
13: fin tq
```

## 3.4 Étude expérimentale

Dans cette Section, nous commencerons par présenter les instances du CPTP. Puis, nous évaluerons la performance des méthodes proposées en les comparant entre elles sur les instances étudiées. Nous montrerons également l'importance de l'hybridation. Après cela, nous choisirons la méthode qui donne les meilleurs résultats et nous la comparerons avec quelques approches proposées dans la littérature pour le même problème.

Dans le but de déterminer rapidement la meilleure méthode implémentée, nous exécutons les méthodes proposées avec un nombre d'itérations relativement petit. Cependant, un plus grand nombre d'itérations est utilisé lors de la comparaison de la meilleure méthode avec celles de la littérature.

Toutes les méthodes sont implémentées en C et exécutées sur un ordinateur de type intel(R) Core (TM) i7-4600U CPU @ 2.10GHz 2.70GHz avec 8.00 Gb de RAM et 64-bit. Étant donné l'aspect aléatoire des méthodes, celles-ci sont exécutées 3 fois pour chaque instance du problème. Les meilleures solutions rencontrées, en termes de

meilleur pourcentage de déviation par rapport aux méthodes d'Archetti et al. [2] sur les instances du CPTP, sont présentées. Un **pourcentage de déviation (gap)** d'une méthode $a$ par rapport à une méthode $b$ est calculé suivant l'Expression $gap = 100 \cdot \frac{z_b - z_a}{z_b}$, où $z_a$ et $z_b$ sont les valeurs de la fonction objectif obtenues avec les méthodes $a$ et $b$ respectivement.

### 3.4.1 Instances du CPTP

Les instances du CPTP ont été proposées par Archetti et al. [2], en modifiant les instances du CVRP proposées par Christofides et al. [21]. Le nombre de clients dans les instances du CPTP appartient à l'ensemble $\{50, 75, 100, 120, 150, 199\}$. Les instances de CPTP proposées par Archetti et al. [2] sont divisées en trois ensembles décrits dans ce qui suit.

Le premier ensemble contient les 10 instances proposées par Christofides et al. [21] avec les mêmes capacités et nombres de véhicules. Le profit $pr_i$ d'un client $i$ est généré depuis les instances de Christofides et al. [21] selon l'Expression $pr_i = (0.5 + h) \cdot d_i$, où $d_i$ est la demande du client $i$ et $h$ est aléatoirement choisi depuis l'intervalle $[0, 1]$.

Dans le second ensemble, pour chaque instance présentée par Christofides et al. [21], les cas de $Q = 50$, $Q = 75$ et $Q = 100$ sont considérés, où $Q$ représente la capacité des véhicules. Pour chaque cas, trois instances sont générées avec différents nombres de véhicules. Ces derniers prennent leurs valeurs dans l'ensemble $\{2, 3, 4\}$. Les profits du second ensemble sont générés de la même façon que pour le premier ensemble. Ainsi, un total de 90 instances sont considérées dans le second ensemble.

Enfin, dans le troisième ensemble, les capacités des véhicules proposées par Christofides et al. [21] sont maintenues. Cependant, le nombre de véhicules varie dans l'ensemble $\{2, 3, 4\}$. Les profits des clients dans le troisième ensemble sont générés de la même façon que pour le premier ensemble. Ainsi, le troisième ensemble contient 30 instances. Donc, un total de 130 instances du CPTP sont utilisées par Archetti et al. [2].

Dans leur travail Archetti et al. [2] ont étudiés deux problèmes : le CPTP et le *problème de la course d'orientation avec contraintes de capacité* (Capacitated Team Orienteering Problem en anglais ou CTOP). Dans le CTOP, il existe certaines contraintes de durée maximum pour chaque tournée. Les instances du CPTP et du CTOP proposées par les auteurs sont presque identiques. La différence entre les deux est que dans les instances du CTOP une limite de durée des tournées de véhicules est introduite. Les instances de types "3" et "8" proposées pour le CTOP se différencient les unes des autres justement par la valeurs attribuées à cette limite de durée des tournées. Concernant le CPTP, comme les contraintes de durées maximum des tournées ne sont pas considérées, les instances de types "3" et "8" sont exactement les mêmes.

Dans ce qui suit, nous ne prenons en considération que les instances de type "8". Nous obtenons un total de 117 instances.

### 3.4.2 Étude de l'heuristique ILS avec un opérateur de voisinage

Les heuristiques combinant ILS à un opérateur de voisinage sont très rapides. Pour cette raison, nous avons choisi de fixer le nombre d'itérations sans améliorations d'ILS à 500 au lieu de 50.

Le Tableau 3.2 affiche les résultats obtenus en termes de pourcentage de déviation moyen (sur toutes les instances) par rapport aux méthodes d'Archetti et al. [2] (noté gap) et en termes de temps de calcul (**CPU**) en secondes.

TABLE 3.2 – *Comparaison entre les combinaisons d'ILS avec les différents opérateurs de voisinage.*

| | ILS_2-Opt | ILS_2-Opt* | ILS_relocate-intra | ILS_relocate-Inter | ILS_Or_Opt | ILS_switch-intra | ILS_switch_inter |
|---|---|---|---|---|---|---|---|
| gap | 5.41 | **4.94** | 5.78 | 5.42 | 5.76 | 5.52 | 5.07 |
| CPU | 15.70 | 14.98 | 15.72 | 16.42 | **14.58** | 15.15 | 15.92 |

Les résultats affichés dans le Tableau 3.2 montrent que l'opérateur 2-Opt* est celui qui donne, en combinaison avec ILS, le meilleur résultat en termes de qualité de la solution. De plus le temps de calcul de la version ILS_2-Opt* est le second meilleur temps par rapport aux autres versions.

### 3.4.3 Étude de l'heuristique ILS_RVND

Tout comme les heuristiques combinant ILS avec un opérateur de voisinage, l'heuristique ILS_RVND est très rapide. En plus d'avoir lancée une version d'ILS_RVND avec 50 itérations sans améliorations, et afin d'avoir des résultats comparables en termes de temps de calcul à ceux déjà obtenus par la combinaison d'ILS avec les opérateurs de voisinages, nous testons une version d'ILS_RVND avec un nombre d'itérations sans améliorations égal à 500 dans ILS.

Le Tableau 3.3 résume les résultats obtenus par ILS_RVND en termes de gap moyen et de temps de calcul en seconde. Ces résultats sont comparés avec ceux de la meilleure version d'ILS obtenue dans la Section précédente.

La comparaison entre les deux versions d'ILS_RVND montre que l'utilisation d'un plus grand nombre d'itération conduit à de meilleurs résultats. Nous remarquons que le temps de calcul de la version avec 50 itérations est beaucoup plus petit. Le temps de calcul de la seconde version reste malgré cela raisonnable.

La comparaison entre ILS_RVND et ILS_2-Opt* montre que les deux heuristiques donnent des résultats assez similaires lorsqu'elles sont exécutées avec le même nombre d'itérations sans amélioration. Nous supposons que la différence entre ces résultats peut être plus évidente si le nombre d'itérations des deux heuristiques augmente et/ou si elles sont hybridées avec d'autres méthodes.

TABLE 3.3 – *Comparaison entre ILS_2-Opt* et les deux versions d'ILS_RVND.*

| | ILS_2-Opt*<br>500 itérations | ILS_RVND<br>50 itérations | ILS_RVND<br>500 itérations |
|---|---|---|---|
| gap | **4.94** | 7.59 | 4.95 |
| CPU | 14.98 | **1.26** | 10.57 |

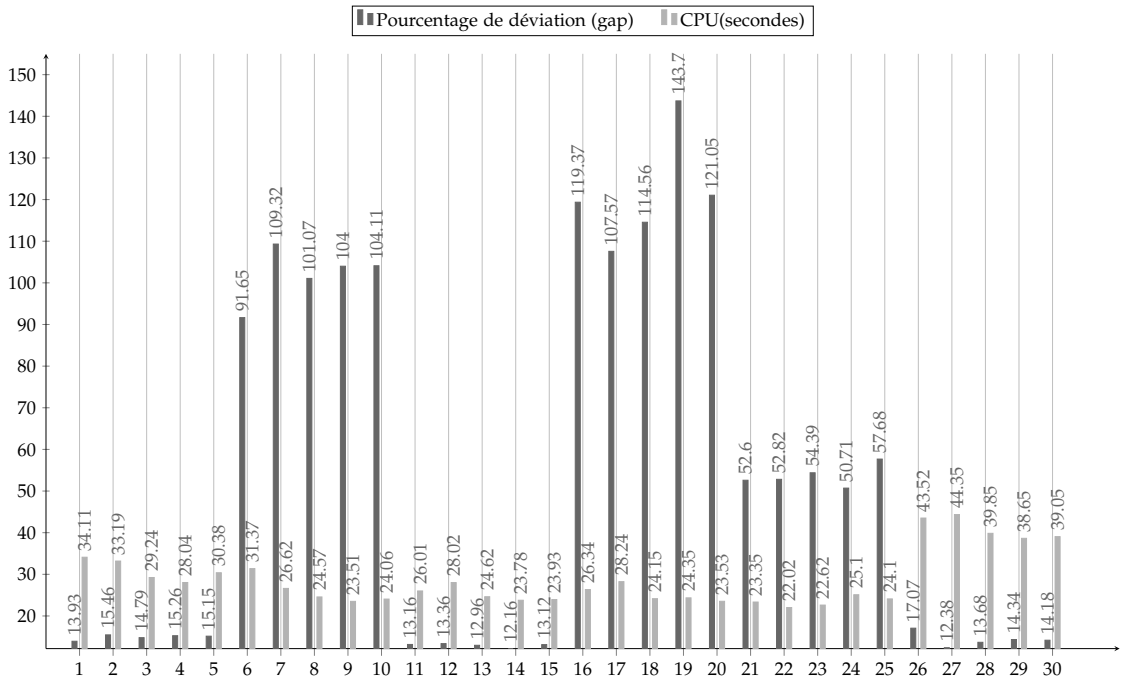
Notons que l'heuristique RVND n'est pas testée seule (sans ILS) étant donné que cette heuristique est incapable de changer par elle-même l'ensemble des clients visités. En effet, l'ensemble des clients d'une solution d'RVND est le même que celui donné par l'heuristique de construction. RVND seule ne peut donc être considérée comme une heuristique adaptée au CPTP.

### 3.4.4 Étude de l'heuristique LNS

Comme mentionné précédemment, nous avons implémenté la plupart des opérateurs de suppression/insertion présentés par Pisinger et Ropke [78] en utilisant la fonction objectif du CPTP. 30 versions d'LNS ont été testées, en combinant les opérateurs de suppression/insertion implémentés. Les LNSs implémentées suppriment entre 1 et 3 clients à chaque itération. Chaque heuristique de type LNS s'arrête lorsque 50000 itérations sans améliorations sont atteintes.

La Figure 3.4 compare le pourcentage de déviation moyen (par rapport aux solutions présentées par Archetti et al. [2]) des 30 versions étudiées. À partir de cette Figure, nous remarquons que la combinaison du *related removal* et de l'opérateur de *regret* utilisant un niveau de regret égal à 4, donne les meilleurs résultats en termes de qualité de la solution (voir version 14). Cette combinaison se termine également en un temps raisonnable. Dans ce qui suit, « LNS » fera référence à cette version.

FIGURE 3.4 – *Comparaison entre les différents couples de suppression/insertion de l'heuristique LNS.*




Le Tableau 3.4 compare les résultats d'LNS avec ceux présentés dans les Sections précédente. Nous remarquons qu'LNS donne de moins bons pourcentages de déviation par rapport à ILS_2-Opt* et ILS_RVND.

TABLE 3.4 – *Comparaison entre ILS_2-Opt*, les deux versions d'ILS_RVND et LNS.*

| | ILS_2-Opt* 500 itérations | ILS_RVND 50 itérations | ILS_RVND 500 itérations | LNS |
|---|---|---|---|---|
| gap | **4.94** | 7.59 | 4.95 | 12.16 |
| CPU | 14.98 | **1.26** | 10.57 | 23.78 |

### 3.4.5 Étude de l'heuristique ILS_LNS

L'LNS implémentée dans l'heuristique hybride ILS_LNS est celle qui utilise le meilleur couple d'opérateurs de suppression/insertion trouvé dans la Section 3.4.4.

Nous avons testé 2 configurations du nombre de clients à supprimer lors de chaque itération. Dans la première configuration, nous considérons LNS comme une procédure de recherche locale simple en supprimant un petit nombre de clients à chaque fois. Plus précisément, nous choisissons à chaque itération et de façon aléatoire le nombre $r$ de clients à supprimer depuis l'intervalle $[1, 3]$. Dans la seconde configuration, $r$ est choisi de façon aléatoire depuis l'intervalle $[1, 0.4 \cdot n^{'}]$, où $n^{'}$ est le nombre de clients dans la solution actuelle. LNS s'arrête lorsque 50 itérations sans améliorations sont atteintes.

À chaque appel de la procédure de perturbation, $r'$ clients sont supprimés, où $r'$ est choisi aléatoirement dans l'intervalle $[0.1 \cdot n^{'}, 0.4 \cdot n^{'}]$ et $n^{'}$ est le nombre de client appartenant à la solution actuelle.

ILS_LNS s'arrête lorsque 50 itérations sans améliorations sont atteintes.

Le Tableau 3.5, compare les résultats des méthodes précédemment testées avec ceux des deux configurations d'ILS_LNS. Dans ce Tableau, gap représente le pourcentage de déviation par rapport aux instances d'Archetti et al. [2] et CPU est le temps de calcul en secondes. $r \in [1, 3]$ signifie que le nombre $r$ de clients à supprimer est aléatoirement choisi dans l'intervalle $[1, 3]$. Tandis que, $r \in [1, 0.4 * n^{'}]$ signifie que $r$ est aléatoirement choisi dans l'intervalle $[1, 0.4 * n^{'}]$. Nous remarquons que la première configuration d'ILS_LNS fournit de meilleurs résultats et a besoin d'un temps de calcul inférieur à celui d'LNS. Lorsque $r$ est choisi dans $[1, 0.4 * n^{'}]$, l'heuristique donne de meilleurs solutions. Cependant, elle a besoin d'un plus grand temps de calcul.

TABLE 3.5 – *Comparaison entre les deux configurations d'ILS_LNS et les heuristiques précédemment étudiées.*

| | ILS_2-Opt* 500 itérations | ILS_RVND 50 itérations | ILS_RVND 500 itérations | LNS | ILS_LNS $r \in [1, 3]$ | ILS_LNS $r \in [1, 0.4 * n^{'}]$ |
|---|---|---|---|---|---|---|
| gap | **4.94** | 7.59 | 4.95 | 12.16 | 5.61 | 5.07 |
| CPU | 14.98 | **1.26** | 10.57 | 23.78 | 14.75 | 36.35 |

Nous remarquons qu'ILS_2-Opt* et ILS_RVND avec 500 itérations sans améliorations donnent de meilleurs résultats que les deux versions d'ILS_LNS en termes de pourcentage de déviation. Le temps de calcul d'ILS_LNS avec $r \in [1, 3]$ est meilleur que le temps de calcul d'ILS_2-Opt*. Cependant, le temps de calcul d'ILS_LNS avec $r \in [1, 0.4 * n^{'}]$ est plus important que celui d'ILS_2-Opt* et ILS_RVND.

Concernant ILS_RVND avec 50 itérations sans améliorations, nous remarquons que le pourcentage de déviation de cette heuristique par rapport aux méthodes d'Archetti

et al. [2] est de 7.59%, ce qui est plus important que le pourcentage de déviation des deux versions d'ILS_LNS. Le temps de calcul d'ILS_RVND avec 50 itérations sans améliorations est quant à lui inférieur à celui des deux versions d'ILS_LNS.

Comme le temps de calcul des deux versions d'ILS_LNS reste raisonnable, nous choisissons de continuer l'étude en utilisant l'ILS_LNS avec $r \in [1, 0.4 * n^{'}]$.

### 3.4.6 Étude de l'heuristique LNS_RVND

LNS_RVND est une heuristique qui combine la meilleure version d'LNS trouvée dans la Section 3.4.4 avec l'heuristique RVND. L'heuristique LNS est exécutée, et à chaque fois qu'$i$ itérations sont atteintes, la solution actuelle est améliorée (si possible) avec RVND utilisant une certaine probabilité $P$. Nous avons étudié les configurations suivantes ($i = 100$, $P = 1/10$), ($i = 1000$, $P = 1/10$) et ($i = 1000$, $P = 1/100$). Nous avons remarqué que la seconde configuration fournit les meilleurs résultats en termes de pourcentage de déviation et de temps de calcul. Cependant, ces résultats n'ont pas été motivants en comparaison avec ceux d'ILS_LNS, ILS_RVND et ILS_2-Opt*, comme montré dans le Tableau 3.6. Dans ce Tableau, le pourcentage moyen de déviation (gap) par rapport aux méthodes d'Archetti et al. [2] est considéré ainsi que le temps de calcul en secondes (CPU). Nous concluons que l'heuristique ILS multi-départ joue un grand rôle pour l'atteinte de solutions de bonne qualité en un temps raisonnable.

Table 3.6 – *Comparaison entre LNS_RVND et les autres heuristiques précédemment étudiées.*

| | ILS_2-Opt* 500 itérations | ILS_RVND 500 itérations | LNS | ILS_LNS $r \in [1, 0.4 * n^{'}]$ | LNS_RVND |
|---|---|---|---|---|---|
| gap | **4.94** | 4.95 | 12.16 | 5.07 | 11.94 |
| CPU | 14.98 | **10.57** | 23.78 | 36.35 | 67.10 |

### 3.4.7 Étude de l'heuristique ILS_2-Opt*_LNS

ILS_2-Opt*_LNS est une heuristique multi-départs de type ILS qui utilise l'heuristique ILS avec une combinaison d'LNS et de 2-Opt* comme procédure de recherche locale. Dans ILS_2-Opt*_LNS, la combinaison d'LNS et de 2-Opt* est répétée jusqu'à l'atteinte d'un nombre $i_{LNS\_2\text{-}Opt*}$ d'itérations. Ce nombre est fixé à 7. Le nombre d'itérations sans améliorations $maxOcc$ d'ILS_2-Opt*_LNS est fixé à 200. Le nombre $maxOcc_{LNS}$ d'itérations sans améliorations dans LNS est fixé à 20.

Le Tableau 3.7 compare les résultats d'ILS_2-Opt*_LNS avec ceux des heuristiques précédemment étudiées. Nous remarquons qu'ILS_2-Opt*_LNS est capable de fournir de bien meilleurs résultats que les autres heuristiques en termes de pourcentage de déviation moyen (gap) par rapport aux instances d'Archetti et al. [2]. Cependant, le temps de calcul en secondes (CPU) de cette heuristique est supérieur aux temps de calcul des autres méthodes.

TABLE 3.7 – *Comparaison entre ILS_2-Opt*_LNS et les heuristiques précédemment étudiées.*

| | ILS_2-Opt* 500 itérations | ILS_RVND 500 itérations | LNS | ILS_LNS $r \in [1, 0.4 * n^{'}]$ | LNS_RVND | ILS_2-Opt*_LNS |
|---|---|---|---|---|---|---|
| gap | 4.94 | 4.95 | 12.16 | 5.07 | 11.94 | **1.72** |
| CPU | 14.98 | **10.57** | 23.78 | 36.35 | 67.10 | 36.64 |

### 3.4.8 Étude de l'heuristique ILS_RVND_LNS

Cette heuristique est la même que la précédente à la seule différence que dans ILS_RVND_LNS, l'heuristique RVND est utilisée au lieu de l'opérateur 2-Opt*.

Notons que nous avons choisi d'arrêter la combinaison d'LNS et d'RVND lorsqu'un certain nombre d'itérations est atteint au lieu d'un certain nombre d'itérations sans améliorations car cela mène à de meilleurs résultats.

Remarquons aussi que, contrairement à l'ILS_RVND étudiée plus haut, cette combinaison d'ILS et d'RVND utilise RVND avec une probabilité égale à 1. Ce changement a été motivé par le fait qu'ILS_RVND ait donné des résultats de bonne qualité par rapport aux autres heuristiques testées utilisant LNS. C'est notamment le cas pour LNS_RVND.

Le Tableau 3.8 compare les résultats d'ILS_RVND_LNS avec ceux des heuristiques précédemment étudiées. Depuis ce Tableau, nous remarquons qu'ILS_RVND_LNS obtient les meilleurs résultats en un temps raisonnable. Nous remarquons également que plus l'heuristique ILS est hybridée, et plus les résultats sont meilleurs. Cependant, cela peut engendrer un temps de calcul additionnel étant donné que si l'on n'hybride pas une heuristique, elle sera bloquée dans un optimum local très rapidement.

TABLE 3.8 – *Comparaison entre ILS_RVND_LNS et les heuristiques précédemment étudiées.*

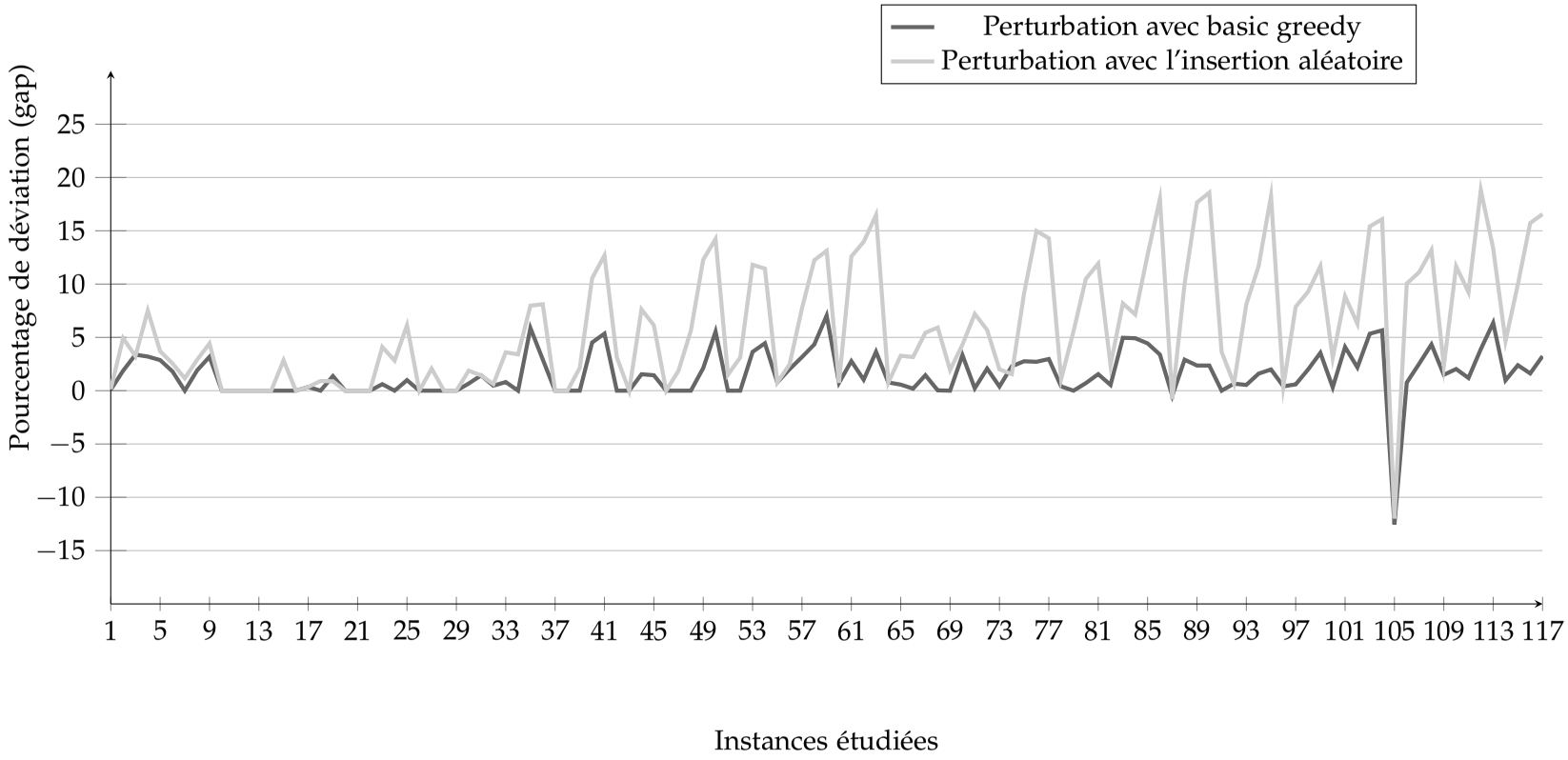
| | ILS_2-Opt* 500 itérations | ILS_RVND 500 itérations | LNS | ILS_LNS $r \in [1, 0.4 * n^{'}]$ | LNS_RVND | ILS_2-Opt*_LNS | ILS_RVND_LNS |
|---|---|---|---|---|---|---|---|
| gap | 4.94 | 4.95 | 12.16 | 5.07 | 11.94 | 1.72 | **1.57** |
| CPU | 14.98 | **10.57** | 23.78 | 36.35 | 67.10 | 36.64 | 28.39 |

En comparant ILS_2-Opt*_LNS avec ILS_RVND_LNS, nous remarquons que ILS_RVND_LNS donne de meilleurs résultats en un meilleur temps de calcul. Cela confirme donc notre supposition que la combinaison d'ILS avec RVND au lieu d'un opérateur de voisinage donne de meilleurs résultats lorsque les deux heuristiques (ILS_RVND et ILS_2-Opt*) sont utilisées avec un plus haut degré d'hybridation et/ou un plus grand nombre d'itérations. Cela est dû au plus grand espace de recherche exploré lorsque plusieurs opérateurs sont utilisés au lieu d'un seul.

### 3.4.9 Procédure de perturbation

Les heuristiques ILS étudiées utilisent une insertion non-aléatoire des clients lors de la phase de perturbation. Le choix d'une telle insertion au lieu d'une insertion aléatoire est dû au fait que l'insertion aléatoire donne de moins bons résultats. La Figure 3.5 justifie cette affirmation. Dans cette Figure, deux versions d'ILS_RVND_LNS avec différents types de perturbation sont présentées.

FIGURE 3.5 – *Comparaison entre l'utilisation de l'opérateur basic greedy et l'insertion aléatoire pour la procédure de perturbation.*



Dans la première version, la perturbation utilise l'opérateur *basic greedy* lors de l'insertion des clients alors que dans la seconde version, une perturbation avec une insertion aléatoire est utilisée. Les deux versions sont évaluées selon leurs pourcentages de déviation par rapport aux solutions d'Archetti et al. [2]. Comme nous pouvons le constater, l'utilisation de l'opérateur *basic greedy* donne, pour la plupart des instances, de meilleurs résultats que l'utilisation de l'insertion aléatoire.

### 3.4.10 Comparaison d'ILS_RVND_LNS avec d'autres méthodes de la littérature

Étant donné qu'ILS_RVND_LNS est la méthode qui donne les meilleurs résultats parmi les approches implémentées, c'est cette méthode qui sera évaluée par rapport aux méthodes d'Archetti et al. [2].

Afin d'avoir des résultats plus ou moins comparables en termes de temps de calcul par rapport aux méthodes d'Archetti et al. [2], ILS_RVND_LNS telle que décrite dans la Section 3.4.8, est exécutée 20 fois.

Le Tableau 3.9 donne une synthèse des résultats obtenus par ILS_RVND_LNS comparés à ceux des méthodes VNS, TF et TA d'Archetti et al. [2]. Dans ce Tableau, *gap* représente le pourcentage de déviation de chaque heuristique par rapport à la meilleure solution obtenue parmi les solutions d'ILS_RVND_LNS, VNS, TF et TA. *CPU(min)* désigne le temps de calcul de chaque heuristique (en minutes).

Nous remarquons qu'ILS_RVND_LNS fournit des résultats compétitifs par rapport aux autres méthodes que ce soit en termes de qualité des solutions ou en termes de temps de calcul. Les résultats détaillés des quatre heuristiques sont donnés en Annexe (voir A.1).

Les résultats détaillés des quatre méthodes montrent qu'ILS_RVND_LNS parvient à obtenir 6 meilleures solutions par rapport aux méthodes d'Archetti et al. [2]. Cependant ILS_RVND_LNS reste incapable d'atteindre certaines solutions de la littérature.

Nous remarquons également que VNS est la méthode qui obtient le meilleur gap. Cependant son temps de calcul reste légèrement supérieur à celui d'ILS_RVND_LNS.

Table 3.9 – *Comparaison entre ILS_RVND_LNS et les méthodes d'Archetti et al. [2].*

| | VNS | TF | TA | ILS_RVND_LNS |
|---|---|---|---|---|
| gap | **0.18** | 0.78 | 0.73 | 0.66 |
| CPU(min) | 10.3 | **2.83** | 8.54 | 9.94 |

## Conclusion

Dans ce Chapitre, nous avons étudié le *problème de la tournée la plus profitable avec contrainte de capacité*. Nous avons proposé plusieurs méthodes pour ce problème. Ces méthodes comprennent l'heuristique ILS, l'heuristique LNS et l'heuristique RVND. Nous avons montré, pour les méthodes implémentées, l'utilité des procédure multi-départs. Nous avons également montré, pour ces méthodes, que l'hybridation mène

à de meilleurs résultats. Cependant le temps de calcul des méthodes hybrides est relativement élevé en comparaison avec les méthodes élémentaires. Enfin, nous avons comparé la meilleure méthode obtenue à celles présentées par Archetti et al. [2]. Les résultats ont montré que l'heuristique proposée est compétitive et que ses résultats sont fournis en un temps raisonnable.

L'étude présentée dans ce Chapitre a fait l'objet d'un article de conférence [20] et d'un chapitre de livre [18].

# 4 Méthodes approchées pour le problème de la tournée la plus profitable avec ramassages et livraisons simultanés

Sommaire



Le problème étudié dans ce Chapitre est une variante du *problème de tournées de véhicules avec ramassages et livraisons simultanés*. Dans cette variante, les véhicules ne sont plus contraints de passer par tous les clients et la visite de chaque client engendre un certain profit. À notre connaissance, c'est la première fois que ce problème est abordé dans la littérature. Dans ce qui suit, nous donnerons donc une description détaillée du nouveau problème. Nous expliquerons aussi les raisons qui nous ont poussés à le traiter en soulignant ses principales différences avec les autres problèmes de la littérature. Nous présenterons par la suite, une formulation mathématique. Puis, nous exposerons la méthode que nous avons proposée et comparerons ses résultats avec *i*) ceux du solveur CPLEX en introduisant le modèle mathématique proposé, *ii*) ceux de différentes

versions de la méthode proposée et *iii*) ceux d'autres versions de la méthode inspirées des approches de la littérature. Nous évaluerons, après cela, la méthode proposée sur d'autres problèmes de la littérature. Nous terminerons le Chapitre en donnant une conclusion ainsi que quelques perspectives de recherche.

## Introduction

Il y a aujourd'hui de plus en plus de problèmes environnementaux causés par les activités industrielles. Afin de lutter contre ce genre de problèmes, plusieurs pays à travers le monde se sont engagés à fournir différentes stratégies de protection environnementale. Parmi ces stratégies, de nouvelles lois ont été considérées, imposant une meilleure gestion des déchets par les entreprises. En effet, dans plusieurs pays, les entreprises de production se retrouvent contraintes d'analyser, de suivre et d'entretenir les cycles de vie de leurs produits. Ainsi, au lieu d'être jetées, les palettes de manutention par exemple (servant à optimiser la distribution, le chargement et le stockage de produits) sont récupérées puis réutilisées.

Étant donné que les coûts de transport augmentent lorsque l'on traite indépendamment le cycle de distribution des produits et celui de la gestion des déchets, les entreprises se voient obligées d'optimiser leurs schémas de transport en y incluant les deux types d'opérations. Cela engendre donc un unique schéma avec des livraisons et des collectes (ou ramassage).

Souvent, les clients approvisionnés imposent aux transporteurs de les visiter une unique fois. Autrement dit, ils imposent aux distributeurs d'effectuer les opérations de livraison et de collecte simultanément.

Dans la littérature le problème de collecte et de livraison de produits simultanément porte le nom de *problème de tournées de véhicules avec ramassages et livraisons simultanés* (Vehicle Routing Problem with Simutlaneous Pick-up and Delivery en anglais ou VRPSPD).

Le VRPSPD est un problème qui a été très étudié et ce, depuis 1989. Cependant, dans toutes les versions traitées, les auteurs n'ont considéré que le cas où la flotte de véhicules utilisée pour le transport est illimitée. Cela signifie que l'on peut utiliser autant de véhicules que nécessaire pour satisfaire les demandes des clients. En pratique, ce n'est généralement pas possible vu que l'acquisition d'une telle flotte engendrent des coûts très élevés.

D'autre part, si l'on fixe le nombre de véhicules à utiliser dès le départ, il serait NP-Difficile de trouver une solution réalisable pour le problème comme l'ont affirmé Bianchessi et Righini [13].

Pour faire face à ce problème, nous proposons donc de créer une nouvelle variante où le nombre de véhicules utilisés est fixé mais dans laquelle on ne serait pas obligé de visiter tous les clients. En effet, l'entreprise peut affecter à chaque client un certain profit et l'objectif sera formulé de sorte à ce qu'on maximise la différence entre la somme des profits des clients visités et la distance totale parcourue par les véhicules. On peut noter que dans le cas où tous les clients possèdent le même profit et où le nombre de véhicules est assez grand pour desservir tous les clients on peut alors tomber sur un problème équivalent au VRPSPD. Il faudra alors que ce profit soit assez grand par rapport à la somme de toutes les distances entre tous les clients par exemple. La fonction objectif qui est une fonction de maximisation obligera donc la prise de tous les clients dans la solution et sera équivalente à la maximisation d'une certaine constante (somme des profits) à laquelle on soustrait la somme des distances incluses dans la

solution. Cette formulation est à son tour équivalente à la minimisation de la somme des distances prises dans la solution.

Dans la littérature, les problèmes ayant pour objectif la maximisation de la différence entre la somme des profits et la somme des distances portent le nom de *problèmes de la tournée la plus profitable* (Profitable Tour Problems en anglais ou PTP). Le PTP incluant des contraintes de capacité est noté CPTP (voir le Chapitre précédent pour davantage de détails).

Nous pouvons facilement constater que le CPTP est un cas particulier de la variante que nous proposons. En effet, si l'on fixe les demandes de ramassage des clients à 0, on retombe directement sur un CPTP.

Par conséquent, nous appellerons notre variante *problème de la tournée la plus profitable avec ramassages et livraisons simultanés* (Profitable Tour Problem with Simutlaneous Pickup and Delivery services en anglais), que nous noterons PTPSPD.

Dans le présent Chapitre nous présentons donc une version réalise du VRPSPD notée PTPSPD. D'un point de vue technique : *i*) Le PTPSPD est une variante du VRP avec une flotte limitée de véhicules de capacités identiques, disponible dans un dépôt, dans le but d'effectuer des opérations de livraison et de ramassage pour un ensemble de clients. *ii*) Dans un PTPSPD, des profits sont affectés aux clients de sorte que, les tournées, commençant et se terminant au dépôt, garantissent que chaque client est visité au plus une seule fois (certains clients peuvent ne pas être visités du tout) et par un unique véhicule. *iii*) Les livraisons sont transportées du dépôt vers les clients, et les ramassages sont transférés des clients au dépôt. *iv*) Il est interdit de dépasser la capacité des véhicules. *v*) L'objectif du problème est de maximiser la différence entre la somme des profits collectés et les coûts de transport.

Le PTPSPD peut être rencontré par exemple dans le commerce des boissons gazeuses où chaque magasin (client) reçoit des bouteilles remplies et remet des bouteilles vides. Le transporteur se chargera de livrer les bouteilles remplies et de ramasser celles qui sont vides pour pouvoir les réutiliser par la suite. Une autre illustration du problème peut être trouvée dans la récupération de certaines machines ou certains appareils à la fin de leurs cycles de vie dans le but de les réparer ou de les recycler. Enfin, comme mentionné précédemment, une autre application pourrait être celle de la livraison de certains produits aux supermarchés ou aux épiceries dans des palettes de manutention et la récupération des palettes des anciennes livraisons afin de les réutiliser dans d'autres opérations de manutention.

## 4.1 État de l'art

Au meilleur de notre connaissance, le PTPSPD n'a pas encore été traité dans la littérature. Dans le Chapitre précédent, une description des méthodes la littérature du CPTP a été fournie. Dans ce qui suit, nous résumons les travaux traitant le VRPSPD ainsi que d'autres variantes de VRP avec ramassages, livraisons et profits.

D'après Salhi et Nagy [84] et Dethloff [29], Min [67] a introduit le VRPSPD pour modéliser un problème de collecte et de livraison de livres à 22 bibliothèques depuis

une bibliothèque centrale. Lors de chaque passage à une bibliothèque, le transporteur devait délivrer un certain nombre de livres et en récupérer d'autres. L'objectif du problème était de construire des routes de coût minimum satisfaisant toutes les demandes de ramassages et de livraisons de façon simultanée. Min [67] utilisa une procédure de résolution basée sur le principe de regroupement des clients d'abord puis, de routage (détermination de l'ordre des clients dans les routes). Ce type de procédures est appelé *cluster first-route second* en anglais. En effet, après avoir procédé au regroupement des clients par blocs, l'auteur a tenté d'effectuer l'opération de routage en résolvant des *problèmes de voyageur de commerce* dans chaque bloc. La réalisabilité de la solution était alors obtenue en pénalisant les arcs dépassant la capacité des véhicules puis en résolvant de nouveau le TSP irréalisable.

Dans l'article de Nagy et Salhi [72], il est rapporté que Halse [48] a résolu des instances de VRPSPD par une procédure de type *cluster first-route second*. La première étape de cette heuristique consiste en l'affectation des clients aux véhicules. Dans la seconde étape, un opérateur de voisinage est utilisé. Halse a été le premier à pouvoir résoudre des instances de VRPSPD contenant jusqu'à 100 clients et ayant plus de deux routes.

Le VRPSPD n'a plus été étudié pendant près d'une décennie. Après cela, Dethloff [29] prit le relais en développant un algorithme de construction basé sur le principe de *l'insertion la moins coûteuse* (cheapest insertion). L'idée principale de l'auteur peut être décrite par l'insertion successive de clients dans une route en cours de construction de sorte que, les routes soient construites (et donc « remplies ») consécutivement et que les critères d'insertion (insertion de moindre coût et respect de la capacité des véhicules) soient respectés. Après cela, Dethloff modifia les critères d'insertion et en ajouta trois nouveaux. Les tests d'évaluation de l'algorithme ont été effectués sur des instances de la littérature proposés par Min [67] et par Salhi et Nagy [84] ainsi que sur des instances que l'auteur généra aléatoirement. Les solutions obtenues ont été meilleures que celles de la littérature sauf pour une petite partie des instances où l'on limitait le nombre de tournées. L'auteur souligna que le temps de calcul de son approche décroissait avec la réduction des capacités des véhicules.

Montané et Galvão [69] ont eux aussi utilisé une méthode de type *cluster first-route second*. Ils regroupèrent les clients de deux façons différentes en s'inspirant de procédures de regroupement utilisées dans d'autres variantes de VRP de la littérature. Puis, pour chaque méthode de regroupement, ils appliquèrent quatre heuristiques de résolution du *problème de voyageur de commerce avec ramassages et livraisons simultanés* (TSPSPD). Ces heuristiques sont issues de travaux de la littérature dédiés au VRP avec ramassages et livraisons que les auteurs ont adapté au problème. Cela donna naissance à huit procédures pour le VRPSPD. Afin de comparer la qualité de ces dernières, les auteurs ont généré des instances pour le problème en adaptant des instances de la littérature destinées au VRP classique.

Trois ans après les travaux de Montané et Galvão, Nagy et Salhi [72] ont étudié plusieurs variantes de VRP avec ramassages et livraisons dont le VRPSPD. Les auteurs ont également étudié une variante du VRPSPD avec une contrainte additionnelle limitant la distance maximum qu'un véhicule pouvait parcourir. Deux nouveaux concepts appelés **weakly feasible route** (tournée faiblement réalisable) et **strongly feasible route** (tournée fortement réalisable) ont été introduits. Le premier concept décrit une tour-

née dans laquelle aucune des sommes des demandes de ses clients (ni la somme des ramassages ni celle des livraisons) ne doit dépasser la capacité du véhicule. Le second concept représente quant à lui une tournée dans laquelle la capacité du véhicule ne doit être excédée en aucun point. Dans les deux concepts, la longueur de la tournée ne doit pas dépasser la longueur maximum. Nagy et Salhi ont ensuite présenté plusieurs heuristiques moyennant un ensemble de stratégies d'amélioration des longueurs de routes et de rectification de la réalisabilité. Les heuristiques proposées permettent d'accepter des solutions irréalisables en fonction d'un certain paramètre. Lors de la phase d'exécution, les auteurs ont adapté, au VRPSPD, des instances de la littérature du VRP et ont comparé leurs heuristiques avec celles de la littérature. Il y eut, dans la plupart des cas, une amélioration remarquable que ce soit en termes de qualité des solutions, de temps de calcul ou de nombre de véhicules utilisés.

Une autre heuristique qui a été appliquée avec succès au VRPSPD est l'ALNS proposée par Ropke et Pisinger [83]. Cette heuristique fait appel à cinq opérateurs de suppression et cinq opérateurs d'insertion. Ceci a permis à l'heuristique de visiter divers voisinages et d'atteindre des solutions non atteintes auparavant.

Montané et Galvão [70] ont développé un algorithme de *recherche taboue* pour le VRPSPD avec et sans contraintes de distance maximum. Les voisinages proposés emploient trois mouvements inter-routes et un mouvement intra-route. Deux stratégies de sélection du prochain mouvement à effectuer ont été développées. L'intensification et la diversification sont quant à elles contrôlées par un schéma de pénalisation des fréquences d'apparitions d'attributs au sein des solutions. Montané et Galvão ont eux aussi adapté des instances de la littérature au VRPSPD et les ont utilisées pour évaluer la performance de leur algorithme. Les auteurs ont également comparé les résultats de leur *recherche taboue* avec ceux de la littérature sur les instances connues.

Dans leur article, Bianchessi et Righini [13] ont présenté et comparé des algorithmes de construction, des algorithmes de recherche locale ainsi que des algorithmes de *recherche taboue*. Plus particulièrement, le travail proposé traite l'application de la *recherche taboue* aux algorithmes basés sur des voisinages complexes et variés. En effet, les auteurs ont combiné des voisinages se basant sur l'échange d'arcs et de sommets en employant des *listes taboues* différentes. Bianchessi et Righini ont précisé que, dans le cas d'un VRPSPD avec une flotte limitée de véhicules, le fait de trouver une solution réalisable pour le problème était un problème NP-Difficile. Dans leur approche de résolution, Bianchessi et Righini ont utilisé le principe de *tournée faiblement réalisable* introduit par Nagy et Salhi [72]. Ainsi, lorsqu'une tournée faiblement réalisable est trouvée, elle est transformée en une tournée fortement réalisable en plaçant les clients de livraisons (dont la livraison est supérieure au ramassage) en début de tournées. Les auteurs ont aussi associé à chaque client un ensemble de variables afin d'accélérer la vérification de la réalisabilité.

Zachariadis et al. [105] ont proposé une méthode hybride, notée (GTS), combinant la *recherche taboue* et la *recherche locale guidée*. Cette combinaison a été faite dans le but de créer un équilibre entre l'intensification et la diversification des deux métaheuristiques. L'heuristique de construction proposée par Paessens [75] a été développée pour générer une solution initiale. Quatre types de mouvements avec des versions inter et intra-route(s) ont également été développés. À chaque itération de la GTS, un des quatre voisinages est sélectionné et appliqué à la solution actuelle. Les mouvements

acceptés par la GTS sont des mouvements réalisables et non tabous. Cependant, le *critère d'aspiration* est employé. Lorsqu'un mouvement est effectué, son inverse est déclaré tabou pour un certain nombre d'itérations de l'algorithme. À chaque itération de la GTS, l'arc de la solution candidate qui maximise la valeur d'une certaine fonction d'utilité est pénalisé. Son coût est alors augmenté à l'aide d'une fonction prédéfinie par les auteurs. Après un nombre fixé d'itérations sans amélioration, la GTS s'arrête, retournant la meilleure solution rencontrée jusque là.

Zachariadis et al. [106] ont développé une *recherche à mémoire adaptative* qui sauvegarde, à chaque fois, un ensemble de routes selon certains critères. L'algorithme extrait ensuite des séquences des routes sauvegardées et les combine pour créer une solution réalisable. Cette dernière est améliorée avec une *recherche taboue*. Les routes de la solution sont alors ajoutées aux routes sauvegardées et la procédure est répétée jusqu'à l'atteinte d'un nombre prédéfini d'itérations. En plus de sauvegarder les meilleures routes rencontrées, la méthode de Zachariadis et al. garde aussi en mémoire la fréquence d'extraction, la taille et le coût de la solution pour chaque séquence utilisée. Ces trois paramètres jouent un rôle important dans le choix des prochaines séquences à sélectionner. Les auteurs ont également présenté, pour chaque opérateur de voisinage utilisé dans la *recherche taboue*, quelques conditions faciles à vérifier pour garantir la réalisabilité des mouvements allant être effectués. Ces conditions permettent d'améliorer considérablement le temps de calcul de la méthode, particulièrement pour les instances de grande taille. Pour réduire aussi le temps de calcul et éviter les dépassements de mémoire, les informations nécessaires au fonctionnement de l'algorithme sont stockées dans une structure sous forme d'arbre.

La même année, Subramanian et al. [92] ont élaboré un algorithme parallèle noté P-ILS-RVND pour résoudre le VRPSPD. Dans cet algorithme, un processus maître oriente la direction de recherche en s'aidant des informations transmises par les sous-processus. Tout d'abord, chaque client est considéré dans une route à part. Après cela, chaque sous-processus exécute une heuristique combinant une ILS et une RVND (notée ILS-RVND) avec seulement une itération sans amélioration, ce qui va juste permettre de réduire le nombre de routes. Les solutions sont retournées au processus maître qui détermine le plus petit nombre de véhicules utilisés. Ce nombre et ensuite transmis aux sous-processus, qui l'utilisent ainsi qu'un autre paramètre noté $\gamma$ pour générer une solution initiale. Celle-ci est, par la suite, améliorée avec une ILS-RVND réitérée un certain nombre de fois sans perturbations. Les solutions obtenues par chaque sous-processus sont transmises au processus maître, qui détermine les trois meilleures solutions et les trois valeurs du paramètre $\gamma$ qui leurs sont associées. Ces valeurs sont transmises aux sous-processus. Chaque sous-processus tire aléatoirement l'une des trois valeurs de $\gamma$ et l'utilise pour générer une nouvelle solution initiale. Celle-ci est aussi améliorée avec une ILS-RVND mais avec des perturbations et beaucoup plus d'itérations cette fois-ci. À partir de là, les sous-processus commencent à envoyer des demandes d'itérations additionnelles au processus maître, et celui-ci décide de la nécessité ou non de la validation de telles requêtes. Après l'épuisement des itérations disponibles, l'algorithme s'arrête.

Zachariadis et Kiranoudis [104] ont implémenté une recherche locale qui utilise un codage spécial de mouvements appelé **Static Move Descriptor** (SMD) ainsi qu'un **concept de promesse**. La recherche locale inclut l'opérateur de voisinage *2-Opt* ainsi qu'un nouvel opérateur appelé *VLBE*. VLBE échange des séquences de clients entre

différentes routes. Le but d'SMD est de réduire le temps de calcul. Le concept de promesse vise, quant à lui, à sortir des optima locaux. Les auteurs ont également défini certaines variables pour accélérer la vérification de la réalisabilité.

Dans leur travail Tasan et Gen [96] ont implémenté un *algorithme génétique*. Les solutions de cet algorithme sont représentées par un vecteur dans lequel on ne peut distinguer de façon explicite les débuts et les fins des différentes routes. Ces débuts et fins de routes sont déterminés à l'aide de la capacité des véhicules utilisés. Les auteurs utilisent dans leur méthode une pénalité relativement élevée pour sanctionner les routes fortement irréalisables (routes non réalisables et non *faiblement réalisables*). Les individus de la population sont sélectionnés grâce à la méthode de la roulette (Roulette-Wheel Selection en anglais). Les opérateurs génétiques étudiés sont le croisement *partial-mapped crossover* (PMX) et la mutation *swap*. PMX sélectionne deux positions dans les chromosomes des parents aléatoirement. Il échange les deux « sous-chromosomes » (définis par les deux positions) des deux parents, puis, procède à l'ajout des parties restantes. La mutation *swap* échange les positions de deux clients dans la solution. Les auteurs ont généré de nouvelles instances pour le problème à partir des instances du CVRP. Les instances générées ont été utilisées pour évaluer les solutions de l'*algorithme génétique* par rapport à celles obtenues avec le solveur CPLEX.

Jun et Kim [58] ont implémenté un algorithme de résolution pour VRPSPD dont la construction des routes se fait à l'aide d'un nouveau balayage noté SWNA. SWNA utilise plusieurs points de départ et insert les clients en fonction de leurs angles polaires par rapport à ces points-là. Plusieurs solutions sont générées et la meilleure est retenue. L'algorithme procède ensuite à l'amélioration de la solution sauvegardée à l'aide d'une série d'opérateurs de voisinage inter et intra-route(s). Une perturbation est ensuite effectuée. Cette dernière consiste en la suppression de certaines routes et de certains clients. Après cela, un problème d'affectation est résolu afin de remettre tous les clients dans la solution. Enfin, la solution est améliorée encore une fois avec les opérateurs de voisinage et les dernières étapes de l'algorithme (suppression, réinsertion et amélioration) sont répétées jusqu'à atteindre un certain nombre d'itérations fixé au préalable.

Goksal et al. [43] ont choisi de développer un algorithme hybride combinant une PSO avec une VND. Une stratégie semblable à celle du *recuit simulé* a été appliquée pour garder la diversification de la PSO. Pour commencer, les auteurs ont proposé un codage des solutions sous forme de tour géant ainsi qu'une procédure rapide de décodage visant à délimiter les routes. La procédure de décodage s'appuie sur la résolution du problème du plus court chemin dans un graphe. L'*heuristique du plus proche voisin* a été utilisée pour déterminer la première particule de la population. Les autres particules sont générées aléatoirement. Les mises à jour des positions des particules de la PSO ont été adaptées au VRPSPD et six structures de voisinage inter et intra-route(s) ont été implémentées dans la recherche locale.

Fard et Akbari [36] se sont intéressés au VRPSPD avec des contraintes relatives à la durée maximum des tournées. Les auteurs ont proposé une *recherche taboue* hybride. Tout d'abord, une heuristique multi-départs ainsi qu'une *heuristique du plus proche voisin* ont été implémentées pour générer plusieurs solutions initiales. Ces dernières sont évaluées, et la meilleure solution rencontrée est considérée comme point de départ de l'heuristique hybride. Ensuite, une *recherche taboue* est appliquée à la solution sélection-

née, puis, les *listes taboues* sont réinitialisées. Après cela, l'heuristique ajuste les durées des *statuts tabous* et utilise les procédures d'intensification puis de diversification proposées par Montané et Galvão [70]. Enfin, les durées des *statuts tabous* sont réinitialisés et une *recherche taboue* est appliquée suivie d'une recherche locale.

Plusieurs méthodes approchées ont été utilisées pour résoudre les *problèmes généraux de ramassages et livraisons* (General Pickup and Delivery Problems en anglais ou GPDPs). On peut citer par exemple, les heuristiques de *recherche taboue* de Nanry et Barnes [73], Hoff et Løkketangen [54], Hoff et al. [53], Brandao [14], Gribkovskaia et al. [45] et Erdoğan et al. [33]. Dans leur papier, Nanry et Barnes [73] ont présenté une *recherche taboue réactive* (Reactive Tabu Search en anglais ou RTS) pour le *problèmes de ramassages et livraisons avec fenêtres de temps* (Pickup and Delivery Problem with Time Windows). Cette heuristique utilise des déplacements et des échanges de paires de ramassage-livraison entre différentes routes. L'heuristique modifie également la position d'un client donné au sein d'une même route. RTS permet l'acceptation de solutions irréalisables en ajoutant un terme de pénalisation à la fonction objectif. La *recherche taboue* de Hoff et Løkketangen [54] génère des solutions de type Lasso pour le *problème de voyageur de commerce avec ramassages et livraisons* (Traveling Salesman Problem with Pickup and Delivery en anglais ou TSPPD), de sorte que la structure de voisinage employée consiste en l'opérateur *2-Opt*. Hoff et al. [53] ont étudié une *recherche taboue* impliquant des ensembles restreints de mouvements pour les opérateurs relocate et swap inter-route. Cette approche est appliquée au *problème de ramassages et livraisons de Lasso un-à-plusieurs-à-un, multi-véhicules* (Multi-Vehicle Lasso One-to-Many-to-One Pickup and Delivery Problem en anglais), au *problème de ramassages et livraisons Hamiltonien un-à-plusieurs-à-un avec demandes combinées* (Hamiltonian One-to-Many-to-One Pickup and Delivery Problem with Combined Demands en anglais) et au *problème de ramassages et livraisons un-à-plusieurs-à-un avec demandes uniques et retours* (One-to-Many-to-One Pickup and Delivery Problem with Single Demands and Backhauls en anglais). Dans le papier de Brandao [14], une *recherche taboue* est présentée pour le *problème de tournées de véhicules avec retours* (Vehicle Routing Problem with Backhauls en anglais). Cette heuristique utilise une solution initiale qui résulte d'une relaxation du problème étudié. Gribkovskaia et al. [45] ont étudié le *problème de tournées à un seul véhicule avec ramassages et livraisons* (Single Vehicle Routing Problem with Pickups and Deliveries en anglais) et ont proposé une *recherche taboue* basée sur l'algorithme de *recherche taboue unifié* (Unified Tabu Search algorithm en anglais) de Cordeau et al. [24]. Erdoğan et al. [33] ont présenté une *recherche taboue probabiliste* (Probabilistic Tabu Search en anglais) ainsi qu'une *recherche locale réitérée* pour le *problème de voyageur de commerce avec ramassages, livraisons et chargement en FIFO* (Pickup and Delivery Traveling Salesman Problem with first-in-first-out loading en anglais).

VNS a aussi été appliquée aux GPDPs. Par exemple, une VNS a été proposée par Mladenović et Hansen [68] pour le *problème de voyageur de commerce avec retours* (Traveling Salesman Problem with Backhauls en anglais). Carrabs et al. [16] ont eux aussi proposé une heuristique de type VNS pour le *problème de voyageur de commerce avec ramassages, livraisons et chargement en LIFO* (Traveling Salesman Problem with Pickups and Deliveries with Last In First Out Loading en anglais). Dans leur méthode, trois nouveaux voisinages ont été utilisés.

Une autre heuristique qui a été appliquée avec succès aux *problèmes de ramassages et livraisons* est l'heuristique ALNS. Ropke et Pisinger [82] ont développé une ALNS pour

le *problème de tournées de véhicules avec ramassages, livraisons et fenêtres de temps* (Vehicle Routing Problem with Pickup and Delivery and Time Windows en anglais). Les auteurs ont implémenté trois heuristiques de suppression et cinq heuristiques d'insertion. La sélection de ces opérateurs dépend de leurs performances durant les itérations antérieures. Ropke et Pisinger [83] ont adapté cet algorithme à une grande classe de *problèmes de tournées de véhicules avec ramassages et livraisons*. Ces auteurs ont développé cinq opérateurs de suppression et ont gardé les mêmes opérateurs d'insertion que ceux de Ropke et Pisinger [82].

Plusieurs algorithmes hybrides ont été développés pour les *problèmes de ramassages et livraisons*. À titre d'exemple, nous pouvons citer l'approche de Bent et Van Hentenryck [9] qui tente de résoudre les *problème de tournées de véhicules avec ramassages, livraisons et fenêtres de temps* (Pickup and Delivery Vehicle Routing Problems with Time Windows and Multiple Vehicles en anglais). Dans leur papier, Bent et Van Hentenryck ont utilisé un algorithme à deux phases qui hybride le *recuit simulé* avec LNS.

Hernández-Pérez et al. [49] ont proposé un autre algorithme hybride pour résoudre le *problème de voyageur de commerce avec ramassages, livraisons et un seul produit* (One-commodity Pickup and Delivery Traveling Salesman Problem en anglais). Cet algorithme combine une GRASP avec une VND. Plus précisément, une première heuristique de type VND est utilisée après un algorithme glouton aléatoire afin de générer un ensemble de solutions réalisables. Cette première VND emploie des versions modifiées de certains opérateurs de voisinage qui permettent de visiter certaines des solutions irréalisables. Après cela, une seconde VND tente d'améliorer la meilleure solution rencontrée en changeant la position des clients dans une même route.

Rodríguez-Martín et Salazar-González [81] ont décrit une approche hybride combinant GRASP avec VND pour résoudre le *Problème voyageur de commerce avec ramassages et livraisons, un-à-un, et multi-produits* (Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem en anglais). Plus précisément, ils ont implémenté deux variantes de cette méthode. Dans la première variante, une procédure de programmation mathématique, agissant tel une recherche locale, est exécutée sur une version relaxée du modèle mathématique. La seconde variante considère, quant à elle, des opérateurs d'échange d'arcs comme procédures de recherche locale.

Récemment, Männel et Bortfeldt [66] ont étudié le *problème de tournées de véhicules avec ramassages, livraisons et chargement à trois dimensions* (Vehicle Routing Problem with Pickup and Delivery and Three-dimensional Loading Constraints en anglais ou 3L-PDP). Leur papier considère trois variantes du 3L-PDP pouvant être classées selon leurs types de rechargement. Les auteurs ont développé un algorithme hybride considérant une version modifiée d'LNS ainsi qu'une heuristique de recherche dans un arbre (Tree Search en anglais). Ghilas et al. [41] ont abordé le *problème de ramassages et livraisons avec fenêtres de temps et lignes programmées* (Pickup and Delivery Problem with Time Windows and Scheduled Lines en anglais ou PDPTW-SL) et ont proposé une heuristique de type ALNS pour le résoudre. Cette heuristique inclut 10 opérateurs de suppression et 5 opérateurs d'insertion. Iassinovskaia et al. [56] ont développé une heuristique de type *cluster first-route second* pour le *le problème de tournées de véhicules et d'inventaire avec fenêtres de temps et ramassages et livraisons simultanés, dans des chaînes logistiques fermées* (inventory-routing problem with time windows and simultaneous pickup and delivery in closed-loop supply chains en anglais ou PDIRPTW) tandis que Zhu et al. [108] ont

proposé un *algorithm mémétique* multi-objectif afin de résoudre le *problème de ramassages et livraisons dynamique, un-à-plusieurs-à-un* (one-to-many-to-one Dynamic Pickup-and-Delivery Problem en anglais ou DPDP). Durant la même année, Hernández-Pérez et al. [50] ont présenté une méthode hybride pour le *problème de voyageur de commerce multi-produits avec ramassages et livraisons* (multi-commodity Pickup-and-Delivery Traveling Salesman Problem en anglais ou m-PDTSP). Cette méthode inclut une phase d'initialisation où un ensemble de solutions sont générées. Après cette phase d'initialisation, une phase d'amélioration utilisant VND est appliquée. Cette dernière inclut également certaines procédures de perturbation afin de ne pas rester coincée dans des optima locaux. Enfin, Zachariadis et al. [107] ont implémenté un système d'optimisation utilisant quelques techniques de mémorisation pour le *problème de tournées de véhicules avec ramassages et livraisons simultanés et chargement à deux dimensions* (Vehicle Routing Problem with Simultaneous Pickups and Deliveries and Two-Dimensional Loading Constraints en anglais ou 2L-SPD) et le *2L-SPD avec chargement en LIFO* (2L-SPD with Last In First Out constraints en anglais). Cette approche consiste en une recherche locale utilisant trois structures de voisinages avec des représentations spéciales ainsi qu'une heuristique de rechargement bidimensionnel permettant d'avoir des solutions réalisables.

Pour davantage de définitions et d'approches de résolution des *problèmes de ramassages et livraisons*, nous référons le lecteur intéressé aux articles de Berbeglia et al. [10] et de Silva et Zuluaga [89].

Comme dit précédemment, une description des travaux de la littérature ayant traité le CPTP a été présenté dans le Chapitre précédent.

Il y a également eu d'autres travaux traitant les *problèmes de ramassages et livraisons avec profits*. On peut citer par exemple le papier de Gribkovskaia et al. [46], qui étudie la variante de *VRP avec un seul véhicule, opérations sélectives de ramassage et opérations de livraison* (Single Vehicle Routing Problem with Deliveries and Selective Pickups en anglais ou SVRPDSP). Cette variante est différente du PTPSPD. En effet dans un SVRPDSP, les opérations de ramassage et livraison peuvent être effectuées indépendamment (pas de contrainte qui impose la simultanéité). De plus, un seul véhicule est requis pour satisfaire toutes les demandes de livraison et quelques demandes de ramassage. Gribkovskaia et al. ont proposé un modèle mathématique, plusieurs heuristiques de construction et d'amélioration ainsi qu'un algorithme de *recherche taboue*. Ce dernier emploie un mécanisme de perturbation permettant de visiter certaines solutions irréalisables ainsi qu'un schéma de pénalisation introduisant un peu de diversification au sein de la recherche. Étant donné que dans un SVRPDSP un client peut être visité deux fois, les voisinages étudiés dans le papier de Gribkovskaia et al. [46], manipulent les insertions et les suppressions des deuxièmes visites des clients.

Un peu plus tard, Coelho et al. [23] ont proposé une heuristique hybride basée sur *recherche générale à voisinage variable* pour le SVRPDSP. Cette approche inclue plusieurs opérateurs d'amélioration ainsi que quelques heuristiques de perturbation. La solution initiale est générée à l'aide d'une méthode exacte.

Gutiérrez-Jarpa et al. [47] ont étudié le *problème de tournées de véhicules avec ramassages sélectifs, livraisons et fenêtres de temps* (Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows en anglais ou VRPDSPTW), une variante de GPDPs avec

profits qui étend le SVRPDSP. En plus des différences mentionnées précédemment entre le SVRPDSP et le PTPSPD, le VRPDSPTW considère également des contraintes de fenêtres de temps qui ne sont pas incluses dans le PTPSPD. Un algorithme exacte a été développé pour résoudre ce problème.

Récemment, un papier traitant une autre autre variante de GPDPs avec profits a été proposé par Assis et al. [4]. Dans ce papier, les auteurs ont décrit le *problème de tournées de véhicules multi-objectif avec livraisons fixées et ramassages optionnels* (multiobjective vehicle routing problem with fixed delivery and optional collections en anglais). Le principal objectif de cette variante est de minimiser le coût de transport ainsi que le nombre de demandes de ramassage non satisfaites. Nous remarquons que ce problème et le PTPSPD ne partagent pas le même objectif. De plus, le *problème de tournées de véhicules multi-objectif avec livraisons fixées et ramassages optionnels* diffère de notre problème dans le fait que les opérations de livraison sont toutes obligatoires. Les auteurs ont développé une *recherche locale réitérée multi-objective* afin d'avoir des solutions efficaces pour leur problème.

Durant la même année, Ting et Liao [97] ont publié une autre variante de GPDPs avec profits appelée *problème de ramassages et livraisons sélectifs* (Selective Pickup and Delivery Problem en anglais ou SPDP). Plus précisément, le SPDP consiste à alimenter tous les clients de livraison à partir des clients de ramassage avec l'objectif de minimiser le coût de transport. Dans cette variante, la visite de tous les clients de ramassage n'est pas requise. Le SPDP diffère du PTPSPD de par son objectif, sa contrainte de visite obligatoire de tous les clients de livraison et des origines de ses produits. Ting et al. ont proposé une formulation mathématique sous forme de programme linéaire en nombres entiers ainsi qu'un *algorithme mémétique*. Ce dernier utilise une nouvelle représentation de solutions qui inclut tous les ramassages et toutes les livraisons. De plus, cet *algorithme mémétique* emploie des opérateurs génétiques et un opérateur *2-Opt* modifié, spécialement conçus pour la nouvelle représentation du SPDP.

Ho et Szeto [52] ont également étudié le SPDP. Ils ont proposé un modèle mathématique pouvant être directement exécuté sous solveur (tel que CPLEX). De plus, les auteurs ont implémenté une *procédure de recherche gloutonne aléatoire adaptative* (Greedy Randomized Adaptive Search Procedure en anglais ou GRASP) avec une *recomposition de chemin* (Path Relinking en anglais). Tout d'abord, cette approche construit des solutions initiales ne contenant que les livraisons, de sorte que les meilleures solutions rencontrées soient stockées dans un **ensemble de référence** (reference set en anglais). Ensuite, une solution est prise de l'*ensemble de référence*. Elle est alors améliorée à l'aide de l'opérateur *2-Opt*. Après cela, la *recomposition de chemin* est appliquée en considérant l'optimum local comme étant une solution initiale et la solution de l'*ensemble de référence* comme une solution de « *guidage* » (guiding solution en anglais). Ceci est répété jusqu'à l'atteinte d'un certain nombre d'itérations sans modifications. Ensuite, la solution de l'*ensemble de référence* devient solution initiale et l'optimum local devient une solution de *guidage*. L'*ensemble de référence* est mis à jour si la solution obtenue est meilleure que la moins bonne solution se trouvant dans cet ensemble. Enfin, les ramassages sont ajoutés à la nouvelle solution. Cela est répété pour toutes les solutions de l'*ensemble de référence*. La *recomposition de chemin* emploie un opérateur de type *2-Opt* qui bloque momentanément l'utilisation de certains attributs.

Qiu et al. [80] ont étudié le *problème de ramassages et livraisons sélectif avec maximi-*

*sation de profits* (Profit-Maximizing Pickup and Delivery Selection Problem en anglais ou PPDSP). L'objectif de ce problème est la maximisation de la différence entre les profits collectés et le coût de transport. Dans un PPDSP, tous les clients sont optionnels et possèdent des demandes de ramassage et de livraison. En comparant le PPDSP au PTPSPD, nous pouvons noter que le PPDSP ne requiert pas la satisfaction simultanée des demandes de ramassage et des demandes de livraison. De plus, dans un PPDSP, des bornes sont définies sur la durée totale des routes et des contraintes de fenêtres de temps sont introduites. Nous remarquons également que la flotte de véhicules utilisés dans un PPDSP est une flotte hétérogène. Qiu et al. ont développé une méthode exacte basée sur une recherche dans un graphe, ainsi qu'une recherche aléatoire et une heuristique gloutonne pour les cas à un seul et à plusieurs véhicules respectivement.

Li et al. [65] ont étudié une variante de GPDPs avec profits incluant des clients réservés et des contraintes de fenêtres de temps (Pickup and Delivery Problem with Time Windows, Profits, and Reserved Requests en anglais ou PDPTWPR). Plus particulièrement, dans un PDPTWPR, chaque transporteur (ou véhicule) possède un ensemble de clients réservés et un ensemble de clients optionnels. Pour chaque transporteur, la visite des clients réservés est obligatoire. Les clients optionnels peuvent, quant à eux, soit être visités par d'autres transporteurs, ou alors ne pas être visité du tout. Chaque client possède une demande de ramassage et une demande de livraison associées à une origine, une destination, deux fenêtres de temps, un temps de service et un paiement (équivalent au profit). L'objectif du PDPTWPR est de construire, pour chaque transporteur, une route réalisable incluant tous les clients obligatoires et éventuellement quelques clients optionnels, dans le but de maximiser la différence entre les profits des clients visités et le coût de transport.

Le PDPTWPR diffère principalement du PTPSPD dans trois points. Le premier point est que certains clients ne sont pas optionnels. Le second point est que des contraintes de fenêtres de temps sont incluses. Le dernier point est que chaque véhicule possède son propre ensemble de clients à visiter.

Li et al. ont proposé un modèle mathématique ainsi qu'une heuristique de type ALNS pour résoudre le problème. L'ALNS utilisé est une heuristique multi-départs. Elle inclut l'opérateur de suppression *random removal*, en combinaison avec certains nouveaux opérateurs de suppression inspirés du *worst removal* et du *Shaw removal* de l'ALNS classique. Concernant les opérateurs dérivés du *worst removal*, les auteurs en ont proposé trois variantes. La première variante, appelée *Least profit removal*, supprime les clients qui contribuent le moins à l'augmentation de la valeur de la fonction objectif (différence entre profits collectés et coûts de transport). La seconde variante, appelée *Least paid removal*, supprime les clients les moins rentables en termes de profits. La troisième variante, appelée *Most expensive removal*, supprime les clients les plus coûteux en termes de coûts de transport. L'opérateur *Shaw removal* de l'ALNS classique n'a pas été modifié. Par contre, Li et al. ont implémenté un nouvel opérateur inspiré du *Shaw removal*. Ce nouvel opérateur supprime les clients qui ont des profits plus ou moins similaires. Il est appelé *Price similarity removal*.

Li et al. ont également proposé des opérateurs d'insertion différents de ceux de l'ALNS classique. En effet, ils ont utilisé une heuristique de type *basic greedy* et une heuristique de type *regret* qui utilisent l'objectif du PDPTWPR dans la sélection de clients.

Un mécanisme d'ajustement dynamique du comportement des opérateurs a été adopté par les auteurs. Ce mécanisme choisit le type de clients à supprimer (obligatoires ou optionnels) et détermine les priorités d'insertions de clients.

Enfin, lorsque l'ALNS de Li et al. se retrouve bloquée dans un optimum local, elle fait appel à une stratégie, appelée **meta-destroy**, qui consiste à combiner deux opérateurs de suppression et un opérateur d'insertion, afin d'introduire un peu plus de diversification. Avant chaque redémarrage de l'heuristique, une procédure de recherche locale est exécutée.

À travers cette revue de la littérature, nous pouvons voir que la méthode LNS (et ALNS en particulier) a été implémentée à plusieurs reprises pour les GPDPs dont le VRPSPD.

Comme les GPDPs avec profits sont des problèmes relativement nouveaux, qui n'ont pas été assez étudiés pour voir quelle méthode fonctionne le mieux, et comme toutes les heuristiques proposées dans la littérature des GPDPs avec profits sont différentes, nous avons décidé d'implémenter une heuristique de type ALNS pour le problème étudié. Notre choix est motivé par le fait qu'ALNS a été capable de donner des résultats compétitifs pour le VRPSPD, pour quelques variantes de GPDPs avec profits et pour plusieurs variantes de GPDP.

## 4.2 Formulation mathématique

Dans cette Section, nous proposons une formulation mathématique pour le PTPSPD. Cette formulation est inspirée du modèle présenté par Dell'Amico et al. [27] pour le VRPSPD. Dans ce qui suit, nous donnons une description de notre modèle.

Soient $n$, $m$ et $Q$ le nombre de clients, le nombre de véhicules et la limite de capacité des véhicules respectivement. Notons par $N = \{1, \ldots, n\}$ l'ensemble des clients. Soit 0 le dépôt. Le PTPSPD peut être défini par un graphe complet non-orienté et valué $G = (V, E, c)$, où $V = N \cup \{0\}$ représente l'ensemble des sommets, $E$ est l'ensemble des arêtes (sections de routes entre les éléments de $V$), et $c : E \rightarrow \mathbb{R}^+$ est une fonction de poids qui associe à chaque arête $\{i, j\} \in E$ un poids $c_{ij} \in \mathbb{R}^+$ représentant le coût de transport entre les sommets $i$ et $j$. Notons que, la matrice des coûts de transport entre les paires de sommets $i \in V$ et $j \in V$ est symétrique. Ainsi, pour chaque $(i, j) \in E$, $c_{ij} = c_{ji}$. Pour chaque client $i \in N$, définissons $p_i \geq 0$, $d_i \geq 0$ et $pr_i \geq 0$ comme étant la demande de ramassage, la demande de livraison et le profit de $i$ respectivement. Notons qu'un client sans demande de ramassage ($p_i = 0$) ni demande de livraison ($d_i = 0$) (avec ou sans profit) ne peut exister étant donné qu'on étudie une variante de problèmes de ramassages et livraisons (GPDP). Ainsi, les variables de décision peuvent être définies comme suit : $x_{ij}$, $(i, j) \in E$, est une variable booléenne qui prend la valeur 1 si un véhicule visite le sommet $j$ directement après avoir visité le sommet $i$ et 0 sinon, $t_i$, $i \in N$, est une autre variable booléenne qui prend la valeur 1 si le sommet $i$ est inclus dans la solution et 0 sinon. Enfin, $y_{ij}$ et $z_{ij}$, avec $(i, j) \in E$, sont des variables réelles positives représentant les demandes collectées en visitant les prédécesseurs du sommet $i$ ($i$ inclus) et qui sont transportés sur l'arc $(i, j)$, et la quantité de demandes à délivrer aux successeurs de $i$ qui sont transportées sur l'arc $(i, j)$ respectivement. Nous obtenons donc le programme linéaire en variables mixtes suivant :

$$\max f(x) = \sum_{i \in N} pr_i \cdot t_i - \sum_{i,j \in V} c_{ij} \cdot x_{ij} \tag{4.1}$$

$$\sum_{i \in V} y_{ji} - \sum_{i \in V} y_{ij} = p_j \cdot t_j \qquad \forall j \in N \tag{4.2}$$

$$\sum_{i \in V} z_{ij} - \sum_{i \in V} z_{ji} = d_j \cdot t_j \qquad \forall j \in N \tag{4.3}$$

$$y_{ij} + z_{ij} \leq Q \cdot x_{ij} \quad \forall i \in V; \forall j \in V; \tag{4.4}$$

$$t_i = \sum_{j \in V} x_{ij} \qquad \forall i \in N; \tag{4.5}$$

$$\sum_{j \in V} x_{ij} = \sum_{j \in V} x_{ji} \qquad \forall i \in V; \tag{4.6}$$

$$\sum_{i \in N} x_{0i} \leq m \tag{4.7}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in V; \forall j \in V; \tag{4.8}$$

$$t_i \in \{0,1\} \qquad \forall i \in N; \tag{4.9}$$

$$y_{ij} \in \Re^+ \quad \forall i \in V; \forall j \in V; \tag{4.10}$$

$$z_{ij} \in \Re^+ \quad \forall i \in V; \forall j \in V; \tag{4.11}$$

La fonction objectif est donnée par la Formule (4.1). Les Contraintes (4.2) et (4.3) représentent les contraintes de satisfaction des demandes de ramassage et livraison. Les Contraintes (4.4) assurent que la capacité des véhicules est respectée. Les Contraintes (4.5) garantissent que si la solution contient un arc sortant d'un sommet $i$ alors le client $i$ sera nécessairement inclus dans la solution. Les Contraintes (4.6) sont celles de conservation des flots. Les Contraintes (4.7) forcent le nombre d'arcs sortant du dépôt à être au plus égal au nombre de véhicules disponibles. Les Contraintes ((4.8)–(4.11)) définissent les variables de décision. De plus, les Contraintes (4.5), (4.8) et (4.9) garantissent que chaque client est visité au plus une seule fois. Enfin, les Contraintes (4.2), (4.3), (4.5), (4.6), (4.8) et (4.9) garantissent l'élimination des sous-tours en supposant que, pour chaque client $i$, $p_i$ et $d_i$ ne peuvent être tout deux égaux à 0. Une preuve d'élimination des sous-tours est donnée dans l'Annexe A.2.

On peut remarquer qu'il existe beaucoup de similarités entre notre modèle et celui proposé par Dell'Amico et al. [27]. En effet, si tous les profits sont fixés à un très grand nombre en comparaison avec les coûts de transport (les coûts de transport peuvent être équivalents aux distances), la solution optimale contiendra tous les clients. Lorsque l'on compare plusieurs solutions contenant tous les clients, la somme des profits ($\sum_{i \in N} pr_i \cdot t_i$) peut être considérée comme une valeur constante. Ainsi, l'objectif sera la maximisation de ($-\sum_{i,j \in V} c_{ij} \cdot x_{ij}$), qui est à son tour équivalente à la minimisation des coûts de transport.

Notons que si tous les profits sont fixés à 0, notre modèle ne sera pas équivalent à celui proposé par Dell'Amico et al. [27]. En effet, lorsque les profits sont fixés à 0, la solution optimale du problème sera une solution sans aucun client, comme il n'y a aucune contrainte qui impose que chaque client doit être visité au moins une fois.

Notre modèle mathématique possède $O(n^2)$ variables et $O(n^2)$ contraintes.

## 4.3 Approche proposée

Pour tenter de résoudre le PTPSPD, nous proposons une extension de l'heuristique ALNS appelée *ALNS sélective* (selective ALNS en anglais ou sALNS). La décision d'adopter une telle approche a été motivée par le fait qu'ALNS ait fait ses preuves dans la résolution de plusieurs variantes de problèmes de tournées de véhicules en général, et de problèmes de ramassages et livraisons en particulier (voir les travaux de Ropke et Pisinger [83], Pisinger et Ropke [78] par exemple).

De plus, l'aspect sélectif du PTPSPD, nous a poussé à penser que la méthode ALNS pouvait y être particulièrement bien adaptée. En effet, comme mentionné dans la Section 2.2.3, la méthode utilise le principe de suppression et de réinsertion des clients dans une solution donnée. En fait, cet algorithme utilise plusieurs opérateurs appelés *opérateurs de destruction* (ou bien *opérateurs de suppression*) qui ont pour but de supprimer un certain nombre de clients de la solution actuelle. Puis, il tente de réinsérer ces clients avec d'autres opérateurs dits *de construction* (ou alors *opérateurs de réparation* ou encore *opérateurs d'insertion*) dans de meilleures positions.

À chaque itération de l'ALNS, on choisit un opérateur de destruction et un opérateur de réparation de façon pseudo-aléatoire, basée sur le score (performance) de chaque opérateur tout au long de la recherche. La solution qui en résulte est alors acceptée ou non selon certains critères et le processus continue jusqu'à l'atteinte des critères d'arrêt.

Nous avons donc supposé qu'en supprimant quelques clients, on pouvait obtenir une solution réalisable qui serait éventuellement de meilleure qualité que la solution de départ, et que cette solution ne pourrait peut être pas être atteinte sans le principe de suppression.

Plus de détails concernant ALNS peuvent être trouvés dans la Section 2.2.3 du Chapitre 2.

L'Algorithme 10 décrit le pseudo-code d'sALNS. Les détails de ses composantes sont présentés dans les Sections suivantes. Afin de faciliter la compréhension, nous donnons, dans le Tableau 4.1, une description de chaque paramètre et chaque variable utilisés dans l'Algorithme 10.

Table 4.1 – *Description des paramètres et des variables utilisés dans l'Algorithme 10.*

| paramètre/variable | description |
|---|---|
| $L_{rem}$ | Liste des opérateurs de suppression |
| $L_{ins}$ | Liste des opérateurs d'insertion |
| $lev$ | un compteur utilisé pour déterminer l'actuelle phase d'sALNS (évaluation ou application) |
| $itr$ | Le nombre d'itérations actuelles |
| $T$ | La température actuelle |
| $IS$ | Le nombre total d'itérations dans la boucle principale d'sALNS |
| $ml$ | Le nombre d'itérations effectuées avant la réinitialisation des scores, défini par l'utilisateur |
| $r$ | Le nombre aléatoire de clients à supprimer $r \in \{a_1, a_2, \ldots, a_k\}$, où $a_1, a_2, \ldots, a_k \in \mathbb{N}$ et $k$ sont des paramètres définis par l'utilisateur |
| $nb_{rh}$ | Le nombre d'opérateurs de suppression |
| $R$ | L'opérateur de suppression choisi |
| $nb_{ih}$ | Le nombre d'opérateurs d'insertion |
| $I$ | L'opérateur d'insertion choisi |
| $c$ | Le coefficient de décroissance, défini par l'utilisateur |
| $T_0$ | Le seuil de température, défini par l'utilisateur |

sALNS commence par l'appel de l'heuristique $H_{const}$ (détaillée dans la Section 4.3.1).

Cette dernière génère une solution réalisable. La boucle principale est alors exécutée durant $IS$ itérations.

Chaque $ml$ itérations, les scores des opérateurs de suppression et d'insertion sont réinitialisés (lignes 5–8). L'algorithme principal choisit ensuite le nombre $r$ de clients à supprimer. Ce nombre est aléatoirement choisi depuis l'ensemble de $k$ éléments $\{a_1, a_2, \ldots, a_k\}$, où $a_1, a_2, \ldots, a_k \in \mathbb{N}$ et $k$ sont définis par l'utilisateur (ligne 9).

Un opérateur de suppression est après cela sélectionné (lignes 10–17). Une liste $L_{rem}$ contenant initialement tous les opérateurs de suppression est utilisée afin de permettre cette sélection. Plus précisément, durant une phase d'évaluation (qui dure $5 \cdot nb_{rh}$ itérations pour les suppressions, où $nb_{rh}$ est le nombre d'opérateurs de suppression), sALNS choisit un opérateur qui n'a pas encore été choisi d'$L_{rem}$ et supprime cet opérateur de la liste. Dans le cas où $L_{rem}$ est vide, tous les opérateurs de suppression sont inclus de nouveau dans $L_{rem}$ (ligne 12). D'autre part, dans une phase d'application, l'algorithme choisit un opérateur de suppression en fonction du score de ce dernier de sorte que, les opérateurs les plus efficaces (avec de grands scores) ont plus de chances d'être sélectionnés (voir Section 4.3.3). L'opérateur de suppression choisi est utilisé pour générer une nouvelle solution en supprimant $r$ clients de la solution actuelle (ligne 18).

Sur la ligne suivante, cette nouvelle solution est acceptée ou non suivant le critère d'acceptation (voir Section 4.3.2). Si la nouvelle solution est meilleure que la solution actuelle alors, le score de l'opérateur de suppression sélectionné est mis à jour (voir 4.3.3) comme monté sur les lignes 20–22.

La sélection d'un opérateur d'insertion est très semblable à celle d'un opérateur de suppression (lignes 23–30). La seule différence est que la phase d'évaluation dure $5 \cdot nb_{ih}$ itérations, où $nb_{ih}$ est le nombre d'opérateurs d'insertion. $L_{ins}$ représente la liste des opérateurs d'insertion. Sur la ligne 31, une nouvelle solution est générée en ajoutant quelques clients (ou aucun client) à la solution actuelle.

La nouvelle solution est acceptée ou pas selon le critère d'acceptation (voir Section 4.3.2). Si la nouvelle solution est plus profitable que la solution actuelle, le score de l'opérateur d'insertion est mis à jour (lignes 33–38). De plus, si le score de l'opérateur de suppression n'a pas été mis à jour lors de l'itération actuelle, la procédure met également à jour ce score (4.3.3).

La température actuelle $T$, qui est initialisée à 1, est décrémentée en fonction de l'Expression $T = T \cdot c$ en utilisant le coefficient de décroissance $c \in ]0, 1[$ (ligne 39). Si le seuil de température $T_0 \leq 1$ est atteint (lignes 40–42), la valeur de la température actuelle $T$ est incrémenté en fonction de la Formule $T = itr \cdot 10$, où $itr$ est le nombre actuel d'itérations effectuées (et qui est incrémenté à chaque itération).

Comme mentionné dans la revue de la littérature, Li et al. [65] ont développé une nouvelle variante d'ALNS pour le PDPTWPR. Dans ce qui suit, nous soulignons les caractéristiques principales de notre heuristique qui ne figurent ni dans l'ALNS classique proposée par Pisinger et Ropke [78] ni dans l'ALNS proposée par Li et al. [65]. *i*) sALNS évalue les solutions et met à jour les scores des opérateurs après chaque appel d'opérateur de suppression et chaque appel d'opérateur d'insertion (voir 4.3.2) ; *ii*) sALNS alterne deux phases différentes pour sélectionner les opérateurs (voir 4.3.3) ; *iii*) sALNS utilise des combinaisons distinctes d'opérateurs de suppression et insertion, qui prennent en compte les profits et les variations des chargements dans leurs critères de sélection (voir 4.3.4) ; *iv*) sALNS redémarre périodiquement depuis une température initiale plus élevée afin d'ajouter plus de diversification à la recherche (voir 4.3.5). De plus, le mécanisme de diversification noté *meta-destroy* proposé par Li et al. [65] peut

**Algorithme 10** *sALNS*

**Entrées:**
Une instance du PTPSPD
Une liste $L_{rem}$ d'opérateurs de suppression
Une liste $L_{ins}$ d'opérateurs d'insertion
$lev = 0$ ; $itr = 0$ ;$T = 1$
**Sorties:**
La meilleure solution rencontrée
**Init**
Générer une solution initiale à l'aide de l'heuristique de construction (4.3.1)
**tq** $itr < IS$ **faire**
  **si** $lev = ml$ **alors** ▷ début de la phase d'évaluation
    Réinitialiser les scores ;
    $lev = 0$ ;
  **fin si**
  Choisir aléatoirement $r$ dans $\{a_1, \ldots, a_k\}$ ;
  **si** $lev < 5 \cdot nb_{rh}$ **alors** ▷ phase d'évaluation
    **si** $L_{rem}$ est vide **alors**
      Remplir $L_{rem}$ avec les opérateurs de suppression ;
    **fin si**
    Choisir aléatoirement un opérateur de suppression $R$ de $L_{rem}$ et le supprimer de la liste ;
  **sinon** ▷ phase d'application
    Choisir un opérateur de suppression $R$ en fonction de son score actuel ; (voir 4.3.3)
  **fin si**
  Générer une nouvelle solution en supprimant $r$ clients à l'aide de $R$ ;
  Accepter ou non la nouvelle solution en fonction du critère d'acceptation ; (voir 4.3.2)
  **si** la nouvelle solution est meilleure que l'actuelle **alors**
    Mettre à jour le sore d'$R$ ; (voir 4.3.3)
  **fin si**
  **si** $lev < 5 \cdot nb_{ih}$ **alors** ▷ phase d'évaluation
    **si** $L_{ins}$ est vide **alors**
      Remplir $L_{ins}$ des opérateurs d'insertion ;
    **fin si**
    Choisir aléatoirement un opérateur d'insertion $I$ de $L_{ins}$ et le supprimer de la liste ;
  **sinon** ▷ phase d'application
    Choisir un opérateur d'insertion $I$ en fonction de son score actuel ; (voir 4.3.3)
  **fin si**
  Générer une nouvelle solution en appliquant $I$ à la solution actuelle ;
  Accepter ou non la solution en fonction du critère d'acceptation ; (voir 4.3.2)
  **si** la nouvelle solution est meilleure que l'actuelle **alors**
    Mettre à jour le score d'$I$ ; (voir 4.3.3)
    **si** le score d'$R$ n'a pas été mis à jour **alors**
      Mettre à jour le score d'$R$ ; (voir 4.3.3)
    **fin si**
  **fin si**
  $T = T \cdot c$ ; $lev++$ ; $itr++$ ;
  **si** $T < T_0$ **alors**
    $T = itr \cdot 10$
  **fin si**
**fin tq**

être considéré comme un cas particulier de notre évaluation des opérateurs (voir 4.3.5).

Dans les Sections suivantes, nous donnons plus de détails concernant nos composantes, il s'agit de l'heuristique de construction, la procédure d'évaluation des solutions, la procédure de sélection des opérateurs, les opérateurs de suppression/insertion et les stratégies de diversification.

### 4.3.1 Heuristique de construction

Dans le but de générer une solution initiale pour notre problème, nous avons implémenté une heuristique assez proche de l'heuristique de construction présentée pour le CPTP (voir Section 3.3.1). Tout comme dans l'heuristique de Solomon [90], notre heuristique de construction pour le PTPSPD possède deux critères principaux $cr_1(i,u,j)$ et $cr_2(i,u,j)$. Le pseudo-code de cette heuristique est donné par l'Algorithme 11.

**Algorithme 11** Heuristique de construction pour le PTPSPD

**Entrées:**
  Une instance de PTPSPD
  Une liste $L_{unr}$ de clients ne faisant pas partie de la solution
  Un nombre $nbRoutes = 0$ de routes présentes dans la solution actuelle
**Sorties:**
  Une solution réalisable
**tq** $nbRoutes <$ nombre de véhicules **faire**
  Créer une nouvelle route ;
  $nbRoutes++$ ;
  Ajouter un « client graine » à la nouvelle route ;
  **tq** $\exists u \in L_{unr}$ pouvant être inséré sans mener à une solution irréalisable **faire**
    Evaluer l'insertion de chaque client non visité $u \in L_{unr}$ dans la route actuelle ;
    Sélectionner, pour chaque $u$, la meilleure position d'insertion selon le critère $cr_1(i,u,j)$ ;
    Sélectionner la meilleure position d'insertion du meilleur client $u^*$ selon le critère $cr_2(i,u,j)$ ;
    Insérer le client $u^*$ dans sa meilleure position dans la route actuelle ;
    Mettre à jour $L_{unr}$ ;
  **fin tq**
**fin tq**

L'heuristique commence la recherche d'une solution réalisable à partir d'une route vide. La construction de cette route commence à partir d'un « client graine ». Puis, l'insertion des clients non encore visités est considérée. Pour chaque client $u$, l'heuristique choisit sa meilleure position d'insertion entre deux clients (ou le dépôt et un client) consécutifs $i$ et $j$ selon un certain critère noté $cr_1(i,u,j)$. Enfin, l'heuristique choisit l'insertion qui optimise un second critère, noté $cr_2(i,u,j)$, parmi les meilleures positions d'insertion trouvées. Ceci est répété jusqu'à ce qu'il n'y ait plus d'insertions possibles de clients dans la route. Les autres routes sont construites de la même manière. Les critères de sélection de l'heuristique proposée comprennent les profits ainsi que les coûts de transport.

De même que dans la Section 3.3.1, dénotons par $(i_0, i_1, \ldots, i_h)$ la route traitée, où $i_\rho$ représente le client placé à la $\rho^{\text{ième}}$ position de la route si $\rho \notin \{0,h\}$, et le dépôt sinon ($i_0 = i_h = 0$). Pour chaque client $u$, l'heuristique sélectionne sa meilleure position d'insertion dans la route suivant les Expressions (4.12)–(4.15), où (sans perte de généralités) $c_{ij}$ représente le coût de transport entre $i$ et $j$, $pr_u$ est le profit du client $u$, et $\alpha_1, \alpha_2, \mu \geq 0$ sont des paramètres définis par l'utilisateur, avec $\alpha_1 + \alpha_2 = 1$ et $\mu \in [0,3]$.

$$cr_1(i(u), u, j(u)) = \max\left\{cr_1(i_{\rho-1}, u, i_\rho), \rho = 1, \ldots, h\right\}; \tag{4.12}$$
$$cr_1(i, u, j) = \alpha_1 \cdot cr_{11}(i, u, j) - \alpha_2 \cdot cr_{12}(i, u, j); \tag{4.13}$$
$$cr_{11}(i, u, j) = pr_u; \tag{4.14}$$
$$cr_{12}(i, u, j) = c_{iu} + c_{uj} - \mu \cdot c_{ij}. \tag{4.15}$$

Afin d'éviter de laisser l'insertion des clients difficiles aux dernières itérations, un second critère $cr_2(i, u, j)$ est utilisé. $cr_2(i, u, j)$ force en quelques sortes l'insertion du client non visité le plus éloigné du dépôt. Plus précisément, parmi les clients non visités, l'optimisation du second critère mène à la sélection du meilleur client $u^*$ suivant les Expressions (4.16) et (4.17). Dans (4.17), $c_{0u}$ représente le coût de transport entre le dépôt et le client $u$, $\lambda \in [0, 1]$ est un paramètre défini par l'utilisateur, et qui règle l'importance du terme $c_{0u}$ lors du choix du client à insérer.

$$cr_2(i(u^*), u^*, j(u^*)) = \max\left\{cr_2(i(u), u, j(u)), u \text{ non visité}\right\}; \tag{4.16}$$
$$cr_2(i(u), u, j(u)) = \lambda \cdot c_{0u} + cr_1(i(u), u, j(u)). \tag{4.17}$$

En plus de la sélection de « clients graines » basé sur le client le plus éloigné du dépôt avec le plus grand profit $max_u \{pr_u + c_{0u}, u \text{ non visité}\}$, nous avons ajouté deux types de sélection des « clients graines ». Dans le premier type, nous choisissons le client avec la plus grande valeur de $pr_u - (c_{0u} + c_{u0})$ tandis que, dans le second type, nous choisissons le client avec le plus grand profit $pr_u$. Un paramètre *type_Select* défini par l'utilisateur est ajouté afin de déterminer la sélection des « clients graines » qui sera prise en considération. *type_Select* peut prendre les valeurs 1 pour la plus grande valeur de $(pr_u + c_{0u})$, 2 pour la plus grande valeur de $pr_u - (c_{0u} + c_{u0})$, et 3 pour le plus grand profit.

L'heuristique proposée est exécutée 10 fois, chaque fois avec une valeur différente des paramètres $\alpha_1$, $\alpha_2$, $\mu$, $\lambda$ et *type_Select*. Le but étant d'avoir différentes solutions initiales en variant les valeurs des paramètres dans l'heuristique de construction. La meilleure solution rencontrée est alors considérée comme point de départ de l'algorithme principal. Notons que, si les paramètres $\alpha_1$, $\alpha_2$, $\mu$, $\lambda$ et *type_Select* sont fixés à $\frac{1}{2}$, $\frac{1}{2}$, 1, 0 et 2 respectivement, l'heuristique insert, parmi les clients non encore visités, ceux qui mènent à la plus grande valeur de la fonction objectif du PTPSPD. Notons que, ceci ne peut être vrai si les paramètres de l'heuristique avaient été normalisés.

### 4.3.2 Évaluation des solutions

L'aspect sélectif du PTPSPD pousse à considérer une solution « détruite » (obtenue par un opérateur de suppression) comme une solution réalisable. Cette dernière peut même être plus profitable qu'une solution « construite » (obtenue via un opérateur d'insertion). En effet, si nous supprimons certains clients d'une solution réalisable donnée, aucune contrainte du problème ne serait violée. Pour cette raison, et contrairement aux

autres heuristiques de type ALNS de la littérature, sALNS évalue les solutions et met à jour les scores des opérateurs après les opérateurs de suppression également. Notons que, dans l'ALNS classique et dans l'ALNS proposée par Li et al. [65], les solutions de cet espace de recherche ne seraient pas atteintes.

L'évaluation de la solution est effectuée avec le principe du *recuit simulé* (voir les travaux de Kirkpatrick et al. [60] et Černỳ [17]). Ainsi, une solution est acceptée si elle est plus profitable que la meilleure solution rencontrée ou la solution actuelle, ou alors, si la **règle de Métropolis** est satisfaite. Autrement dit, les mauvaises solutions sont acceptées avec la probabilité donnée par la Formule (4.18).

$$\exp^{\left(f(x')-f(x)\right)/T} \tag{4.18}$$

où $f()$ représente la fonction objectif, $x'$ et $x$ représentent la nouvelle et l'actuelle solutions respectivement et $T > 0$ est la température actuelle.

Notons que, la règle de Métropolis est appliquée uniquement après les opérateurs d'insertion car nous avons remarqué que cela mène à de meilleurs résultats. Si une mauvaise solution n'est pas acceptée, sALNS continue avec la solution précédemment acceptée (pas forcément la meilleure). À chaque itération, la température actuelle $T$ est décrémentée à l'aide du coefficient de décroissance $c \in ]0, 1[$ suivant l'Expression $T = T \cdot c$.

### 4.3.3 Sélection des opérateurs

Tout au long du processus de recherche, sALNS utilise alternativement deux phases pour permettre une bonne sélection des opérateurs, il s'agit des phases d'**évaluation** et d'**application**.

Dans chaque phase d'évaluation, les opérateurs sont aléatoirement sélectionnés indépendamment de leurs performances (ou scores). Au début de cette phase, les scores de tous les opérateurs sont ré-initialisés, puis recalculés. Dans chaque phase d'application, la sélection des opérateurs dépend de leurs scores précédemment calculés. Les scores sont également calculés durant les phases d'application.

Contrairement aux autres ALNS de la littérature, sALNS ne divise pas la recherche en segments (nombre donné d'itérations dans lesquelles de nouveaux scores sont calculés) et attend la fin d'un segment actuel avant d'utiliser les nouveaux scores pour sélectionner les opérateurs. Au lieu de cela, et plus précisément dans une phase d'application d'sALNS, les scores obtenus des itérations précédentes (incluant la phase d'évaluation précédente et la phase d'application actuelle) sont considérés pour la sélection des opérateurs.

#### Mise à jour des scores

Dans les phases d'évaluation et d'application, les scores sont calculés de la même façon. Ce calcul diffère des autres mises à jour des scores de la littérature.

Plus précisément, un vecteur *vec* est utilisé pour sauvegarder les scores. Au début de la recherche, ce vecteur est vide, et à chaque fois qu'un opérateur donne un bon résultat (fournit une meilleure solution que l'actuelle), le nom de cet opérateur est

considéré comme un nouvel élément de *vec*. Ceci équivaut à mettre à jour les scores des opérateurs étudiés en les incrémentant. Ainsi, lorsque sALNS doit sélectionner un opérateur selon ses scores précédents, un nombre *rdm* est aléatoirement choisi depuis l'ensemble $\{0, \ldots, s_{vec} - 1\}$ où $s_{vec}$ représente le nombre d'éléments non vides de *vec*. Le nom de l'opérateur sélectionné est celui du $rdm^{\text{ième}}$ élément de *vec*. Donc, les opérateurs ayant de grands scores ont plus de chances d'être sélectionnés.

Remarquons que les scores des opérateurs de suppression et d'insertion sont mis à jour séparément. Plus précisément, chaque fois qu'un opérateur de suppression donne un bon résultat, son score est mis à jour. Lorsqu'une solution est examinée après un opérateur d'insertion, nous pouvons distinguer deux cas. Dans le premier cas, la solution a déjà été améliorée à l'aide d'un opérateur de suppression. Ici, seul le score de l'opérateur d'insertion est mis à jour. Cependant, si l'opérateur de suppression n'a pas fourni un bon résultat mais que l'opérateur d'insertion si, les scores des deux opérateurs sont mis à jour. Cette différente mise à jour des scores entre les suppression et les insertions réside dans le fait que nous ne savons pas quel opérateur a mené au résultat trouvé.

#### Durées des phases

Les phases d'évaluations sont exécutées pour un certain nombre d'itérations noté *eval*, tel que $eval = nb_H \cdot l$, où $l$ est une constante fixée par l'utilisateur, et $nb_H$ est le nombre d'opérateurs utilisés. Lors des calculs, $nb_H$ prend la valeur $nb_{rh} = 6$ pour les suppressions ($nb_{rh}$ est le nombre d'opérateurs de suppression utilisés dans sALNS) et la valeur $nb_{ih} = 4$ pour les insertions ($nb_{ih}$ est le nombre d'opérateurs d'insertion d'sALNS). Après chaque phase d'évaluation, sALNS retourne à la phase d'application. Chaque *ml* itérations, l'heuristique retourne de nouveau à la phase d'évaluation. Le paramètre *ml* est défini par l'utilisateur.

### 4.3.4 Opérateurs de suppression/insertion

Afin de créer une approche adaptée au PTPSPD, nous avons modifié certains opérateurs de l'ALNS classique. Dans ce qui suit, nous donnons une description des opérateurs utilisés.

#### Opérateurs de suppression

Les opérateurs de suppression suppriment un certain nombre de clients (noté $r$) de la solution actuelle. Dans sALNS, six opérateurs de suppression sont utilisés. Ces derniers sont inspirés de ceux présentés par Pisinger et Ropke [78]. Ils consistent en le *random removal*, le *worst removal*, le *related removal*, l'*historical node-pair removal*, l'*historical request-pair removal* et le *cluster removal*. Les critères de sélection du *related removal* et du *cluster removal* sont modifiés dans le présent travail. Pour les autres opérateurs de suppression, la fonction objectif du PTPSPD est considérée au lieu de la minimisation des coûts de transport.

Dans le travail présenté par Pisinger et Ropke [78], deux paramètres de contrôle de l'aléatoire sont présentés. Le premier (noté $p_w$) contrôle le *worst removal*, alors que le second (noté $p_r$) contrôle le *related removal*. Pour les petites valeurs de ces paramètres de contrôle de l'aléatoire, l'opérateur de suppression impliqué supprime le meilleur client en termes de critère de sélection de cet opérateur. Alors que, pour de plus grandes

valeurs de ces paramètres, l'opérateur impliqué supprime les clients de mauvaise qualité en termes de critère de sélection de l'opérateur. Ainsi, les grandes valeurs de ce paramètre induisent plus de diversification. Dans notre travail, le paramètre de contrôle de l'aléatoire est considéré pour tous les opérateurs de suppression.

Les détails concernant notre *related removal* et notre *cluster removal* sont donnés dans les paragraphes suivants. Les autres opérateurs de suppression sont les mêmes que ceux décrits dans la Section 3.3.4 avec la fonction objectif du PTPSPD.

***Related removal :*** Cet opérateur vise à supprimer les clients qui sont en quelques sortes similaires afin de pouvoir échanger leurs positions facilement à l'aide des opérateurs d'insertion, générant ainsi une nouvelle solution. Dans le présent travail, cette similarité (appelée aussi relatedness) est calculée suivant la Formule (4.19)

$$r_{ij} = \left| pr_i^{'} - pr_j^{'} \right| + c_{ij}^{'} \tag{4.19}$$

où $r_{ij}$ désigne la similarité entre un client $i$ et un client $j$. Pour un client $i$ donné, $pr_i^{'} = pr_i / pr_{biggest}$, où $pr_i$ représente le profit de $i$ et $pr_{biggest}$ est le plus grand profit de l'instance étudiée. $c_{ij}^{'} = c_{ij}/c_{max}$, où $c_{ij}$ est le coût de transport entre $i$ et $j$, et $c_{max}$ représente le plus grand coût de transport entre deux sommets dans l'instance étudiée. Notons que $pr_i^{'}$, $pr_j^{'}$ et $c_{ij}^{'}$ appartiennent à $[0, 1]$ quels que soient les clients $i$ et $j$.

Cette normalisation permet d'utiliser des valeurs comparables pour les profits et les coûts de transport. Donc, un client $j$ est similaire à un client $i$ si $r_{ij}$ est relativement petite. Cette similarité favorise la suppression de clients qui ne se trouvent pas loin les uns des autres et qui ont presque les mêmes profits.

***Cluster removal :*** Cet opérateur supprime $r$ clients de la solution. La première route étudiée est aléatoirement sélectionnée. Ensuite, l'algorithme de Kruskal pour le problème de l'arbre couvrant de poids minimum (Minimum Spanning Tree problem ou MST en anglais, voir la Section 2.2.4) est utilisé pour déterminer les clients à supprimer de la route. Dans le *cluster removal*, l'algorithme de Kruskal détermine l'MST sur un sous-graphe défini par les sommets et les arcs de la route actuelle. Le poids de chaque arc (les poids sont utilisés pour calculer l'MST) est défini par la similarité entre les deux clients de ses extrémités (initiale et terminale). L'algorithme de Kruskal s'arrête lorsque deux composantes disjointes (clusters en anglais) sont retrouvées lors de la construction de l'MST. Après cela, l'une des deux composantes est aléatoirement supprimée. Puis, un client $i$ est aléatoirement sélectionné depuis la composante supprimée. $j$, le client le plus similaire à $i$, est sélectionné depuis une route différente de celle de $i$. La route de $j$ est alors considérée comme la prochaines route à étudier. L'opérateur continue de cette façon jusqu'à la suppression d'$r$ clients. Notons que la similarité entre deux clients $i$ et $j$ est définie de la même façon que celle décrite dans *related removal* (voir le paragraphe précédent).

**Opérateurs d'insertion**

Nous avons implémenté quatre opérateurs d'insertion, il s'agit de *basic greedy 1*, *basic greedy 2*, *regret* et *H_insert*. Pisinger et Ropke [78] n'ont utilisé qu'un seul type de *basic greedy* et un ensemble d'opérateurs de *regret* qui diffèrent les uns des autres dans le degré de regret (nombre de meilleures positions d'insertion examinées pour chaque client). Li et al. [65] ont également utilisé *basic greedy* et l'ensemble des opérateurs de *regret* avec la fonction objectif du PDPTWPR comme critère de sélection. Dans le présent travail, nous proposons deux différents types de *basic greedy* ainsi qu'un opérateur de *regret*. Ces opérateurs considèrent, à chaque itération, plusieurs routes à la fois (le nombre de routes est égal au nombre de véhicules) lors de l'insertion d'un client donné. Afin de gérer les contraintes du PTPSPD, les deux *basic greedy* et l'opérateur de *regret* incluent des critères de sélection basés sur les profits, les coûts de transport et les chargements des véhicules. Afin d'éviter d'avoir des temps de calcul trop importants, seul l'opérateur *regret-2*, qui considère dans ses critères de sélection la seconde meilleure position d'insertion, est utilisé.

D'un autre côté, *H_insert* a été proposée par Hifi et Wu [51] dans une métaheuristique hybride pour tenter de résoudre le VRPTW. Dans le présent travail, *H_insert* est adaptée au PTPSPD. Au meilleur de notre connaissance, c'est la première fois que l'heuristique *H_insert* est utilisée au sein d'un algorithme de type ALNS.

Dans nos opérateurs d'insertion, la contrainte d'insertion de tous les clients est relâchée. De plus, s'il y a moins de routes dans la solution actuelle que le nombre de véhicules disponibles, l'insertion de clients au sein de routes vides est aussi considérée.

Une description de *basic greedy 1*, *basic greedy 2*, *regret* et *H_insert* est donnée dans ce qui suit.

***Basic greedy 1 :*** Cet opérateur insert chaque client dans sa meilleure position d'insertion, et dans sa meilleure route. Comme dit plus haut, *basic greedy 1* inclue dans ses critères de sélection les profits, les coûts de transport et les chargements des véhicules. Plus précisément, les critères d'insertion de *basic greedy 1* sont définis de manière assez similaire à ceux présentés dans la Section 4.3.1. La différence est que le premier critère $cr_1^{global}(i,u,j)$ est défini suivant les Expressions (4.20)–(4.24), où $\rho$ peut appartenir à n'importe quelle route de l'ensemble des routes disponibles, $\alpha_3 \geq 0$ est un paramètre défini par l'utilisateur, $Max\_LB(i)$ et $Max\_LA(i)$ sont le chargement maximum du véhicule collecté avant et après le client $i$ respectivement, les termes/paramètres restants sont les mêmes que ceux définis dans la Section 4.3.1.

$$cr_1^{global}(i(u),u,j(u)) = \max\left\{cr_1^{global}(i_{\rho-1},u,i_\rho,\rho=1,\ldots,h\right\}; \tag{4.20}$$

$$cr_1^{global}(i,u,j) = \alpha_1 \cdot cr_{11}(i,u,j) - \alpha_2 \cdot cr_{12}(i,u,j) - \alpha_3 \cdot cr_{13}(i,u,j); \tag{4.21}$$

$$cr_{11}(i,u,j) = pr_u; \tag{4.22}$$

$$cr_{12}(i,u,j) = c_{iu} + c_{uj} - \mu \cdot c_{ij}; \tag{4.23}$$

$$cr_{13}(i,u,j) = max\left\{Max\_LB(i) + d_u;\ Max\_LA(i) + p_u\right\}. \tag{4.24}$$

Le second critère parmi toutes les routes $cr_2^{global}(i,u,j)$ est défini suivant les Expressions (4.25) et (4.26), où $\lambda$ et $c_{0u}$ sont définis de la même façon que dans la Section 4.3.1.

$$cr_2^{global}(i(u^*), u^*, j(u^*)) = \max\left\{cr_2^{global}(i(u), u, j(u)), u \text{ non visité}\right\}; \quad (4.25)$$

$$cr_2^{global}(i, u, j) = \lambda \cdot c_{0u} + cr_1^{global}(i(u), u, j(u)). \quad (4.26)$$

Remarquons que, lorsque la valeur d'$\alpha_3$ est fixée à 1 et que les autres paramètres sont fixés à 0, l'opérateur sélectionne le client qui produit le plus petit chargement maximum dans le véhicule.

Après quelques tests expérimentaux, nous avons remarqué que la configuration $\alpha_1 = \alpha_2 = \alpha_3 = 1$ donne de meilleurs résultats en comparaison avec les configurations où $\alpha_1 + \alpha_2 + \alpha_3 = 1$. Ainsi, les valeurs des paramètre de *basic greedy 1* sont réglées comme suit $\alpha_1 = \alpha_2 = \alpha_3 = 1$, $\mu$ et $\lambda$ sont aléatoirement générés des intervalles $[0, 3]$ et $[0, 1]$ respectivement.

***Basic greedy 2 :*** Cet opérateur est pratiquement le même que *basic greedy 1*. La différence entre les deux opérateurs réside dans le réglage des paramètres. En effet, dans *basic greedy 2*, $\alpha_1 = \alpha_2 = \alpha_3 = 1$, $\mu = 1$ et $\lambda$ est choisi aléatoirement de l'intervalle $[0, 1]$. On peut remarquer que *basic greedy 1* génère plus de diversification que *basic greedy 2*.

***Opérateur de regret :*** L'opérateur de regret sélectionne les clients qui seront les plus regrettés s'ils ne sont pas insérés à l'itération actuelle. Plus précisément, soit $cr_1^{global}(i, u, j)$ le premier critère d'insertion d'un client $u$ entre $i$ et $j$, en considérant toutes les routes disponibles, comme défini dans les Expressions (4.20)–(4.24). Le second critère considérant toutes les routes disponibles $cr_2^{global}(i, u, j)$ est alors défini suivant les Expressions (4.25) et (4.26). Soit $\zeta$ la meilleure route de la meilleure insertion, obtenue en évaluant les Expressions (4.25) et (4.26) et en tenant compte de toutes les routes. Définissions $cr_1^{global-\zeta}(i, u, j)$ comme étant le premier critère d'insertion du client $u$ entre $i$ et $j$ en considérant toutes les routes sauf $\zeta$. Soit $cr_2^{global-\zeta}(i, u, j)$ le second critère correspondant. Le meilleur client à insérer suivant le critère $cr_2^{global-\zeta}(i, u, j)$ est obtenu en considérant toutes les routes sauf $\zeta$ suivant les Expressions (4.27) et (4.28).

$$cr_2^{global-\zeta}(i(u^*), u^*, j(u^*)) = \max\left\{cr_2^{global-\zeta}(i, u, j), u \text{ non visité}\right\}; \quad (4.27)$$

$$cr_2^{global-\zeta}(i, u, j) = \lambda \cdot c_{0u} + cr_1^{global-\zeta}(i(u), u, j(u)). \quad (4.28)$$

L'opérateur de *regret* insert le client qui sera le plus regretté, c'est à dire, celui avec la plus grande valeur de $cr_2^{global}(i(u^*), u^*, j(u^*)) - cr_2^{global-\zeta}(i(u^*), u^*, j(u^*))$.

L'opérateur de *regret* utilise les mêmes valeurs des paramètres que l'opérateur *basic greedy 2*.

***H_insert :*** Cet opérateur examine tous les clients non visités. Les clients sont examinés suivant leur ordre de suppression. À chaque fois qu'un client est considéré, *H_insert* l'insert dans sa meilleure position d'insertion parmi toutes les routes disponibles si son

insertion améliore la solution actuelle. Dans *H_insert*, une meilleure position est une position qui maximise la fonction objectif du PTPSPD. Notons que chaque client n'est examiné qu'une seule fois.

### 4.3.5 Stratégies de diversification

Plusieurs stratégies de diversification sont utilisées dans l'approche proposée. Elles sont détaillées dans ce qui suit.

#### Diversification via mise à jour de la température

À chaque fois que le seuil de température $T_0$ est atteint, le *recuit simulé* est appelé avec une plus grande température « initiale » $T = itr \cdot 10$, où $itr$ est le nombre d'itérations actuelles. Ainsi, sALNS peut être vue comme une sorte d'heuristique multi-départs dans laquelle les solutions « actuelles » deviennent de nouvelles solutions initiales. Lorsque sALNS est exécutée avec une plus grande température de départ, l'heuristique accepte plus de solutions de mauvaise qualité à cause de la règle de Métropolis. Certaines de ces « mauvaises solutions » aident l'algorithme à échapper aux optima locaux et mènent à des solutions plus profitables. Notons que, lorsque la température est réinitialisée, la probabilité de sélection des opérateurs de suppression/insertion est maintenue.

#### Diversification via les opérateurs de suppression

Dans le cas où une solution est acceptée après une suppression mais pas après une insertion, une nouvelle itération commence et un opérateur de suppression est exécuté de nouveau. La nouvelle solution est alors acceptée si elle est meilleure que l'ancienne solution, et l'opérateur d'insertion est exécuté. Cela résulte en une combinaison de deux suppression et une insertion, ce qui équivaut au mécanisme de diversification *meta-destroy* proposé par Li et al. [65]. De plus, dans sALNS, les solutions de plusieurs opérateurs de suppression successifs peuvent être acceptées avant de passer à une solution générée par un opérateur d'insertion. Cela constitue un grand avantage pour l'approche proposée, étant donné qu'il n'est plus nécessaire de tester la suppression d'un grand nombre de clients, ce qui peut engendrer un temps de calcul supplémentaire. Ainsi, l'approche reste rapide vu qu'elle commence par la suppression de petits nombres de clients, et s'il s'avère plus avantageux de supprimer plus de clients, cela se fera par l'acceptation des solutions de plusieurs opérateurs de suppression successifs. Cette stratégie mène à une recherche plus diversifiée.

#### Diversification via les termes de perturbation

Dans l'ALNS classique présentée par Pisinger et Ropke [78], un terme de perturbation (noise term en anglais) est inclus dans les critères de sélection des opérateurs d'insertion afin d'ajouter davantage de diversification. Dans le présent travail, ce terme de perturbation est aussi ajouté aux critères de sélection des opérateurs d'insertion et il est calculé de la même façon. En effet, à chaque fois que le coût d'insertion d'un client $u$ est calculé suivant le premier critère $cr_1^{global}(i,u,j)$ (tel que décrit dans les Expressions (4.20)–(4.24)), un terme de perturbation $\delta$ est ajouté pour générer une version modifiée du premier critère $cr_1^{'global}(i,u,j) = cr_1^{global}(i,u,j) + \delta$. $\delta$ est aléatoirement choisi depuis l'intervalle $[-N_{max}, N_{max}]$, $N_{max} = \eta \cdot \max\{c_{ij}, (i,j) \in E\}$, où $c_{ij}$ est le coût de transport

entre les clients $i$ et $j$, et $\eta$ est un paramètre de contrôle de la quantité de perturbation (défini par l'utilisateur).

Lorsqu'un opérateur d'insertion est sélectionné, l'algorithme décide s'il utilise le terme de perturbation ou non selon la performance des opérateurs d'insertion avec et sans perturbation. Cette décision est faite suivant le principe de la *roulette* (roulette wheel selection en anglais). Plus précisément, nous considérons $\omega^+$ et $\omega^-$ comme étant les poids de l'utilisation et de la non-utilisation des termes de perturbation respectivement. Un poids reflète la performance d'une version (avec ou sans perturbation) durant les itérations précédentes. Lors d'une itération courante, les décisions d'utilisation ou non de la perturbation sont faites suivant les probabilités $\frac{\omega^+}{\omega^+ + \omega^-}$ et $\frac{\omega^-}{\omega^+ + \omega^-}$ respectivement. $\omega^+$ et $\omega^-$ sont fixés à 1 au début de la recherche. Les valeurs d'$\omega^+$ et $\omega^-$ sont fixées jusqu'à ce que le couple de phases d'évaluation-application (noté e-a) soit terminé. À la fin de chaque e-a, les valeurs d'$\omega^+$ et $\omega^-$ sont ajustées suivant le score de l'utilisation de la perturbation (noté $sc_{\omega^+}$) et le score de la non-utilisation de la perturbation (noté $sc_{\omega^-}$) obtenus dans cette e-a respectivement. Au début de chaque e-a, les scores sont remis à la valeur 0. Les scores sont, après cela, incrémentés à chaque itération en ajoutant les valeurs $\sigma_1$, $\sigma_2$ ou $\sigma_3$.

$\sigma_1$ est ajoutée à $sc_{\omega^+}$ (ou à $sc_{\omega^-}$) si une meilleure solution par rapport à la meilleure solution globale est obtenue avec (ou sans) le terme de perturbation. $\sigma_2$ est ajoutée à $sc_{\omega^+}$ (ou à $sc_{\omega^-}$) si la valeur de la fonction objectif de la solution qui a été obtenue avec (ou sans) le terme de perturbation est meilleure que la valeur de la fonction objectif de la solution actuelle. Enfin, $\sigma_3$ est ajoutée à $sc_{\omega^+}$ (ou à $sc_{\omega^-}$) si une solution obtenue avec (ou sans) le terme de perturbation, est acceptée bien que la valeur de la fonction objectif de cette solution soit moins bonne que la valeur de la fonction objectif de la solution actuelle.

Comme mentionné plus haut, les valeurs d'$\omega^+$ et $\omega^-$ sont mises à jour à la fin de chaque e-a. $sc_{\omega^+}$ et $sc_{\omega^-}$ sont les scores de l'utilisation et de la non-utilisation de la perturbation à la fin de l'e-a actuelle respectivement. Soit $times_{\omega^+}$ le nombre de fois où un opérateur a utilisé le terme de perturbation, et soit $times_{\omega^-}$ le nombre de fois où un opérateur n'a pas utilisé le terme de perturbation dans l'e-a venant de se terminer. Soit $\rho \in [0, 1]$ un facteur de réaction qui contrôle la vitesse à laquelle l'algorithme réagit aux changements des scores. $\omega^+$ et $\omega^-$ sont alors ajustés suivant les Expressions (4.29) et (4.30). Notons que, si la valeur de $\rho$ est fixée à 1, la sélection dépend uniquement des scores obtenus durant l'e-a qui vient de se terminer. Si la valeur de $\rho$ est fixé à 0, la sélection dépend uniquement des poids obtenus lors des couples précédents d'e-a sans considérer les scores obtenus durant l'e-a qui vient juste de se terminer.

$$\omega^+ = (1 - \rho) \cdot \omega^+ + \rho \cdot \frac{sc_{\omega^+}}{times_{\omega^+}}. \tag{4.29}$$

$$\omega^- = (1 - \rho) \cdot \omega^- + \rho \cdot \frac{sc_{\omega^-}}{times_{\omega^-}}. \tag{4.30}$$

## 4.4 Résultats expérimentaux

Dans cette Section, nous comparons les résultats d'sALNS avec ceux de CPLEX 12.2, ceux de l'ALNS classique présenté par Pisinger et Ropke [78] et ceux d'une autre

version d'ALNS notée *Li*. *Li* est une variante d'sALNS qui utilise les opérateurs de suppression/insertion de l'ALNS proposée par Li et al. [65]. Nous n'avons pas pu implémenter l'ALNS proposée par Li et al. [65] pour le PTPSPD car les composantes de cette heuristique prennent en compte des caractéristiques du PDPTWPR qui n'existent pas pour le PTPSPD. Par exemple, dans un PDPTWPR, chaque véhicule possède un ensemble de clients optionnels qui peuvent être visités par d'autres véhicules, et un ensemble de clients réservés qui ne doivent être visités que par ce véhicule. De plus, l'ALNS proposée par Li et al. [65] emploie un mécanisme d'ajustement dynamique du comportement des opérateurs selon les statuts (optionnel ou réservé). Ce mécanisme ne peut être utilisé pour le PTPSPD. Nous ne pouvons pas non plus appliquer sALNS au PDPTWPR étant donné que cette heuristique peut supprimer certains clients réservés, ce qui donnera une solution irréalisable. Néanmoins, nous remarquons que les opérateurs de suppression/insertion de l'ALNS proposée par Li et al. [65] peuvent être appliqués au PTPSPD, si nous forçons les opérations de ramassage et de livraison de chaque client à être effectuées simultanément. Cela donnera donc un ensemble d'opérateurs de suppression/insertion qui sera différent que celui proposé dans ce Chapitre. Comme nous voulions tester l'ensemble des opérateurs de Li et al. [65] sur le PTPSPD, nous avons donc créé la version *Li* d'sALNS, qui utilise les opérateurs de Li et al. [65].

Dans ce qui suit, nous commençons par décrire les instances proposées. Nous donnerons, après cela, des détails concernant la configuration de l'heuristique de construction et de l'ALNS classique (voir le travail de Pisinger et Ropke [78]). Un réglage des paramètres, une évaluation des composantes et une comparaison entre sALNS, ALNS ([78]) et *Li* seront considérés. Par la suite, nous comparerons les résultats d'sALNS avec ceux de CPLEX 12.2 et de l'ALNS classique. Enfin, comme il y a quelques similarités entre notre modèle mathématique et celui présenté par Dell'Amico et al. [27], nous évaluerons la performance d'sALNS sur les instances proposées par Dell'Amico et al. [27]. sALNS est aussi évaluée sur les instances du CPTP.

Toutes les heuristiques sont implémentées en C et exécutées sur un ordinateur portable muni d'un intel(R) Core (TM) i7-4600U CPU @ 2.10 GHz 2.70 GHz avec 8.00 GB de RAM et 64-bit. CPLEX est également exécuté sur le même ordinateur. Les paramètres par default de CPLEX sont utilisés et le solveur est arrêté après deux heures de calcul.

À cause de l'aspect aléatoire des heuristiques implémentées, chacune d'elles est lancée 10 fois pour chaque instance. Chaque heuristique est lancée durant 900000 itérations.

Comme l'utilisation de 900000 itérations lors du réglage des paramètres pourrait prendre beaucoup trop de temps, nous n'utilisons que 2000 itérations avec 10 exécution chacune pour ce réglage. Concernant l'évaluation des composantes d'sALNS, nous voulons voir jusqu'où peut aller chaque configuration d'sALNS en fonction des composantes incluses. Nous fixons alors le nombre d'itérations lors de l'évaluation des composantes à 200000 et le nombre d'exécution à 5.

### 4.4.1 Génération d'instances pour le PTPSPD

Afin de générer des instances pour le PTPSPD, nous avons modifié celles du CPTP proposées par Archetti et al. [2] et décrites dans la Section 3.4.1.

Afin d'adapter les instances du CPTP au PTPSPD, nous générons les demandes

de ramassage et de livraison en utilisant la méthode présentée par Salhi et Nagy [84]. Cette dernière a initialement été proposée pour le VRPSPD. Pour générer leur groupe d'instances, Salhi et Nagy [84] ont modifié celles de Christofides et al. [21] qui étaient initialement utilisées pour le *problème de tournées de véhicules avec contraintes de capacité*. Pour chaque client, les auteurs ont donc créé une demande de ramassage et une demande de livraison. Plus précisément, leur méthode commence par calculer un ratio $r_i$ pour chaque client $i$ comme étant le minimum entre $x_i/y_i$ et $y_i/x_i$ où $(x_i, y_i)$ sont les coordonnées du client $i$. Après cela, la demande de livraison $d_i$ est calculée suivant la Formule $d_i = r_i \cdot dem_i$ et le ramassage $p_i$ par $p_i = (1 - r_i) \cdot dem_i$ où $dem_i$ est la demande initiale du client $i$.

Les demandes moyennes de ramassage et de livraison (respectivement $moyp_i$ et $moyd_i$) des instances du PTPSPD, et qui sont basées sur le premier ensemble d'instances d'Archetti et al. [2] (avec les nombres initiaux de véhicules et limites de capacités) sont données dans le Tableau 4.2.

Le Tableau 4.2 rapporte également le nombre de clients $n$, la limite de capacité $Q$, le nombre de véhicules $m$, la valeur minimum des livraisons $\min d_i$, la valeur maximum des livraisons $\max d_i$, la valeur minimum des ramassages $\min p_i$ et la valeur maximum des ramassages $\max p_i$ pour chaque instance. Le nombre de clients et l'information concernant les demandes de livraison et de ramassage des instances restantes sont les mêmes que celles décrites dans le Tableau 4.2.

Pour les mêmes raisons que celles citées dans la Section 3.4.1. Nous ne prenons en compte que les instances de type « 8 » au lieu de prendre « 8 » et « 3 ». Nous obtenons un total de 117 instances. Notons que nous considérons les instances avec des demandes non-arrondies.

TABLE 4.2 – *Description des instances du PTPSPD dérivées du premier ensemble d'instances présentées par Archetti et al. [2].*

| Instance | n | Q | v | $moyd_i$ | $\min d_i$ | $\max d_i$ | $moyp_i$ | $\min p_i$ | $\max p_i$ |
|---|---|---|---|---|---|---|---|---|---|
| 6 | 50 | 160 | 10 | 9.210 | 0.86 | 30.00 | 6.330 | 0.00 | 22.84 |
| 7 | 75 | 140 | 20 | 10.892 | 0.25 | 31.31 | 7.295 | 0.00 | 22.60 |
| 8 | 100 | 200 | 15 | 8.379 | 0.04 | 29.82 | 6.201 | 0.00 | 27.35 |
| 9 | 150 | 200 | 10 | 8.656 | 0.04 | 30.00 | 6.244 | 0.00 | 27.35 |
| 10 | 199 | 200 | 20 | 9.402 | 0.04 | 31.31 | 6.608 | 0.00 | 27.35 |
| 13 | 120 | 200 | 15 | 5.092 | 0.08 | 18.09 | 6.367 | 0.00 | 24.00 |
| 14 | 100 | 200 | 10 | 9.420 | 0.00 | 40.00 | 8.680 | 0.00 | 32.00 |
| 15 | 150 | 200 | 15 | 8.656 | 0.04 | 30.00 | 6.244 | 0.00 | 27.35 |
| 16 | 199 | 200 | 20 | 9.402 | 0.04 | 31.31 | 6.608 | 0.00 | 27.35 |

### 4.4.2 Configuration de l'heuristique de construction

Les paramètres de l'heuristique de construction sont aléatoirement choisis depuis leurs intervalles respectifs. Voir Section 4.3.1 pour plus de détails concernant les intervalles utilisés.

Cette heuristique est exécutée 10 fois. Notons que, les valeurs des paramètres peuvent changer à chaque exécution. La meilleure solution rencontrée est considérée comme point de départ de l'algorithme principal.

### 4.4.3 Configuration de l'ALNS classique

Nous avons implémenté une version de l'ALNS classique proposée par Pisinger et Ropke [78] adaptée au PTPSPD. Comme la version classique utilise la minimisation des coûts de transport sans considérer les profits, nous avons remplacé ses opérateurs de suppression par ceux présentés dans la Section 4.3.4. Cette version adaptée d'ALNS utilise aussi cinq insertions. La première insertion est *basic greedy* avec des valeurs de paramètres égales à 1 pour $\mu$, $\alpha_1$ et $\alpha_2$, et à 0 pour $\alpha_3$ et $\lambda$. Les autres insertions sont *regret-2*, *regret-3*, *regret-4* et *regret-m*, où $m$ est le nombre de véhicules. Les opérateurs de *regret* considèrent les mêmes valeurs des paramètres que *basic greedy*. Notons qu'avec ces valeurs des paramètres, nous obtenons les mêmes insertions que celles de l'ALNS classique utilisant la fonction objectif du PTPSPD. La sélection des opérateurs, la mise à jour des scores, l'évaluation des solutions et la mise à jour des températures sont identiques à celles utilisées dans l'ALNS classique. Comme il n'y a pas de modifications majeures dans cette version adaptée de l'ALNS classique, elle sera notée *ALNS classique* ou simplement *ALNS* dans le reste de ce document.

#### Réglage des paramètres pour ALNS

Les paramètres de l'ALNS classique sont décrits dans le Tableau 4.3. $p_r$ et $p_w$ sont les paramètres de contrôle de l'aléatoire comme décrits dans la Section 4.3.4. $T_s$ est la température de départ d'ALNS et $c$ est le coefficient de décroissance qui sert à réduire la valeur de la température actuelle à chaque itération. $q$ est un nombre aléatoire qui représente le nombre de clients à supprimer lors de chaque itération. Les définitions des paramètres $\sigma_1$, $\rho$ et $\eta$ peuvent être retrouvées dans la Section 4.3.5, en considérant l'évaluation de la performance de différents opérateurs au lieu de l'évaluation de la performance d'utilisation ou non du terme de perturbation. Les définitions de $\sigma_2$ et $\sigma_3$ sont cependant un peu différentes. En effet, $\sigma_2$ et $\sigma_3$ sont ajoutées au score d'un opérateur si une solution qui n'a pas été acceptée avant est obtenue avec l'opérateur en question, de telle sorte que la valeur de la fonction objectif est : $i$) meilleure que celle de la solution actuelle pour $\sigma_2$, et $ii$) moins bonne que celle de la solution actuelle, mais la solution est quand même acceptée pour $\sigma_3$. Pour vérifier si une solution a déjà été acceptée ou non, les valeurs de la fonction objectif des solutions précédentes sont sauvegardées.

Table 4.3 – *Description des paramètres d'ALNS.*

| paramètre | description |
|---|---|
| $p_r$ | Le paramètre de contrôle de l'aléatoire du *related removal* |
| $p_w$ | Le paramètre de contrôle de l'aléatoire du *worst removal* |
| $T_s$ | La température initiale |
| $c$ | Le coefficient de décroissance |
| $\sigma_1$ | Les scores des opérateurs de suppression/insertion sont incrémentés avec $\sigma_1$ s'ils donnent une nouvelle meilleure solution globale |
| $\sigma_2$ | Les scores des opérateurs de suppression/insertion sont incrémentés avec $\sigma_2$ s'ils donnent une solution qui n'a pas été acceptée précédemment et la valeur de la fonction objectif de cette solution est meilleure que celle de la solution actuelle |
| $\sigma_3$ | Les scores des opérateurs de suppression/insertion sont incrémentés avec $\sigma_3$ s'ils donnent une solution qui n'a pas été acceptée auparavant, la valeur de la fonction objectif de cette solution est moins bonne que celle de la solution actuelle, mais la solution est acceptée |
| $\rho$ | Le facteur de réaction : utilisé pour contrôler la vitesse à laquelle l'algorithme réagit aux changements de scores (voir la Section 4.3.5) |
| $q$ | Le nombre de clients à supprimer |
| $\eta$ | Le facteur de perturbation : qui contrôle la quantité de perturbation |

Afin de générer une version efficace d'ALNS, nous avons étudié plusieurs configurations de ses paramètres comme montré dans le Tableau 4.4. Dans ce Tableau, $C_x$ représente le nom de la configuration, et $f$ est la valeur de la solution initiale. $v1$ est un nombre aléatoire compris entre $a$ et $b$ où $a$ est égal à $0.1 \cdot n$ si $0.1 \cdot n$ est plus petit que 30, $a$ est égal à 30 sinon, et $b$ est égal à $0.4 \cdot n$ si $0.4 \cdot n$ est plus petit que 60, $b$ est égal à 60 sinon. $v2$ est aléatoirement choisi depuis l'ensemble $\{2, \ldots, 7\}$. $v3$ est aléatoirement choisi depuis l'ensemble $\{1, 2\}$. Le paramètre $T_s$ varie dans l'ensemble $\{V_1, V_2, V_3\}$, où $V_1 = 0.0072 \cdot f$, $V_2 = 0.0010 \cdot f$ et $V_3 = 0.0005 \cdot f$, où $f$ est la valeur de la fonction objectif de la solution initiale. Le paramètre $c$ varie dans l'ensemble $\{V1, V2, V3, V4\}$, où $V1 = 0.99975$, $V2 = 0.9997$, $V3 = 0.9998$ et $V4 = 0.99985$. Le paramètre $\eta$ varie dans l'ensemble $\{v_1, v_2, v_3\}$, où $v_1 = 0.025$, $v_2 = 0.03$ et $v_3 = 0.02$. $C1$ est la configuration donnée par Pisinger et Ropke [78]. Pour générer les configurations restantes, nous modifions un paramètre à la fois (ou un ensemble de paramètres qui agissent sur une même composante) depuis la meilleure configuration rencontrée jusque-là.

Table 4.4 – *Configuration des paramètres pour l'ALNS classique.*

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | C17 | C18 | C19 | C20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_r$ | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 3 | 10 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $p_w$ | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 6 | 10 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $T_s$ | $V_1$ | $V_1$ | $V_1$ | $V_2$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ | $V_3$ |
| $c$ | $V1$ | $V1$ | $V1$ | $V1$ | $V1$ | $V2$ | $V3$ | $V4$ | $V3$ | $V3$ | $V3$ | $V3$ | $V3$ | $V3$ | $V3$ | $V3$ | $V3$ | $V3$ | $V3$ | $V3$ |
| $\sigma_1$ | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 50 | 20 | 33 | 33 | 33 | 33 |
| $\sigma_2$ | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 0 | 9 | 9 | 0 | 9 | 9 | 9 |
| $\sigma_3$ | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 0 | 13 | 13 | 13 | 0 | 13 | 13 | 13 |
| $\rho$ | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.2 | 0.1 | 0.1 |
| $q$ | $v1$ | $v2$ | $v3$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ |
| $\eta$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_2$ | $v_3$ |

Les résultats des configurations étudiées sont présentés dans le Tableau 4.5. ALNS est lancée 10 fois avec chaque configuration sur toutes les instances étudiées et durant 2000 itérations. Les résultats sont comparés à ceux de CPLEX 12.2 utilisant le modèle mathématique proposé (CPLEX n'a pu démontrer l'optimalité que pour 8 solutions).

Plus précisément, les pourcentages de déviation moyens (parmi toutes les instances et les 10 exécutions) par rapport à la dernière solution obtenue par CPLEX sont présentés. Le **pourcentage de déviation (gap)** d'une solution $sol_1$ par rapport à une solution $sol_2$ est calculé suivant l'Expression $gap = 100 \cdot (z_{sol_2} - z_{sol_1}) / |z_{sol_2}|$, où $z_{sol_1}$ et $z_{sol_2}$ sont les valeurs de la fonction objectif de $sol_1$ et $sol_2$ respectivement. Dans le reste de ce Chapitre, le gap d'une solution donnée par rapport à une autre solution sera calculé de la même manière.

Dans les trois premières lignes du Tableau 4.5, les meilleurs gaps (meil), les gaps moyens (moy) et les moins bons (pire) sont donnés. Le meilleur temps de calcul (meil), le temps de calcul moyen (moy) et le temps de calcul le moins bon (pire) en secondes sont présentés dans les trois dernières lignes. Notons que, le temps de calcul moyen de CPLEX est de 1.91 heures. Dans certains cas, CPLEX s'est arrêté avant 2 heures de calcul avec un statut de dépassement mémoire (out of memory). Afin de calculer le temps moyen de CPLEX en tenant compte de toutes les instances, le temps de calcul pour les cas de dépassement mémoire est considéré égal à 2 heures.

Nous pouvons voir que $C9$ est la meilleure configuration en termes de qualité des solutions dans le meilleur cas et le cas moyen. $C9$ atteint la $4^{\text{ième}}$ meilleure position

TABLE 4.5 – *Comparaison entre les configurations des paramètres pour ALNS.*

| | | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | C17 | C18 | C19 | C20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| moy | meil | -8.30 | -14.51 | -8.85 | -13.85 | -14.36 | -13.99 | -14.16 | -14.55 | **-14.74** | -14.19 | -14.53 | -14.56 | -14.27 | -14.49 | -14.10 | -14.35 | -14.27 | -14.04 | -14.63 | -14.48 |
| gap | moy | 1.88 | -4.78 | 4.31 | -4.86 | -5.16 | -4.83 | -5.34 | -4.97 | **-5.48** | -4.85 | -4.85 | -5.30 | -5.24 | -5.26 | -5.05 | -5.14 | -4.81 | -4.93 | -4.89 | -5.03 |
| | pire | 12.08 | 6.71 | 21.07 | **5.61** | 6.04 | 6.44 | 5.67 | 6.34 | 6.25 | 7.64 | 6.45 | 6.68 | 6.47 | 6.60 | 7.56 | 7.69 | 7.05 | 6.27 | 7.93 | 6.04 |
| moy | meil | 8.46 | 1.56 | **0.78** | 1.57 | 1.63 | 1.58 | 1.61 | 1.57 | 1.56 | 1.56 | 1.56 | 1.57 | 1.62 | 1.60 | 1.60 | 1.60 | 1.60 | 1.60 | 1.59 | 1.68 |
| CPU | moy | 9.44 | 1.75 | **0.89** | 1.76 | 1.88 | 1.79 | 1.82 | 1.76 | 1.76 | 1.74 | 1.74 | 1.75 | 1.85 | 1.81 | 1.80 | 1.78 | 1.80 | 1.80 | 1.79 | 1.91 |
| (sec) | pire | 10.84 | 2.01 | **1.04** | 2.05 | 2.23 | 2.06 | 2.15 | 2.05 | 2.05 | 1.98 | 1.98 | 1.98 | 2.18 | 2.11 | 2.07 | 2.04 | 2.07 | 2.11 | 2.06 | 2.24 |

en termes de gap dans le pire cas. De plus, $C9$ obtient le second meilleur temps de calcul après $C3$, et le troisième temps de calcul moyen après $C3$ et $C10$. Pour toutes ces raisons, $C9$ sera retenue pour les comparaisons futures présentées dans ce Chapitre. Nous pouvons également voir que $C1$ et $C3$ sont les moins bonnes configurations en termes de qualité des solutions. De plus, $C1$ est la moins bonne configuration en termes de temps de calcul. Cela montre l'importance d'un bon choix du nombre de clients à supprimer pour le problème étudié.

### 4.4.4 Configuration d'sALNS

Dans cette Section, nous exécutons, en utilisant les valeurs des paramètres de la configuration $P1$ (voir Tableau 4.7), différentes versions d'sALNS afin de montrer l'importance de chaque nouvelle composante. Après cela, nous sélectionnons les meilleures versions et les combinons en un unique algorithme. Enfin, nous réglons les paramètres de cet algorithme final.

Le Tableau 4.6 est fourni afin de faciliter la compréhension des sections suivantes. Ce Tableau décrit les paramètres de l'approche proposée.

TABLE 4.6 – *Description des paramètres d'sALNS.*

| paramètre | description |
|---|---|
| $p_r$ | Le paramètre de contrôle de l'aléatoire du *related removal* |
| $p_w$ | Le paramètre de contrôle de l'aléatoire du *worst removal* |
| $p_{hn}$ | Le paramètre de contrôle de l'aléatoire de l'*historical node-pair removal* |
| $p_{hr}$ | Le paramètre de contrôle de l'aléatoire de l'*historical request-pair removal* |
| $p_c$ | Le paramètre de contrôle de l'aléatoire du *cluster removal* |
| $c$ | Le coefficient de décroissance |
| $T_0$ | Le seuil de température |
| $ml$ | Le nombre d'itérations effectuées avant que les scores ne soient remis à zéro |
| $\sigma_1$ | Les scores des opérateurs d'insertion avec perturbation (sans perturbation) sont incrémentés de $\sigma_1$ si l'utilisation (non-utilisation) de la perturbation résulte en une nouvelle meilleure solution globale |
| $\sigma_2$ | Les scores des opérateurs d'insertion avec perturbation (sans perturbation) sont incrémentés de $\sigma_2$ si l'utilisation (non-utilisation) de la perturbation résulte en une solution avec une valeur de la fonction objectif meilleure que celle de la solution actuelle |
| $\sigma_3$ | Les scores des opérateurs d'insertion avec perturbation (sans perturbation) sont incrémentés de $\sigma_3$ si l'utilisation (non-utilisation) de la perturbation résulte en une solution avec une valeur de la fonction objectif moins bonne que celle de la solution actuelle, mais que cette solution soit quand même acceptée |
| $\rho$ | Le facteur de réaction : utilisé pour contrôler la vitesse de réaction de l'algorithme aux changements des scores (comme dans l'ALNS classique, sauf que dans sALNS il n'est utilisé que pour le terme de perturbation) |
| $\eta$ | Le facteur de perturbation : qui contrôle la quantité de perturbation |
| $l$ | La constante contrôlant la durée d'une phase d'évaluation |
| $r$ | Le nombre de clients à supprimer |

#### Évaluation des nouvelles composantes

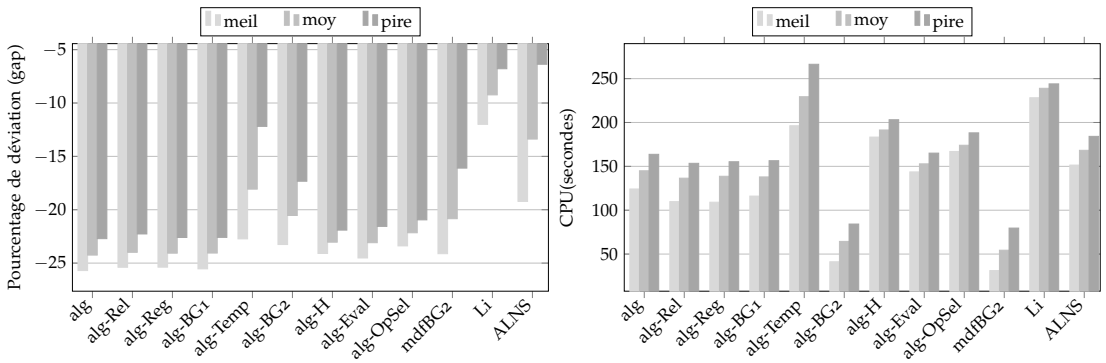
Afin d'évaluer les composantes ajoutées et pour créer une version plus performante d'sALNS, nous exécutons huit versions de l'algorithme. La première version représente le cas où toutes les composantes sont utilisées. *alg* désigne cette version. Pour les autres versions, nous supprimons, à chaque fois, une composante d'*alg*. Nous remplaçons les procédures de sélection des opérateurs et de mise à jour des scores proposées par celles de l'ALNS classique. Cette version est appelée *alg-OpSel*. Nous supprimons l'évaluation des solutions après les opérateurs de suppression d'*alg*. Cette version est appelée

*alg-Eval*. Chaque opérateur d'insertion est supprimé d'*alg* et nous obtenons *alg-BG1*, *alg-BG2*, *alg-Reg* et *alg-H* qui désignent respectivement *alg* sans *basic greedy 1*, *alg* sans *basic greedy 2*, *alg* sans l'opérateur de *regret* et *alg* sans *H_insert* respectivement. Nous évaluons également l'algorithme avec et sans le *related removal*. La version sans *related removal* est appelée *alg-Rel*. De plus, nous considérons la suppression de notre diversification via la mise à jour de la température d'*alg*. Cette version est notée *alg-Temp*. Nous étudions aussi la version d'*alg* qui n'utilise que les opérateurs de suppression/insertion présentés par Li et al. [65]. Pour le *Shaw removal*, nous avons seulement enlevé les termes en rapport avec les fenêtres de temps. Cette version est appelée *Li* et elle est également comparée aux autres versions. De plus, nous ajoutons une version notée *mdfBG2* qui représente *alg* sans *basic greedy 2*, ni *cluster removal* ni *historical request-pair removal*. Nous expliquerons par la suite les raisons qui nous ont poussés à générer la version *mdfBG2*. Enfin, nous exécutons l'ALNS classique décrite dans la Section 4.4.3. Toutes les versions étudiées sont exécutées 5 fois durant 200000 itérations.

Dans la partie gauche de la Figure 4.1, nous donnons le gap moyen de chaque version par rapport aux solutions de CPLEX sur toutes les intances. Sur la partie droite de la Figure 4.1, nous donnons le temps de calcul moyen en secondes pour chaque version. Plus précisément, nous montrons pour les deux parties de la Figure, les meilleurs résultats (meil), les résultats moyens (moy) et les résultats dans le pire cas (pire). Dans un premier temps, nous commençons par comparer les différentes versions de notre algorithme sans considérer *Li* et l'ALNS classique. Ces dernières versions seront étudiées à la fin de la Section actuelle.

De la Figure 4.1, nous pouvons voir qu'*alg* fournit des résultats légèrement meilleurs que *alg-Rel*, *alg-Reg* et *alg-BG1* en termes de qualité des solutions pour les trois cas (meilleur, moyen, pire). En effet, le meilleur gap, le gap moyen et le moins bon gap d'*alg* sont de $-25.7\%$, $-24.25\%$ et $-22.71\%$ respectivement, tandis que le meilleur gap, le gap moyen et le moins bon prennent les valeurs $-25.39\%$, $-23.99\%$ et $-22.27\%$ respectivement pour *alg-Rel*, $-25.38\%$, $-24.07\%$ et $-22.6\%$ respectivement pour *alg-Reg*, et $-25.54\%$, $-24.06\%$ et $-22.59\%$ respectivement pour *alg-BG1*. Nous remarquons aussi qu'*alg* fournit de meilleurs résultats que les versions restantes dans le meilleur cas, le cas moyen et le pire cas.

FIGURE 4.1 – *Évaluation des composantes d'sALNS en termes de pourcentage de déviation (à gauche) et de temps de calcul (à droite).*




De la Figure 4.1, nous remarquons également qu'*alg* s'arrête en un temps de calcul raisonnable. En effet, si l'on ne tient pas compte d'*alg-BG2* et *mdfBG2* qui sont les

versions les plus rapides, nous constatons que le temps de calcul d'*alg* n'est pas trop éloigné de celui d'*alg-Rel*, *alg-Reg* et *alg-BG1*. Nous remarquons qu'*alg* est plus rapide que les versions restantes. Le fait que les gaps d'*alg-Rel*, *alg-Reg* et *alg-BG1* soient un peu moins bons que ceux d'*alg* nous a encouragés à garder *related removal*, l'opérateur de *regret* et *basic greedy 1* au sein de l'algorithme principal. Nous espérons que l'algorithme puisse se sortir des optima locaux grâce à l'aspect aléatoire de ces opérateurs.

Lorsque nous évaluons la mise à jour de la température, nous remarquons que la qualité des solutions et le temps de calcul d'*alg-Temp* sont beaucoup moins bons que ceux d'*alg*. Cela prouve que le redémarrage de la version *alg* depuis des températures plus élevées incorpore un certain équilibre entre l'intensification et la diversification pour l'approche étudiée. Nous pensons que lorsque la mise à jour de température proposée est utilisée, le temps de calcul décroît car de bonnes solutions sont atteintes rapidement. En effet, après qu'une solution de bonne qualité soit atteinte, la probabilité d'avoir une meilleure solution, que ce soit après une suppression ou une insertion, est réduite. Ceci résulte en moins de mises à jour de la meilleure et de l'actuelle solutions et leurs « métriques » (les métriques sont des vecteurs utilisés pour vérifier la réalisabilité d'une solution rapidement). Si une solution $S$ n'est améliorée ni après une suppression ni après une insertion, il y aura une unique mise à jour de la solution obtenue $S^{'}$ ($S^{'} = S$) et ses métriques à la fin de l'itération actuelle. Cependant, si $S$ et la meilleure solution globale $S^{meil}$ sont toutes deux améliorées après une suppression et après une insertion, il y aura une mise à jour de $S$, $S^{meil}$ et leurs métriques respectives après la suppression et après l'insertion, ce qui résulte en une augmentation du temps de calcul.

Concernant *alg-BG2*, nous pouvons voir que la suppression de *basic greedy 2* d'*alg* génère de plus grands gaps en comparaison avec *alg*. Cela souligne l'importance de cet opérateur au sein de la version *alg*. D'autre part, les versions *alg-BG2* et *mdfBG2* constituent les versions les plus rapides. Ceci prouve que, lorsque l'opérateur *basic greedy 2* est utilisé, l'algorithme le favorise par rapport aux autres opérateurs d'insertion dû à son efficacité. Plus précisément, l'aspect parallèle (l'insertion de chaque client est évaluée sur plusieurs routes et ré-évaluée aux itérations suivantes s'il n'a pas été inséré) de l'opérateur *basic greedy 2* implique un plus grand temps de calcul lorsque l'algorithme le favorise, tandis qu'une sélection plus équilibrée des opérateurs d'insertion génère une approche plus rapide, particulièrement quand on utilise une heuristique rapide comme *H_insert*.

En ce qui concerne *alg-H*, il est clair que la suppression d'*H_insert* dégrade la qualité de la solution. Cela est souligné par la différence entre les gaps d'*alg-H* et d'*alg*. Aussi, *alg-H* requiert plus d'effort de calcul. Cela est dû au fait qu'*H_insert* soit un opérateur utilisant des insertions séquentielles, et la suppression d'*H_insert* implique l'utilisation exclusive d'opérateurs d'insertions parallèles, ce qui demande beaucoup de temps de calcul.

En considérant la version *alg-Eval*, nous pouvons voir que l'évaluation de la solution après les opérateurs de suppression mène à l'exploration de certaines régions omises de l'espace de recherche. En effet, cela peut être vu à travers les gaps obtenus lors de la suppression de cette évaluation de l'algorithme principal. En outre, le temps d'exécution de cette version est plus grand que celui d'*alg*.

Il est clair que la procédure de sélection des opérateurs proposée a un impact positif

sur la performance de la version *alg*. Cela peut être observé depuis la différence de gaps entre *alg* et *alg-OpSel*. En plus, la suppression de cette composante induit plus de temps de calcul.

Bien qu'*alg-BG2* fournisse, en moyenne, des solutions de moins bonne qualité en comparaison avec *alg*, la suppression de l'opérateur *basic greedy 2* d'*alg* permet à la version *alg-BG2* d'atteindre de meilleures solutions pour 18 instances. Pour six d'entre elles, même lorsqu'*alg* est exécutée avec un plus grand nombre d'itérations, la version reste coincée dans des optima locaux. Nous avons remarqué que la version *mdfBG2* est aussi capable d'atteindre des solutions de bonne qualité pour ces instances. De plus, *mdfBG2* fournit de meilleurs gaps et est plus rapide qu'*alg-BG2*.

Concernant la version *Li*, on peut voir que le remplacement des opérateurs de suppression/insertion proposés dans ce travail par ceux présentés par Li et al. [65] détériore la qualité de la solution et le temps de calcul. En effet, la version *Li* donne les moins bons résultats parmi tous ceux qui sont étudiés. Cela prouve que les opérateurs proposés dans cette étude sont adaptés au PTPSPD.

Lors de la comparaison d'ALNS avec *Li*, nous remarquons qu'ALNS donne de meilleurs résultats en termes de qualité de la solution et de temps de calcul. Cependant, ALNS est moins bonne que toutes les autres versions en termes de qualité de la solution.

L'étude de toutes ces versions nous a encouragé à créer un algorithme contenant les versions *alg* et *mdfBG2*, de telle sorte que, au début de la recherche, cet algorithme choisit de façon aléatoire la version à exécuter. Dans ce qui suit, sALNS désignera cet algorithme.

**Réglage des paramètres pour l'algorithme final**

Afin d'avoir un bon réglage des paramètres pour l'algorithme proposé, nous commençons à partir d'sALNS avec une configuration arbitraire notée $P1$ puis, nous modifions la meilleure configuration obtenue en changeant un paramètre à la fois. Chaque configuration est exécutée 10 fois sur toutes les instances durant 2000 itérations.

Le Tableau 4.7 donne les valeur des paramètres de chaque configuration. Sur ce Tableau, $v1$, $v2$ et $v3$ représentent les mêmes valeurs de paramètres que celles décrites dans la Section 4.4.3. Le paramètre $c$ varie dans l'ensemble $\left\{V_1^{'}, V_2^{'}, V_3^{'}\right\}$, où $V_1^{'} = 0.99$, $V_2^{'} = 0.98$ et $V_3^{'} = 0.9998$. Le paramètre $\eta$ varie dans l'ensemble $\{v_1, v_2, v_3\}$, où $v_1 = 0.025$, $v_2 = 0.03$ et $v_3 = 0.02$. $Px$ est le nom de la $x^{\text{ième}}$ configuration, où $x \in \{1, 2, \ldots, 21\}$.

Dans le Tableau 4.8, nous présentons les résultats de chaque configuration. Les lignes de ce Tableau sont définies de la même façon que celles du Tableau 4.5.

À partir du Tableau 4.8, nous remarquons que la configuration $P20$ mène aux meilleurs résultats en termes de pourcentage de déviation moyen. Cependant, le temps de calcul de $P20$ atteint presque 4 fois le temps de calcul de la configuration $P1$. $P1$ diffère de $P20$ uniquement dans la valeur du paramètre $r$. Nous avons décidé d'augmenter le nombre d'itérations de $P1$ et nous avons comparé les résultats obtenus avec

TABLE 4.7 – *Configuration des paramètres pour sALNS.*

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P17 | P18 | P19 | P20 | P21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_r$ | 0 | 0 | 6 | 6 | 6 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $p_w$ | 6 | 6 | 3 | 6 | 6 | 50 | 6 | 0 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $p_{hn}$ | 50 | 50 | 0 | 0 | 6 | 50 | 50 | 50 | 0 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| $p_{hr}$ | 6 | 6 | 0 | 0 | 6 | 50 | 50 | 50 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $p_c$ | 6 | 6 | 0 | 0 | 6 | 50 | 50 | 50 | 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $c$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_2'$ | $V_3'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ | $V_1'$ |
| $T_0$ | 1 | 0.5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $ml$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{8}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ | $\frac{itr}{4.5}$ |
| $\sigma_1$ | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 33 |
| $\sigma_2$ | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 0 | 9 | 9 | 9 | 9 | 9 | 9 | 9 |
| $\sigma_3$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $\rho$ | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.1 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |
| $l$ | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 10 | 1 | 5 | 5 | 5 | 5 |
| $\eta$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_1$ | $v_2$ | $v_3$ | $v_1$ | $v_1$ |
| $r$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v2$ | $v1$ | $v3$ |

ceux de $P20$. Nous remarquons que cette augmentation des nombre d'itérations de $P1$ mène à de meilleurs résultats en comparaison avec $P20$. En effet, $P1$ atteint les valeurs $-21.86\%$, $-15.57\%$ et $-6.57\%$ pour le meilleur gap, le gap moyen et le gap dans le pire cas respectivement. Tandis que le temps de calcul de $P1$ atteint les valeurs 3.34, 6.00 et 9.09 secondes pour le meilleur temps, le temps moyen et le temps dans le pire cas respectivement. Comme nous pouvons le voir, le temps de calcul de $P1$ est plus petit que celui de $P20$.

Si l'on ne tient pas compte de $P20$, nous pouvons noter qu'aucune configuration ne domine les autres en termes de meilleur gap, de gap moyen et de gap dans le pire cas simultanément. Alors, nous choisissons la meilleure configuration en termes de gap moyen. La configuration $P1$ est celle qui possède le meilleur gap moyen. Donc, elle sera retenue pour l'algorithme final.

Rappelons que dans la Section 4.3.4, les paramètres $\alpha_1$, $\alpha_2$, $\alpha_3$ et $\lambda$ sont utilisés dans les critères de sélection des opérateurs d'insertion. La valeurs des paramètres $\alpha_1$, $\alpha_2$, $\alpha_3$ sont toutes fixées à 1, et $\lambda$ est choisie aléatoirement dans l'intervalle $[0, 1]$. La raison est que nous avons commencé depuis les valeurs $\alpha_1 = \frac{1}{2}$, $\alpha_2 = \frac{1}{2}$, $\alpha_3 = 0$ et $\lambda = 0$, qui correspondent à une sélection des clients basée sur la fonction objectif uniquement, et nous avons tenté d'améliorer ces valeurs.

Le Tableau 4.9 rapporte les résultats des configurations de paramètres que nous avons essayées, en termes de meilleur gap (meil), gap moyen (moy) et gap dans le pire cas (pire) par rapport aux solutions de CPLEX, et en termes de meilleur temps de calcul, de temps de calcul moyen et de temps de calcul dans le pire cas en secondes.

Soit $Con_1$ la configuration avec les valeurs $\alpha_1 = \frac{1}{2}$, $\alpha_2 = \frac{1}{2}$, $\alpha_3 = 0$ et $\lambda = 0$. Nous avons supposé que la diversification induite par d'autres valeurs du paramètre $\lambda$ pourrait fournir de meilleurs résultats. Nous avons donc testé la configuration $Con_2$ qui diffère de $Con_1$ dans la valeur du paramètre $\lambda$. Dans $Con_2$, $\lambda$ est aléatoirement sélectionné dans l'intervalle $[0, 1]$.

Comme les résultats obtenus ont confirmé notre hypothèse (voir Tableau 4.9), nous avons continué les tests depuis $Con_2$, nous avons essayé la configuration $Con_3$, avec les paramètres suivants $\alpha_1 = \alpha_2 = \alpha_3 = \frac{1}{3}$ et $\lambda$ est aléatoirement choisi dans l'intervalle $[0, 1]$. Comme les résultats de $Con_3$ sont meilleurs que ceux de $Con_2$ (voir Tableau 4.9),

TABLE 4.8 – *Comparaison entre les configurations des paramètres d'sALNS.*

| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P17 | P18 | P19 | P20 | P21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| moy | meil | -18.92 | **-18.97** | -15.74 | -16.66 | -18.19 | -17.63 | -18.90 | -18.84 | -17.80 | -18.22 | -18.87 | -18.66 | -18.90 | -18.74 | -18.78 | -18.78 | -18.49 | -18.46 | -18.89 | -17.68 | -14.60 |
| gap | moy | -10.61 | -10.30 | -5.44 | -4.11 | -7.57 | -8.31 | -9.42 | -9.98 | -6.15 | -9.49 | -8.38 | -10.24 | -10.51 | -10.39 | -10.07 | -9.69 | -9.66 | -10.17 | -9.45 | **-12.03** | 1.21 |
| | pire | 0.92 | 2.27 | 6.06 | 15.32 | 6.64 | 5.11 | 1.06 | 3.03 | 9.14 | 2.95 | 6.81 | 1.11 | 1.87 | 1.35 | 3.25 | 2.12 | 2.04 | 2.55 | 2.90 | **-6.31** | 25.02 |
| moy | meil | 1.05 | 1.15 | 0.91 | 1.06 | 1.05 | 1.02 | 1.12 | 1.15 | 1.16 | 0.94 | 1.67 | 1.29 | 1.05 | 1.02 | 1.03 | 1.06 | 1.05 | 1.06 | 1.01 | 4.81 | **0.65** |
| CPU | moy | 1.65 | 1.78 | 1.50 | 1.69 | 1.67 | 1.66 | 1.74 | 1.85 | 1.73 | 1.69 | 2.26 | 1.80 | 1.64 | 1.66 | 1.67 | 1.60 | 1.62 | 1.60 | 1.53 | 6.94 | **1.11** |
| (sec) | pire | 2.43 | 2.63 | 2.22 | 2.45 | 2.54 | 2.55 | 2.53 | 2.79 | 2.53 | 2.70 | 3.14 | 2.50 | 2.46 | 2.43 | 2.51 | 2.24 | 2.37 | 2.30 | 2.10 | 9.78 | **1.75** |

TABLE 4.9 – *Comparaison entre les configurations des paramètres pour les opérateurs d'insertion.*

| | | $Con_1$ | $Con_2$ | $Con_3$ | $Con_4$ |
|---|---|---|---|---|---|
| moy | meil | -5.94 | -14.95 | -15.74 | **-18.92** |
| gap | moy | 2.80 | -5.38 | -5.44 | **-10.61** |
| | pire | 11.52 | 8.02 | 6.06 | **0.92** |
| moy | meil | **0.74** | 0.93 | 0.91 | 1.05 |
| CPU | moy | **1.11** | 1.49 | 1.50 | 1.65 |
| (sec) | pire | **1.60** | 2.16 | 2.22 | 2.43 |

nous avons essayé une autre configuration (notée $Con_4$), dans laquelle la somme d'$\alpha_1$, $\alpha_2$ et $\alpha_3$ est supérieure à 1. L'idée derrière la génération de $Con_4$ et que nous voulons que la diversification induite par $\lambda$ soit plus équilibrée. Ainsi, $Con_4$ possède les valeurs suivantes $\alpha_1 = \alpha_2 = \alpha_3 = 1$ et $\lambda$ est aléatoirement choisie dans l'intervalle $[0, 1]$. Du Tableau 4.9, nous pouvons voir que $Con_4$ mène aux meilleurs résultats.

Dans la Section 4.3.2, nous avons dit que la règle de Métropolis était seulement appliquée après les opérateurs d'insertion parce que cela menait à de meilleurs résultats. Afin de démontrer cela, quelques tests expérimentaux sont effectués. En effet, un paramètre $metro \in \{0, 1\}$ est ajouté à sALNS, de telle sorte que, si *metro* prend la valeur 0, sALNS est exécutée sans la règle de Métropolis après les opérateurs de suppression. Sinon, sALNS utilise la règle de Métropolis après les opérateurs de suppression.

Les tests décrits ci-dessus montrent que lorsque $metro = 0$, sALNS obtient les valeurs $-18.92\%$, $-10.61\%$ et $0.92\%$ pour le meilleur gap, le gap moyen et le moins bon gap respectivement. Lorsque $metro = 1$, sALNS obtient les valeurs $-2.15\%$, $10.73\%$ et $23.08\%$ pour le meilleur gap, le gap moyen et le moins bon gap respectivement. Concernant le temps de calcul moyen, lorsque $metro = 0$, sALNS obtient les valeurs 1.05, 1.65 et 2.43 secondes pour le meilleur temps, le temps moyen et le moins bon temps de calcul respectivement. Lorsque $metro = 1$, sALNS obtient les valeurs 0.53, 0.83, 1.28 secondes pour le meilleur temps, le temps moyen et le moins bon temps de calcul respectivement.

Ces résultats confirment que, même si le temps de calcul est légèrement plus grand lorsque $metro = 1$, de bien meilleurs résultats sont obtenus lorsque la règle de Métropolis n'est appliquée qu'après les opérateurs d'insertion.

### 4.4.5 Étude de la performance d'sALNS

Nous évaluons, sur les instances générées, la performance d'sALNS, ALNS et CPLEX en utilisant le modèle mathématique proposé pour le PTPSPD. De plus, nous exécutons notre heuristique de construction en utilisant le critère $cr_1^{global}(i, u, j)$ comme défini dans les Expressions (4.20)–(4.24), et avec toutes les combinaisons possibles des valeurs des paramètres. Les meilleures solutions rencontrée sont retournées.

Table 4.10 – *Évaluation des heuristiques suivant le nombre de véhicules et la limite de capacité.*

| | CPLEX | $H_{const}^{1run}$ | | $H_{cosnt}$ | | ALNS | | sALNS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Set | $\%^{UB}$ | $\%^{cpx}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | $\%^{UB}$ | $\%^{ALNS}$ | CPU |
| X-Y | 36.02 | -110.9 | **0.06** | -163.11 | 603.09 | -197.56 | 182.23 | **-213.33** | 6.95 | -9.39 | 149.41 |
| 50-2 | 10.42 | 137.52 | **0.01** | 36.91 | 90.89 | 5.57 | 589.74 | **-3.11** | 8.01 | -9.52 | 167.93 |
| 50-3 | 14.11 | 142.53 | **0.01** | 44.19 | 127.73 | 4.60 | 927.26 | **-3.12** | 11.70 | -8.46 | 367.1 |
| 50-4 | 18.36 | 149.65 | **0.01** | 50.62 | 158.07 | 5.00 | 1033.14 | **-3.93** | 15.46 | -9.42 | 710.39 |
| 75-2 | 12.07 | 93.19 | **0.01** | 20.77 | 118.02 | -1.2 | 630.23 | **-4.12** | 8.71 | -2.87 | 274.18 |
| 75-3 | 15.20 | 102.67 | **0.01** | 29.41 | 179.84 | -1.35 | 831.44 | **-5.64** | 10.66 | -4.5 | 723.67 |
| 75-4 | 20.83 | 99.55 | **0.02** | 31.97 | 216.39 | -3.79 | 1062.02 | **-10.53** | 13.18 | -7.7 | 793.11 |
| 100-2 | 13.41 | 66.58 | **0.01** | 14.58 | 162.98 | -5.19 | 590.49 | **-7.38** | 7.72 | -2.06 | 444.9 |
| 100-3 | 15.36 | 67.76 | **0.02** | 19.58 | 223.92 | -.8 | 886.37 | **-5.99** | 10.49 | -5.33 | 702.83 |
| 100-4 | 20.63 | 82.26 | **0.02** | 24.29 | 268.25 | -5.38 | 931.22 | **-11.43** | 11.92 | -6.1 | 1344.18 |
| X-2 | 11.37 | 49.43 | **0.02** | 9.24 | 328.87 | -4.21 | 643.49 | **-5.49** | 6.78 | -1.29 | 638.52 |
| X-3 | 18.13 | 45.71 | **0.03** | 7.36 | 583.93 | -11.02 | 695.28 | **-14.24** | 7.44 | -2.92 | 654.13 |
| X-4 | 28.73 | 26.04 | **0.06** | -7.53 | 568.00 | -18.22 | 669.69 | **-25.75** | 7.89 | -8.26 | 601.66 |
| moy | 18.05 | 73.23 | **0.02** | 9.1 | 279.23 | -17.97 | 744.05 | **-24.16** | 9.76 | -5.99 | 582.46 |

Dans le Tableau 4.10 les meilleurs résultats de toutes les approches sont comparés. Les résultats détaillés dont donnés dans l'Annexe A.3. Sur le Tableau 4.10, *sALNS* désigne l'algorithme proposé, *ALNS* désigne l'ALNS classique. CPLEX représente les résultats du solveur CPLEX 12.2. $H^{1run}_{const}$ et $H_{const}$ sont l'heuristique de construction proposée pour le PTPSPD exécutée une seule fois avec des valeurs des paramètres générées aléatoirement et cette même heuristique utilisant toutes les combinaisons possibles des valeurs des paramètres (comme décrite plus haut) respectivement. Les instances sont divisées en 13 ensembles selon leurs nombres de véhicules et leurs limites de capacité. Dans chaque ensemble, il y a 9 instances avec 50 à 199 clients. Sur la colonne *Set*, les ensembles de type « *Q-m* » sont présentés, où $Q$ représente la limite de capacité et $m$ est le nombre de véhicules. Lorsque $Q$ prend les valeurs $X$, cela signifie que la limite de capacité des instances d'origine est utilisée (voir le travail de Christofides et al. [21]). De la même façon, lorsque $m$ prend les valeurs $Y$ cela signifie que les nombres de véhicules d'origine sont utilisés. Pour chaque approche, nous donnons le gap moyen par rapport à CPLEX dans la colonne $\%^{cpx}$ et le temps de calcul moyen CPU en secondes. $\%^{UB}$ représente le gap moyen par rapport à la borne supérieure de CPLEX 12.2 (seules 8 solutions de CPLEX sont optimales). $\%^{alns}$ désigne le gap moyen entre sALNS et ALNS. Enfin, *moy* désigne les résultats moyens sur tous les ensembles d'instances.

Du Tableau 4.10 et l'Annexe A.3, nous remarquons qu'$H^{1run}_{const}$ fournit de meilleures solutions que CPLEX dans deux cas pour les instances avec de grand(e)s (d'origine) limites de capacité et nombres de véhicules. Pour les autres ensembles, $H^{1run}_{const}$ ne fournit de meilleures résultats par rapport à CPLEX que dans un seul cas (dernière instance dans l'Annexe A.3). Cependant, l'heuristique est très rapide.

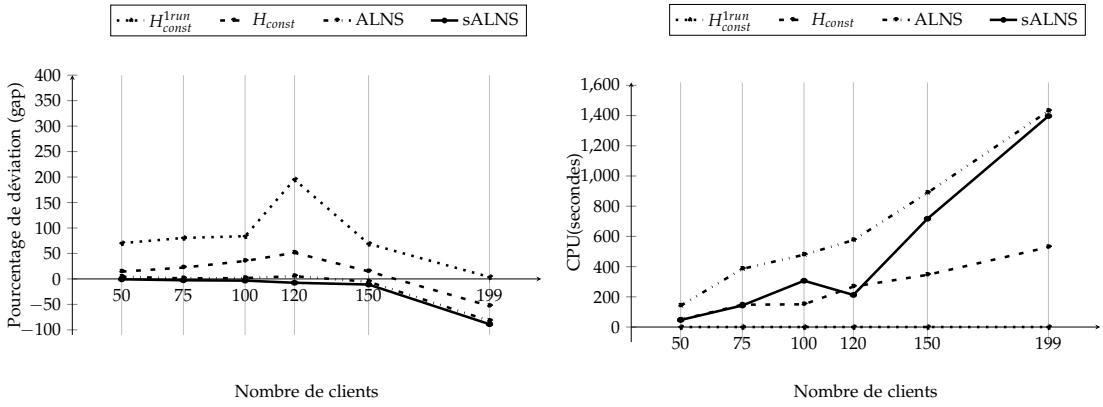
Concernant $H_{const}$, nous pouvons noter que cette heuristique donne, en moyenne, de meilleurs résultats que CPLEX pour les instances avec une grande limite de capacité et un nombre de véhicules plus grand ou égal à 4. Le temps de calcul de cette heuristique est plus petit que celui d'ALNS et d'sALNS.

L'ALNS classique est plus lente que les autres heuristiques. Cette heuristique parvient à atteindre de meilleures solutions que CPLEX dans tous les ensembles sauf lorsque la limite de capacité est égale à 50.

sALNS fournit les meilleurs résultats parmi toutes les approches étudiées, et particulièrement pour les ensembles avec un grand nombre de véhicules et une grande capacité. Plus précisément, l'heuristique sALNS proposée donne un meilleur gap par rapport à la borne supérieure de CPLEX. sALNS fournit de meilleures solutions que CPLEX dans 97 cas et des résultats identiques dans 17 cas (en incluant les solutions optimales). sALNS ne parvient pas à atteindre les solutions de CPLEX pour seulement 3 instances, il s'agit des instances 9-150-50-3, 9-150-50-4 et 15-150-50-4. Les gaps d'sALNS par rapport à CPLEX pour ces instances sont égaux à 0.44%, 3.64% et 0.77% respectivement (voir l'Annexe A.3 pour les résultats détaillés). Le gap entre sALNS et ALNS montre que la première heuristique est plus efficace. En outre, sALNS est généralement plus rapide qu'ALNS sauf pour l'ensemble avec un nombre de véhicules égal à 4 et une capacité de 100.

Dans la Figure 4.2, nous donnons les moyennes des meilleurs résultats des heuristiques selon le nombre de clients pour les instances étudiées. En fait, les instances sont divisées en 5 ensembles selon leurs nombres de clients (avec 50, 75, 100, 120, 150 et

FIGURE 4.2 – *Évaluation des heuristiques selon le nombre de clients.*




199 clients). Dans la partie gauche de la Figure, le gap moyen par rapport à CPLEX de l'heuristique de construction du PTPSPD exécutée une seule fois ($H^{1run}_{const}$), celui de l'heuristique de construction du PTPSPD exécutée avec toutes les combinaisons possibles des valeurs des paramètres ($H_{const}$), celui d'ALNS et celui d'sALNS sont donnés. Dans la partie droite de la Figure, le temps de calcul moyen en secondes de ces heuristiques est donné.

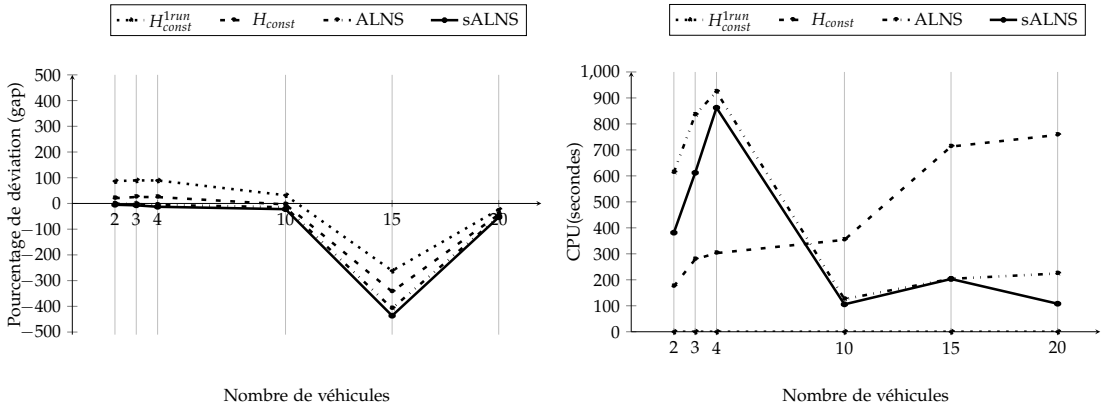
De la Figure 4.2, nous remarquons que le gap d'sALNS est toujours négatif. De plus, sALNS atteint toujours les valeurs minimales en comparaison avec les autres heuristiques. Nous remarquons également que la déviation d'sALNS par rapport à CPLEX augmente avec l'augmentation du nombre de clients. Cela est dû au fait que CPLEX ne parvient pas à trouver des solutions de bonne qualité avec seulement 2 heures d'exécution pour ces instances.

Concernant l'ALNS classique, son gap par rapport à CPLEX atteint des valeurs négatives seulement lorsque le nombre de clients est plus grand ou égal à 150. Nous pensons que c'est parce que l'heuristique ne peut visiter certaines régions de l'espace de recherche à cause des raisons suivantes. 1) ALNS n'évalue pas les solutions après les suppressions. Ainsi, le mécanisme de diversification via les opérateurs de suppression ne peut être utilisé. Donc, ALNS ne peut accepter de solutions avec un nombre inférieur de clients, ce qui peut être une caractéristique principale d'une solution optimale. 2) Comme les opérateurs d'insertion d'ALNS ne gèrent pas les variations des chargements au sein des solutions, ALNS ne peut créer davantage d'espace dans les solutions et insérer des clients additionnels. ALNS peut passer à coté de certaines solutions de bonne qualité, particulièrement s'il y a des clients avec de grands profits dont l'insertion ne détériore pas le coût de transport total de façon significative s'ils sont insérés aux bonnes positions. 3) La non-utilisation de la diversification via la mise à jour de température, laisse ALNS coincée dans des optima locaux. 4) La sélection des opérateurs d'ALNS n'est pas adéquate pour le PTPSPD.

Concernant les deux versions de l'heuristique de construction, nous remarquons qu'$H_{const}$ n'atteint de valeurs négatives que lorsque le nombre de clients est égal à 199. $H^{1run}_{const}$ n'atteint les solutions de CPLEX pour aucun ensemble.

En ce qui concerne le temps de calcul, nous remarquons qu'$H^{1run}_{const}$ est l'heuristique

FIGURE 4.3 – *Évaluation des heuristiques selon le nombre de véhicules.*




la plus rapide.

Les temps de calcul d'sALNS sont assez similaires à ceux d'$H_{const}$ lorsque le nombre de véhicules est égal à 50, 75 et 120. Cependant, sALNS est moins rapide qu'$H_{const}$ pour les autres ensembles d'instances. Lorsque nous la comparons avec ALNS, nous remarquons qu'sALNS est plus rapide sur tous les ensembles étudiés.

Dans la Figure 4.3, nous donnons le gap moyen par rapport à CPLEX dans le meilleur des cas (à gauche) et le temps de calcul en secondes (à droite) des heuristiques étudiées en fonction du nombre de véhicules. Toutes les instances sont étudiées. Six ensembles d'instances sont formés en fonction du nombre de véhicules (avec 2, 3, 4, 10, 15 et 20 véhicules). Nous remarquons qu'sALNS donne de meilleurs résultats que les autres heuristiques en termes de gap, pour tous les ensembles d'instances. En ce qui est du temps de calcul, nous pouvons voir qu'sALNS est plus rapide qu'ALNS. sALNS est moins rapide qu'$H_{const}$ pour les instances avec de petits nombres de véhicules (2, 3 et 4). Cependant, lorsque le nombre de véhicules augmente, sALNS devient plus rapide.

Nous remarquons qu'ALNS fournit toujours des gaps négatifs par rapport à CPLEX. De plus, ALNS donne de meilleurs résultats que les deux versions des heuristiques de construction, pour tous les ensembles d'instances. Concernant le temps de calcul, tout comme sALNS, ALNS est moins rapide qu'$H_{const}$ pour les instances avec de petits nombres de véhicules (2, 3 et 4). ALNS est plus rapide pour les instances avec 10 véhicules et plus.

Nous pensons que les temps de calcul d'ALNS et d'sALNS décroissent pour les instances avec un nombre de véhicules plus grand ou égal à 10 parce que de telles instances ont les valeurs d'origine de nombre de véhicules et de limite de capacité (voir Section 4.4.1). Les solutions de ce type d'instances contiennent généralement tous les clients. Ainsi, le problème devient plus facile à résoudre étant donné qu'ALNS et sALNS n'ont qu'à réordonner les clients au lieu de sélectionner le meilleur ensemble de clients puis de le réordonner. D'autre part, il n'y aurait pas de mise à jour de la meilleure solution et de la solution actuelle et leurs métriques après les opérateurs de suppression. D'un autre coté, comme des solutions de bonne qualité sont atteintes rapidement, il y aurait également moins de mises à jour après les insertions. Cela

résulterait en une réduction du temps de calcul.

Nous remarquons que les gaps d'$H_{const}$ par rapport à CPLEX atteignent des valeurs négatives seulement lorsque le nombre de véhicules est plus grand ou égal à 10. Les gaps d'$H^{1run}_{const}$ par rapport à CPLEX atteignent des valeurs négatives pour les instances avec 15 et 20 véhicules.

Depuis le Tableau 4.10, la Figure 4.2 et la Figure 4.3, nous pouvons dire que l'efficacité d'sALNS en termes de qualité des solutions n'est pas affectée par les changements de nombres de clients, de nombres de véhicules ou de limite de capacité. Cependant, le temps de calcul d'sALNS dépend fortement de ces données.

### 4.4.6 Évaluation de la performance d'sALNS sur les instances du VRPSPD

Comme nous l'avons dit dans la Section 4.2, le modèle proposé dans ce travail peut être équivalent à celui proposé par Dell'Amico et al. [27] si tous les profits sont fixés à une très grande valeur en comparaison avec les coûts de transport (ou distances). Alors, nous avons voulu évaluer la performance d'sALNS sur les instances de Dell'Amico et al. [27] avec une matrice symétrique des coûts de transport. Cela est fait en fixant les profits à un très grand nombre.

Dans ce qui suit, nous décrivons les instances utilisées. Ensuite, nous donnons les résultats d'sALNS sur ces instances là. Les résultats d'sALNS sont comparés à ceux obtenus par Dell'Amico et al. [27] en utilisant une méthode de *branch-and-price* (B&P) avec une limite de temps égale à 1 heure.

#### Description des instances du VRPSPD

Nous utilisons les instances de Classe 1 présentées par Dell'Amico et al. [27] pour le VRPSPD, et qui sont basées sur 3 instances de VRPTW (r101, c101 et rc101) proposées par Solomon [90]. Afin d'adapter les instances du VRPTW au VRPSPD, Dell'Amico et al. [27] ont considéré les 20 premiers et les 40 premiers clients de chaque instance. Les fenêtres de temps ont été négligées. Les demandes des instances d'origine sont considérées comme demandes de livraison. Pour chaque client $i$, sa demande de ramassage $p_i$ est générée depuis sa demande de livraison $d_i$ suivant les Expressions $p_i = \lfloor(1-\gamma)\cdot d_i\rfloor$ si $i$ est pair, et $p_i = \lfloor(1+\gamma)\cdot d_i\rfloor$ si $i$ est impair. Le paramètre $\gamma$ prend soit la valeur 0.2 ou bien la valeur 0.8. Cela donne un total de 12 instances différentes. Dell'Amico et al. [27] ont arrondi la distance Euclidienne entre les clients au premier entier supérieur.

Nous testons également les instances de la Classe 3S et la Classe 3C. Les instances de la Classe 3S sont basées sur celles du CVRP qui sont disponibles sur la « VRPLIB » [102], avec 20 et 40 clients. Pour chaque instance du CVRP, le taux de clients de livraison peut prendre les valeurs $\frac{1}{2}$, $\frac{2}{3}$ ou $\frac{4}{5}$, et la capacité varie dans l'ensemble $\{150, 200, 300\}$. Un total de 18 instances sont générées.

La Classe 3C est générée depuis la Classe 3S comme suit : chaque demande de Classe 3S est considérée comme demande de livraison. Pour chaque client $i$, sa demande de ramassage $p_i$ est calculée suivant l'Expression $p_i = (0.5 + r)\cdot d_i$, où $d_i$ est la demande de livraison de $i$ et $r$ est choisi aléatoirement suivant une distribution

uniforme dans l'intervalle $[0, 1]$. La Classe 3C contient 18 instances différentes.

Dans les instances des Classes 3S et 3C, le nombre de véhicules a été calculé comme le nombre minimum de véhicules pour lequel une solution réalisable existe. Voir le travail de Dell'Amico et al. [27] pour plus de détails.

### Étude du comportement d'sALNS sur les instances du VRPSPD

Afin de tester sALNS sur les instances du VRPSDP, nous considérons les instances de Dell'Amico et al. [27] comme des instances de PTPSPD en ajoutant un profit à chaque client. Pour forcer sALNS à favoriser les solutions contenant tous les clients et ainsi, fournir des solutions réalisables pour le VRPSPD, nous ajoutons une grande valeur de profit aux clients des instances étudiées. Les profits sont fixés à 10000. Lorsque sALNS s'arrête, nous vérifions la réalisabilité de la solution obtenue.

Table 4.11 – *Étude du comportement d'sALNS sur les instances du VRPSPD de Classe 1.*

| | | B&P | sALNS | | |
|---|---|---|---|---|---|
| ins | m | coût | coût | gap | CPU |
| c101_20_02 | 4 | 272 | 272 | **0.00** | 35.12 |
| c101_20_08 | 4 | 279 | 279 | **0.00** | 34.42 |
| r101_20_02 | 3 | 329 | 329 | **0.00** | 27.59 |
| r101_20_08 | 3 | 342 | 342 | **0.00** | 30.42 |
| rc101_20_02 | 5 | 428 | 428 | **0.00** | 41.55 |
| rc101_20_08 | 5 | 458 | 458 | **0.00** | 40.54 |
| c101_40_02 | 8 | 551 | 551 | **0.00** | 65.92 |
| c101_40_08 | 8 | 569 | 589 | 3.40 | 68.14 |
| r101_40_02 | 6 | 601 | 596 | **-0.84** | 64.23 |
| r101_40_08 | 6 | 629 | 627 | **-0.32** | 70.55 |
| rc101_40_02 | 9 | 886 | 886 | **0.00** | 72.02 |
| rc101_40_08 | 9 | 926 | 926 | **0.00** | 79.26 |
| moy | | | | 0.19 | 52.48 |

Table 4.12 – *Étude du comportement d'sALNS sur les instances du VRPSPD de Classe 3S et de Classe 3C.*

| | | B&P | sALNS | | | | | B&P | sALNS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ins | m | coût | coût | gap | CPU | ins | m | coût | coût | gap | CPU |
| 3S_20_50_1 | 6 | 8769 | 8769 | **0.00** | 32.82 | 3C_20_50_1 | 11 | 12720 | 12720 | **0.00** | 56.56 |
| 3S_20_50_2 | 4 | 7986 | 7986 | **0.00** | 28.07 | 3C_20_50_2 | 7 | 11559 | 10461 | **-10.50** | 47.85 |
| 3S_20_50_3 | 3 | 6445 | 6445 | **0.00** | 23.70 | 3C_20_50_3 | 6 | 8387 | 8387 | **0.00** | 36.00 |
| 3S_20_66_1 | 7 | 9129 | 9129 | **0.00** | 37.16 | 3C_20_66_1 | 12 | 14578 | 14578 | **0.00** | 80.68 |
| 3S_20_66_2 | 5 | 7470 | 7470 | **0.00** | 33.30 | 3C_20_66_2 | 8 | 11178 | 11176 | **-0.02** | 46.76 |
| 3S_20_66_3 | 3 | 8346 | 7036 | **-18.62** | 26.87 | 3C_20_66_3 | 5 | 8160 | 8160 | **0.00** | 33.03 |
| 3S_20_80_1 | 8 | 10707 | 10707 | **0.00** | 52.23 | 3C_20_80_1 | 11 | 12802 | 12802 | **0.00** | 59.31 |
| 3S_20_80_2 | 6 | 10093 | 9024 | **-11.85** | 37.33 | 3C_20_80_2 | 8 | 10087 | 10087 | **0.00** | 46.10 |
| 3S_20_80_3 | 4 | 7058 | 7058 | **0.00** | 27.63 | 3C_20_80_3 | 5 | 8317 | 8317 | **0.00** | 33.59 |
| 3S_40_50_1 | 10 | 18282 | 18282 | **0.00** | 89.41 | 3C_40_50_1 | 22 | 27559 | 27245 | **-1.15** | 134.04 |
| 3S_40_50_2 | 8 | 14603 | 14603 | **0.00** | 66.40 | 3C_40_50_2 | 16 | 21773 | 21773 | **0.00** | 129.70 |
| 3S_40_50_3 | 5 | 11610 | 11343 | **-2.35** | 55.04 | 3C_40_50_3 | 10 | 15629 | 15523 | **-0.68** | 80.71 |
| 3S_40_66_1 | 12 | 18370 | 18003 | **-2.04** | 78.88 | 3C_40_66_1 | 22 | 25981 | 25981 | **0.00** | 126.07 |
| 3S_40_66_2 | 9 | 15307 | 15307 | **0.00** | 67.95 | 3C_40_66_2 | 15 | 21319 | 21377 | 0.27 | 106.91 |
| 3S_40_66_3 | 6 | 11725 | 11725 | **0.00** | 61.72 | 3C_40_66_3 | 10 | 15293 | 15293 | **0.00** | 79.43 |
| 3S_40_80_1 | 17 | 20665 | 20665 | **0.00** | 120.32 | 3C_40_80_1 | 21 | 26273 | 26617 | 1.29 | 121.09 |
| 3S_40_80_2 | 12 | 17599 | 17503 | **-0.55** | 112.29 | 3C_40_80_2 | 16 | 20652 | 20652 | **0.00** | 108.24 |
| 3S_40_80_3 | 8 | 13317 | 13221 | **-0.73** | 81.83 | 3C_40_80_3 | 10 | 15365 | 15365 | **0.00** | 82.41 |
| moy | | | | **-2.01** | 57.39 | moy | | | | **-0.60** | 78.25 |

Les résultats obtenus sont donnés dans les Tableaux 4.11 et 4.12. Le Tableau 4.11 affiche les résultats pour la Classe 1, tandis que les résultats des Classes 3S et 3C sont affichés sur le Tableau 4.12.

Dans ces Tableaux, *B&P* désigne la meilleure solution obtenue avec la méthode de *branch-and-price* présentée par Dell'Amico et al. [27]. *sALNS* représente l'heuristique proposée dans ce Chapitre. *coût* est le meilleur coût de transport (ou meilleure distance) obtenu, et *CPU* est le temps de calcul correspondant en secondes. *ins* désigne le nom de l'instance, telle qu'une instance $tw\_n\_val_\gamma$ de Classe 1 est une instance générée en utilisant l'instance de VRPTW $tw$ avec $n$ clients et $\gamma$ est égal à $val_\gamma$. Dans une instance $Class\_n\_frac\_Q$ de Classes 3S ou 3C, $Class$ désigne le nom de la classe (3S ou 3C), $n$ est le nombre de clients, $frac \in \{50, 66, 80\}$ est le taux de clients de livraison, où 50 représente la proportion $\frac{1}{2}$, 66 représente $\frac{2}{3}$ et 80 représente $\frac{4}{5}$. $Q$ est la limite de capacité. $m$ désigne le nombre de véhicules. $gap$ représente le gap d'sALNS par rapport à la meilleure solution obtenue par Dell'Amico et al. [27]. Le gap d'une solution d'sALNS par rapport à une solution de B&P est calculé suivant l'Expression $gap = 100 \cdot (z_{sALNS} - z_{B\&P}) / |z_{sALNS}|$, où $z_{sALNS}$ et $z_{B\&P}$ sont les valeurs de la fonction objectif atteintes par sALNS et B&P respectivement.

Du Tableau 4.11, nous remarquons que, pour les instances de Classe 1, sALNS atteint pratiquement toutes les solutions fournies par B&P en un temps de calcul raisonnable. sALNS n'atteint pas les solutions de B&P pour l'instance c101_40_08 et obtient un gap de 3.40%. Nous remarquons que pour cette instance, sALNS utilise un véhicule de plus (9 au lieu de 8) que B&P. D'autre part, sALNS fournit une meilleure solution que B&P pour les instances r101_40_02 et r101_40_08. Notons que, dans l'instance r101_40_08, sALNS utilise plus de véhicules que B&P (7 au lieu de 6).

Du Tableau 4.12, nous pouvons voir qu'sALNS donne des solutions de meilleure qualité ou bien des solutions égales à celle de B&P pour les instances de Classe 3S. D'un autre coté, pour les instances de Classe 3C, sALNS n'arrive pas à atteindre les solutions de B&P pour les instances 3C_40_66_2 et 3C_40_80_1.

### 4.4.7 Évaluation de la performance d'sALNS sur les instances du CPTP

Nous avons également exécuté sALNS sur les instances du CPTP proposées par Archetti et al. [2] et nous avons comparé les résultats obtenus avec ceux d'Archetti et al. [2] ainsi qu'avec ceux d'ILS_RVND_LNS.

Le Tableau 4.13 résume les résultats obtenus. Les résultats détaillés sont donnés en Annexe (voir A.4). Dans le Tableau 4.13, *gap* représente le pourcentage moyen de déviation de chaque heuristique par rapport à la meilleure solution obtenue parmi les solutions de VNS, TF, TA, ILS_RVND_LNS et sALNS. *CPU(min)* désigne le temps de calcul de chaque heuristique en minutes.

Depuis le Tableau 4.13, nous remarquons qu'sALNS possède le plus petit pourcentage de déviation moyen par rapport aux autres approches. De plus, le temps de calcul d'sALNS est le second plus petit temps de calcul.

Les tests effectués ont montré qu'sALNS parvenait à atteindre les solutions des

Table 4.13 – *Comparaison d'sALNS avec les méthodes d'Archetti et al. [2] et ILS_RVND_LNS sur les instances du CPTP.*

| | VNS | TF | TA | ILS_RVND_LNS | sALNS |
|---|---|---|---|---|---|
| gap | 0.28 | 0.87 | 0.83 | 0.76 | **0.07** |
| CPU (min) | 10.3 | **2.83** | 8.54 | 9.94 | 5.82 |

autres approches dans 83 cas sur 117 et à fournir de meilleures solutions dans 17 cas. sALNS n'a pu atteindre les solutions des autres approches dans 17 cas. Le temps de calcul d'sALNS reste raisonnable pour le CPTP.

## Conclusion

Dans ce Chapitre, nous avons décrit le *problème de la tournée la plus profitable avec ramassages et livraisons simultanés* (Profitable Tour Problem with Simultaneous Pickup and Delivery services en anglais ou PTPSPD) et nous avons proposé une formulation mathématique sous forme de programme linéaire en variables mixtes. Nous avons donné une nouvelle heuristique de construction ainsi qu'une *recherche adaptative sélective à grand voisinage* (selective Adaptive Large Neighborhood Search en anglais ou sALNS).

sALNS utilise plusieurs composantes. Les plus importantes sont la procédure d'évaluation de la solution, la procédure de sélection des opérateurs, les opérateurs de suppression/insertion modifiés et la diversification via la mise à jour de la température.

sALNS est testée sur 117 instances basées sur celles de la littérature. Certains paramètres sont ajoutés aux instances de la littérature pour les rendre compatibles avec le PTPSPD.

Pour montrer l'efficacité de l'heuristique proposée, nous avons d'abord évalué ses composantes. Après cela, nous avons implémenté une version d'sALNS (notée *Li*) qui utilise les opérateurs de suppression/insertion présentés par Li et al. [65]. Les solutions de *Li* sont comparées avec celles d'sALNS. Par la suite, nous avons exécuté la *recherche adaptative à grand voisinage* (ALNS) ainsi que notre modèle mathématique en utilisant le solveur CPLEX.

Les résultats expérimentaux ont montré qu'sALNS donne des résultats compétitifs en comparaison avec *Li*, ALNS et CPLEX, que ce soit en termes de qualité des solutions ou de temps de calcul.

Nous avons également évalué la performance d'sALNS sur des instances du *problème de tournées de véhicules avec ramassages et livraisons simultanés* (Vehicle Routing Problem with Simultaneous Pickup and Delivery services en anglais ou VRPSPD) et du *problème de la tournée la plus profitable avec contrainte de capacité* (Capacitated Profitable Problem en anglais ou CPTP) issues de la littérature. Les expérimentations ont montré que l'heuristique proposée donne des résultats de bonne qualité en un temps raisonnable.

Un prochain travail peut considérer l'application d'sALNS à d'autres variantes du *problème de ramassages et livraisons avec profits* (Pickup and Delivery Problem with Pro-

fits en anglais), incluant différentes contraintes réalistes telles que la durée maximum des tournées, l'utilisation d'une flotte hétérogène de véhicules et la considération de fenêtres de temps. Cela pourrait se faire en incorporant ces contraintes dans les critères de sélection des opérateurs d'insertion par exemple.

D'autre part, le PTPSPD pourrait être étudié avec des contraintes réalistes additionnelle telles les fenêtres de temps, ou avec l'objectif de maximisation des profits sous contraintes de durée maximum des tournées. Cela résulterait en un *problème de la course d'orientation avec ramassages et livraisons simultanés* (ou Team Orienteering Problem with Simultaneous Pickup and Delivery services en anglais).

L'étude présentée dans ce Chapitre a fait l'objet d'un article publié [19].

# Conclusion générale

Dans cette thèse nous nous sommes intéressés à deux variantes du *problème de tournées de véhicules* (VRP) avec opérations de livraison et/ou de ramassage. Ce type de problèmes vise à concevoir des schémas de transport efficaces incluant des opérations de ramassage et/ou de livraison.

Les problèmes étudiés sont particulièrement importants vu qu'ils prennent en compte certaines contraintes environnementales. En effet, d'une part, les entreprises s'engagent à prendre en charge leurs produits tout au long de leurs cycles de vie grâce aux opérations de ramassage des déchets ou appareils usés. Ces opérations peuvent être combinées, au sein d'une même tournée, avec des livraisons de produits finis. D'autre part, les problèmes étudiés cherchent à optimiser l'utilisation des capacités des véhicules en visitant des clients additionnels sur le chemin du retour. Ceci contribue à la réduction des embouteillages et des émissions de gaz à effets de serre, en évitant de mobiliser davantage de véhicules pour les opérations additionnelles.

Les variantes de VRP traitées dans cette thèse sont le *problème de la tournée la plus profitable avec contrainte de capacité* (CPTP) et le *problème de la tournée la plus profitable avec ramassages et livraisons simultanés* (PTPSPD). Ces deux variantes font également partie de la famille des *problèmes de tournées de véhicules avec profits*.

Dans le Chapitre 1, nous avons donné quelques définitions et notions de bases de programmation linéaire et de théorie des graphes. Ces définitions ont pour but d'aider le lecteur à se familiariser avec les termes techniques pour bien comprendre les autres Chapitres. Nous avons également présenté la famille des *problèmes de tournées de véhicules*.

Dans le Chapitre 2, nous avons décrit les principales méthodes de résolution du *problème de tournées de véhicules*. Ceci a pour but d'aider également le lecteur à comprendre les autres Chapitres.

Le Chapitre 3 traite le CPTP qui est un problème connu dans la littérature. Dans un CPTP, il est question de sélectionner un ensemble de clients à visiter afin de concevoir des tournées profitables. Le choix de sélection des clients à visiter se fait en fonction des profits obtenus lors de la visite de ces derniers et en fonction du coût de transport engendré lors de chaque visite.

Afin de tenter de résoudre le CPTP, nous avons d'abord revu les travaux de la littérature. Puis, nous avons proposé plusieurs approches utilisant la *recherche locale réitérée* (ILS), la *recherche à grand voisinage* (LNS), plusieurs opérateurs de voisinage et la *descente à voisinage variable avec ordre aléatoire des voisinages* (RVND). Plusieurs combinaisons des méthodes implémentées ont été étudiées et testées sur les instances du CPTP de la littérature.

Nous avons remarqué que plus une méthode était hybridée et plus elle arrivait à de meilleurs résultats. Cependant, le temps de calcul des méthodes hybrides pouvait être plus important que celui des méthodes élémentaires. Nous avons supposé que cela était dû au fait que les méthodes élémentaires se retrouvaient rapidement coincées dans des optima locaux.

La meilleure méthode parmi celles étudiées en termes de qualité des solution est donnée par l'hybridation d'ILS, RVND et LNS. Cette méthode est notée ILS_RVND_LNS.

ILS_RVND_LNS a aussi été comparée avec d'autres méthodes de la littérature du CPTP. ILS_RVND_LNS a pu atteindre des résultats compétitifs, que ce soit en termes de temps de calcul ou de qualité des solutions.

Dans le Chapitre 4, nous avons étudié le PTPSPD. Au meilleur de notre connaissance, ce problème n'a jamais été étudié auparavant. Le PTPSPD est une version étendue du CPTP dans laquelle des opérations de ramassage et de livraison sont requises pour chaque client. Ces opérations doivent être effectuées simultanément (en une seule visite).

Nous avons, dans un premier temps, présenté une revue de la littérature pour les problèmes plus ou moins similaires au PTPSPD. Après cela, nous avons proposé une formulation mathématique pour le problème et nous avons mis en place une heuristique de type *recherche adaptative à grand voisinage* (ALNS). L'heuristique proposée prend en charge l'aspect sélectif du PTPSPD. Pour cette raison, nous l'avons appelée *recherche adaptative sélective à grand voisinage* (sALNS). Plusieurs composantes de l'ALNS classique ont été modifiées pour rendre la nouvelle heuristique sALNS plus adaptée au PTPSPD.

Afin d'évaluer sALNS, nous avons étudié plusieurs versions de cette heuristique avec et sans chaque composante ajoutée. Les versions étudiées ont été comparées entre elles, avec l'ALNS classique et avec une autre version d'sALNS utilisant des opérateurs de suppression/insertion proposés dans la littérature pour une autre variante du *problème de tournées de véhicules* avec ramassages, livraisons et profits.

Nous avons aussi comparé les résultats d'sALNS avec ceux du solveur CPLEX utilisant le modèle mathématique proposé et avec l'ALNS classique. Plusieurs analyses des méthodes en fonction des données des instances ont été faites. Cela a montré que les changements de paramètres dans les données du problème n'affectaient pas la qualité des solutions d'sALNS. Cependant, le temps calcul de la méthode est sensible à ces changements.

Après cela, nous avons exécuté sALNS sur des instances de la littérature du *problème de tournées de véhicules avec ramassages et livraisons simultanés* (VRPSPD) et du CPTP. L'étude expérimentale a permis de montrer que les résultats de l'heuristique proposée sont compétitifs.

Dans la continuité directe de notre travail, nous pouvons tester les méthodes proposées sur d'autres variantes du *problème de tournées de véhicules* et notamment sur des *problèmes de tournées de véhicules* avec ramassages, livraisons et profits.

Nous pensons également qu'il serait possible d'étendre le PTPSPD en ajoutant certaines contraintes réalistes telles que les celles de fenêtres de temps ou de flotte hétérogène de véhicules ou encore, définir le problème avec un objectif de minimisation de la consommation du carburant.

# Annexes

A

## A.1 Comparaison des résultats d'ILS_RVND_LNS avec ceux d'Archetti et al. [2] sur les instances du CPTP

Dans le Tableau suivant, Ins est le nom de l'instance telle que dans un nom de type $px - y - z$, $x$ représente le numéro de l'instance, $y$ est le nombre de véhicules disponibles et $z$ est la capacité des véhicules. $n$ est le nombre de clients. *VNS*, *TF* et *TA* font respectivement référence à la *recherche à voisinage variable*, la *recherche taboue à solutions réalisables* et la *recherche taboue à solutions irréalisables* d'Archetti et al. [2]. Alors que *ILS_RVND_LNS* est la meilleures heuristique parmi celles que nous avons proposées dans le Chapitre 3 . *CPU* désigne le temps de calcul (en secondes) de la méthode proposée et la dernière ligne donne les temps de calcul moyens (en minutes) pour chaque approche. Enfin, une valeur affichée en gras indique que la meilleure solution est atteinte.

Table A.1 – *Résultats détaillés d'ILS_RVND_LNS sur les instances du CPTP.*

| Instance | n | VNS | TF | TA | ILS_RVND_LNS | CPU |
|---|---|---|---|---|---|---|
| p06-10-160 | 50 | 258.97 | 258.97 | 255.38 | **259.12** | 356.73 |
| p07-20-140 | 75 | **534.81** | 525.06 | 527.90 | 524.39 | 1006.98 |
| p08-15-200 | 100 | **663.98** | 657.31 | 656.32 | 649.67 | 2330.63 |
| p09-10-200 | 150 | 1189.33 | **1192.68** | 1143.65 | 1162.55 | 4194.31 |
| p10-20-200 | 199 | **1773.65** | 1761.37 | 1759.81 | 1741.43 | 10736.14 |
| p13-15-200 | 120 | 284.71 | 269.74 | 274.28 | **289.59** | 2417.94 |
| p14-10-200 | 100 | **890.44** | 886.78 | 888.18 | **890.44** | 1283.71 |
| p15-15-200 | 150 | **1168.63** | 1156.01 | 1134.17 | 1157.38 | 4945.23 |
| p16-20-200 | 199 | **1791.78** | 1764.15 | 1776.41 | 1747.06 | 8435.07 |
| p06-2-50 | 50 | **33.88** | **33.88** | **33.88** | **33.88** | 38.71 |
| p07-2-50 | 75 | **49.18** | **49.18** | **49.18** | **49.18** | 51.73 |
| p08-2-50 | 100 | **57.75** | **57.75** | **57.75** | **57.75** | 76.92 |
| p09-2-50 | 150 | **65.03** | 63.89 | **65.03** | **65.03** | 97.6 |
| p10-2-50 | 199 | **70.87** | **70.87** | **70.87** | **70.87** | 129.57 |
| p13-2-50 | 120 | **64.12** | **64.12** | **64.12** | **64.12** | 129.66 |
| p14-2-50 | 100 | **43.26** | **43.26** | **43.26** | **43.26** | 76.34 |
| p15-2-50 | 150 | **64.98** | **64.98** | **64.98** | **64.98** | 107.39 |
| p16-2-50 | 199 | **66.81** | **66.81** | **66.81** | 66.39 | 142.77 |
| p06-3-50 | 50 | **40.95** | **40.95** | **40.95** | **40.95** | 56.78 |
| p07-3-50 | 75 | **69.94** | **69.94** | **69.94** | **69.94** | 74.12 |
| p08-3-50 | 100 | **80.82** | **80.82** | **80.82** | **80.82** | 121.37 |
| p09-3-50 | 150 | **96.16** | **96.16** | **96.16** | **96.16** | 162.38 |
| p10-3-50 | 199 | **103.79** | **103.79** | **103.79** | **103.79** | 209.51 |
| p13-3-50 | 120 | **87.25** | **87.25** | **87.25** | **87.25** | 207.5 |
| p14-3-50 | 100 | **59.43** | **59.43** | **59.43** | **59.43** | 74.87 |
| p15-3-50 | 150 | **96.42** | **96.42** | **96.42** | **96.42** | 173.42 |
| p16-3-50 | 199 | **99.7** | **99.7** | **99.7** | **99.7** | 228.97 |
| p06-4-50 | 50 | **45.43** | **45.43** | **45.43** | **45.43** | 77.98 |
| p07-4-50 | 75 | **90.65** | **90.65** | **90.65** | **90.65** | 99.74 |
| p08-4-50 | 100 | **100.36** | 98.47 | **100.36** | 99.76 | 174.71 |
| p09-4-50 | 150 | **121.35** | **121.35** | **121.35** | **121.35** | 247.87 |
| p10-4-50 | 199 | **134.81** | **134.81** | **134.81** | **134.81** | 297.66 |

TABLE A.1 – *Résultats détaillés d'ILS_RVND_LNS sur les instances du CPTP.*

| Instance | n | VNS | TF | TA | ILS_RVND_LNS | CPU |
|---|---|---|---|---|---|---|
| p13-4-50 | 120 | **104.18** | 103.73 | 103.72 | 103.34 | 243.39 |
| p14-4-50 | 100 | **68.63** | **68.63** | **68.63** | **68.63** | 133.17 |
| p15-4-50 | 150 | **124.02** | **124.02** | **124.02** | 119.52 | 252.23 |
| p16-4-50 | 199 | **131.37** | **131.37** | **131.37** | **131.37** | 327.39 |
| p06-2-75 | 50 | **72.28** | **72.28** | **72.28** | **72.28** | 49.4 |
| p07-2-75 | 75 | **92.44** | **92.44** | **92.44** | **92.44** | 69.34 |
| p08-2-75 | 100 | **106.15** | **106.15** | **106.15** | **106.15** | 143.26 |
| p09-2-75 | 150 | **117.66** | **117.66** | **117.66** | **117.66** | 136.25 |
| p10-2-75 | 199 | **124.85** | **124.85** | **124.85** | **124.85** | 167.55 |
| p13-2-75 | 120 | **110.12** | **110.12** | **110.12** | **110.12** | 155.25 |
| p14-2-75 | 100 | **77.09** | **77.09** | **77.09** | **77.09** | 70.38 |
| p15-2-75 | 150 | **120.93** | **120.93** | **120.93** | **120.93** | 144.84 |
| p16-2-75 | 199 | **123.38** | **123.38** | **123.38** | **123.38** | 179.58 |
| p06-3-75 | 50 | **92.32** | **92.32** | **92.32** | **92.32** | 75.7 |
| p07-3-75 | 75 | **131.12** | **131.12** | **131.12** | **131.12** | 106.59 |
| p08-3-75 | 100 | **147.55** | **147.55** | 145.87 | **147.55** | 238.69 |
| p09-3-75 | 150 | **160.96** | **160.96** | **160.96** | 160.66 | 238.63 |
| p10-3-75 | 199 | **177.9** | **177.9** | 176.50 | 176.22 | 263.4 |
| p13-3-75 | 120 | **139.37** | 137.95 | 137.45 | **139.37** | 375.34 |
| p14-3-75 | 100 | **112.56** | 112.51 | **112.56** | **112.56** | 104.9 |
| p15-3-75 | 150 | **174.58** | **174.58** | **174.58** | **174.58** | 225.78 |
| p16-3-75 | 199 | **179.55** | **179.55** | 179.23 | 177.35 | 279.69 |
| p06-4-75 | 50 | **99.37** | **99.37** | **99.37** | **99.37** | 100.6 |
| p07-4-75 | 75 | **158.11** | **158.11** | **158.11** | **158.11** | 147.84 |
| p08-4-75 | 100 | **185.27** | **185.27** | **185.27** | 181.42 | 400.73 |
| p09-4-75 | 150 | **204.25** | 203.24 | 203.24 | 201.47 | 335.87 |
| p10-4-75 | 199 | **229.27** | **229.27** | **229.27** | 225.22 | 381.49 |
| p13-4-75 | 120 | **161.62** | 160.68 | 157.98 | 161.59 | 548.71 |
| p14-4-75 | 100 | **139.88** | 139.67 | 139.83 | **139.88** | 187.11 |
| p15-4-75 | 150 | **219.22** | **219.22** | 216.61 | **219.22** | 318.75 |
| p16-4-75 | 199 | **235.03** | **235.03** | **235.03** | 228.49 | 392.83 |
| p06-2-100 | 50 | **100.27** | 99.50 | 99.50 | **100.27** | 63.84 |
| p07-2-100 | 75 | **132.7** | **132.7** | **132.7** | **132.7** | 103.65 |
| p08-2-100 | 100 | **158.21** | **158.21** | **158.21** | **158.21** | 134.63 |
| p09-2-100 | 150 | **161.23** | **161.23** | **161.23** | 161.15 | 176.16 |
| p10-2-100 | 199 | **171.24** | **171.24** | **171.24** | 171.19 | 189.8 |
| p13-2-100 | 120 | **145.75** | 145.67 | 145.67 | **145.75** | 248.41 |
| p14-2-100 | 100 | **125.29** | **125.29** | **125.29** | **125.29** | 123.27 |
| p15-2-100 | 150 | **169.71** | **169.71** | **169.71** | **169.71** | 181.41 |
| p16-2-100 | 199 | **177.23** | **177.23** | 175.57 | 173.56 | 217.8 |
| p06-3-100 | 50 | **134.72** | **134.72** | **134.72** | **134.72** | 93.74 |
| p07-3-100 | 75 | **185.25** | 184.88 | **185.25** | 184.88 | 167.88 |
| p08-3-100 | 100 | **218.63** | **218.63** | 218.33 | 218.43 | 284.91 |
| p09-3-100 | 150 | **230.49** | 229.61 | 229.61 | 229.58 | 284.75 |
| p10-3-100 | 199 | **250.18** | 246.56 | 246.95 | 246.56 | 316.77 |
| p13-3-100 | 120 | **181.63** | 177.76 | 180.04 | 180.79 | 625.97 |
| p14-3-100 | 100 | **182.31** | 179.48 | **182.31** | **182.31** | 195.82 |

Table A.1 – *Résultats détaillés d'ILS_RVND_LNS sur les instances du CPTP.*

| Instance | n | VNS | TF | TA | ILS_RVND_LNS | CPU |
|---|---|---|---|---|---|---|
| p15-3-100 | 150 | **244.08** | 241.84 | **244.08** | 243.89 | 279.08 |
| p16-3-100 | 199 | **258.07** | 257.10 | 252.44 | 255.38 | 333.9 |
| p06-4-100 | 50 | **153.3** | **153.3** | 152.97 | 152.97 | 120.51 |
| p07-4-100 | 75 | **233.4** | **233.4** | 232.05 | 226.61 | 225.8 |
| p08-4-100 | 100 | **268.34** | 266.23 | 266.08 | 259.2 | 504.64 |
| p09-4-100 | 150 | **290.54** | **290.54** | 290.15 | 285.3 | 433.15 |
| p10-4-100 | 199 | **324.02** | 321.17 | 321.03 | 320.07 | 508.78 |
| p13-4-100 | 120 | 200.62 | 178.82 | 183.66 | **202.21** | 1208.47 |
| p14-4-100 | 100 | **237.68** | 236.50 | **237.68** | **237.68** | 251.91 |
| p15-4-100 | 150 | **308.07** | 305.30 | 304.81 | 302.78 | 437.42 |
| p16-4-100 | 199 | **336.24** | 328.20 | 329.53 | 328.29 | 515.7 |
| p06-2-9 | 50 | **168.6** | **168.6** | **168.6** | **168.6** | 93.97 |
| p07-2-9 | 75 | **199.97** | **199.97** | **199.97** | **199.97** | 131.79 |
| p08-2-9 | 100 | **330.14** | 319.28 | 319.28 | 328.37 | 242.88 |
| p09-2-9 | 150 | **347.9** | 347.43 | **347.9** | 343.72 | 299.39 |
| p10-2-9 | 199 | **382.41** | 378.32 | 379.81 | 376.35 | 362.3 |
| p13-2-9 | 120 | **239.57** | 238.58 | 230.59 | 238.58 | 1225.12 |
| p14-2-9 | 100 | **303.17** | 302.94 | **303.17** | 303.14 | 201.63 |
| p15-2-9 | 150 | **378.09** | **378.09** | **378.09** | 376.16 | 322 |
| p16-2-9 | 199 | **394.05** | 390.47 | 391.71 | 389.21 | 372.33 |
| p06-3-9 | 50 | **219.36** | 218.96 | 218.96 | 218.67 | 153.06 |
| p07-3-9 | 75 | **274.8** | **274.8** | **274.8** | 273.27 | 207.18 |
| p08-3-9 | 100 | **447.15** | 444.82 | 433.38 | 444.87 | 538.03 |
| p09-3-9 | 150 | **500.17** | 496.84 | 500.12 | 488.79 | 521.54 |
| p10-3-9 | 199 | **559.8** | 549.83 | 551.44 | 533.15 | 608.92 |
| p13-3-9 | 120 | 250.69 | 234.99 | 244.96 | **283.15** | 1955.02 |
| p14-3-9 | 100 | 418.28 | 416.32 | 417.32 | **419.63** | 331.84 |
| p15-3-9 | 150 | **519.39** | 517.18 | 512.83 | 513.09 | 534.16 |
| p16-3-9 | 199 | **567.24** | 558.61 | 558.10 | 556.53 | 608.02 |
| p06-4-9 | 50 | **258.97** | **258.97** | 254.47 | **258.97** | 213.45 |
| p07-4-9 | 75 | **344.35** | 343.12 | 339.95 | 342.7 | 303.67 |
| p08-4-9 | 100 | 536.64 | **537.66** | 536.13 | 535.26 | 902.31 |
| p09-4-9 | 150 | **639.72** | 635.67 | 633.64 | 621.47 | 785.67 |
| p10-4-9 | 199 | **723.47** | 710.59 | 719.13 | 684.68 | 904.46 |
| p13-4-9 | 120 | 279.43 | 264.46 | 294.46 | **295.77** | 2016.09 |
| p14-4-9 | 100 | **537.24** | 516.20 | 531.53 | 531.94 | 444.02 |
| p15-4-9 | 150 | 653.22 | **654.94** | 652.58 | 651.14 | 803.79 |
| p16-4-9 | 199 | 729.40 | **731.14** | 726.22 | 719.14 | 888.78 |
| CPU (min) | | 10.30 | 2.83 | 8.54 | 9.94 | |

## A.2 Preuve d'élimination des sous-tours

Les Contraintes (4.5), (4.6), (4.8) et (4.9) du modèle mathématique proposé dans la Section 4.2 peuvent se traduire comme suit : le nombre d'arcs entrants un sommet $i \in N$ doit être égal au nombre d'arc sortants de $i$. De plus, ce nombre doit être égal à 0 ou 1. Ainsi, si un client $i \in N$ est inclus dans la solution, il y aura exactement un arc entrant

FIGURE A.1 – *Exemples de configurations de solutions interdites moyennant les Contraintes* (4.5), (4.6), (4.8) *et* (4.9).

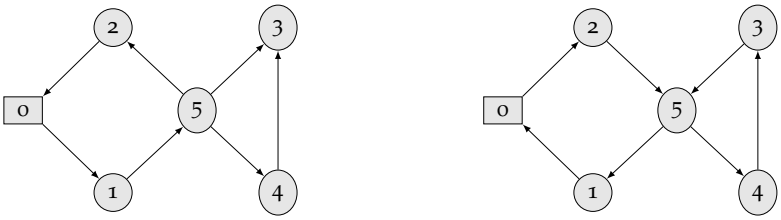



FIGURE A.2 – *Exemple de configurations de solutions interdites qui satisfont les Contraintes* (4.5), (4.6), (4.8) *et* (4.9) *mais qui contiennent quand même un sous-tour.*


et un arc sortant de $i$. Sinon, aucun arc ne pourra ni entrer si sortir de $i$. Donc, le modèle mathématique proposé interdit les solutions (et donc, les sous-tours) incluant différents nombres d'arcs entrants et sortants des sommets de la solution. Un exemple de ce genre de sous-tours est donné dans la Figure A.1.

Dans la Figure A.1, deux graphes avec cinq clients (1, 2, 3, 4 et 5) et un dépôt (0) sont donnés. Sur la partie gauche de la Figure il y a différents nombres d'arcs entrants et sortants du sommet 5, ce qui viole les Contraintes (4.6). Sur la partie droite de la Figure, les mêmes nombres d'arcs entrent et sortent du sommet 5 mais ces nombres sont supérieurs à 1. Si l'on tient compte des Contraintes (4.5) et (4.8), si de telles solutions existent, cela impliquerait qu'il existe un $t_i$ plus grand que 1. Ainsi, les Contraintes (4.9) seraient violées.

Notons que tous les sous-tours ne sont pas pris en compte par les Contraintes (4.5), (4.6), (4.8) et (4.9). Un exemple de sous-tour qui n'est pas pris en compte par ces Contraintes est donné dans la Figure A.2.

Les Contraintes (4.2) et (4.3) prennent en compte les sous-tours restants en supposant que, pour chaque client $i \in N$, les demandes de livraison et de ramassage de $i$ ne peuvent être toutes deux égales à 0. Cette élimination des sous-tours est démontrée comme suit.

Considérons les Contraintes (4.2) et (4.3) du modèle mathématique proposé. Supposons que, pour chaque client $i \in N$ du PTPSPD, la demande de ramassage $p_i$ et la demande de livraison $d_i$ de $i$ ne peuvent être toutes deux égales à 0.

Considérons, à présent, une solution contenant un sous-tour $a_1 - a_2 - a_3 - \ldots - a_{k-1} - a_k - a_1$ avec $k$ clients : $a_1, a_2, a_3, \ldots, a_{k-1}$ et $a_k$. Avec les Contraintes (4.2) et (4.3) du modèle mathématique proposé, nous aurons deux systèmes ($S1$) et ($S2$) respectivement.

$$(S1)\begin{cases} y_{a_1a_2} - y_{a_ka_1} = p_{a_1} \\ y_{a_2a_3} - y_{a_1a_2} = p_{a_2} \\ \vdots \\ y_{a_ka_1} - y_{a_{k-1}a_k} = p_{a_k}. \end{cases} \qquad (S2)\begin{cases} z_{a_ka_1} - z_{a_1a_2} = d_{a_1} \\ z_{a_1a_2} - z_{a_2a_3} = d_{a_2} \\ \vdots \\ z_{a_{k-1}a_k} - z_{a_ka_1} = d_{a_k}. \end{cases}$$

La somme des équations dans $(S1)$ et la somme des équations dans $(S2)$ donnent $\sum_{l=1}^{k} p_{a_l} = 0$ et $\sum_{l=1}^{k} d_{a_l} = 0$ respectivement.

Comme $p_i \geq 0$ et $d_i \geq 0$ pour tout $i \in N$, alors $p_i = 0$ et $d_i = 0$ pour tout $i \in N$, ce qui contredit notre hypothèse de départ. Donc, les solutions du modèle mathématique proposé ne peuvent contenir des sous-tours.

## A.3 Résultats détaillés sur les instances du PTPSPD

Les résultats de tous les algorithmes étudiés pour le PTPSDP sont donnés dans le Tableau A.2. La première colonne affiche le nom des instances, où *ins* représente le nom de l'instance d'origine (voir le travail d'Archetti et al. [2]), $n$ est le nombre de clients, $Q$ est la capacité des véhicules et $m$ est le nombre de véhicules. Le reste des notations sont définies de la même façon que dans le Tableau 4.10. Une valeur affichée en gras représente le meilleur gap atteint pour l'instance étudiée par rapport à la borne supérieure donnée par CPLEX ou par rapport aux solutions de CPLEX.

Table A.2 – *Résultats détaillés de toutes les heuristiques étudiées pour le PTPSPD .*

| | CPLEX | | | $H^{1run}_{const}$ | | $H_{const}$ | | ALNS | | sALNS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ins-n-Q-m | UB | $\%^{UB}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | $\%^{UB}$ | $\%^{ALNS}$ | CPU |
| 6-50-160-10 | 320.31 | **0.00** | 911.98 | 58.69 | 0.00 | 16.11 | 83.62 | 11.45 | 88.64 | **0.00** | **0.00** | -12.93 | 39.56 |
| 7-75-140-20 | 708.96 | 12.45 | 7200.00 | 52.34 | 0.01 | 14.42 | 136.84 | -2.62 | 181.13 | **-7.3** | **6.06** | -4.56 | 54.27 |
| 8-100-200-15 | 777.38 | 5.50 | 7200.00 | 56.99 | 0.03 | 21.56 | 278.91 | 8.22 | 148.36 | **-0.48** | **5.05** | -9.48 | 137.34 |
| 9-150-200-10 | 1430.16 | 41.12 | 7200.00 | 17.90 | 0.06 | -34.68 | 713.52 | -53.53 | 175.52 | **-61.81** | **4.72** | -5.39 | 192.47 |
| 10-199-200-20 | 2166.39 | 100.00 | 7200.00 | **-100.00** | 0.13 | **-100.00** | 1378.49 | **-100.00** | 267.39 | **-100.00** | **6.69** | -3.69 | 161.49 |
| 13-120-200-15 | 610.96 | 37.27 | – | 97.12 | 0.05 | 12.51 | 463.60 | 9.61 | 176.74 | **-23.5** | **22.52** | -36.64 | 355.19 |
| 14-100-200-10 | 1133.48 | 11.18 | 7200.00 | 19.58 | 0.03 | 7.32 | 264.18 | -2.18 | 112.86 | **-5.02** | **6.72** | -2.77 | 84.58 |
| 15-150-200-15 | 1388.34 | 21.90 | 7200.00 | 19.63 | 0.06 | -0.49 | 725.98 | -15.67 | 201.53 | **-21.63** | **5.01** | -5.15 | 151.57 |
| 16-199-200-15 | 2163.25 | 94.77 | 7200.00 | -1220.36 | 0.13 | -1404.72 | 1382.69 | -1633.37 | 287.93 | **-1700.19** | **5.8** | -3.86 | 168.23 |
| 6-50-50-2 | 74.29 | **0.00** | 65.34 | 111.32 | 0.00 | 28.38 | 17.38 | 5.01 | 154.03 | **0.00** | **0.00** | -5.27 | 55.20 |
| 7-75-50-2 | 92.11 | **0.00** | 3249.42 | 86.44 | 0.00 | 29.67 | 31.08 | 7.93 | 289.83 | **0.00** | **0.00** | -8.61 | 24.71 |
| 8-100-50-2 | 91.51 | 12.17 | 7200.00 | 145.19 | 0.01 | 47.21 | 60.17 | 14.12 | 404.24 | **-3.07** | **9.47** | -20.02 | 38.26 |
| 9-150-50-2 | 132.89 | **10.29** | 7200.00 | 138.82 | 0.00 | 19.62 | 121.39 | 15.30 | 727.73 | **0.00** | **10.29** | -18.06 | 73.41 |
| 10-199-50-2 | 143.51 | 9.39 | 7200.00 | 105.46 | 0.02 | 48.76 | 152.32 | 6.03 | 964.45 | **-0.6** | **8.85** | -7.05 | 551.21 |
| 13-120-50-2 | 137.36 | **5.23** | – | 289.03 | 0.00 | 40.09 | 90.97 | 1.29 | 500.71 | **0.00** | **5.23** | -1.31 | 71.72 |
| 14-100-50-2 | 107.47 | **8.86** | 7200.00 | 196.32 | 0.02 | 84.28 | 42.82 | 8.48 | 328.58 | **0.00** | **8.86** | -9.27 | 138.60 |
| 15-150-50-2 | 125.20 | 23.93 | 7200.00 | 96.73 | 0.00 | 13.81 | 133.41 | 1.84 | 819.85 | **-10.23** | **16.15** | -12.29 | 301.23 |
| 16-199-50-2 | 151.95 | 23.92 | 7200.00 | 68.32 | 0.01 | 20.40 | 168.48 | -9.87 | 1118.25 | **-14.08** | **13.2** | -3.83 | 257.02 |
| 6-50-50-3 | 101.50 | **0.00** | 1370.81 | 133.79 | 0.00 | 28.99 | 24.52 | 3.67 | 191.75 | **0.00** | **0.00** | -3.80 | 24.60 |
| 7-75-50-3 | 142.30 | 9.85 | 7200.00 | 141.43 | 0.00 | 45.01 | 113.30 | 6.22 | 532.88 | **-1.65** | **8.36** | -8.39 | 30.28 |
| 8-100-50-3 | 141.63 | 15.78 | 7200.00 | 114.03 | 0.01 | 48.88 | 81.71 | 15.26 | 560.94 | **-0.01** | **15.77** | -18.02 | 43.47 |
| 9-150-50-3 | 193.09 | **12.13** | 7200.00 | 118.84 | 0.00 | 42.69 | 153.38 | 11.93 | 950.32 | **0.44** | 12.52 | -13.05 | 326.25 |
| 10-199-50-3 | 215.51 | 18.92 | 7200.00 | 127.23 | 0.02 | 49.37 | 205.94 | 2.21 | 1400.97 | **-8.74** | **11.83** | -11.20 | 1554.35 |
| 13-120-50-3 | 180.66 | 8.23 | – | 236.55 | 0.00 | 20.09 | 126.41 | 2.00 | 735.96 | **-0.05** | **8.18** | -2.10 | 68.45 |
| 14-100-50-3 | 163.29 | **15.73** | 7200.00 | 177.86 | 0.01 | 106.08 | 54.12 | 8.70 | 514.35 | **0.00** | **15.73** | -9.53 | 386.71 |
| 15-150-50-3 | 184.86 | 20.16 | 7200.00 | 143.39 | 0.02 | 44.23 | 159.15 | 6.95 | 1163.33 | **-1.5** | **18.97** | -9.08 | 619.08 |
| 16-199-50-3 | 222.55 | 26.16 | 7200.00 | 89.68 | 0.01 | 12.35 | 231.08 | -15.52 | 2294.87 | **-16.59** | **13.91** | -0.93 | 250.69 |
| 6-50-50-4 | 131.55 | 13.30 | – | 70.02 | 0.00 | 16.33 | 30.38 | 3.15 | 211.58 | **-1.91** | **11.64** | -5.22 | 27.06 |
| 7-75-50-4 | 184.82 | 14.69 | 7200.00 | 160.51 | 0.02 | 48.93 | 54.99 | 7.58 | 528.80 | **-0.75** | **14.05** | -9.02 | 149.21 |
| 8-100-50-4 | 184.22 | 16.99 | 7200.00 | 127.76 | 0.00 | 55.98 | 109.26 | 11.86 | 733.41 | **-0.09** | **16.91** | -13.55 | 53.81 |
| 9-150-50-4 | 244.85 | **12.22** | 7200.00 | 113.82 | 0.02 | 45.58 | 196.97 | 9.45 | 1231.30 | **3.64** | 15.41 | -6.42 | 1391.83 |
| 10-199-50-4 | 286.57 | 24.57 | 7200.00 | 143.41 | 0.03 | 42.77 | 262.27 | -2.56 | 1779.49 | **-13.98** | **14.02** | -11.13 | 1890.69 |
| 13-120-50-4 | 205.53 | 11.82 | – | 295.44 | 0.01 | 58.03 | 156.62 | 7.36 | 813.76 | **-1.07** | **10.88** | -9.09 | 86.79 |
| 14-100-50-4 | 197.89 | 24.21 | 7200.00 | 220.24 | 0.00 | 124.79 | 67.97 | 3.23 | 627.07 | **-4.63** | **20.7** | -8.13 | 444.00 |

TABLE A.2 – *Résultats détaillés de toutes les heuristiques étudiées pour le PTPSPD .*

| | CPLEX | | | $H_{const}^{1run}$ | | $H_{const}$ | | ALNS | | sALNS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ins-n-Q-m | UB | $\%^{UB}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | $\%^{UB}$ | $\%^{ALNS}$ | CPU |
| 15-150-50-4 | 240.54 | **19.64** | 7200.00 | 114.83 | 0.02 | 43.47 | 197.23 | 11.56 | 1269.55 | **0.77** | 20.26 | -12.20 | 117.68 |
| 16-199-50-4 | 288.94 | 27.77 | 7200.00 | 100.78 | 0.03 | 19.72 | 346.98 | -6.66 | 2103.29 | **-17.31** | **15.27** | -9.98 | 2232.42 |
| 6-50-75-2 | 131.71 | **2.22** | 7200.00 | 87.95 | 0.00 | 6.60 | 25.21 | **0.00** | 153.33 | **0.00** | **2.22** | 0.00 | 58.41 |
| 7-75-75-2 | 175.91 | **1.97** | 7200.00 | 89.50 | 0.00 | 21.03 | 41.95 | 2.66 | 434.68 | **0.00** | **1.97** | -2.73 | 63.74 |
| 8-100-75-2 | 188.40 | 11.63 | 7200.00 | 77.33 | 0.02 | 21.78 | 80.34 | 0.23 | 425.11 | **-1.12** | **10.64** | -1.35 | 72.28 |
| 9-150-75-2 | 234.40 | 19.81 | 7200.00 | 59.50 | 0.01 | 6.81 | 149.55 | -4.17 | 722.07 | **-7.94** | **13.44** | -3.63 | 284.70 |
| 10-199-75-2 | 259.15 | 18.76 | 7200.00 | 83.85 | 0.02 | 12.94 | 209.64 | -4.04 | 1062.37 | **-11.00** | **9.83** | -6.69 | 597.95 |
| 13-120-75-2 | 200.64 | 7.95 | – | 191.76 | 0.00 | 20.72 | 124.46 | 0.11 | 536.49 | **-1.06** | **6.97** | -1.18 | 137.26 |
| 14-100-75-2 | 207.02 | 12.24 | 7200.00 | 95.23 | 0.02 | 56.67 | 54.97 | **-1.88** | 337.41 | **-1.88** | **10.59** | 0.00 | 312.52 |
| 15-150-75-2 | 224.71 | 13.21 | 7200.00 | 95.30 | 0.00 | 34.77 | 152.80 | 3.16 | 743.90 | **-2.59** | **10.96** | -5.94 | 727.41 |
| 16-199-75-2 | 261.15 | 20.85 | 7200.00 | 58.25 | 0.02 | 5.58 | 223.27 | -6.89 | 1256.67 | **-11.46** | **11.79** | -4.27 | 213.37 |
| 6-50-75-3 | 187.08 | 2.88 | 7200.00 | 64.73 | 0.00 | 10.32 | 85.15 | **-0.07** | 165.61 | **-0.07** | **2.81** | 0.00 | 77.43 |
| 7-75-75-3 | 262.35 | 7.80 | 7200.00 | 97.73 | 0.00 | 29.47 | 147.00 | 2.80 | 480.24 | **-0.4** | **7.43** | -3.30 | 51.32 |
| 8-100-75-3 | 280.70 | 17.37 | 7200.00 | 84.36 | 0.01 | 21.01 | 107.08 | -3.11 | 550.24 | **-5.12** | **13.14** | -1.95 | 531.91 |
| 9-150-75-3 | 337.07 | 21.50 | 7200.00 | 55.29 | 0.02 | 20.22 | 203.10 | -6.42 | 1030.84 | **-12.57** | **11.64** | -5.78 | 957.41 |
| 10-199-75-3 | 381.17 | 18.21 | 7200.00 | 75.73 | 0.02 | 17.64 | 290.02 | -6.9 | 1495.39 | **-9.5** | **10.44** | -2.43 | 1623.35 |
| 13-120-75-3 | 244.19 | 11.44 | 7200.00 | 274.51 | 0.01 | 81.04 | 177.22 | 14.72 | 709.52 | **-1.56** | **10.06** | -19.09 | 173.12 |
| 14-100-75-3 | 293.85 | 17.75 | 7200.00 | 98.30 | 0.02 | 44.83 | 81.20 | -0.4 | 463.89 | **-1.1** | **16.85** | -0.70 | 547.76 |
| 15-150-75-3 | 331.42 | 19.55 | 7200.00 | 88.40 | 0.01 | 29.46 | 215.90 | -4.17 | 1025.20 | **-10.1** | **11.42** | -5.69 | 1059.11 |
| 16-199-75-3 | 381.47 | 20.34 | 7200.00 | 84.95 | 0.02 | 10.68 | 311.89 | -8.64 | 1562.04 | **-10.31** | **12.13** | -1.54 | 1491.62 |
| 6-50-75-4 | 223.79 | 12.06 | – | 102.02 | 0.02 | 2.83 | 39.32 | 1.99 | 196.89 | **-2.69** | **9.7** | -4.78 | 28.88 |
| 7-75-75-4 | 326.24 | 9.93 | 7200.00 | 94.86 | 0.00 | 32.30 | 180.40 | 3.95 | 452.75 | **-0.69** | **9.3** | -4.83 | 51.48 |
| 8-100-75-4 | 350.81 | 19.73 | 7200.00 | 61.73 | 0.03 | 34.45 | 132.04 | 1.56 | 987.78 | **-5.12** | **15.62** | -6.78 | 591.81 |
| 9-150-75-4 | 436.45 | 26.50 | 7200.00 | 66.05 | 0.01 | 13.59 | 252.43 | -14.6 | 1237.55 | **-18.62** | **12.81** | -3.51 | 1093.71 |
| 10-199-75-4 | 494.68 | 29.19 | 7200.00 | 71.57 | 0.02 | 17.43 | 367.43 | -22.49 | 1835.17 | **-25.53** | **11.11** | -2.48 | 1954.93 |
| 13-120-75-4 | 265.00 | 18.42 | 7200.00 | 261.64 | 0.03 | 103.00 | 218.34 | 27.39 | 880.01 | **-0.82** | **17.75** | -38.85 | 239.66 |
| 14-100-75-4 | 359.96 | 22.46 | 7200.00 | 126.27 | 0.02 | 48.07 | 106.43 | -6.08 | 918.87 | **-6.72** | **17.24** | -0.60 | 532.04 |
| 15-150-75-4 | 432.34 | 17.18 | 7200.00 | 80.48 | 0.01 | 34.25 | 263.62 | -1.65 | 1177.79 | **-3.99** | **13.88** | -2.30 | 621.64 |
| 16-199-75-4 | 494.95 | 32.05 | 7200.00 | 31.34 | 0.03 | 1.77 | 387.54 | -24.19 | 1871.36 | **-30.62** | **11.24** | -5.18 | 2023.83 |
| 6-50-100-2 | 189.62 | **3.78** | 7200.00 | 37.60 | 0.00 | 6.06 | 33.09 | 0.14 | 145.06 | **0.00** | **3.78** | -0.14 | 87.57 |
| 7-75-100-2 | 252.08 | **0.00** | 6443.98 | 55.06 | 0.01 | 15.30 | 134.34 | 0.86 | 264.61 | **0.00** | **0.00** | -0.87 | 98.81 |
| 8-100-100-2 | 282.20 | 9.45 | 7200.00 | 61.79 | 0.00 | 11.85 | 102.45 | 0.02 | 406.08 | **-3.64** | **6.15** | -3.66 | 373.40 |
| 9-150-100-2 | 331.86 | 14.83 | 7200.00 | 68.15 | 0.02 | 4.80 | 194.13 | -4.2 | 717.37 | **-6.5** | **9.3** | -2.21 | 763.87 |
| 10-199-100-2 | 363.83 | 21.52 | 7200.00 | 62.51 | 0.01 | 4.22 | 271.48 | -17.26 | 1098.47 | **-18.04** | **7.36** | -0.67 | 1149.03 |
| 13-120-100-2 | 247.99 | **8.9** | 7200.00 | 130.69 | 0.00 | 50.97 | 168.03 | 3.39 | 539.02 | **0.00** | **8.9** | -3.51 | 86.22 |
| 14-100-100-2 | 301.58 | 13.12 | 7200.00 | 65.73 | 0.00 | 35.23 | 75.54 | **-0.42** | 346.12 | **-0.42** | **12.76** | 0.00 | 351.65 |
| 15-150-100-2 | 324.59 | 19.57 | 7200.00 | 57.28 | 0.01 | 13.75 | 200.73 | -8.32 | 716.00 | **-11.93** | **9.97** | -3.33 | 779.67 |
| 16-199-100-2 | 367.92 | 29.50 | 7200.00 | 60.41 | 0.02 | -10.99 | 287.00 | -20.92 | 1081.64 | **-25.89** | **11.24** | -4.11 | 313.92 |
| 6-50-100-3 | 251.97 | **4.64** | 7200.00 | 57.58 | 0.00 | 12.29 | 41.66 | 7.98 | 152.19 | **0.00** | **4.64** | -8.67 | 28.00 |
| 7-75-100-3 | 369.46 | 8.92 | 7200.00 | 63.38 | 0.01 | 14.12 | 181.18 | 0.36 | 316.90 | **-1.79** | **7.29** | -2.15 | 353.71 |
| 8-100-100-3 | 397.92 | 12.85 | 7200.00 | 49.37 | 0.02 | 14.80 | 138.92 | 2.16 | 516.27 | **-2.61** | **10.57** | -4.87 | 448.71 |
| 9-150-100-3 | 483.41 | 20.58 | 7200.00 | 61.72 | 0.02 | 5.60 | 262.86 | -10.86 | 992.21 | **-14.69** | **8.91** | -3.46 | 1064.25 |
| 10-199-100-3 | 534.03 | 15.89 | 7200.00 | 70.75 | 0.03 | 12.81 | 379.44 | -5.93 | 1532.58 | **-9.08** | **8.25** | -2.97 | 1609.77 |
| 13-120-100-3 | 314.81 | 24.73 | 7200.00 | 132.89 | 0.01 | 68.26 | 229.85 | 8.61 | 634.27 | **-5.98** | **20.23** | -15.97 | 199.71 |
| 14-100-100-3 | 422.27 | 17.78 | 7200.00 | 40.95 | 0.00 | 30.71 | 106.00 | -0.74 | 677.68 | **-1.29** | **16.72** | -0.54 | 365.88 |
| 15-150-100-3 | 475.73 | 12.40 | 7200.00 | 83.07 | 0.03 | 15.87 | 275.62 | 0.74 | 950.99 | **-4.92** | **8.09** | -5.71 | 1102.24 |
| 16-199-100-3 | 534.67 | 20.46 | 7200.00 | 50.15 | 0.03 | 1.72 | 399.75 | -9.55 | 2204.22 | **-13.55** | **9.69** | -3.66 | 1153.19 |
| 6-50-100-4 | 285.70 | 12.37 | – | 82.35 | 0.00 | 17.25 | 59.21 | -1.07 | 117.45 | **-4.78** | **8.18** | -3.67 | 32.78 |
| 7-75-100-4 | 452.14 | 11.89 | 7200.00 | 60.86 | 0.01 | 11.15 | 93.94 | -0.85 | 363.35 | **-4.71** | **7.74** | -3.83 | 310.26 |
| 8-100-100-4 | 493.59 | 17.80 | 7200.00 | 57.56 | 0.02 | 15.10 | 183.98 | -0.46 | 601.87 | **-6.19** | **12.72** | -5.70 | 523.06 |
| 9-150-100-4 | 621.41 | 22.41 | 7200.00 | 63.48 | 0.03 | 6.73 | 346.95 | -11.36 | 1126.15 | **-14.71** | **11.00** | -3.00 | 983.30 |
| 10-199-100-4 | 691.78 | 28.93 | 7200.00 | 68.33 | 0.03 | 3.00 | 505.25 | -24.33 | 1856.49 | **-27.13** | **9.64** | -2.26 | 3987.88 |
| 13-120-100-4 | 343.80 | 30.72 | 7200.00 | 275.39 | 0.03 | 117.18 | 273.10 | 9.76 | 796.74 | **-13.56** | **21.32** | -25.84 | 240.75 |
| 14-100-100-4 | 529.81 | 22.41 | 7200.00 | 42.55 | 0.01 | 26.67 | 130.44 | -3.96 | 531.08 | **-4.83** | **18.67** | -0.84 | 451.15 |
| 15-150-100-4 | 610.57 | 15.06 | 7200.00 | 58.64 | 0.03 | 16.28 | 330.23 | -4.92 | 1133.26 | **-7.42** | **8.76** | -2.37 | 1226.91 |
| 16-199-100-4 | 695.01 | 24.06 | 7200.00 | 31.20 | 0.03 | 5.23 | 491.18 | -11.26 | 1854.61 | **-19.52** | **9.24** | -7.42 | 4341.49 |
| 6-50-160-2 | 291.51 | **0.00** | 654.86 | 26.54 | 0.01 | 9.52 | 48.34 | 5.01 | 112.28 | **0.00** | **0.00** | -5.27 | 71.23 |
| 7-75-140-2 | 375.96 | 3.17 | 7200.00 | 55.96 | 0.00 | 9.39 | 173.85 | 1.07 | 390.16 | **-0.76** | **2.43** | -1.85 | 199.67 |
| 8-100-200-2 | 568.60 | 10.30 | 7200.00 | 42.22 | 0.02 | 6.91 | 205.47 | **-5.29** | 623.76 | -4.27 | **6.47** | 0.97 | 434.40 |
| 9-150-200-2 | 700.42 | 14.59 | 7200.00 | 36.04 | 0.03 | -2.64 | 403.70 | -8.71 | 772.06 | **-9.08** | **6.83** | -0.34 | 869.22 |
| 10-199-200-2 | 772.22 | 18.22 | 7200.00 | 40.34 | 0.03 | -3.36 | 621.16 | -13.88 | 1156.45 | **-14.72** | **6.18** | -0.74 | 1257.43 |
| 13-120-200-2 | 475.20 | 24.46 | 7200.00 | 138.41 | 0.03 | 37.73 | 350.81 | -9.13 | 472.45 | **-9.84** | **17.03** | -0.65 | 497.88 |
| 14-100-200-2 | 646.13 | 11.27 | 7200.00 | 40.80 | 0.00 | 15.11 | 159.68 | **-0.19** | 334.11 | **-0.19** | **11.1** | 0.00 | 311.42 |
| 15-150-200-2 | 679.83 | 7.89 | 7200.00 | 29.40 | 0.03 | 8.88 | 416.60 | -1.09 | 798.48 | **-3.35** | **4.81** | -2.23 | 830.04 |
| 16-199-200-2 | 760.76 | 12.46 | 7200.00 | 35.13 | 0.05 | 1.62 | 580.23 | -5.64 | 1131.63 | **-7.21** | **6.15** | -1.48 | 1275.39 |
| 6-50-160-3 | 320.31 | **0.00** | 3248.91 | 26.26 | 0.00 | 15.16 | 53.92 | 5.35 | 97.41 | **0.00** | **0.00** | -5.66 | 36.84 |
| 7-75-140-3 | 506.04 | 10.20 | 7200.00 | 42.97 | 0.02 | 12.33 | 231.44 | -4.25 | 384.98 | **-4.45** | **6.21** | -0.19 | 260.67 |

Table A.2 – *Résultats détaillés de toutes les heuristiques étudiées pour le PTPSPD .*

| | CPLEX | | | $H^{1run}_{const}$ | | $H_{const}$ | | ALNS | | sALNS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ins-n-Q-m | UB | $\%^{UB}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | CPU | $\%^{cpx}$ | $\%^{UB}$ | $\%^{ALNS}$ | CPU |
| 8-100-200-3 | 719.14 | 11.77 | 7200.00 | 54.38 | 0.02 | 9.74 | 252.36 | -6.38 | 406.59 | **-9.15** | **3.69** | -2.61 | 110.62 |
| 9-150-200-3 | 975.64 | 15.05 | 7200.00 | 42.73 | 0.03 | 4.88 | 518.02 | -8.03 | 879.10 | **-11.4** | **5.37** | -3.12 | 697.61 |
| 10-199-200-3 | 1094.48 | 25.08 | 7200.00 | 24.11 | 0.05 | -4.81 | 1233.32 | -20.5 | 1480.06 | **-22.34** | **8.35** | -1.52 | 1507.86 |
| 13-120-200-3 | 595.72 | 35.95 | 7200.00 | 109.37 | 0.05 | 40.35 | 642.66 | -16.67 | 343.44 | **-23.69** | **20.78** | -6.01 | 370.13 |
| 14-100-200-3 | 885.82 | 14.67 | 7200.00 | 49.44 | 0.02 | 5.42 | 324.44 | -2.71 | 443.57 | **-3.84** | **11.39** | -1.10 | 355.40 |
| 15-150-200-3 | 943.14 | 20.44 | 7200.00 | 33.98 | 0.03 | 0.90 | 793.81 | -16.17 | 817.76 | **-20.57** | **4.07** | -3.79 | 982.33 |
| 16-199-200-3 | 1082.56 | 30.02 | 7200.00 | 28.16 | 0.05 | -17.74 | 1205.36 | -29.82 | 1404.58 | **-32.77** | **7.09** | -2.27 | 1565.73 |
| 6-50-160-4 | 320.31 | **0.00** | 2956.28 | 53.73 | 0.01 | 16.11 | 85.45 | 11.45 | 49.16 | **0.00** | **0.00** | -12.93 | 37.41 |
| 7-75-140-4 | 605.26 | 16.77 | 7200.00 | 45.15 | 0.02 | 6.44 | 399.91 | -10.83 | 385.05 | **-11.86** | **6.89** | -0.93 | 206.12 |
| 8-100-200-4 | 776.30 | 11.00 | 7200.00 | 37.88 | 0.03 | 15.56 | 442.58 | 1.23 | 218.38 | **-8.2** | **3.71** | -9.55 | 107.98 |
| 9-150-200-4 | 1177.92 | 20.99 | 7200.00 | 20.52 | 0.06 | -0.49 | 950.05 | -16.79 | 918.31 | **-18.63** | **6.27** | -1.57 | 705.87 |
| 10-199-200-4 | 1366.78 | 38.73 | 7200.00 | 6.04 | 0.06 | -27.57 | 929.45 | -47.93 | 1510.91 | **-51.15** | **7.39** | -2.18 | 1530.40 |
| 13-120-200-4 | 610.15 | 33.63 | 7200.00 | 104.82 | 0.03 | 17.18 | 455.74 | 11.90 | 340.17 | **-17.11** | **22.28** | -32.93 | 234.93 |
| 14-100-200-4 | 1064.50 | 17.95 | 7200.00 | 37.57 | 0.03 | 5.56 | 266.69 | 1.14 | 226.01 | **-8.07** | **11.33** | -9.31 | 218.15 |
| 15-150-200-4 | 1141.46 | 19.49 | 7200.00 | 28.63 | 0.11 | -0.55 | 648.02 | -14.15 | 802.87 | **-16.75** | **6.00** | -2.28 | 720.01 |
| 16-199-200-4 | 1361.27 | 100.00 | 7200.00 | **-100.00** | 0.16 | **-100.00** | 934.11 | **-100.00** | 1576.37 | **-100.00** | **7.18** | -2.66 | 1654.06 |
| moy | | 18.05 | 6869.24 | 73.23 | 0.02 | 9.10 | 279.23 | -17.97 | 744.05 | -24.16 | 9.76 | -5.99 | 582.46 |

## A.4 Comparaison des résultats d'sALNS avec ceux d'Archetti et al. [2] et d'ILS_RVND_LNS sur les instances du CPTP

Les résultats d'sALNS sur les instances du CPTP sont donnés dans la colonne *sALNS* du Tableau suivant. *CPU* représente le temps de calcul d'sALNS en seconde. Les autres éléments du Tableau sont décrits de la même façon que dans l'Annexe A.1. Une valeur affichée en gras indique que la meilleure solution est atteinte.

Table A.3 – *Résultats détaillés d'sALNS sur les instances du CPTP.*

| Instance | n | VNS | TF | TA | ILS_RVND_LNS | sALNS | CPU |
|---|---|---|---|---|---|---|---|
| p06-10-160 | 50 | 258.97 | 258.97 | 255.38 | 259.12 | **259.24** | 35.98 |
| p07-20-140 | 75 | 534.81 | 525.06 | 527.9 | 524.39 | **537.57** | 94.19 |
| p08-15-200 | 100 | **663.98** | 657.31 | 656.32 | 649.67 | 659.46 | 132.1 |
| p09-10-200 | 150 | 1189.33 | 1192.68 | 1143.65 | 1162.55 | **1203.36** | 141.42 |
| p10-20-200 | 199 | **1773.65** | 1761.37 | 1759.81 | 1741.43 | 1769.47 | 313.8 |
| p13-15-200 | 120 | 284.71 | 269.74 | 274.28 | 289.59 | **308.55** | 783.24 |
| p14-10-200 | 100 | **890.44** | 886.78 | 888.18 | **890.44** | **890.44** | 100.92 |
| p15-15-200 | 150 | 1168.63 | 1156.01 | 1134.17 | 1157.38 | **1176.38** | 154.81 |
| p16-20-200 | 199 | **1791.78** | 1764.15 | 1776.41 | 1747.06 | 1785.15 | 170 |
| p06-2-50 | 50 | **33.88** | **33.88** | **33.88** | **33.88** | **33.88** | 15.59 |
| p07-2-50 | 75 | **49.18** | **49.18** | **49.18** | **49.18** | **49.18** | 15.91 |
| p08-2-50 | 100 | **57.75** | **57.75** | **57.75** | **57.75** | **57.75** | 25.19 |
| p09-2-50 | 150 | **65.03** | 63.89 | **65.03** | **65.03** | **65.03** | 53.86 |
| p10-2-50 | 199 | **70.87** | **70.87** | **70.87** | **70.87** | **70.87** | 53.55 |
| p13-2-50 | 120 | **64.12** | **64.12** | **64.12** | **64.12** | **64.12** | 76.73 |
| p14-2-50 | 100 | **43.26** | **43.26** | **43.26** | **43.26** | **43.26** | 300.24 |
| p15-2-50 | 150 | **64.98** | **64.98** | **64.98** | **64.98** | **64.98** | 49.47 |
| p16-2-50 | 199 | **66.81** | **66.81** | **66.81** | 66.39 | **66.81** | 51.06 |
| p06-3-50 | 50 | **40.95** | **40.95** | **40.95** | **40.95** | **40.95** | 18.09 |
| p07-3-50 | 75 | **69.94** | **69.94** | **69.94** | **69.94** | **69.94** | 22.33 |
| p08-3-50 | 100 | **80.82** | **80.82** | **80.82** | **80.82** | **80.82** | 34.44 |
| p09-3-50 | 150 | **96.16** | **96.16** | **96.16** | **96.16** | **96.16** | 55.89 |

Table A.3 – *Résultats détaillés d'sALNS sur les instances du CPTP.*

| Instance | n | VNS | TF | TA | ILS_RVND_LNS | sALNS | CPU |
|---|---|---|---|---|---|---|---|
| p10-3-50 | 199 | **103.79** | **103.79** | **103.79** | **103.79** | **103.79** | 69.16 |
| p13-3-50 | 120 | **87.25** | **87.25** | **87.25** | **87.25** | **87.25** | 49.8 |
| p14-3-50 | 100 | **59.43** | **59.43** | **59.43** | **59.43** | **59.43** | 378.26 |
| p15-3-50 | 150 | **96.42** | **96.42** | **96.42** | **96.42** | **96.42** | 52.8 |
| p16-3-50 | 199 | **99.7** | **99.7** | **99.7** | **99.7** | **99.7** | 71.41 |
| p06-4-50 | 50 | **45.43** | **45.43** | **45.43** | **45.43** | **45.43** | 23.56 |
| p07-4-50 | 75 | **90.65** | **90.65** | **90.65** | **90.65** | **90.65** | 26.7 |
| p08-4-50 | 100 | **100.36** | 98.47 | **100.36** | 99.76 | **100.36** | 40.7 |
| p09-4-50 | 150 | **121.35** | **121.35** | **121.35** | **121.35** | **121.35** | 60.47 |
| p10-4-50 | 199 | **134.81** | **134.81** | **134.81** | **134.81** | **134.81** | 77.88 |
| p13-4-50 | 120 | **104.18** | 103.73 | 103.72 | 103.34 | **104.18** | 50.94 |
| p14-4-50 | 100 | **68.63** | **68.63** | **68.63** | **68.63** | **68.63** | 487.18 |
| p15-4-50 | 150 | **124.02** | **124.02** | **124.02** | 119.52 | **124.02** | 60.61 |
| p16-4-50 | 199 | **131.37** | **131.37** | **131.37** | **131.37** | **131.37** | 83.28 |
| p06-2-75 | 50 | **72.28** | **72.28** | **72.28** | **72.28** | **72.28** | 40.08 |
| p07-2-75 | 75 | **92.44** | **92.44** | **92.44** | **92.44** | **92.44** | 23.13 |
| p08-2-75 | 100 | **106.15** | **106.15** | **106.15** | **106.15** | **106.15** | 39.28 |
| p09-2-75 | 150 | **117.66** | **117.66** | **117.66** | **117.66** | **117.66** | 62.28 |
| p10-2-75 | 199 | **124.85** | **124.85** | **124.85** | **124.85** | **124.85** | 78.27 |
| p13-2-75 | 120 | **110.12** | **110.12** | **110.12** | **110.12** | **110.12** | 59.09 |
| p14-2-75 | 100 | **77.09** | **77.09** | **77.09** | **77.09** | **77.09** | 365.25 |
| p15-2-75 | 150 | **120.93** | **120.93** | **120.93** | **120.93** | **120.93** | 64.42 |
| p16-2-75 | 199 | **123.38** | **123.38** | **123.38** | **123.38** | **123.38** | 114.89 |
| p06-3-75 | 50 | **92.32** | **92.32** | **92.32** | **92.32** | **92.32** | 30.8 |
| p07-3-75 | 75 | **131.12** | **131.12** | **131.12** | **131.12** | **131.12** | 29.48 |
| p08-3-75 | 100 | **147.55** | **147.55** | 145.87 | **147.55** | **147.55** | 46.3 |
| p09-3-75 | 150 | **160.96** | **160.96** | **160.96** | 160.66 | **160.96** | 72.56 |
| p10-3-75 | 199 | **177.9** | **177.9** | 176.5 | 176.22 | **177.9** | 124.41 |
| p13-3-75 | 120 | **139.37** | 137.95 | 137.45 | **139.37** | **139.37** | 57.02 |
| p14-3-75 | 100 | **112.56** | 112.51 | **112.56** | **112.56** | **112.56** | 507.52 |
| p15-3-75 | 150 | **174.58** | **174.58** | **174.58** | **174.58** | **174.58** | 70.02 |
| p16-3-75 | 199 | **179.55** | **179.55** | 179.23 | 177.35 | **179.55** | 585.9 |
| p06-4-75 | 50 | **99.37** | **99.37** | **99.37** | **99.37** | **99.37** | 29.27 |
| p07-4-75 | 75 | **158.11** | **158.11** | **158.11** | **158.11** | **158.11** | 31.36 |
| p08-4-75 | 100 | **185.27** | **185.27** | **185.27** | 181.42 | **185.27** | 49.91 |
| p09-4-75 | 150 | **204.25** | 203.24 | 203.24 | 201.47 | **204.25** | 689.52 |
| p10-4-75 | 199 | **229.27** | **229.27** | **229.27** | 225.22 | **229.27** | 1753.88 |
| p13-4-75 | 120 | **161.62** | 160.68 | 157.98 | 161.59 | **161.62** | 60.63 |
| p14-4-75 | 100 | **139.88** | 139.67 | 139.83 | **139.88** | **139.88** | 681.35 |
| p15-4-75 | 150 | **219.22** | **219.22** | 216.61 | **219.22** | **219.22** | 758.82 |
| p16-4-75 | 199 | **235.03** | **235.03** | **235.03** | 228.49 | 234.35 | 312.41 |
| p06-2-100 | 50 | **100.27** | 99.5 | 99.5 | **100.27** | **100.27** | 26.19 |
| p07-2-100 | 75 | **132.7** | **132.7** | **132.7** | **132.7** | **132.7** | 28.23 |
| p08-2-100 | 100 | **158.21** | **158.21** | **158.21** | **158.21** | **158.21** | 51.31 |
| p09-2-100 | 150 | **161.23** | **161.23** | **161.23** | 161.15 | **161.23** | 94.8 |
| p10-2-100 | 199 | **171.24** | **171.24** | **171.24** | 171.19 | **171.24** | 97.99 |
| p13-2-100 | 120 | **145.75** | 145.67 | 145.67 | **145.75** | **145.75** | 73.3 |

TABLE A.3 – *Résultats détaillés d'sALNS sur les instances du CPTP.*

| Instance | n | VNS | TF | TA | ILS_RVND_LNS | sALNS | CPU |
|---|---|---|---|---|---|---|---|
| p14-2-100 | 100 | **125.29** | **125.29** | **125.29** | **125.29** | **125.29** | 399.72 |
| p15-2-100 | 150 | **169.71** | **169.71** | **169.71** | **169.71** | **169.71** | 99.83 |
| p16-2-100 | 199 | **177.23** | **177.23** | 175.57 | 173.56 | **177.23** | 200.97 |
| p06-3-100 | 50 | **134.72** | **134.72** | **134.72** | **134.72** | **134.72** | 27.14 |
| p07-3-100 | 75 | **185.25** | 184.88 | **185.25** | 184.88 | **185.25** | 33.39 |
| p08-3-100 | 100 | **218.63** | **218.63** | 218.33 | 218.43 | **218.63** | 57.86 |
| p09-3-100 | 150 | **230.49** | 229.61 | 229.61 | 229.58 | **230.49** | 141.21 |
| p10-3-100 | 199 | **250.18** | 246.56 | 246.95 | 246.56 | **250.18** | 167.1 |
| p13-3-100 | 120 | **181.63** | 177.76 | 180.04 | 180.79 | **181.63** | 82.89 |
| p14-3-100 | 100 | **182.31** | 179.48 | **182.31** | **182.31** | **182.31** | 482.58 |
| p15-3-100 | 150 | **244.08** | 241.84 | **244.08** | 243.89 | **244.08** | 171.32 |
| p16-3-100 | 199 | 258.07 | 257.1 | 252.44 | 255.38 | **259.25** | 126.95 |
| p06-4-100 | 50 | **153.3** | **153.3** | 152.97 | 152.97 | **153.3** | 29.63 |
| p07-4-100 | 75 | **233.4** | **233.4** | 232.05 | 226.61 | **233.4** | 365.72 |
| p08-4-100 | 100 | **268.34** | 266.23 | 266.08 | 259.2 | 266.98 | 57.02 |
| p09-4-100 | 150 | **290.54** | **290.54** | 290.15 | 285.3 | **290.54** | 981.59 |
| p10-4-100 | 199 | 324.02 | 321.17 | 321.03 | 320.07 | **324.83** | 1696.32 |
| p13-4-100 | 120 | 200.62 | 178.82 | 183.66 | 202.21 | **202.36** | 128.17 |
| p14-4-100 | 100 | **237.68** | 236.5 | **237.68** | **237.68** | **237.68** | 632.62 |
| p15-4-100 | 150 | 308.07 | 305.3 | 304.81 | 302.78 | **309.75** | 895.04 |
| p16-4-100 | 199 | 336.24 | 328.2 | 329.53 | 328.29 | **337.8** | 1924.64 |
| p06-2-9 | 50 | **168.6** | **168.6** | **168.6** | **168.6** | **168.6** | 54.64 |
| p07-2-9 | 75 | **199.97** | **199.97** | **199.97** | **199.97** | **199.97** | 44.27 |
| p08-2-9 | 100 | **330.14** | 319.28 | 319.28 | 328.37 | **330.14** | 387.51 |
| p09-2-9 | 150 | **347.9** | 347.43 | **347.9** | 343.72 | 347.79 | 743.58 |
| p10-2-9 | 199 | **382.41** | 378.32 | 379.81 | 376.35 | **382.41** | 554.35 |
| p13-2-9 | 120 | 239.57 | 238.58 | 230.59 | 238.58 | **240.11** | 717.87 |
| p14-2-9 | 100 | 303.17 | 302.94 | 303.17 | 303.14 | **303.37** | 378.57 |
| p15-2-9 | 150 | **378.09** | **378.09** | **378.09** | 376.16 | **378.09** | 107.25 |
| p16-2-9 | 199 | **394.05** | 390.47 | 391.71 | 389.21 | **394.05** | 783.49 |
| p06-3-9 | 50 | **219.36** | 218.96 | 218.96 | 218.67 | **219.36** | 70.94 |
| p07-3-9 | 75 | **274.8** | **274.8** | **274.8** | 273.27 | 271.69 | 247.88 |
| p08-3-9 | 100 | **447.15** | 444.82 | 433.38 | 444.87 | 444.39 | 75.64 |
| p09-3-9 | 150 | **500.17** | 496.84 | 500.12 | 488.79 | 499.23 | 1230.56 |
| p10-3-9 | 199 | 559.8 | 549.83 | 551.44 | 533.15 | **560.12** | 1574.94 |
| p13-3-9 | 120 | 250.69 | 234.99 | 244.96 | 283.15 | **283.7** | 273.86 |
| p14-3-9 | 100 | 418.28 | 416.32 | 417.32 | 419.63 | **421.79** | 483.35 |
| p15-3-9 | 150 | **519.39** | 517.18 | 512.83 | 513.09 | 517.46 | 456.46 |
| p16-3-9 | 199 | **567.24** | 558.61 | 558.1 | 556.53 | 565.16 | 869.84 |
| p06-4-9 | 50 | **258.97** | **258.97** | 254.47 | **258.97** | **258.97** | 29.77 |
| p07-4-9 | 75 | **344.35** | 343.12 | 339.95 | 342.7 | 343.12 | 176.03 |
| p08-4-9 | 100 | 536.64 | **537.66** | 536.13 | 535.26 | 536.78 | 84 |
| p09-4-9 | 150 | **639.72** | 635.67 | 633.64 | 621.47 | 639.21 | 1420 |
| p10-4-9 | 199 | 723.47 | 710.59 | 719.13 | 684.68 | **724.47** | 3492.82 |
| p13-4-9 | 120 | 279.43 | 264.46 | 294.46 | 295.77 | **298.69** | 218.33 |
| p14-4-9 | 100 | **537.24** | 516.2 | 531.53 | 531.94 | 532.98 | 470.15 |
| p15-4-9 | 150 | 653.22 | **654.94** | 652.58 | 651.14 | 645.54 | 1146.17 |

TABLE A.3 – *Résultats détaillés d'sALNS sur les instances du CPTP.*

| Instance | n | VNS | TF | TA | ILS_RVND_LNS | sALNS | CPU |
|---|---|---|---|---|---|---|---|
| p16-4-9 | 199 | 729.4 | **731.14** | 726.22 | 719.14 | 722.71 | 4725.08 |
| CPU (min) | | 10.30 | 2.83 | 8.54 | 9.94 | 5.82 | |

# Bibliographie


[1] S. Akpinar. Hybrid large neighbourhood search algorithm for capacitated vehicle routing problem. *Expert Systems with Applications*, 61 :28–38, 2016. (Cité page 44.)

[2] C. Archetti, D. Feillet, A. Hertz, and M. G. Speranza. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society*, 60(6) :831–842, 2009. (Cité pages iv, v, vi, 43, 44, 57, 58, 59, 60, 61, 64, 65, 93, 94, 110, 111, 116, 117, 120, 122, 123, 124 et 125.)

[3] C. Archetti, M. G. Speranza, and D. Vigo. Vehicle routing problems with profits. *Vehicle Routing : Problems, Methods, and Applications*, 18 :273, 2014. (Cité pages 16 et 17.)

[4] L. P. Assis, A. L. Maravilha, A. Vivas, F. Campelo, and J. A. Ramírez. Multiobjective vehicle routing problem with fixed delivery and optional collections. *Optimization Letters*, 7(7) :1419–1431, 2013. (Cité pages 44 et 77.)

[5] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6) :621–636, 1989. (Cité page 17.)

[6] J. Bauer, T. Bektaş, and T. G. Crainic. Minimizing greenhouse gas emissions in intermodal freight transport : an application to rail service design. *Journal of the Operational Research Society*, 61(3) :530–542, 2010. (Cité page 18.)

[7] R. Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10) :767–769, 1956. (Cité page 22.)

[8] E. J. Beltrami and L. D. Bodin. Networks and vehicle routing for municipal waste collection. *Networks*, 4(1) :65–94, 1974. (Cité page 18.)

[9] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4) :875–893, 2006. (Cité page 75.)

[10] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems : a classification scheme and survey. *Top*, 15(1) :1–31, 2007. (Cité page 76.)

[11] C. Berge. Graphes et hypergraphes. 1973. (Cité page 6.)

[12] J.-F. Bérubé, M. Gendreau, and J.-Y. Potvin. An exact $\epsilon$-constraint method for bi-objective combinatorial optimization problems : Application to the traveling salesman problem with profits. *European Journal of Operational Research*, 194(1) :39–50, 2009. (Cité page 17.)

[13] N. Bianchessi and G. Righini. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers & Operations Research*, 34(2) :578–594, 2007. (Cité pages 68 et 71.)

[14] J. Brandao. A new tabu search algorithm for the vehicle routing problem with backhauls. *European Journal of Operational Research*, 173(2) :540–555, 2006. (Cité page 74.)

[15] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part i : Route construction and local search algorithms. *Transportation science*, 39(1) :104–118, 2005. (Cité pages 23 et 27.)

[16] F. Carrabs, J.-F. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4) :618–632, 2007. (Cité page 74.)

[17] V. Černỳ. Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1) :41–51, 1985. (Cité pages 31 et 86.)

[18] H. Chentli, R. Ouafi, and W. R. Cherif-Khettaf. Impact of iterated local search heuristic hybridization on vehicle routing problems : Application to the capacitated profitable tour problem. In *Communications in Computer and Information Science*. Springer. Forthcoming chapter. (Cité page 65.)

[19] H. Chentli, R. Ouafi, and W. R. Cherif-Khettaf. A selective adaptive large neighborhood search heuristic for the profitable tour problem with simultaneous pickup and delivery services. *RAIRO-Operations Research*. Forthcoming article. (Cité page 112.)

[20] H. Chentli, R. Ouafi, and W. R. Cherif-Khettaf. Behaviour of a hybrid ils heuristic on the capacitated profitable tour problem. In *Proceedings of the 7th International Conference on Operations Research and Enterprise Systems - Volume 1 : ICORES,*, pages 115–123. INSTICC, SciTePress, 2018. (Cité page 65.)

[21] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979. (Cité pages 57, 94 et 105.)

[22] G. u. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4) :568–581, 1964. (Cité page 24.)

[23] I. M. Coelho, P. L. A. Munhoz, M. N. Haddad, M. J. F. Souza, and L. S. Ochi. A hybrid heuristic based on general variable neighborhood search for the single vehicle routing problem with deliveries and selective pickups. *Electronic Notes in Discrete Mathematics*, 39 :99–106, 2012. (Cité page 76.)

[24] J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8) :928–936, 2001. (Cité page 74.)

[25] D. P. Cuervo, P. Goos, K. Sörensen, and E. Arráiz. An iterated local search algorithm for the vehicle routing problem with backhauls. *European Journal of Operational Research*, 237(2) :454–464, 2014. (Cité page 44.)

[26] M. Dell'Amico, F. Maffioli, and P. Värbrand. On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, 2(3) :297–308, 1995. (Cité page 43.)

[27] M. Dell'Amico, G. Righini, and M. Salani. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transportation Science*, 40(2) :235–247, 2006. (Cité pages 79, 80, 93, 108, 109 et 110.)

[28] G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006. (Cité page 22.)

[29] J. Dethloff. Vehicle routing and reverse logistics : the vehicle routing problem with simultaneous delivery and pick-up. *OR-Spektrum*, 23(1) :79–96, 2001. (Cité pages 69 et 70.)

[30] O. Dominguez, D. Guimarans, A. A. Juan, and I. de la Nuez. A biased-randomised large neighbourhood search for the two-dimensional vehicle routing problem with backhauls. *European Journal of Operational Research*, 255(2) :442–462, 2016. (Cité page 44.)

[31] M. Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992. (Cité page 36.)

[32] B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem : A taxonomic review. *Computers & Industrial Engineering*, 57(4) :1472–1483, 2009. (Cité page 13.)

[33] G. Erdoğan, J.-F. Cordeau, and G. Laporte. The pickup and delivery traveling salesman problem with first-in-first-out loading. *Computers & Operations Research*, 36(6) :1800–1808, 2009. (Cité pages 44 et 74.)

[34] S. Erdoğan and E. Miller-Hooks. A green vehicle routing problem. *Transportation Research Part E : Logistics and Transportation Review*, 48(1) :100–114, 2012. (Cité page 18.)

[35] B. Escoffier and O. Spanjaard. *Dynamic Programming*, pages 71–98. ISTE – Wiley, 2010. (Cité page 22.)

[36] M. K. Fard and M. R. Akbari. A hybrid tabu search algorithm for the vehicle routing problem with simultaneous pickup and delivery and maximum tour time length. *African Journal of Business Management*, 7(11) :801, 2013. (Cité page 73.)

[37] T. A. Feo and M. G. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2) :67–71, 1989. (Cité page 35.)

[38] V. François, Y. Arda, Y. Crama, and G. Laporte. Large neighborhood search for multi-trip vehicle routing. *European Journal of Operational Research*, 255(2) :422–441, 2016. (Cité page 44.)

[39] M. Gansterer, M. Küçüktepe, and R. F. Hartl. The multi-vehicle profitable pickup and delivery problem. *OR Spectrum*, 39(1) :303–319, 2017. (Cité page 44.)

[40] M. Garey and D. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Series of books in the mathematical sciences. W. H. Freeman, 1979. (Cité page 12.)

[41] V. Ghilas, E. Demir, and T. Van Woensel. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research*, 72 :12–30, 2016. (Cité page 75.)

[42] F. Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3) :190–206, 1989. (Cité page 31.)

[43] F. P. Goksal, I. Karaoglan, and F. Altiparmak. A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. *Computers & Industrial Engineering*, 65(1) :39–53, 2013. (Cité page 73.)

[44] P. Grangier, M. Gendreau, F. Lehuédé, and L.-M. Rousseau. A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking. *Computers & Operations Research*, 84 :116–126, 2017. (Cité page 44.)

[45] I. Gribkovskaia, Ø. Halskau, G. Laporte, and M. Vlček. General solutions to the single vehicle routing problem with pickups and deliveries. *European Journal of Operational Research*, 180(2) :568–584, 2007. (Cité page 74.)

[46] I. Gribkovskaia, G. Laporte, and A. Shyshou. The single vehicle routing problem with deliveries and selective pickups. *Computers & Operations Research*, 35(9) :2908–2924, 2008. (Cité pages 19 et 76.)

[47] G. Gutiérrez-Jarpa, G. Desaulniers, G. Laporte, and V. Marianov. A branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows. *European Journal of Operational Research*, 206(2) :341–349, 2010. (Cité page 76.)

[48] K. Halse. *Modeling and solving complex vehicle routing problems*. PhD thesis, Technical University of Denmark, 1992. (Cité page 70.)

[49] H. Hernández-Pérez, I. Rodríguez-Martín, and J. J. Salazar-González. A hybrid grasp/vnd heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 36(5) :1639–1645, 2009. (Cité pages 44 et 75.)

[50] H. Hernández-Pérez, I. Rodríguez-Martín, and J.-J. Salazar-González. A hybrid heuristic approach for the multi-commodity pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 251(1) :44–52, 2016. (Cité page 76.)

[51] M. Hifi and L. Wu. A hybrid metaheuristic for the vehicle routing problem with time windows. In *Control, Decision and Information Technologies (CoDIT), 2014 International Conference on*, pages 188–194. IEEE, 2014. (Cité pages 26 et 89.)

[52] S. C. Ho and W. Szeto. Grasp with path relinking for the selective pickup and delivery problem. *Expert Systems With Applications*, 51 :14–25, 2016. (Cité page 77.)

[53] A. Hoff, I. Gribkovskaia, G. Laporte, and A. Løkketangen. Lasso solution strategies for the vehicle routing problem with pickups and deliveries. *European Journal of Operational Research*, 192(3) :755–766, 2009. (Cité page 74.)

[54] A. Hoff and A. Løkketangen. Creating lasso-solutions for the traveling salesman problem with pickup and delivery by tabu search. *Central European Journal of Operations Research*, 14(2) :125–140, 2006. (Cité page 74.)

[55] J. H. Holland. Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI : University of Michigan Press*, 1975. (Cité page 37.)

[56] G. Iassinovskaia, S. Limbourg, and F. Riane. The inventory-routing problem of returnable transport items with time windows and simultaneous pickup and delivery in closed-loop supply chains. *International Journal of Production Economics*, 2016. (Cité page 75.)

[57] S. Irnich, P. Toth, and D. Vigo. The family of vehicle routing problems. *Vehicle Routing : Problems, Methods, and Applications*, 18 :1, 2014. (Cité page 6.)

[58] Y. Jun and B.-I. Kim. New best solutions to vrpspd benchmark problems by a perturbation based algorithm. *Expert Systems with Applications*, 39(5) :5641–5648, 2012. (Cité page 73.)

[59] J. Kennedy and R. Eberhart. Particle swarm optimization. proceddings of ieee international conference on neural networks,(pp. 1942–1948). *Perth, WA, Australia*, 1995. (Cité page 36.)

[60] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983. (Cité pages 31 et 86.)

[61] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1) :48–50, 1956. (Cité page 38.)

[62] Y. Kuo. Using simulated annealing to minimize fuel consumption for the time-dependent vehicle routing problem. *Computers & Industrial Engineering*, 59(1) :157–165, 2010. (Cité page 18.)

[63] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica : Journal of the Econometric Society*, pages 497–520, 1960. (Cité page 22.)

[64] J. Li, P. M. Pardalos, H. Sun, J. Pei, and Y. Zhang. Iterated local search embedded adaptive neighborhood selection approach for the multi-depot vehicle routing problem with simultaneous deliveries and pickups. *Expert Systems with Applications*, 42(7) :3551–3561, 2015. (Cité page 44.)

[65] Y. Li, H. Chen, and C. Prins. Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. *European Journal of Operational Research*, 252(1) :27–38, 2016. (Cité pages 78, 82, 86, 89, 91, 93, 99, 101 et 111.)

[66] D. Männel and A. Bortfeldt. A hybrid algorithm for the vehicle routing problem with pickup and delivery and three-dimensional loading constraints. *European Journal of Operational Research*, 254(3) :840–858, 2016. (Cité page 75.)

[67] H. Min. The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research Part A : General*, 23(5) :377–386, 1989. (Cité pages 69 et 70.)

[68] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097–1100, 1997. (Cité pages 34 et 74.)

[69] F. Montane and R. D. Galvão. Vehicle routing problems with simultaneous pick-up and delivery service. *OPSEARCH-NEW DELHI-*, 38(6/1) :19–33, 2002. (Cité page 70.)

[70] F. A. T. Montané and R. D. Galvao. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers & Operations Research*, 33(3) :595–619, 2006. (Cité pages 71 et 74.)

[71] V. W. Morais, G. R. Mateus, and T. F. Noronha. Iterated local search heuristics for the vehicle routing problem with cross-docking. *Expert Systems with Applications*, 41(16) :7495–7506, 2014. (Cité page 44.)

[72] G. Nagy and S. Salhi. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European journal of operational research*, 162(1) :126–141, 2005. (Cité pages 70 et 71.)

[73] W. P. Nanry and J. W. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B : Methodological*, 34(2) :107–121, 2000. (Cité page 74.)

[74] U. D. of Energy. Fuel economy guide, 2008. (Cité page 17.)

[75] H. Paessens. The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, 34(3) :336–344, 1988. (Cité pages 24 et 71.)

[76] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1) :21–51, 2008. (Cité pages 15 et 16.)

[77] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1) :81–117, 2008. (Cité pages 15 et 16.)

[78] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8) :2403–2435, 2007. (Cité pages 33, 34, 51, 59, 81, 82, 87, 89, 91, 92, 93, 95 et 96.)

[79] C. Pronello and M. André. Pollutant emissions estimation in road transport models. *INRETS-LTE report*, 2007, 2000. (Cité page 18.)

[80] X. Qiu, S. Feuerriegel, and D. Neumann. Making the most of fleets : A profit-maximizing multi-vehicle pickup and delivery selection problem. *European Journal of Operational Research*, 2016. (Cité page 77.)

[81] I. Rodríguez-Martín and J. J. Salazar-González. A hybrid heuristic approach for the multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *Journal of Heuristics*, 18(6) :849–867, 2012. (Cité pages 44 et 75.)

[82] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4) :455–472, 2006. (Cité pages 32, 74 et 75.)

[83] S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3) :750–775, 2006. (Cité pages 71, 75 et 81.)

[84] S. Salhi and G. Nagy. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the operational Research Society*, pages 1034–1042, 1999. (Cité pages 69, 70 et 94.)

[85] A. Sbihi and R. W. Eglese. Combinatorial optimization and green logistics. *4OR*, 5(2) :99–116, 2007. (Cité page 18.)

[86] F. Schultmann, M. Zumkeller, and O. Rentz. Modeling reverse logistic tasks within closed-loop supply chains : An example from the automotive industry. *European journal of operational research*, 171(3) :1033–1050, 2006. (Cité page 19.)

[87] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming-CP98*, pages 417–431. Springer, 1998. (Cité page 32.)

[88] M. M. Silva, A. Subramanian, and L. S. Ochi. An iterated local search heuristic for the split delivery vehicle routing problem. *Computers & Operations Research*, 53 :234–249, 2015. (Cité page 44.)

[89] P. P. B. Silva and A. H. E. Zuluaga. Description of the classification of publications and the models used in solving of the vehicle routing problem with pickup and delivery. *Revista Ingenierías Universidad de Medellín*, 15(28), 2016. (Cité page 76.)

[90] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2) :254–265, 1987. (Cité pages 26, 45, 84 et 108.)

[91] A. Subramanian and M. Battarra. An iterated local search algorithm for the travelling salesman problem with pickups and deliveries. *Journal of the Operational Research Society*, 64(3) :402–409, 2013. (Cité page 44.)

[92] A. Subramanian, L. M. d. A. Drummond, C. Bentes, L. S. Ochi, and R. Farias. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 37(11) :1899–1911, 2010. (Cité pages 35, 44 et 72.)

[93] A. Subramanian, E. Uchoa, and L. S. Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10) :2519–2531, 2013. (Cité page 44.)

[94] É. D. Taillard, L. M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive memory programming : A unified view of metaheuristics. *European Journal of Operational Research*, 135(1) :1–16, 2001. (Cité page 36.)

[95] E.-G. Talbi. *Metaheuristics : from design to implementation*, volume 74. John Wiley & Sons, 2009. (Cité pages 30, 35 et 36.)

[96] A. S. Tasan and M. Gen. A genetic algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries. *Computers & Industrial Engineering*, 62(3) :755–761, 2012. (Cité page 73.)

[97] C.-K. Ting and X.-L. Liao. The selective pickup and delivery problem : Formulation and a memetic algorithm. *International Journal of Production Economics*, 141(1) :199–211, 2013. (Cité page 77.)

[98] P. Toth and D. Vigo. *Vehicle Routing : Problems, Methods, and Applications*, volume 18. SIAM, 2014. (Cité page 15.)

[99] S. Ubeda, F. Arcelus, and J. Faulin. Green logistics at eroski : A case study. *International Journal of Production Economics*, 131(1) :44–51, 2011. (Cité page 18.)

[100] F. Vanderbeck and L. A. Wolsey. An exact algorithm for ip column generation. *Operations research letters*, 19(4) :151–159, 1996. (Cité page 22.)

[101] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European journal of operational research*, 113(2) :469–499, 1999. (Cité page 35.)

[102] VRPLIB. http ://or.dei.unibo.it/library/vrplib-vehicle-routing-problem-library. (Cité page 108.)

[103] Y. Xiao, Q. Zhao, I. Kaku, and Y. Xu. Development of a fuel consumption optimization model for the capacitated vehicle routing problem. *Computers & Operations Research*, 39(7) :1419–1431, 2012. (Cité page 18.)

[104] E. E. Zachariadis and C. T. Kiranoudis. A local search metaheuristic algorithm for the vehicle routing problem with simultaneous pick-ups and deliveries. *Expert Systems with Applications*, 38(3) :2717–2726, 2011. (Cité page 72.)

[105] E. E. Zachariadis, C. D. Tarantilis, and C. T. Kiranoudis. A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. *Expert Systems with applications*, 36(2) :1070–1081, 2009. (Cité page 71.)

[106] E. E. Zachariadis, C. D. Tarantilis, and C. T. Kiranoudis. An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries. *European Journal of Operational Research*, 202(2) :401–411, 2010. (Cité page 72.)

[107] E. E. Zachariadis, C. D. Tarantilis, and C. T. Kiranoudis. The vehicle routing problem with simultaneous pick-ups and deliveries and two-dimensional loading constraints. *European Journal of Operational Research*, 251(2) :369–386, 2016. (Cité page 76.)

[108] Z. Zhu, J. Xiao, S. He, Z. Ji, and Y. Sun. A multi-objective memetic algorithm based on locality-sensitive hashing for one-to-many-to-one dynamic pickup-and-delivery problem. *Information Sciences*, 329 :73–89, 2016. (Cité page 75.)