Recherche à voisinage variable pour des problèmes de routage avec ou sans gestion de stock

Anis Mjirda

Université de Valenciennes et du Hainaut-Cambrésis

Thèse de doctorat

Pour obtenir le grade de Docteur de l'Université de

VALENCIENNES ET DU HAINAUT-CAMBRESIS

Sciences et Technologie, Mention : Informatique

Présentée et soutenue par Anis MJIRDA

Le 19/09/2014, à Valenciennes

Ecole doctorale :
Sciences Pour l'Ingénieur (SPI)

Equipe de recherche, Laboratoire :
Décision, Intéraction et Mobilité (DIM)
Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH)

Recherche à voisinage variable pour des problèmes de routage avec ou sans gestion de stock

JURY

Président du jury
- FREVILLE Arnaud. Professeur des universités. Université de Valenciennes.

Rapporteurs
- MOUKRIM Aziz. Professeur des universités. Université de Technologie de Compiègne.
- SALHI Said. Professeur des universités. Université du Kent, Royaume-Uni.

Examinateurs
- JARBOUI Bassem. Professeur des universités. Université de Sfax, Tunisie.
- MLADENOVIC Nenad. Professeur des universités. Université de Valenciennes.
- VASQUEZ Michel. Enseignant-Chercheur, HDR. Ecole des Mines d'Alès

Directeur de thèse

- HANAFI Saïd. Professeur des universités. Université de Valenciennes.

Co-encadrant
- WILBAUT Christophe. Maître de conférences. Université de Valenciennes.

Résumé

Dans cette thèse nous nous intéressons à l'étude et à la résolution de problèmes d'optimisation dans le domaine du transport. La première problématique concerne le problème d'élaboration de tournées avec gestion des stocks, et nous considérons dans une seconde partie le problème du voyageur de commerce avec tirant d'eau. Nous avons développé des approches basées sur la recherche à voisinage variable pour résoudre ces problèmes NP-Difficiles, en proposant différentes structures de voisinages et schémas de résolution efficaces. L'évaluation globale des approches proposées sur des instances de la littérature montre leur efficacité. En particulier, nos algorithmes ont amélioré les résultats obtenus par les meilleures approches existantes pour ces deux problèmes.

Mots clés : Optimisation - Routage - Gestion de Stock - Heuristique - Métaheuristique - Recherche à Voisinage Variable.

Abstract

This thesis deals with the study of optimization problems in the transportation domain. We first address the inventory routing problem and we consider the traveling salesman problem with draft limits in a second part. In both cases we have developed methods based on the variable neighborhood search to solve these NP-hard problems. We have proposed several efficient neighborhood structures and solving frameworks. The global evaluation of the proposed approach on sets of benchmarks available in the litterature shows a remarkable efficency and effectiveness. In particular, our algorithms have improved the results obtained by the current best approaches for these two problems.

Keywords : Optimization - Routing - Inventory Management - Heuristic - Metaheuristic - Variable Neighborhood Search

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Introduction générale

Ce mémoire présente une synthèse de mes travaux de thèse au sein du Laboratoire d'Automatique, de Mécanique et d'Informatique industrielles et Humaines, UMR CNRS 8201 (LAMIH). Les problématiques abordées rentrent dans le cadre de l'optimisation combinatoire. L'optimisation combinatoire est une branche de l'optimisation qui traite un ensemble de problèmes pour lesquels l'ensemble des solutions réalisables est fini et discret. La plupart des problèmes étudiés en optimisation combinatoire sont connus comme étant *difficiles* à résoudre, en théorie et/ou en pratique. Cette difficulté se traduit souvent par un nombre exponentiel de solutions à explorer, ce qui rend l'identification d'une solution optimale assez complexe, voir impossible dans certains cas. A partir de ce constat, plusieurs méthodes sont développées pour résoudre ces problèmes de façon efficace. En effet, il existe trois grandes classes de méthodes de résolution de problèmes d'optimisation. La première regroupe les méthodes de résolution dites exactes. Ces méthodes sont basées sur des algorithmes issus de la programmation mathématique, et elles garantissent l'optimalité de la solution obtenue. En contre partie, le temps de résolution est malheureusement généralement important, surtout pour des instances de problèmes de grande taille. La deuxième classe est composée des méthodes de résolution dites approchées, qui regroupe les heuristiques et les métaheuristiques. Ces dernières visent à trouver une solution de "bonne" qualité en un temps raisonnable, sans aucune garantie de son optimalité. La dernière classe de méthodes consiste à hybrider les méthodes de résolution exactes avec les méthodes heuristiques ou les métaheuristiques. Cette classe est connue dans la littérature sous la terminologie *matheuristique*.

Un des problèmes le plus étudié par la communauté de la recherche opérationnelle est sans doute le problème d'élaboration de tournées. Dans ce type de problème, une compagnie cherche à optimiser un certain objectif (minimiser les coûts de transport par exemple), en visitant un ensemble de clients pour livrer (ou ramasser) un ou plusieurs produits. Ce problème se modélise sous la forme d'un graphe dans lequel il s'agit alors de déterminer les itinéraires, visitant un ensemble de sommets, à moindre coût dans un réseau.

Le problème d'élaboration de tournées (ou plus simplement de tournées) apparaît dans plusieurs domaines d'applications réels comme le transport de marchandises, la télécommunication, etc. Le problème du voyageur de commerce est à l'origine de la majorité des problèmes de tournées. Malgré la simplicité de son énoncé, la résolution d'instances de grande taille est loin d'être triviale, même pour certaines approches récentes. Au fur et à mesure du temps et dans le but de s'approcher de la pratique, des contraintes additionnelles ont été ajoutées dans ce problème, ce qui a donné naissance à un grand nombre de variantes de problèmes de tournées.

Dans le cadre de cette thèse, nous nous intéressons à l'étude et à la résolution de deux problèmes de tournées et nous proposons des approches de résolution basées sur la métaheuristique appelée recherche à voisinage variable. Nous nous intéressons en premier lieu à la problématique de l'élaboration de tournées avec prise en compte de la gestion des stocks. Ce problème est connu dans la littérature sous la terminologie anglaise *Inventory Routing Problem (IRP)*. Nous étudions ensuite une variante du problème du voyageur de commerce se rapportant au contexte maritime. Dans ce problème, des contraintes supplémentaires, dites de tirant d'eau, doivent être prises en compte pour déterminer une tournée optimale d'un navire. Ce problème est connu sous la dénomination anglaise *Traveling Salesman Problem with Draft Limits (TSPDL)*.

Le chapitre 1 de cette thèse est centré autour de la recherche à voisinage variable. Après une introduction sur les éléments fondamentaux de la méthode, nous parcourons un ensemble de variantes issues de cette métaheuristique proposée à la fin des années 1990, et nous citons quelques applications existantes.

Dans le chapitre 2, nous nous intéressons à une revue de la littérature sur les problèmes d'élaboration de tournées. Après avoir rapidement passé en revue le problème du voyageur de commerce et la version de base du problème de tournées de véhicules, nous considérons quelques unes des nombreuses variantes traitées par la communauté depuis l'apparition de ce problème dans les années 1950. Nous abordons ensuite les problèmes incluant la gestion de stocks, en mettant en évidence quelques unes des caractéristiques majeures de ces problèmes.

Nous décrivons ensuite une première méthode de résolution pour un problème de routage avec gestion de stocks dans le chapitre 3. Le problème traité est un problème de ramassage dans lequel une flotte homogène de véhicules est disponible pour récupérer chez des fournisseurs un ensemble de produits nécessaires à une usine d'assemblage. La rupture de stock au niveau de l'usine n'est pas autorisée, de manière à garantir une continuité de la production. L'approche que nous proposons se décompose en deux phases : nous construisons une solution initiale à l'aide d'un premier algorithme de Recherche à Voisinage Variable (RVV) dans lequel nous ne considérons pas les coûts de stockage, ce qui permet de se ramener à un problème d'élaboration de tournées pour chaque période de l'horizon de planification. Nous proposons ensuite un second algorithme de RVV au sein duquel nous utilisons une heuristique pour déterminer la quantité de produit à ramasser chez chaque fournisseur. Les résultats obtenus montrent le bon comportement de la méthode, qui visite plusieurs nouvelles meilleures solutions sur un ensemble d'instances existantes.

Le chapitre 4 est dédié à la présentation d'un autre algorithme de RVV pour traiter le même problème. En fait, les résultats obtenus précédemment sont encourageants mais nous proposons une autre stratégie d'exploration de l'espace de recherche pour améliorer ces résultats. La méthode repose sur la mise en œuvre de structures de voisinage utilisant des mouvements de quantités de produits entre les tournées de véhicules. Cela est rendu possible par l'introduction d'une gestion des quantités à l'aide de variables entières, et non plus continues. Les résultats obtenus sur les mêmes instances montrent l'efficacité de la méthode qui améliore pratiquement toutes les meilleures solutions connues.

Finalement, nous considérons le TSPDL dans le chapitre 5 de cette thèse. Pour résoudre efficacement ce problème, nous avons proposé deux approches basées sur la RVV. La première variante utilise un ensemble de voisinages que nous avons définis pour ce problème, et met en œuvre une stratégie du premier voisin améliorant. La seconde approche explore une partie seulement des voisinages, en restant dans le même voisinage tant qu'il est possible d'améliorer la solution courante. Les résultats obtenus sur des instances existantes montrent l'efficacité des méthodes proposées qui convergent, en pratique, vers

toutes les solutions optimales en un temps restreint à 100 secondes. Nous avons également évalué le comportement de nos méthodes sur de nouvelles instances de plus grande taille. Les résultats confirment le bon comportement général de nos algorithmes.

Chapitre 1

Recherche à voisinage variable : un aperçu

1.1 Introduction

Lorsque nous cherchons à résoudre un problème d'optimisation, nous devons mettre en œuvre une méthode (un algorithme) capable de fournir une solution optimale de ce problème en un temps raisonnable. Au delà de la difficulté de trouver cette solution optimale, prouver son optimalité est également une tâche généralement complexe. Différentes classifications des méthodes de résolution peuvent être recensées dans la littérature. En général, deux grandes classes d'approches sont mises en avant : (*i*) les méthodes de résolution dites exactes, et (*ii*) les méthodes de résolution dites approchées. Chacune de ces deux catégories peut à son tour être divisée en plusieurs sous-catégories, avec par exemple les heuristiques, les métaheuristiques ou les méthodes hybrides pour les méthodes approchées. Le fait de pouvoir trouver plusieurs classifications s'explique en partie par le fait qu'il n'est pas toujours facile de choisir une "catégorie" pour une méthode. Ainsi par exemple, un autre terme est apparu dans la littérature il y a quelques années pour désigner les approches combinant des techniques issues de la programmation mathématique et les heuristiques : les *matheuristiques*. Certaines matheuristiques convergent théoriquement vers une solution optimale du problème, et peuvent donc être également vue comme une méthode exacte.

De façon générale, chacune de ces techniques présente des avantages et des inconvénients. Ainsi il est bien connu que les méthodes issues de la programmation mathématique garantissent l'optimalité de la solution retournée. En contre partie, le temps de calcul et/ou les ressources en mémoire nécessaires augmentent en général de façon exponentielle avec la taille des instances

traitées. De l'autre côté, les méthodes approchées, et en particulier les métaheuristiques, permettent en général de fournir des solutions de bonne qualité en un temps acceptable, sans toutefois pouvoir en garantir l'optimalité (et même souvent sans pouvoir garantir un certain écart par rapport à la valeur optimale).

Comme nous l'avons précisé dans l'introduction générale de ce mémoire, nous nous intéressons dans cette thèse à la résolution de deux problèmes de transport difficiles : le premier combine la gestion des stocks et la construction de tournées de ramassage, alors que le second est une variante du problème du voyageur de commerce appliquée dans le secteur maritime. Nous reviendrons plus en détails sur ces problèmes dans les chapitres 3 et 5. Ils font partis de la classe des problèmes NP-difficiles, et nous nous sommes donc concentrés sur la proposition d'approches heuristiques efficaces pour les traiter. Notre attention s'est plus précisément portée sur une métaheuristique aujourd'hui très connue et utilisée, la *Recherche à Voisinage Variable* (RVV) (*Variable Neighborhood Search* selon la terminologie anglaise). Ce premier chapitre est dédié à une présentation générale des concepts intervenants dans la RVV. Nous commençons donc par introduire quelques notions élémentaires dans la section 1.2. Nous abordons ensuite les techniques classiques de descente et de recherche locale dans les sections 1.3-1.4, avant de discuter de la RVV et de passer en revue quelques unes des nombreuses variantes existantes dans la section 1.5.

1.2 Définitions préliminaires

1.2.1 Problème d'optimisation linéaire

Nos travaux se placent dans le contexte de la résolution d'un problème d'optimisation linéaire. Il s'agit de trouver une solution qui minimise (ou maximise) une fonction linéaire, en satisfaisant un ensemble de contraintes linéaires. Dans toute la suite, nous supposerons (sauf mention contraire) que le problème à résoudre est un problème de minimisation. Un problème P peut être représenté par le couple $< \mathcal{S}, f >$, où $\mathcal{S}$ est l'ensemble des solutions réalisables (c'est-à-dire satisfaisant l'ensemble des contraintes), et $f : \mathcal{S} \rightarrow \mathbb{R}$

est la fonction à minimiser (traduisant l'objectif du problème). A chaque solution $x \in \mathcal{S}$ est associé un coût $f(x)$.

Une solution optimale, généralement notée $x^* \in \mathcal{S}$ vérifie l'inégalité suivante : $\forall x \in \mathcal{S}, f(x^*) \leq f(x)$, $f^* = f(x^*)$ est la valeur optimale. L'ensemble des solutions optimales de P est noté $\mathcal{S}^* = \{x \in \mathcal{S} | f(x) = f^*\}$.

1.2.2 Notion de voisinage

Les heuristiques et les métaheuristiques manipulent une ou plusieurs solutions, et mettent souvent en œuvre des stratégies de mouvement d'une solution vers une autre. C'est en particulier le cas avec les techniques basées sur la recherche locale. Cette notion de mouvement d'une solution vers une autre est liée à la définition du voisinage de la solution considérée. Le voisinage d'une solution x correspond à l'ensemble des solutions accessibles depuis x à l'aide d'un mouvement élémentaire. Plus formellement, soit $< \mathcal{S}, f >$ l'instance du problème d'optimisation P traité, et soit $d : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+$ une fonction qui calcule la distance entre deux solutions de $\mathcal{S}$. Un voisinage d'une solution $x \in \mathcal{S}$, est un ensemble $\mathcal{V}(x) \subseteq \mathcal{S}$ des solutions "proches" de x, c'est-à-dire à une distance inférieure à ε donné :

$$\mathcal{V}(x) = \{y \in \mathcal{S} | d(x, y) \leq \varepsilon, \varepsilon \in \mathbb{R}^+\} \tag{1.1}$$

Illustrons cette notion de voisinage sur le problème du voyageur de commerce. L'ensemble des solution S de ce problème est constitué de toutes les permutations possibles des villes. Un voisinage classique pour ce problème appelé $2-opt$ consiste à supprimer deux arêtes de la tournée courante et en ajouter deux autres arêtes.

1.2.3 Optimum local

Pour un problème d'optimisation P donné, un optimum local associé à un voisinage $\mathcal{V}$ est une solution $x^L \in \mathcal{S}$ tel que $f(x^L) \leq f(x)$, $\forall x \in \mathcal{V}(x^L)$.

La notion d'optimum local représente une limite dans l'utilisation et l'efficacité des méthodes de recherche locale que nous abordons dans la suite. Cependant, comme nous le verrons un peu plus loin, différentes techniques

permettent de sortir de ces optima locaux et de relancer la recherche.

1.3 Méthode de descente

Une méthode de recherche locale consiste à explorer le voisinage de la solution courante pour trouver une meilleure solution. Ce processus peut être répété au sein d'une méthode de descente. Il s'agit alors d'une heuristique itérative d'amélioration qui consiste, à partir d'une solution initiale, à répéter le mouvement vers une meilleure solution (du point de vue de l'objectif du problème) dans le voisinage de la solution courante, tant qu'il est possible de trouver une telle solution améliorant le coût. Elle peut être résumée en trois étapes :

1. *Génération d'une solution voisine de* x : la solution courante x est modifiée selon le voisinage $\mathcal{V}(x)$ pour fournir une nouvelle solution x'.
2. *Test* : il faut ensuite décider si le mouvement de la solution x vers la solution x' est accepté. Cette décision dépend, comme nous allons le voir, de la stratégie mise en œuvre. Si le déplacement est accepté, x' remplace x. Sinon, l'évaluation du voisinage (étape 1) se poursuit, au plus jusqu'à avoir évalué toutes les solutions de $\mathcal{V}(x)$.
3. *Condition d'arrêt* : le critère d'arrêt d'une recherche locale "basique" est la non amélioration de la solution courante. Si la solution x est restée inchangée à la fin de l'étape 2, alors l'algorithme s'arrête et la solution courante est retournée. Sinon, le processus reprend à l'étape 1.

Une méthode de descente consiste donc à générer un chemin depuis une solution initiale vers un optimum local dans le voisinage considéré. Deux stratégies d'exploration du voisinage $\mathcal{V}(x)$ peuvent généralement être utilisées : l'exploration du meilleur voisin (*best improvement*) ce qui implique que la structure de voisinage notée $\mathcal{V}$ soit complètement explorée à chaque itération. Un pseudo-code de cette configuration est présenté dans l'algorithme 1.1, avec x la solution initiale.

La seconde stratégie d'exploration consiste à se déplacer vers le premier voisin améliorant. Cette variante est plus rapide puisque le voisinage n'est gé-

Algorithme 1.1 : Descente avec la stratégie du meilleur voisin.

Fonction Meilleur_voisin_améliorant(x)
répéter
  $x' \leftarrow x$;
  $x \leftarrow \text{argmin}_{y \in \mathcal{V}(x')} f(y)$;
jusqu'à $f(x') \leq f(x)$;
retourner x';

néralement pas exploré dans sa totalité. Les différentes étapes sont présentées dans l'algorithme 1.2.

Algorithme 1.2 : Descente avec la stratégie du premier voisin améliorant.

Fonction Premier_voisin_améliorant(x)
répéter
  Soit $\mathcal{V}(x) = \{x^0, x^1, \ldots, x^p\}$;
  $i \leftarrow 0$;
  $x^i \leftarrow x$;
  répéter
    $i \leftarrow i + 1$;
    si $f(x^i) < f(x)$ **alors**
      $x \leftarrow x^i$;
  jusqu'à $f(x) \leq f(x^{i-1})$ ou $i = p$;
jusqu'à $f(x^0) \leq f(x)$;
retourner x^0 ;

Nous abordons dans la suite la notion de changement de voisinage, idée sur laquelle repose la RVV et ses variantes.

1.4 Descente à voisinage variable

A partir des algorithmes de descente présentés précédemment, il est possible de définir une variante dans laquelle plusieurs structures de voisinages sont utilisées. C'est l'idée de base de la descente à voisinage variable [94].

1.4.1 Présentation de la méthode

La descente à voisinage variable (*Variable Neighborhood Descent* (VND) en anglais) consiste donc à explorer l'espace de recherche en utilisant plusieurs structures de voisinages (au minimum deux). Soit $V = \{\mathcal{V}_1, \ldots, \mathcal{V}_{k_{max}}\}$ l'ensemble des structures de voisinages à utiliser. Le changement de voisinage se fait de façon systématique. Pour clarifier la présentation, nous introduisons la procédure de changement de voisinage présentée dans [58]. Les différentes étapes sont décrites dans l'algorithme 1.3. Celui-ci prend en entrée la solution courante, x, une solution x' voisine de x, et une valeur entière notée k, qui correspond à l'indice du voisinage courant ($1 \leq k \leq k_{max}$). Les sorties de l'algorithmes sont la solution x (modifiée si x' est meilleure), et la nouvelle valeur de k.

Algorithme 1.3 : Algorithme de changement de voisinage.

```
Procédure Changer_de_voisinage(x, x', k)
si f(x') < f(x) alors
    x ← x' ;
    k ← 1 ;
sinon
    k ← k + 1 ;
```

Une présentation de la Descente à Voisinage Variable (DVV) est donnée dans l'algorithme 3.4. Cette fonction prend deux paramètres en entrée : la solution initiale, x, et le nombre maximum de structures de voisinages utilisées, k_{max}. L'algorithme va explorer itérativement chacun des voisinages, en recommençant au premier à chaque fois qu'une meilleure solution peut être trouvée (d'après la fonction de changement de voisinage décrite dans l'algorithme 1.3). La méthode s'arrête lorsqu'il n'est plus possible de trouver de meilleure solution sur l'ensemble des k_{max} structures de voisinage.

La descente à voisinage variable a été appliquée à de nombreux problèmes. Nous pouvons par exemple citer le travail de Osman et Ahmadi [101] dans lequel les auteurs ont étudié plusieurs stratégies de sélection pour le problème de la *p-médiane*. Nous pouvons également citer le travail de Cote et al. [35] dans lequel les auteurs ont utilisé une DVV simplifiée et hybridée avec un algorithme évolutionnaire pour la résolution d'un problème spécifique d'emploi

Algorithme 1.4 : Algorithme de descente à voisinage variable.

```
Fonction DVV(x, k_max)
répéter
    stop ← faux ;
    k ← 1 ;
    x' ← x ;
    répéter
        x'' ← argmin_{y∈V(x)} f(y) ;
        Changer_de_voisinage(x, x'', k) ;
    jusqu'à k = k_max ;
    si f(x') ≤ f(x) alors
        stop ← vrai ;
jusqu'à stop = vrai ;
retourner x' ;
```

du temps. La DVV a également été utilisée pour résoudre des problématiques de routage dans divers contextes. Elle a par exemple été utilisée pour résoudre le problème de tournées de véhicules avec capacité dans [20]. Les auteurs ont proposé une méthode hybride qui combine une recherche locale itérative et la DVV. La méthode a aussi été mise en œuvre dans [110] pour résoudre un problème de routage dans un réseaux informatique.

Une des questions soulevées lors du développement d'un algorithme de descente à voisinage variable est l'ordre de parcours des différents voisinages. Plusieurs solutions ont été évoquées dans la littérature, comme dans le travail de Hu et al. [65]. Les auteurs ont défini une méthode auto adaptative pour le choix du voisinage. Une comparaison est effectuée par rapport au choix séquentiel et les résultats montrent que leur approche permet un gain.

1.4.2 Quelques variantes de la descente à voisinage variable

Nous pouvons trouver quelques variantes de la DVV dans la littérature. Nous pouvons en particulier remarquer que la version présentée précédemment est un algorithme séquentiel, dans le sens où chaque voisinage est considéré l'un après l'autre. Deux variantes remettant en cause ce procédé ont été

récemment proposées.

Descente à voisinage variable imbriquée

La première variante est présentée dans Ilic et al. [68]. Les auteurs se sont intéressés à la résolution du problème de la p-médiane avec pivot (p-*hub median problem* en anglais). La méthode consiste à introduire une exploration imbriquée des voisinages dans la DVV, plutôt que l'exploration séquentielle classique décrite précédemment. Pour simplifier et en se limitant à deux voisinages, le principe est d'appliquer le mouvement associé au premier voisinage en chaque point obtenu en appliquant le mouvement associé au second voisinage. Nous pouvons donc observer que dans ce cas le nombre de solutions à considérer dans l'approche séquentielle est égal à $|\mathcal{V}_1| + |\mathcal{V}_2|$, alors qu'il est égal à $|\mathcal{V}_1| \times |\mathcal{V}_2|$ dans cette variante. En généralisant au cas où k_{max} structures de voisinage sont utilisées, le nombre total de solutions à visiter est égal à $\prod_{k=1}^{k_{max}} |\mathcal{V}_k(x)|, x \in \mathcal{S}$. Il est donc clair que la mise en œuvre d'une telle approche avec un nombre de voisinages relativement faible entraîne une augmentation rapide du temps de calcul nécessaire à l'exploration complète de l'espace de recherche.

L'algorithme 1.5 résume les différentes étapes de cette variante, x étant la solution initiale et k_{max} le nombre de structures de voisinages utilisées. Cet algorithme est adapté de celui présenté dans [68].

Descente à voisinage variable mixte

Comme indiquée précédemment, la DVV imbriquée risque d'être difficilement utilisable en pratique, dès que le nombre de voisinages augmente. C'est pourquoi les auteurs de [68] ont proposé dans le même article une autre variante, appelée DVV mixte, dans laquelle plusieurs stratégies sont proposées pour réduire la taille de l'ensemble des solutions à évaluer. Le processus le plus naturel est l'introduction d'un paramètre, notée b par les auteurs, qui correspond au niveau à partir duquel la recherche imbriquée est stoppée, pour laisser place à la recherche séquentielle classique. La valeur de ce paramètre b vérifie nécessairement $1 \leq b \leq k_{max}$, et si $b = 1$ l'algorithme obtenu correspond alors à la DVV séquentielle, alors que lorsque $b = k_{max}$ l'algorithme

Algorithme 1.5 : Descente à voisinage variable imbriquée.

Fonction DVVI(x, k_{max})
$x^* \leftarrow x$;
$k \leftarrow k_{max}$;
répéter
  si $\mathcal{V}_k(x)$ *est complètement exploré* **alors**
    $k \leftarrow k + 1$;
  sinon si $k \geq 2$ **alors**
    $x' \leftarrow x_k \in \mathcal{V}_k(x)$;
    $k \leftarrow k - 1$;
  si $k = 1$ et $f(x') < f(x^*)$ **alors**
    $x^* \leftarrow x'$;
jusqu'à $k = k_{max} + 1$;
retourner x^* ;

obtenu est la DVV imbriquée "totale". La taille de l'ensemble des solutions à explorer avec cette variante est bornée par la valeur suivante :

$$\sum_{k=b}^{k_{max}} |\mathcal{V}_k(x)| \times \prod_{k=1}^{b-1} |\mathcal{V}_k(x)|, x \in \mathcal{S}.$$

1.5 Recherche à voisinage variable

1.5.1 Schéma de base

La recherche à voisinage variable est une métaheuristique dont le schéma de base a été proposé par Mladenović et Hansen [94] dans les années 90. Cette méthode peut être décomposée en deux phases : une phase déterministe au sein de laquelle une recherche locale permet de converger vers un optimum local, et une phase stochastique mise en place pour s'en échapper.

La partie stochastique de l'algorithme consiste à générer, à partir de la solution courante x, une nouvelle solution x' selon un voisinage donné. Cette phase est appelée phase de perturbation (*shaking* en anglais).

Comme pour la descente à voisinage variable, la recherche à voisinage variable utilise une solution initiale x comme point de départ, et un ensemble

de k_{max} voisinages $V = \{\mathcal{V}_1, \ldots, \mathcal{V}_{k_{max}}\}$. A chaque itération, une solution aléatoire x' est générée à partir du voisinage courant $\mathcal{V}_k$. Ensuite, une recherche locale est appliqué sur x' ce qui génère une nouvelle solution x''. Si cette nouvelle solution x'' est meilleure que x', une mise à jour est effectuée et le processus reprend avec le premier voisinage. Dans le cas contraire, les mêmes étapes sont réitérées mais en passant au voisinage suivant $\mathcal{V}_{k+1}$. L'algorithme 1.6 résume les différentes étapes de la méthode. Dans cet algorithme, la notation *rand(S)* désigne la génération aléatoire d'une solution dans l'ensemble S. Notons que la notation *Recherche_Locale* fait ici référence à l'appel d'une méthode d'amélioration qui peut prendre différentes formes : recherche locale simple, méthode de descente, descente à voisinage variable, *etc.* La condition d'arrêt de l'algorithme dépend de l'application et peut par exemple correspondre à un nombre maximum d'itérations ou un temps de calcul alloué.

Algorithme 1.6 : Recherche à voisinage variable.

```
Fonction RVV(x, k_max)
stop ← faux ;
répéter
    k ← 1 ;
    tant que k ≤ k_max faire
        x' ← rand(V_k(x)) ;
        x'' ← Recherche_Locale(x') ;
        Changer_de_voisinage(x, x'', k) ;
        Mise à jour de la variable stop ;
jusqu'à stop = vrai ;
retourner x ;
```

La recherche à voisinage variable a été utilisée pour résoudre de nombreux problèmes d'optimisation dans divers domaines. Parmi les premières applications, nous pouvons citer les travaux de Hansen et Mladenovic [59, 60] pour la résolution du problème du voyageur de commerce. Depuis la proposition et la formalisation de cette métaheuristique, de nombreuses variantes ont émergées dans la littérature. Nous en passons quelques unes en revue dans les sections suivantes.

1.5.2 Recherche à voisinage variable générale

Dans cette variante, une descente à voisinage variable est utilisée à la place de la recherche locale. La méthode nécessite la définition de deux ensembles de structures de voisinage (éventuellement identiques). Le premier ensemble est utilisé dans la phase de perturbation, alors que le second concerne la descente à voisinage variable. La RVV générale est décrite dans l'algorithme 1.7.

Algorithme 1.7 : Recherche à voisinage variable générale.

```
Fonction RVV-G (x, k_max, k'_max)
stop ← faux ;
répéter
    k ← 1 ;
    tant que k ≤ k_max faire
        x' ← rand(V_k(x)) ;
        x'' ← DVV(x',k'_max) ;
        Changer_de_voisinage(x, x'', k) ;
        Mise à jour de la variable stop ;
jusqu'à stop = vrai ;
retourner x ;
```

Parmi les travaux récents dans lesquels une RVV générale a été proposée, nous pouvons citer le travail de Mladenovic et al. [96] pour la résolution du problème du voyageur de commerce avec fenêtre de temps, ou encore le travail présenté dans [97] pour la résolution du problème du voyageur de commerce avec collecte et livraison. Salhi et al. [118] ont aussi proposé récemment une méthode de recherche à voisinage variable pour résoudre efficacement le problème de tournées de véhicule avec flotte hétérogène. D'autres problèmes difficiles qui combinent plusieurs activités de la chaîne logistique ont été résolu avec la RVV générale comme le travail de Jarboui et al [69] pour la résolution d'un problème de localisation et routage.

1.5.3 Recherche à voisinage variable réduite

La version dite réduite de la recherche à voisinage variable consiste à ne considérer que la phase d'exploration des voisinages, et donc d'oublier la

phase d'amélioration. Une nouvelle solution est choisie aléatoirement dans le voisinage courant $\mathcal{V}_k(x)$ de la solution courante x. Cette nouvelle solution est comparée avec la solution x, et une mise à jour (c'est-à-dire un mouvement) est effectué si une amélioration est observée. Les différentes étapes sont présentées dans l'algorithme 1.8.

Algorithme 1.8 : Recherche à voisinage variable réduite.

```
Fonction RVVR(x, k_max)
stop ← faux ;
répéter
    k ← 1 ;
    tant que k ≤ k_max faire
        x' ← rand(V_k(x)) ;
        Changer_de_voisinage(x, x', k) ;
        Mise à jour de la variable stop ;
jusqu'à stop = vrai ;
retourner x ;
```

La RVV réduite a été utilisé pour résoudre divers problèmes, comme dans les travaux de Sevkli et Sevilgen [121] où les auteurs ont proposé une méthode hybride combinant la RVV réduite avec un algorithme d'essaims de particules. Xiao et al. [132] ont aussi proposé un algorithme basé sur cette approche pour résoudre le problème de dimensionnement de lot multi-niveaux.

1.5.4 Recherche à voisinage variable biaisée

Une des difficultés rencontrées par les métaheuristiques, et donc par la RVV, est le fait de pouvoir explorer et évaluer de façon efficace des parties de l'espace de recherche contenant de très bonnes solutions (et éventuellement une solution optimale) très éloignées de la meilleure solution courante. La RVV biaisée (connue sous la terminologie anglaise *Skewed VNS*) vise à résoudre au mieux ce problème. Pour cela, l'idée est de recentrer la recherche autour d'une solution de bonne qualité, mais pas nécessairement meilleure que la solution courante. Cela est rendu possible en intégrant dans l'évaluation des solutions une distance entre la meilleure solution courante et la solution à évaluer. Cette distance est utilisée pour autoriser le mouvement

vers une solution non nécessairement meilleure que la solution courante. Ce processus de diversification nécessite une modification de la fonction d'évaluation des voisins. La version modifiée de cet algorithme est présentée dans l'algorithme 1.9. Dans cet algorithme un paramètre supplémentaire noté α est utilisé pour pondérer l'importance de la distance entre les solutions par rapport à leur coût. La notation $d(x, y)$ fait référence à une distance entre les solutions x et y. Cette distance dépend notamment du problème traité.

Algorithme 1.9 : Algorithme de changement de voisinage pour la RVV biaisée.

Fonction Changer_de_voisinage_RVV_Biaisée(x, x', k, α)
si $f(x') - \alpha d(x, x') < f(x)$ **alors**
  $x \leftarrow x'$;
  $k \leftarrow 1$;
sinon
  $k \leftarrow k + 1$;

L'algorithme 1.10 présente les grandes étapes de la RVV biaisée. Dans cette variante, une phase de mise à jour de la meilleure solution est ajoutée juste après la recherche locale, et avant l'éventuel déplacement, de sorte à ne pas risquer de perdre la meilleure solution.

Algorithme 1.10 : Recherche à voisinage variable biaisée.

Fonction RVV-B (x, k_{max}, α)
$x^* \leftarrow x$;
$stop \leftarrow faux$;
répéter
  $k \leftarrow 1$;
  tant que $k \leq k_{max}$ **faire**
    $x' \leftarrow rand(\mathcal{V}_k(x))$;
    $x'' \leftarrow$ Recherche_Locale(x') ;
    si $f(x'') < f(x^*)$ **alors**
      $x^* \leftarrow x''$;
    Changer_de_voisinage_ RVV_Biaisée(x, x'', k, α) ;
    Mise à jour de la variable *stop* ;
jusqu'à *stop = vrai ;*
retourner x^* ;

Cette variante a notamment été utilisée pour résoudre le problème d'arbre couvrant de poids minimal par Souza et Martins [39]. Brimberg et al. [16] ont appliqué cette approche pour résoudre un problème de k-cardinalité dans un graphe.

1.5.5 Recherche à voisinage variable avec décomposition

Cette variante a été proposée par Hansen et al. en 2001 [61]. L'objectif était d'étendre la RVV en un schéma à deux niveaux, incluant une phase de décomposition du problème. La méthode est très proche de la RVV classique, mais l'idée est en fait de ne pas appliquer la phase de recherche locale sur tout l'espace de recherche (qui peut être très grand pour certains problèmes), mais uniquement pour résoudre un sous-problème correspondant à une plus petite partie de l'espace de recherche. L'algorithme 1.11 résume les étapes de cette variante. Dans cet algorithme, la solution y correspond à la solution obtenue par la phase de décomposition. Cette solution contient k attributs distincts de la solution x à l'itération courante. La recherche locale est alors appliquée sur cette solution, et donc sur un espace de recherche réduit. La solution x'' est la solution sur l'espace de recherche S obtenue en complétant les deux solutions partielles, et la fonction de changement de voisinage classique peut alors être appelée.

Parmi les problèmes qui ont été traités par la RVVD, nous pouvons citer le travail original de Hansen et al. [61] pour la résolution du problème de la *p-médiane*, ou encore le travail de Costa et al. (2002) [34] pour un problème de disposition de câbles. Urovsevic et al. [129] on également étudié un problème de k-cardinalité dans un graphe. Nous pouvons aussi citer le travail de Lejeune [83] dans le cadre de l'optimisation de la planification de la chaîne logistique incluant trois activités principales : la gestion de stock, la production et la distribution. Un des travaux intéressant à citer est celui de Lazić et al. (2010) [81] dans lequel les auteurs ont proposé une heuristique hybride pour la résolution de problèmes en variables 0-1 mixtes en se basant sur le principe de la recherche à voisinage variable avec décomposition.

Algorithme 1.11 : Recherche à voisinage variable avec décomposition.

Fonction RVVD(x, k_{max})
$stop \leftarrow faux$;
répéter
  $k \leftarrow 1$;
  répéter
    $x' \leftarrow rand(\mathcal{V}_k(x))$;
    $y \leftarrow x' \setminus x$;
    $y' \leftarrow$ Recherche_Locale(y) ;
    $x'' = (x' \setminus y) \cup y'$;
    Changer_de_voisinage(x,x'',k) ;
  jusqu'à $k = k_{max}$;
  Mise à jour de la variable *stop* ;
jusqu'à *stop = vrai ;*
retourner x ;

1.5.6 Recherche à voisinage variable primale-duale

La recherche à voisinage variable primale duale a été introduite par Hansen et al. [57]. Cette variante a été proposée de manière à pouvoir fournir une évaluation de la qualité des solutions obtenues par la RVV. En effet, comme pour les autres métaheuristiques, il n'est en général pas facile (voir impossible) de donner avec précision l'écart entre la valeur de la solution finale retournée par l'approche heuristique et la valeur optimale. Pour résoudre ce problème, les auteurs proposent l'approche suivante : dans une première phase un algorithme de RVV est appliqué de manière à obtenir une solution réalisable du problème (primal). Ensuite, cette solution est utilisée pour déduire une solution du problème dual ou d'une relaxation de ce problème. Finalement, et selon le problème traité, les conditions des écarts de complémentarité peuvent éventuellement être utilisées pour améliorer les bornes obtenues. Notons qu'il est aussi possible d'utiliser une autre variante que la RVV pour la phase primale ou la phase duale.

Cette approche a été appliquée initialement par Hansen et al. [57] pour résoudre un problème de localisation. Les auteurs ont développé une RVV ainsi qu'une RVV réduite et une RVV avec décomposition. L'approche a permis d'obtenir des solutions approchées avec une erreur bornée à 4% pour des ins-

tances comptant jusqu'à 15 000 utilisateurs et autant de sites potentiels. Les mêmes auteurs se sont intéressés dans [58] à un problème de regroupement de données qu'ils ont traité comme un problème de *p-médiane*. Différentes approches sont proposées (RVV réduite et RVV avec décomposition). Un procédé de calcul d'une solution du dual directement à partir de celle obtenue pour le primal est également mise en œuvre, permettant de justifier un écart entre la valeur de la solution finale et la valeur optimale est inférieure à 1% pour la grande majorité des instances.

1.5.7 Recherche à formulation variable

La recherche à formulation variable a été introduite par Mladenovic et al. [95], et repose sur l'observation suivante : de nombreux problèmes d'optimisation peuvent être formulés différemment, et il n'est pas toujours aisé de savoir si une formulation domine les autres sur toutes les instances du problème traité. L'idée est donc d'utiliser plusieurs formulations disponibles, et de changer de formulation en fonction de l'état de la recherche. Cette méthode est intéressante en particulier pour les problèmes pour lesquels il est facile de transformer une solution d'une formulation vers une autre, et pour lesquels les techniques de recherche locale se comportent différemment selon les formulations utilisées. Cette approche a également été appliquée récemment par Pardo et al. [103] pour résoudre un problème de découpe. La mise en place de cette méthode demande quelques ajustements de sorte à considérer les différentes formulations du problème. Une façon simple de procéder est de suivre la démarche proposée dans [103], qui consiste à ajouter dans chacune des trois étapes principales de la RVV (la méthode de perturbation, la recherche locale et la fonction de changement de voisinage) l'utilisation de la fonction d'acceptation décrite dans l'algorithme 1.12.

Dans cet algorithme, n_f désigne le nombre de formulations utilisées pour résoudre le problème, alors que la notation $f_i(x)$ est associée à la valeur de la solution x en considérant la formulation f_i. Cette fonction permet donc de rejeter un mouvement si la solution courante n'est pas améliorée pour au moins une formulation.

Algorithme 1.12 : Fonction d'acceptation de mouvement.

```
Fonction Accepter_Mouvement(x, x', n_f)
i ← 1 ;
tant que i ≤ n_f faire
    si f_i(x') < f_i(x) alors
        retourner vrai ;
    sinon si f_i(x') > f_i(x) alors
        retourner faux ;
    i ← i + 1 ;
retourner faux ;
```

1.6 Conclusion

Depuis son apparition dans les années 90, la recherche à voisinage variable a montré son efficacité pour la résolution de nombreux problèmes d'optimisation difficiles. Elle est présente dans la littérature sous plusieurs variantes et versions. Nous avons essayé de présenter dans ce chapitre un résumé des principales variantes de la RVV, en commençant par la version initiale de cette métaheuristique, et en parcourant les variantes les plus récentes et les plus utilisées dans la littérature. Nous avons également cité quelques références pour chacune des variantes, de manière à montrer la diversité des domaines d'application de la RVV.

Devant ce constat, nous avons utilisé la recherche à voisinage variable pour résoudre deux problèmes. Avant de détailler ces approches dans les chapitres 3-5, nous passons en revue dans le prochain chapitre un ensemble de références existantes dans le domaine applicatif qui nous concerne, c'est-à-dire les problèmes d'élaborations de tournées.

Chapitre 2

Revue de la littérature sur les problèmes de tournées

2.1 Introduction

Ce deuxième chapitre de la thèse est centré autour de la problématique de notre sujet, à savoir l'optimisation des problèmes de tournées. Nous commençons par considérer le problème du voyageur de commerce (*Traveling Salesman Problem* selon la terminologie anglaise, TSP), problème à l'origine de nombreuses variantes encore d'actualité. Nous reviendrons en particulier dans le chapitre 5 sur une variante récemment proposée dans le contexte du transport maritime et pour laquelle nous avons proposé une nouvelle approche de résolution. Nous présentons une formulation mathématique du TSP, et donnons quelques références de méthodes de résolution exacte. Nous passons ensuite en revue quelques heuristiques permettant de construire une solution initiale et/ou d'améliorer une solution donnée à l'aide de mouvements plus ou moins simples. Nous terminons par quelques exemples de métaheuristiques développées pour le TSP. La seconde partie de ce chapitre est consacrée au problème d'élaboration de tournées, pour lequel nous suivons une démarche similaire à celle employée pour le TSP. Nous insistons un peu plus sur quelques unes des nombreuses variantes existantes dans la littérature. Finalement, la troisième partie de ce chapitre est dédiée aux problèmes combinant la problématique du routage avec celle de la gestion de stocks, qui est au coeur des chapitre 3 et 4 de ce mémoire.

2.2 Problème du voyageur de commerce

2.2.1 Formulation et méthodes exactes

Le problème du voyageur de commerce, connu en anglais sous la dénomination *"Traveling Salesman Problem"* (TSP), a été introduit au 19$^{\text{ème}}$ siècle par le mathématicien allemand W. R. Hamilton. La version la plus simple de ce problème très connu peut être énoncée comme suit. Soit un ensemble de villes, toutes reliées entre elles, et pour lesquelles nous connaissons un coût associé au trajet entre tout couple de villes (une distance, un temps de trajet, ...). Un voyageur de commerce, initialement situé dans une des villes, doit déterminer une tournée passant une et une seule fois par chaque ville, de sorte à revenir dans sa ville de départ et en minimisant le coût total (c'est-à-dire la distance totale parcourue, la durée totale des trajets, ...).

Pour introduire de façon plus formelle le TSP, nous considérons dans un premier temps le problème symétrique dans lequel le coût pour aller d'une ville i à une autre ville j est le même que celui pour aller de j à i. Le problème peut alors être représenté par un graphe $G = (S, A)$, où $S = \{1, 2, \ldots, n\}$ représente l'ensemble des sommets (des villes) et A est l'ensemble des arêtes (par convention l'arête $(i, j) \in A$ si $i < j$). Nous associons à chaque arête $e = (i, j)$ une valeur $c_e = c_{ij}$ représentant le coût du déplacement de i à j.

Le voyageur fait face à un nombre élevé de choix lorsqu'il doit décider du parcours (du cycle hamiltonien) qu'il va suivre. En fait, il y a $(n-1)!/2$ cycles possibles dans un graphe non orienté comportant n villes. Il est trivial d'observer que ce nombre augmente exponentiellement avec n, et qu'un problème portant uniquement sur 10 villes présente (déjà) 181 440 solutions potentielles. Le TSP est un problème NP-Difficile, et il peut être modélisé sous la forme d'un programme linéaire en nombres entiers, comme proposé par Dantzig et al. [37]. Nous introduisons cette formulation à partir d'une variable binaire x_e associée à chaque arête $e = (i, j)$ de A, égale à 1 si l'arête (i, j) est visitée par le voyageur (c'est-à-dire si elle fait partie du cycle choisi), et à 0 sinon. En notant $\delta(i)$ l'ensemble des arêtes ayant le sommet i comme extrémité, et $A(V)$ l'ensemble des arêtes internes à l'ensemble V, la formulation du problème est alors la suivante :

$$\min \quad \sum_{e \in A} c_e x_e \tag{2.1}$$

$$\text{sous les contraintes : } \sum_{e \in \delta(i)} x_e = 2 \quad \forall i \in S \tag{2.2}$$

$$\sum_{e \in A(V)} x_e \leq |V| - 1 \quad V \subset S, 3 \leq |V| \leq \lfloor n/2 \rfloor \tag{2.3}$$

$$x_e \in \{0, 1\} \quad e \in A \tag{2.4}$$

Il est clair que la fonction objectif (2.1) minimise le coût total associé aux déplacements du voyageur, ce qui revient à minimiser le coût du cycle obtenu sur le graphe G. Les contraintes (2.2) assurent que chaque ville est visitée une et une seule fois (chaque sommet admet exactement deux arêtes adjacentes choisies dans le cycle). L'élimination des sous-tours est garantie par les contraintes (2.3). En effet, s'il existe un sous-tour associé à un sous-ensemble V de sommets, le nombre de sommets visités dans ce sous-ensemble est égal à $|V|$, et la contrainte (2.3) empêche la formation de ce sous-tour. Les contraintes (2.4) imposent l'intégralité des variables.

Cette formulation s'avère difficilement telle quelle en pratique, en raison de la taille du modèle dès que le nombre de villes augmente. Cette augmentation est en particulier liée à la génération des contraintes d'élimination de sous-tours. Plusieurs travaux ont ainsi consisté à résoudre le problème à partir de cette formulation, mais en relaxant les contraintes d'élimination de sous-tours. Cette relaxation résulte en un problème d'affectation ("*Assignement Problem*" (*AP*) selon la terminologie anglaise). En se basant sur cette relaxation (*AP-relaxation*), plusieurs algorithmes par séparation et évaluation (encore appelés *Branch & Bound*) ont été proposés. Nous pouvons par exemple les travaux de Eastman [42] ou plus récemment de Miller et Pekny [90].

Une des approches les plus efficaces pour résoudre les instances du TSP de façon exacte est la méthode *Concorde* proposée par Applegate et al. [4]. La méthode, basée sur un algorithme de type *Branch & Cut & Price*, permet de résoudre la version symétrique du TSP pour des instances comportant jusqu'à 85 900 villes [6]. L'approche a également été utilisée pour résoudre

d'autres problèmes (en particulier des problèmes de tournées).

2.2.2 Heuristiques constructives et d'amélioration

Le TSP étant un problème NP-Difficile, et les instances de grande taille ayant posé de nombreux problèmes aux méthodes exactes les plus efficaces, de nombreux chercheurs se sont naturellement tournés vers la conception et le développement de méthodes heuristiques pour le résoudre. Une des premières heuristiques a été proposée par Rosen et al. [116] et est appelée algorithme du plus proche voisin ou "*Nearest Neighbour Algorithm*". Cette heuristique constructive consiste, à chaque itération, à choisir le sommet non encore visité le plus proche du sommet courant, et à l'ajouter à la tournée. Cette heuristique simple peut également être utilisée pour générer une solution de départ du problème. Il a toutefois été montré [114] qu'elle pouvait conduire à des solutions de très mauvaise qualité. Nous pouvons également citer les heuristiques d'insertion [100, 116] dans lesquelles un sommet est inséré dans une solution partielle en fonction d'un critère choisi (par exemple le sommet le plus proche ou le plus éloigné d'un sommet dans le tour partiel, le sommet qui engendre la plus petite ou la plus grande augmentation du coût du cycle partiel, un sommet choisi aléatoirement, ...). D'autres approches peuvent être listées comme l'heuristique de Christofides [23]. Cet algorithme repose sur la constatation suivante : si nous enlevons une arête à un cycle hamiltonien de poids minimum, alors nous obtenons un arbre couvrant de poids minimum pour le graphe restant. L'approche consiste à générer dans un premier temps un arbre couvrant de poids minimum du graphe, puis à coupler les sommets de degré impair de cet arbre par un couplage de poids minimum. Si le cycle obtenu n'est pas hamiltonien, il suffit alors de ne pas reconsidérer les sommets déjà visités pour en obtenir un. Cette heuristique permet d'obtenir un cycle de longueur au plus 1,5 fois celle d'un cycle optimal.

Ces heuristiques permettent de garantir l'obtention d'une solution réalisable du problème, mais comme nous l'avons indiqué la qualité de cette solution n'est pas nécessairement toujours très bonne. Plusieurs heuristiques d'amélioration, de type recherche locale, ont donc été proposées pour le TSP. Parmi celles-ci nous pouvons citer les heuristiques d'insertion d'un sommet ou

d'une arête. La méthode d'insertion d'un sommet (resp. d'une arête) consiste à supprimer un sommet (resp. une arête) de la solution courante et à le (resp. la) réinsérer à la meilleure position possible (celle engendrant le plus grand gain à partir de la solution courante). La complexité de cette heuristique est clairement en $O(n^2)$ si nous utilisons une stratégie d'exploration du meilleur voisin, puisqu'il faut examiner toutes les positions possibles de toutes les villes (resp. les arêtes). D'autres heuristiques d'amélioration peuvent être trouvées. En particulier, l'heuristique 2-opt [36] consiste à supprimer deux arêtes dans le cycle courant, puis a reconnecter les deux chemins obtenus d'une autre manière de sorte à ce qu'ils reforment un cycle. Ici encore la complexité de l'algorithme est en $O(n^2)$ puisqu'il faut examiner tous les couples d'arêtes. Dans le même cadre, l'heuristique 3-opt [71] consiste à casser le cycle courant en trois morceaux (en supprimant 3 arêtes), puis à recombiner ces morceaux pour obtenir le cycle de coût minimal. La complexité de cette heuristique augmente logiquement par rapport au 2-opt, et elle est en $O(n^3)$ si nous souhaitons évaluer tous les mouvements possibles. Ces deux méthodes peuvent être généralisées avec l'heuristique k-opt. Nous reviendrons sur ces approches dans le chapitre 5 au sein duquel nous avons utilisé des mouvements de ce type pour résoudre une variante du TSP. Nous terminons ce paragraphe par un autre algorithme d'amélioration efficace pour le TSP, l'heuristique de Lin-Kernighan [85]. Cette heuristique peut également être vue comme une généralisation des heuristiques 2-opt et 3-opt, dans le sens où elle consiste à construire une nouvelle solution à partir de modifications simples, même si certaines de ces modifications n'entraînent pas de diminution du coût. En fait, la méthode change le nombre d'arêtes à supprimer et à réinsérer à chaque itération, de façon adaptative, jusqu'à ce qu'un critère d'arrêt soit atteint. Différentes versions peuvent être obtenues en fonction des mouvements de base utilisés (en particulier 2-opt, 3-opt, ou insertion). Helsgaun a notamment proposé dans [62] une variante permettant d'obtenir de bons résultats sur les instances de la littérature.

2.2.3 Métaheuristiques

L'apparition et le développement des métaheuristiques, à partir de la fin des années 70, a poussé la communauté à mettre au point d'autres approches de résolution pour les problèmes d'optimisation difficiles. Ces approches permettent en particulier d'obtenir des solutions de bonne qualité pour des instances trop grandes pour être résolues efficacement par des méthodes exactes. Plusieurs classifications des métaheuristiques peuvent être trouvées dans la littérature. Nous nous contenterons ici de distinguer les approches travaillant sur une seule solution des approches reposant sur l'évolution d'une population de solutions. Pour la première catégorie, et parmi les métaheuristiques qui ont été appliquées au TSP, nous pouvons citer le recuit simulé [73]. Cette approche, inspirée d'un processus utilisé en métallurgie, a été utilisée dans de nombreux travaux pour le TSP (par exemple dans [19, 70, 73]). La recherche tabou, autre métaheuristique très connue [52, 56], a également été appliquée au TSP à différentes reprises (voir par exemple [74, 88]). En ce qui concerne les approches évolutionnaires, utilisant une population de solutions, nous pouvons en particulier citer les algorithmes génétiques qui ont également été utilisés pour résoudre le TSP ([54, 70]]).

Le nombre de références traitant du TSP dans la littérature est impressionnant. Il est également possible de trouver plusieurs articles présentant une vue d'ensemble des travaux sur le TSP (par exemple [76, 112]) ainsi que plusieurs livres traitant ce sujet (par exemple [5, 55]).

Différentes variantes du TSP ont été proposées plus ou moins récemment dans la littérature. Des contraintes utilisées de façon classique dans les problèmes d'élaboration de tournées comme les fenêtres de temps peuvent évidemment être ajoutées dans le TSP. D'autres contraintes peuvent aussi être insérées. Ainsi par exemple, le *Steiner* TSP est une variante dans laquelle un sous-ensemble de villes doit être visité obligatoirement. L'objectif est de trouver un cycle de coût minimum, pas nécessairement hamiltonien, qui visite chacune des villes obligatoires au moins une fois (les villes et les arêtes pouvant être visitées plusieurs fois ici). Cette variante a été introduite dans les années 80 ([33]) et Letchford et al. ont par exemple récemment adapté dans [84] des formulations compactes initialement proposées pour le TSP à

ce problème. Erdogan et al. ont récemment introduit dans [43] une autre variante appelée le TSP attractif. Dans ce problème, l'ensemble des sommets du graphe est partitionné en deux : un sous-ensemble de sommets correspondant aux usines, et un autre aux clients. L'objectif est de construire une tournée qui génère le profit maximum sur un sous-ensemble des sommets associés aux usines. Le profit est déterminé à partir d'une fonction d'*attraction* : chaque sommet visité attire une partie du profit associé aux clients, en fonction de la distance séparant les clients de cette usine, et de l'attractivité de l'usine. D'autres variantes peuvent être trouvées dans la littérature, en particulier lorsque des contraintes utilisées dans les problèmes d'élaboration de tournées sont également prises en compte (fenêtres de temps, livraison et ramassage, *etc*). Nous abordons quelques unes de ces contraintes dans la section suivante.

2.3 Problème d'élaboration de tournées

2.3.1 Formulation et méthodes exactes

Le problème d'élaboration de tournées (*Vehicle Routing Problem* selon la terminologie anglaise, VRP) est une extension du TSP dans laquelle nous considérons un ensemble de véhicules pour desservir un ensemble de clients. Il s'agit donc d'un problème NP-Difficile. Le VRP peut, comme le TSP, être défini sur un graphe $G = (S, A)$ avec $S = \{0, \ldots, n\}$ l'ensemble des sommets, c'est-à-dire l'ensemble des n clients ici. Par convention, le sommet 0 représente le dépôt. L'ensemble A est l'ensemble des arêtes (ou des arcs si nous considérons l'orientation) reliant les couples de sommets du graphe. Ce problème a été introduit par Dantzig et al. [38] à la fin des années 1950. Nous pouvons énoncer une version courante du VRP plus précisément comme suit : à partir d'un dépôt central, où est localisée une flotte homogène de K véhicules ($1 \leq K \leq n$) avec une certaine capacité C, comment planifier les tournées des véhicules de manière à satisfaire la demande (supposée connue et notée d_i pour le sommet $i \in S - \{0\}$) d'un ensemble de clients répartis géographiquement, de sortes à minimiser les coûts de transport (connaissant le coût de transport unitaire entre deux sommets du graphe, noté c_{ij} pour $e = (i, j) \in A, i \neq j$). Cette variante du VRP est le VRP avec capacité, qui

contraint au respect de la capacité des véhicules utilisés. Quelques contraintes classiques du problème peuvent ainsi être listées :

- un client (c'est-à-dire $i \in S - \{0\}$) doit être visité une et une seule fois par un et un seul véhicule.
- tous les véhicules partent et reviennent au dépôt.
- la capacité de tous les véhicules doit être respectée.

Plusieurs formulations mathématiques du VRP ont été proposées dans la littérature. Nous pouvons par exemple considérer une formulation proche de celle décrite pour le TSP (voir section 2.2.1), dans le cas orienté, et si le nombre de véhicules disponibles est connu. Pour cela nous définissons la variable binaire $x_{ij} = 1$ si l'arc (i, j) est dans une solution optimale, 0 sinon. La formulation, notée VRP1, est alors la suivante :

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.5}$$

$$\text{s.c} \sum_{i\in S-\{j\}} x_{ij} = 1 \quad \forall j \in S - \{0\} \tag{2.6}$$

$$\sum_{j\in S-\{i\}} x_{ij} = 1 \quad \forall i \in S - \{0\} \tag{2.7}$$

$$\sum_{i\in S-\{0\}} x_{i0} = K \tag{2.8}$$

$$\sum_{j\in S-\{0\}} x_{0j} = K \tag{2.9}$$

$$\sum_{i\notin V}\sum_{j\in V} x_{ij} \geq \upsilon(V) \quad S - \{0\}, V \neq \oslash \tag{2.10}$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \tag{2.11}$$

Dans cette formulation nous cherchons à minimiser la somme des coûts des arcs empruntés (2.5), en respectant le fait d'entrer et sortir de chacun des sommets associés aux clients (2.6) - (2.7). Les contraintes (2.8) et (2.9) assurent l'utilisation des K véhicules, alors que les contraintes (2.10) permettent de garantir l'élimination de sous-tours pour les différents véhicules, et le respect de la capacité des véhicules. Pour cela, il faut que la valeur $\upsilon(V)$ corresponde au nombre minimum de véhicules pour assurer la visite des

clients de l'ensemble V. Cette valeur peut par exemple être fixée à $\left\lceil \frac{\sum_{i \in V} d_i}{C} \right\rceil$.

Nous pouvons également aborder le VRP en faisant un parallèle avec le TSP, en nous disant que chaque véhicule du VRP peut être assimilé à un voyageur de commerce, et doit effectuer une tournée à partir d'un dépôt. Cela revient à considérer le VRP comme une suite de K TSP à résoudre. Laporte et al. [80] ont exploité cette relation entre le K-TSP, qui est une relaxation du VRP, et le VRP. Le modèle obtenu, dans le cas où le nombre de véhicules à utiliser n'est pas fixé, est un problème d'affectation légèrement modifié puisqu'il comporte les contraintes d'élimination de sous-tours en plus. Ils ont intégré cette formulation dans un algorithme de séparation et évaluation, et ils ont pu résoudre des instances contenant jusqu'à 260 sommets.

D'autres formulations peuvent être trouvées dans la littérature, notamment dans les travaux de Christofides [24]. si le nombre exact de véhicules à utiliser n'est pas connu mais borné par la taille de la flotte, notée K, nous pouvons par exemple définir une variable x_{ij}^k égale à 1 si le véhicule $k \in \{1, \ldots, K\}$ traverse l'arc (i, j) (pour $i, j \in \{1, \ldots, n\}, i \neq j$), et 0 sinon. Le VRP peut alors être formulé comme suit (nous notons VRP2 ce modèle) :

$$\min \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{k=1}^{K} c_{ij} x_{ij}^k \tag{2.12}$$

$$\text{s.c} \sum_{j=0}^{n} \sum_{k=1}^{K} x_{ij}^k = 1 \quad \forall i \in S - \{0\} \tag{2.13}$$

$$\sum_{i=0}^{n} \sum_{k=1}^{K} x_{ij}^k = 1 \quad \forall j \in S - \{0\} \tag{2.14}$$

$$\sum_{i=0}^{n} x_{ip}^k - \sum_{j=0}^{n} x_{pj}^k = 0 \quad \forall p \in S - \{0\}, k \in \{1, \ldots, K\} \tag{2.15}$$

$$\sum_{j=1}^{n} d_j (\sum_{i=0}^{n} x_{ij}^k) \leq C \quad \forall k \in \{1, \ldots, K\} \tag{2.16}$$

$$\sum_{j=1}^{n} x_{i0}^k \leq 1 \quad \forall k \in \{1, \ldots, K\} \tag{2.17}$$

$$\sum_{i=1}^{n} x_{0j}^{k} \leq 1 \quad \forall k \in \{1, \ldots, K\} \tag{2.18}$$

$$x_{ij}^{k} \in \{0, 1\} \quad \forall i, j \in \{1, \ldots, n\}, k \in \{1, \ldots, k\} \tag{2.19}$$

L'objectif (2.12) consiste bien à minimiser la somme des coûts des arcs traversés. Les contraintes (2.13) et (2.14) sont similaires aux contraintes (2.6) et (2.7) du modèle précédent et assurent que chaque client est visité une fois par un seul véhicule. La continuité des routes est assurée par la contrainte (2.15), alors que (2.16) garantie le respect de la capacité des véhicules. Finalement, les contraintes (2.17) et (2.18) assurent le fait qu'un véhicule n'effectue pas plus d'une tournée.

Parmi les autres méthodes basées sur des formulations du VRP, nous pouvons citer le travail de Christofides et al. [25], pour le cas symétrique du problème. La formulation proposée repose sur le partitionnement de l'ensemble des arêtes du graphe en 4 sous-ensembles, définissant notamment un arbre couvrant sur le graphe dans lequel le degré du dépôt est égal à $2K - y$, pour $0 \leq y \leq K$. Leur approche est basée sur l'utilisation d'une relaxation Lagrangienne de ce modèle leur permettant d'obtenir une borne inférieure du problème pour une valeur y fixée.

D'après l'article de Laporte [77], Balinski et Quandt [9] ont été parmi les premiers à modéliser le VRP comme un problème de partitionnement d'ensemble. L'inconvénient majeur de ce type de formulation pour le VRP réside dans la taille exponentielle du modèle, puisqu'il faut associer une variable binaire à chaque route potentielle. Ces formulations se prêtent bien à l'application de techniques de génération de colonnes (voir par exemple [40]).

Finalement, notons que d'autres formulations basées notamment sur l'écoulement d'un flot dans un réseau peuvent aussi être recensées dans la littérature. Nous pouvons citer dans ce contexte le travail de Macedo et al. [87] pour une variante de VRP avec des routes multiples.

2.3.2 Heuristiques et métaheuristiques

Devant la difficulté de résolution d'instances de taille moyenne du VRP, différentes heuristiques et métaheuristiques ont été proposées pour générer

une solution réalisable du problème et/ou pour obtenir des solutions de très bonne qualité en des temps de calcul raisonnables. En pratique, certaines de ces approches sont des extensions de méthodes proposées pour résoudre le TSP. Selon l'article de Laporte et al. [79] du début des années 2000, la majorité des heuristiques constructives et d'amélioration utilisées pour résoudre le VRP ont été proposées entre 1960 et 1990, alors que les métaheuristiques dédiées au VRP se sont développées principalement à partir de 1990.

Parmi les heuristiques constructives, l'heuristique proposée par Clarke et Wright [26] est probablement la plus connue. Celle-ci repose sur une phase de pré-traitement au cours de laquelle la valeur $s_{ij} = c_{0i} + c_{j0} - c_{ij}$ est calculée pour tout $i, j = 1, \ldots, n$ et $i \neq j$. Les valeurs s_{ij} correspondent aux "économies" réalisées si un véhicule visite les deux clients i et j l'un après l'autre et dans cet ordre, plutôt que d'utiliser deux véhicules pour visiter le client i d'un côté et le client j de l'autre. Après avoir trié les valeurs des s_{ij} dans l'ordre décroissant et avoir initialisé la solution avec n tournées ne contenant chacune qu'un client distinct, la méthode consiste à évaluer s'il est possible de fusionner deux tournées, l'une commençant par l'arc (ou l'arête) (1,i) et l'autre se terminant par (j,1). Dans ce cas la nouvelle tournée est obtenue : en supprimant (0,i) et (1,j) des deux tournées existantes, et en ajoutant (j, i) dans la tournée résultante. Le processus est répété tant qu'il est possible de trouver deux routes candidates pour la fusion.

Plusieurs variantes et extensions de cette heuristique ont été proposées dans la littérature. Ainsi par exemple, Mole et Jameson ont généralisé dans [99] la notion d'économie en introduisant deux paramètres permettant de contrôler le choix des arcs pendant la construction de la solution. Holmes et Parker [63] ont introduit un schéma de perturbation de la solution dans le but d'éliminer les routes de mauvaise qualité. Paessens et Heinrich [102] ont évalué de façon détaillée le comportement de cette approche pour le VRP. Notons qu'il est possible de trouver plusieurs articles traitant du VRP et dans lesquels différentes versions et implémentations de cette heuristiques sont abordées (par exemple dans [78]).

D'autres heuristiques constructives ont été proposées pour le VRP. Nous pouvons citer en particulier l'heuristique dite du *petal* qui repose sur la méthode balayage initialement proposée dans [50]. L'algorithme de balayage

s'applique au VRP lorsque les sommets sont représentés par des coordonnées polaires. Après avoir choisi aléatoirement un sommet client, noté i^*, et avoir calculé l'angle correspondant pour tous les sommets par rapport au rayon $(0, i^*)$, la méthode consiste à construire une route à la fois, en ajoutant de façon itérative et tant que cela est possible le sommet non encore visité ayant le plus petit angle dans la route courante. Une phase d'amélioration peut être appliquée à chaque route, en résolvant le TSP correspondant. L'heuristique du pétal consiste à généraliser cette approche en générant plusieurs routes plutôt qu'une seule à la fois, et en résolvant un problème de partitionnement d'ensemble pour choisir la route à retenir. Quelques extensions de cette approche ont permis d'obtenir des résultats compétitifs par rapport aux heuristiques basées sur le critère d'économie [115, 117].

Parmi les métaheuristiques, la recherche tabou a été appliquée sous différentes variantes et avec un certain succès au VRP [48, 123, 126, 128]. Nous pouvons également recenser différents algorithmes génétiques [13, 109]. La recherche à voisinage variable a aussi été appliquée avec succès pour traiter des instances de grande taille du VRP (par exemple dans [75]). Notons aussi le travail de Pisinger et Ropke [107] dans lequel les auteurs proposent un ensemble de méthodes génériques reposant sur l'exploration de voisinages adaptatifs de grande taille. Citons pour terminer l'article de Gendreau et al. [49] qui passe en revue un ensemble de métaheuristiques proposées dans la littérature pour le VRP et ses extensions.

2.3.3 Quelques variantes

Depuis l'apparition du VRP, de nombreuses variantes et extensions ont été proposées, en fonction des caractéristiques du problème. Cette section vise à illustrer rapidement quelques unes de ces variantes, sans chercher à toutes les énumérer. Plusieurs auteurs ont proposé des classifications des problèmes d'élaboration de tournées en fonction de leurs caractéristiques. En nous référant à Housroum (2005) [64], nous pouvons par exemple résumer quelques unes des principales caractéristiques des variantes du VRP dans le tableau 2.1.

Une des variantes les plus connues et les plus étudiées et le VRP dans

Caractéristique	**Options possibles**
Nombre de véhicules	- 1, - K, - Sans restriction
Flotte de véhicules	- Homogène / Hétérogène - Avec compartiments - Avec / sans capacité
Dépôts	- Unique - Multi-dépôts
Demandes des clients	- Déterministe / Stochastique - Avec fenêtres de temps - Produit unique / multiples
Services proposés	- Ramassage / Livraison - Ramassage et livraison
Horizon de planification	- Une seule période - Multi-périodes
Types de tournées	- Fermées / Ouvertes - Distance maximale
Objectif	- Minimisation des coûts de transport - Minimisation du nombre de véhicules - Plusieurs objectifs

TABLEAU 2.1 – Quelques caractéristiques majeures du VRP

lequel des fenêtres de temps sont associées à chaque client (voir par exemple [67, 124, 125]). Plus formellement, à chaque nœud i du graphe ($i \in \{1, \ldots, n\}$) est associé deux valeurs o_i et f_i correspondant à une date d'ouverture et une date de fermeture, respectivement. Deux cas peuvent être différenciés, selon que ces contraintes sont souples ou fortes. Dans le premier cas, il est possible pour le véhicule d'arriver chez le client i avant o_i, ce qui entraînera une phase d'attente avant de pouvoir livrer (ou ramasser) le produit et une pénalité introduite dans la fonction objectif. Le deuxième cas correspond à la situation où aucune arrivée n'est tolérée en dehors de la fenêtre de temps du client. Les contraintes de fenêtre de temps peuvent être assez facilement ajoutées à la seconde modélisation du VRP (VRP2) présentée dans la section 2.3.1. Pour cela, nous introduisons les variables et notations suivantes :

– a_i : variable indiquant la date d'arrivée chez le client i

– w_i : le temps d'attente chez le client i ($w_i = max\{0, o_i - a_i\}$)

Le modèle (VRP2) peut alors être étendu en ajoutant les contraintes suivantes :

$$a_0 = w_0 = 0 \tag{2.20}$$

$$\sum_{k=1}^{K} \sum_{i=0, i\neq j}^{n} x_{ij}^k (a_i + w_i + c_{ij}) \leq a_j \quad \forall j \in \{1, \ldots, n\} \tag{2.21}$$

$$o_i \leq a_i + w_i \leq f_i \quad \forall i \in \{1, \ldots, n\} \tag{2.22}$$

Les contraintes (2.20) assurent que les véhicules sont prêts au début de la période au dépôt, alors que les contraintes (2.21) permettent de déterminer les heures d'arrivée chez chaque client. Les contraintes (2.22) assurent le service de chaque client dans sa fenêtre de temps. Notons qu'il est courant d'ajouter dans la contrainte (2.21) la prise en compte d'un temps de service chez le client, de manière à ne pas considérer la livraison ou le ramassage comme instantané.

Plusieurs problèmes traitent le cas d'une flotte hétérogène. Dans ce cas, nous avons à notre disposition un ensemble de véhicules de T types différents. La capacité d'un véhicule dépend alors de son type. Dans ce type de problème, un coût d'usage d'un véhicule est généralement utilisé pour différencier l'utilisation des différents types de véhicules. En notant C_k la capacité du véhicule k et f_k le coût fixe d'utilisation du véhicule k (C_k et f_k dépendant du type du véhicule k), la fonction objective (2.12) devient (2.23) :

$$\min \sum_{i=0}^{n} \sum_{j=0, j\neq i}^{n} \sum_{k=1}^{K} c_{ij} x_{ij}^k + \sum_{k=1}^{K} \sum_{j=1}^{n} f_k x_{0j}^k \tag{2.23}$$

Et les contraintes de capacité (2.16) deviennent (2.24) :

$$\sum_{j=1}^{n} d_j (\sum_{i=0}^{n} x_{ij}^k) \leq C_k \quad \forall k \in \{1, \ldots, K\} \tag{2.24}$$

Baldacci et al. [8] présentent dans leur article un ensemble de références

de travaux traitant les variantes du VRP dans lesquelles la flotte de véhicules est hétérogène.

Plusieurs objectifs peuvent également être pris en compte simultanément, ce qui génère tout un ensemble supplémentaires de variantes du VRP. Comme souligné dans [72], de nombreuses variantes intégrant plusieurs objectifs sont des extensions de la variante mono-objectif correspondante, avec pour ambition de rendre plus applicable en pratique les méthodes proposées dans le cas mono-objectif, ou de généraliser ces problèmes. Parmi les objectifs utilisés dans les différents problèmes nous pouvons par exemple citer, en plus du coût total de transport et de la distance totale parcourue par la flotte des véhicules, maximiser la satisfaction des clients, minimiser le temps d'attente des véhicules, optimiser le nombre de clients visités par les véhicules, ...

Un problème récent intéressant à citer est le *Swap-Body Vehicle Routing Problem*. Ce problème est définit dans le challenge VeRoLog 2014 [130]. Sa spécificité réside sur la flotte de véhicule utilisée pour la livraison des demandes clients. En effet, un véhicule est composé toujours par un camion et une caisse mobile avec la possibilité d'ajouter une remorque. La présence d'une remorque dans un véhicule donne ce qu'on appelle un train routier. Cependant, certains clients ne peuvent pas être visités avec les trains routiers. En outre, il existe des lieux d'échange pour garer ou récupérer une remorque. L'objectif est de minimiser l'ensemble des coûts associés à ce problème.

Nous terminons ce court passage en revue en mentionnant le problème du VRP périodique. Dans cette variante, l'objectif est de construire les tournées des véhicules sur un horizon comprenant plusieurs périodes. En général, chaque client doit être visité un certain nombre de fois (connu) sur l'horizon de planification. Les contraintes classiques du VRP (respect de la capacité des véhicules et livraison/ramassage de la demande des clients) sont également prises en compte. Pour résoudre ce problème nous pouvons, en considérant un client donné, chercher à l'affecter à un planning de visites, c'est-à-dire un ensemble de jours durant lesquels un véhicule le visitera. Une solution du problème revient alors à considérer trois aspects simultanément [46] :

- Choisir un planning de visites pour chaque client ;
- Affecter un ensemble de clients à chaque véhicule pour chaque période de l'horizon ;

– Définir les tournées des véhicules sur chaque période de l'horizon.

Nous pouvons facilement voir que c'est le premier point qui différencie ce problème du VRP. Différentes approches heuristiques ont été proposées pour traiter ce problème et sa variante intégrant les fenêtres de temps. Ces méthodes reposent en particulier sur la recherche tabou [30, 31, 32] et sur la recherche à voisinage variable [105, 106].

Nous avons abordé ce problème périodique car il se rapproche d'un problème qui nous intéresse plus précisément dans cette thèse, et qui combine le routage des véhicules avec la gestion des stocks chez les clients. Ce problème, par nature multi-périodique, peut aussi être vu comme une extension du VRP. Comme nous allons le voir dans la section suivante, il implique lui aussi plusieurs décisions à prendre simultanément.

2.4 Problème de tournées avec gestion de stock

2.4.1 Définition et caractéristique

Les problèmes de tournées avec gestion de stock sont des problèmes issus de la même famille de problèmes de tournées "classiques". De façon générale ils sont connus sous le nom anglais *Inventory Routing Problem* (IRP). Ces problèmes combinent deux activités de la chaine logistique : la gestion du stock chez le client et le routage des véhicules. Dans les problèmes de tournées de véhicules, la distribution de produit se fait après une commande client. Le fournisseur doit les assurer par achat ou par fabrication ; il doit assembler les commandes clients, charger les véhicules, définir un plan de routage et établir la livraison.

Nous ne parlons plus de commande client dans l'IRP. La gestion de stock est alors confiée au fournisseur. Il détermine la quantité à livrer pour chacun de ses clients selon un horizon décisionnel. C'est en 1983 que Bell et al. [12] ont donné la définition du problème de tournées avec gestion de stock. La distribution se fait à partir d'un seul dépôt (dépôt central) avec une flotte infinie et homogène de véhicules et elle ne concerne qu'un seul produit. Initialement, les véhicules sont localisés dans le dépôt, et les tournées des véhicules sont organisées sur plusieurs périodes de temps. La consommation de chaque

produit par chaque client se fait suivant un taux constant (déterministe) au cours du temps. La capacité du dépôt est supérieure à celle du site de stockage des clients. L'objectif consiste alors à minimiser les coûts de distribution vers l'ensemble des clients sans avoir de rupture de stock. Pour cela, il faut apporter des réponses à trois questions fondamentales :

- Quand visiter chaque client ?
- Quelle quantité livrer à chaque client ?
- Quelle tournée associer à quel véhicule ?

La première question est liée à la rupture de stock des clients. Connaissant le taux de consommation des clients, nous pouvons déterminer la date limite de visite d'un client pour éviter la rupture de stock. Cependant, nous ne sommes pas obligés d'attendre la date limite pour réapprovisionner le client. Cela offre plus de flexibilité sur la quantité de produit à livrer, mais en contre partie complique le problème. La deuxième question est liée à la quantité à livrer ; la définition de cette quantité n'est pas triviale. En effet, une grande quantité livrée pour un client peut réduire le nombre de visites, mais ce n'est pas toujours vrai si le taux consommé est très grand. Une fois l'ensemble des dates de livraison et les quantités à livrer déterminées sur l'horizon de planification, nous pouvons résoudre un problème visant à optimiser les tournées de véhicules.

L'IRP se présente dans la littérature sous plusieurs variantes. Ces variantes dépendent principalement de quatre caractéristiques fondamentales qui sont : la demande, la flotte de véhicules, l'horizon décisionnel et la livraison. Le tableau 2.2 illustre l'ensemble de ces principales caractéristiques de l'IRP.

- **La demande :** elle correspond au taux de consommation hebdomadaire pour un problème de distribution. La majorité des problèmes IRP étudiés suppose que cette demande est connue ou déterministe. Néanmoins, il existe des travaux qui traitent le cas stochastique qui semble plus proche de la réalité où nous ne pouvons pas prévoir d'une façon exacte la date de rupture de stock. Dans les problèmes de ramassage, la demande correspond à la quantité de produit à ramasser chez chaque client ou fournisseur. Selon le type de problème, le nombre de produits diffère. Dans le cas général, un seul produit est associé. Mais plusieurs

Caractéristique	**Options possibles**
Demande	- Déterministe / Stochastique / Dynamique - Produit unique / Plusieurs produits
Flotte de transport	- Limité / Illimité - Homogène / Hétérogène - Propre au fournisseur / Sous-traitance
Horizon	- Fini / Infini
Livraison	- Directe / Multiple / Continue

TABLEAU 2.2 – Différentes caractéristiques de l'IRP

travaux que nous allons citer par la suite considèrent le transport de plusieurs produits, comme par exemple pour la livraison de produits pétroliers vers les stations services [2].

– **La flotte de transport :** dans un problème routier la flotte considérée est un ensemble de véhicules qui assurent le transport du produit. Les problèmes d'IRP concernent également le domaine maritime. La flotte de véhicules est considérée dans la majorité des problèmes d'IRP comme un élément non contraint, avec une flotte infinie disponible pour la livraison ou la collecte du produit. Dans certaines études, une contrainte sur le nombre de véhicules est prise en considération. Ce nombre peut être égal à un seul véhicule ou à un nombre fini. Finalement, la flotte peut être homogène ou hétérogène.
– **L'horizon :** c'est la période décisionnelle sur laquelle le problème est résolu. Selon la nature du problème, nous pouvons distinguer deux grandes classes d'horizon : fini ou infini. Les décisions prisent peuvent être des décisions à court terme ou des décision à long terme. Plusieurs approches ont été développées afin de réduire les décisions à long terme en décisions à court terme.
– **La livraison :** elle concerne la façon d'approvisionner ou de ramasser le produit de l'ensemble des clients (i.e. fournisseurs). Dans certains cas, il peut être nécessaire d'associer un seul client à une tournée. Nous parlons alors de livraison directe (ou visite simple). Dans le cas général les tournées regroupent un ensemble de clients (livraisons multiples). De façon générale, une tournée est établie sur une seule période de

temps sur l'horizon décisionnel, mais cela est possible seulement dans le transport routier. Dans le transport maritime, le transport du produit peut durer plusieurs périodes, ce qui correspond à la terminologie de livraison continue.

A partir de ces quatre caractéristiques fondamentales, nous pouvons affirmer qu'il existe plusieurs variantes de l'IRP et que chacune prend en considération un ou plusieurs de ces critères. Nous pouvons ajouter à ces caractéristiques trois autres qui concernent la structure du problème, la politique de la gestion de stock associée et la décision au niveau gestion de stock en se basant sur le travaux de Coelho et al. [27]. Le tableau 2.3 résume ces trois caractéristiques.

Structure	- Un à un (*one to one*) - Un vers plusieurs (*one to many*) - Plusieurs vers plusieurs (*many to many*)
Politique de gestion de stock	- Niveau maximal - *Order up to level*
Décision sur le stock	- *Lost sales* - *Back-order* - *Non-negative*

TABLEAU 2.3 – Caractéristiques de l'IRP [27]

- La **structure** fait référence au nombre de fournisseurs et de clients, cela change selon le problème. Dans le cas où un seul fournisseur approvisionne un seul client, le problème est de structure un à un (*one to one*). Un à plusieurs (*one to many*) est le cas le plus étudié qui concerne un fournisseur et un ensemble de clients. La dernière structure est plusieurs à plusieurs (*many to many*), qui est sans doute la moins étudiée dans la littérature et concerne les problèmes avec plusieurs fournisseurs et plusieurs clients à la fois.
- Il existe **deux politiques de gestion de stock**, la première est le niveau maximal qui consiste à déterminer la quantité de produit à maintenir afin d'éviter le coût de stockage non nécessaire. Le niveau de remplissage est flexible mais il est borné par la capacité de stockage interne chez le client. La deuxième politique, dite "*Order Up-to-level*

(OU)", consiste à amener le stock client à son niveau maximal chaque fois qu'il est visité.

– Les **décisions prises sur le stock** impactent le modèle de gestion des stocks. Dans le cas où la rupture de stock est autorisée, nous parlons de deux décisions : la première est connue sous la nomination anglaise "*Back-Order*" ou retour de la demande. Elle se produit si les demandes seront livrées dans une période ultérieure lorsque le client peut attendre. Si les demandes ne sont pas livrées, alors nous parlons de "*lost sales*" ou ventes perdus. Dans ces deux cas, il peut exister une pénalité pour la rupture de stock. Dans le cas général, la majorité des problèmes considèrent la rupture de stock comme une contrainte fondamentale de l'IRP : le "*Back order*" et les "*lost sales*" ne sont pas permis.

2.4.2 Formulation Mathématique

Différentes formulations mathématiques peuvent être recensées, en fonction de la variante étudiée. Pour illustrer cela, nous nous basons sur la formulation proposée par Archetti et al. [7]. Le problème considéré concerne la livraison d'un produit à partir d'un seul fournisseur vers un ensemble de clients géographiquement dispersés en utilisant une flotte de véhicules homogène et avec capacité. Il est défini sur un graphe $G = (S, A)$ où $S = \{0, \ldots, n\}$ est l'ensemble des sommets et $A = \{(i, j) : i, j \in S, i \neq j\}$ est l'ensemble des arcs liant les nœuds de G. On note par $S' = S - \{0\}$ l'ensemble des clients sur G. Sur chaque arc, un coût c_{ij} de transport est associé. Soit h_i le coût de stockage associé à chaque unité de produit pour chaque période de temps $t \in \tau = 1, \ldots, T$. Nous considérons que le coût de stockage est associé aux fournisseurs et aux clients et chaque client admet une capacité de stockage interne notée C_i. Le niveau de stock initial pour le fournisseur est noté I_0^0 et pour chaque client I_i^0. Le niveau de stockage à la fin de chaque période t est noté par I_{it}. Une hypothèse à prendre en considération concerne la disponibilité du produit chez le fournisseur qui est suffisant pour assurer la demande sur l'horizon décisionnel. La quantité de produit disponible chez le fournisseur est notée r^t pour la période t. Chaque client i a une demande d_{it} à la période t qui est connue (problème déterministe). La distribution du produit

se fait par une flotte de véhicules $K = \{1, \ldots, k\}$ homogène avec capacité Q localisée initialement chez le fournisseur correspondant au sommet 0 de G. On note par :

- q_{it} la quantité livrée au client i durant la période t.
- x_{ijt} est égale à 1 si l'arc (i, j) est traversé à la période t, 0 sinon.
- y_{it} est égale à 1 si seulement le client i est visité à la période t.

L'objectif du problème est de minimiser les coûts de distribution et de stockage sans avoir de rupture de stock et en prenant en compte les contraintes suivantes :

- Le niveau de stock chez le client ne dépasse jamais sa capacité interne.
- Le niveau de stock doit être toujours positif.
- Chaque véhicule assure une seule tournée pour chaque période commençant par le fournisseur 0 et finissant au même nœud.
- La capacité des véhicules doit être respectée.

Le problème peut alors être formulé comme suit :

$$\min \quad z = \sum_{i \in S} \sum_{t \in \tau} h_i I_{it} + \sum_{i \in S} \sum_{j \in S, i<j} \sum_{t \in \tau} c_{ij} x_{ijt} \tag{2.25}$$

$$\text{s.c} \quad I_{0t} = I_{0,t-1} + r^t - \sum_{i \in S'} q_{it}, \quad \forall t \in \tau \tag{2.26}$$

$$I_{0t} \geq 0, \quad \forall t \in \tau \tag{2.27}$$

$$I_{it} = I_{0,t_1} + q_{it} - d_{it}, \quad \forall i \in S', \quad \forall t \in \tau \tag{2.28}$$

$$I_{it} \geq 0, \quad \forall i \in S', \quad \forall t \in \tau \tag{2.29}$$

$$q_{it} \geq C_i y_i^t - I_{i,t-1}, \quad \forall i \in S', \quad \forall t \in \tau \tag{2.30}$$

$$q_{it} \geq C_i - I_{i,t-1}, \quad \forall i \in S', \quad \forall t \in \tau \tag{2.31}$$

$$\sum_{i \in S} q_{it} \leq Q y_0^t, \quad \forall t \in \tau \tag{2.32}$$

$$\sum_{j \in S', i<j} x_{ijt} + \sum_{j \in S', i>j} x_{jit} = 2y_i^t, \quad \forall i \in S', \quad \forall t \in \tau \tag{2.33}$$

$$\sum_{i \in \vartheta} \sum_{j \in \vartheta, i<j} x_{ijt} \leq \sum_{i \in \vartheta} y_i^t - y_m^t, \quad \vartheta \subseteq S, \quad t \in \tau, \quad m \in \vartheta \tag{2.34}$$

$$q_{it} \geq 0, \quad \forall i \in S, \quad \forall t \in \tau \tag{2.35}$$

$$x_{ijt} \in \{0, 1\}, \quad \forall i, j \in S, i \neq j, \quad \forall t \in \tau \tag{2.36}$$

$$x_{i0t} \in \{0,1,2\}, \forall i \in S, \forall t \in \tau \tag{2.37}$$

$$y_i^t \in 0,1, \quad \forall i \in S, \quad \forall t \in \tau \tag{2.38}$$

La fonction objective (2.25) minimise le coût de stockage chez le client et chez le fournisseur ainsi que les coûts de transport associés. La contrainte (2.26) définit le niveau de stock chez le fournisseur à la fin de la période t. La contrainte (2.27) garantit la disponibilité de produit chez le fournisseur. De même pour la contrainte (2.28) et (2.29), mais au niveau de l'ensemble des clients. Les contraintes (2.30) à (2.31) définissent les quantités livrées. Elles assurent l'utilisation de la politique "*Order-Up-to level*" dans la gestion de stock. La contrainte (2.32) assure que la capacité des véhicules est respectée. La contraintes (2.33) assure la faisabilité de la tournée, alors que la contrainte (2.34) élimine les sous-tours. Les contraintes (2.35)-(2.38) sont les contraintes de non négativité et d'intégralité des variables.

2.4.3 Algorithmes de résolution exacte pour l'IRP

Le modèle présenté précédemment concerne la version standard de l'IRP avec la politique de gestion de stock "*Order-Up-to level*". Pour résoudre le problème avec la politique de niveau maximal, il suffit d'éliminer les contraintes (2.30) et (2.32). Archetti et al. [7] ont utilisé une méthode de *Branch & Cut* qui consiste à relaxer la contrainte (2.36) et de l'ajouter comme une coupe dans l'arbre de recherche afin d'éliminer les solutions non réalisables. Ils ont proposé des inégalités valides et ils étaient capables de résoudre des instances contenant 50 clients sur un horizon composé de trois périodes ainsi que des instances contenant 30 clients sur un horizon de six périodes sur une durée d'exécution de deux heures.

Campbell et al. [17] ont proposé une méthode de résolution basée sur la programmation linéaire en deux phases. Dans la première phase, l'approche détermine la date de visite ainsi que la quantité à livrer pour chaque client. La deuxième phase consiste à déterminer les routes. Evidemment, l'optimalité de la deuxième phase dépend du choix établit dans la première phase.

Le modèle présenté ci-dessus prend en charge un seul véhicule pour la livraison. Récemment, Coelho et Laporte [29] ainsi que Adulyasak et al. [1]

ont proposé des extensions de ce modèle sous les deux politiques de gestion de stock (*Order-Up to level* et niveau maximal) en tenant compte de la disponibilité de plusieurs véhicules. Ils ont utilisé pour la résolution une méthode *Branch & Cut*. Ils ont pu résoudre des problèmes contenant plus que 45 clients, trois périodes et trois véhicules disponibles pour la livraison.

2.4.4 Méthodes heuristiques pour la résolution de l'IRP

Nous avons essayé de présenter dans la section précédente une formulation mathématique générale du problème tirée de [7]. Le problème de tournées avec gestion de stock est clairement un problème NP-difficile, puisqu'il inclut le problème d'élaboration de tournées lui même NP-Difficile. Plusieurs travaux sur l'IRP ont donc porté sur l'utilisation de méthodes heuristiques ou le développement de métaheuristiques pour résoudre l'IRP.

Nous allons faire une classification partielle des méthodes évoquées dans la littérature et des algorithmes utilisés pour les diverses variantes de l'IRP en se basant sur le travail de Michel [89]. Nous prenons comme critère de classification le nombre de produits à livrer ou à ramasser. Nous considérons dans un premier temps les problèmes qui ne traitent qu'un produit unique pour le transport, et dans un deuxième temps les problèmes qui concernent plusieurs produits.

2.4.5 Problèmes à produit unique

En examinant la littérature, nous trouvons que la majorité des problèmes traités prennent comme hypothèse un seul produit à transporter et une demande ou un taux d'usage connu (cas déterministe). Dans cette partie, nous allons présenter les algorithmes utilisés pour la résolution des problèmes de type IRP qui considèrent un seul produit quelque soit le type de demande (déterministe ou stochastique) et quel que soit le type d'horizon (fini ou infini).

Les premiers auteurs qui ont étudié l'IRP étaient Bell et al. [12]. Leur étude du problème est basée sur une prévision du taux de consommation du produit par le client (cas déterministe). Afin de résoudre ce problème, ils ont

considéré que chaque tournée comporte quatre clients au maximum à approvisionner. Un module d'optimisation définit les routes, les dates de visites pour chaque route et affecte les véhicules aux routes. Ils avaient également défini une méthode basée sur la relaxation Lagrangienne pour résoudre le problème.

Ce travail a été suivi par celui de Federgruen et Zipkin [44]. Ces derniers se sont intéressés à adapter des heuristiques pour le problème de tournées de véhicules proposées par Fisher et Jaikumar [45] au cas de l'IRP. Ils traitent un cas stochastique avec une flotte hétérogène pour la livraison du produit. La méthode heuristique utilisée consiste à choisir et à assigner les clients à un véhicule dans un premier temps et à définir les quantités à livrer pour chacun. Un problème de voyageur de commerce est ensuite résolu pour chaque véhicule.

Golden et al. [53] traitent un cas stochastique de l'IRP dans lequel une notion d'urgence est intégrée pour les clients. Il s'agit de définir un seuil d'alerte pour chacun des clients. Ce seuil permet de définir une fenêtre de temps pour chacun des clients. Finalement, les clients sont choisis selon leur degré d'urgence et sont associés à des routes en utilisant l'algorithme de Clarke and Wright [26].

Dror et Ball [41] traitent un cas stochastique d'IRP avec une flotte de véhicules. Leur idée était de transformer le problème, initialement sur une période à long terme, en un problème à court terme. L'horizon de planification est ainsi réduit d'une année à quelques jours. Le jour optimal de livraison de chaque client est défini selon une probabilité. Le principe est d'estimer le jour où le client peut être en rupture de stock. Si ce jour est dans la période de planification, il sera visité. Si la livraison est faite avant la date de rupture, une pénalité est appliquée. Une fois que les clients à visiter sont déterminés, les tournées des véhicules sont obtenues en résolvant un problème de VRP.

Chien et al. [21] considèrent un problème avec une disponibilité limitée du produit au niveau du dépôt (problème similaire à celui de Federgruen and Zipkin [44]). Ils ont donné une formulation sous forme de programme linéaire en nombre entier mixte pour ce problème qui permet de définir le client à visiter et les quantités à livrer, ainsi que d'établir le routage d'une flotte de véhicules hétérogènes.

Anily et al. [3] traitent un problème composé d'un seul dépôt et d'un ensemble de magasins qui ont une demande connue. La livraison se fait grâce à une flotte de véhicules infinie avec capacité. Afin de résoudre le problème, ils ont utilisé une méthode qui permet d'associer les magasins à des clusters en se basant sur la fréquence de livraison. La quantité livrée pour chaque cluster est le minimum entre la capacité du véhicule et la quantité définie par le modèle économique.

Bramel et Simchi-Levi (1995) [15] ont proposé une méthode heuristique appelée "*Location Based Heuristic*" ou heuristique basée sur le problème de localisation, et l'ont appliquée sur un problème de tournées de véhicules ainsi qu'un problème d'IRP. La résolution du problème d'IRP se fait en trois phases. Dans une première phase, ils calculent la période de visite optimale en se basant sur la formule de Wilson (*Economic Order Quantity Model*) et la capacité des véhicules. Ils définissent par la suite les centres qui peuvent être formés et finalement ils définissent les clusters clients par la résolution d'un problème de localisation.

Webb et Larson (1995) [131] étudient un IRP dans lequel ils cherchent à minimiser le coût d'utilisation des véhicules. En se basant sur la politique "*order up to level*", ils cherchent à identifier un ensemble de clusters clients et à trouver un ensemble de routes pour chaque cluster qui puisse être répété de façon périodique.

Witucki et al. (1997) [86] étudient un problème de distribution de gaz industriel. Pour cela, ils proposent une méthode de résolution en deux phases. Durant la première phase, un plan de visite des clients est établi pour une semaine. Ils utilisent une méthode basée sur la génération de colonnes pour optimiser les tournées en prenant l'hypothèse qu'un client est desservi une seule fois par semaine. Dans la deuxième phase, en connaissant les quantités réelles livrées chaque jour, ils ré-optimisent le planning par une autre méthode de génération de colonnes.

Bard et al. [10] présentent une étude sur le problème de l'IRP avec des installations intermédiaires dans lesquel les véhicules récupèrent les produits. Cette variante considère un seul dépôt, un ensemble de clients, un ensemble d'installations intermédiaires et une flotte de véhicules homogène. Les tournées commencent à partir du dépôt, et un véhicule assure la livraison vers un

premier ensemble de clients. Une fois que le véhicule est vide, la prochaine destination est nécessairement une installation intermédiaire afin de recharger le produit, et une nouvelle tournée commence. La résolution de ce problème peut se résumer en deux grandes phases. La première consiste à identifier les clients qui seront livrés et les associer à des clusters. La deuxième phase consiste à résoudre un problème de tournées de véhicules avec installations intermédiaire et l'améliorer avec des méthodes de recherches locales basées sur les permutations de clients entre les routes et les périodes.

Christiansen et Nygreen (1998) [22] étudient un problème d'IRP dans le transport maritime. Le problème étudié comporte un seul produit à livrer et c'est un problème de chargement et déchargement entre différents ports, avec également des fenêtres de temps. Les problèmes de transports maritimes diffèrent des problèmes de transport routiers sur plusieurs plans : il n'y a pas un dépôt central ; la fin de la tournée n'est pas forcément le point de départ, *etc.* L'objectif étudié consiste à minimiser le coût total des tournées pour l'ensemble des navires. Pour la résolution, ils ont proposé un modèle mathématique qui définit les routes pour chaque navire ainsi que la gestion de stock pour chaque port. Un algorithme de type *Branch and Price* est utilisé pour la résolution.

Bertazzi et al. [14] étudient un problème avec plusieurs dépôts. Ils analysent différentes fonctions objectives pour ce problème qui consistent à minimiser le coût de transport et/ou le coût de stockage chez le fournisseur et chez les clients. Le problème est résolu avec une méthode heuristique en deux phases. La première phase vise uniquement à créer une solution réalisable. La deuxième phase est une phase d'amélioration qui consiste à effectuer des mouvements en changeant la date de livraison pour certains clients. A notre connaissance, cette étude était la première à considérer le coût de stockage chez le fournisseur et chez le client.

Campbell et Savelsbergh [18] étudient un problème de distribution de gaz liquide. Ils utilisent une idée similaire à celle présentée dans [17]. La nouveauté était de définir des clusters à servir par un véhicule. La première phase consistait alors à regrouper les clients dans des clusters. La deuxième phase déterminait l'ordre de visite des clients dans un cluster ainsi que la quantité à livrer pour chacun.

Savelsbergh et al. [119] considèrent un problème d'IRP avec *mouvements continus* d'une compagnie de production de gaz industriel. Dans ce problème il existe deux types de clients : les clients proches du dépôt pour lesquels la livraison est possible sur une période ; et les autres clients qui sont très éloignés et pour lesquels la livraison peut demander plusieurs périodes. Cela donne un mouvement dit *continu* aux véhicules. Afin de résoudre ce problème, les auteurs ont proposé trois méthodes heuristiques qui définissent les routes et les quantités à livrer. Suite aux résultats obtenus par ces approches, une formulation mathématique en nombres entiers mixtes a aussi été proposée afin d'optimiser les quantités à livrer. Dans un deuxième article [120], les auteurs ont proposé une formulation basée sur un modèle de flot pour le même problème afin d'appuyer la métaheuristique *GRASP* et améliorer les résultats obtenus.

Finalement, Raa et Aghezzaf [111] étudient un IRP cyclique (ou avec des tours multiples). Dans ce problème un ensemble de clients avec une demande déterministe et un horizon fini sont définis. L'originalité réside dans le fait qu'un véhicule peut assurer plusieurs tournées par période. Le temps entre deux tournées consécutives est appelé un cycle. Pour la résolution du problème, ils ont développé une méthode de génération de colonnes qui permet d'assigner les clients à des véhicules.

2.4.6 Problèmes à plusieurs produits

Dans cette section nous passons en revue quelques autres travaux traitant de problèmes à plusieurs produits.

Allah et al. [2] étudient le problème d'approvisionnement de stations services au Canada. Le problème est traité comme étant un IRP. Pour chaque période (une journée), ils cherchent à déterminer la quantité à livrer vers l'ensemble des stations et à élaborer les tournées de façon à ce que le coût soit minimisé. Dans leur article, ils ont présenté une formulation mathématique en nombres entiers mixtes et ont traité deux cas : la livraison simple et la livraison multiples.

Persson et al. [104] ont proposé une méthode de génération de colonnes pour une variante de l'IRP dans le domaine maritime. La particularité du

problème est la prise en compte d'une variation du taux de production dans les différents ports, celle-ci étant quand même connu à l'avance.

Huang et Lin [66] ont proposé un algorithme de colonie de fourmis pour résoudre un problème d'IRP avec une incertitude sur la demande des clients. Ce travail a été suivi par celui de Popovic et al. [108] dans lequel les auteurs ont considéré le problème de distribution de carburant vers les stations services, la flotte de véhicules étant homogène et les véhicules compartimentés. Ils ont pris l'hypothèse que chaque compartiment fait l'objet d'une quantité à livrer pour une station donnée. Ils ont proposé une formulation sous la forme d'un programme linéaire en nombres entiers mixtes. Ils ont ensuite défini différentes structures de voisinages et une méthode de recherche à voisinage variable.

Coelho et al. [28] ont proposé un algorithme de *branch & cut* pour résoudre un problème multi-produits de l'IRP. Les auteurs ont également généré des instances pour ce problème et ils ont résolu à l'optimal des problèmes comportant 10 clients, 7 périodes, 5 produits et 5 véhicules.

Finalement, Song et Furman [122] ont étudié un cas industriel de l'IRP dans le contexte maritime. Ils ont développé une méthode heuristique pour la résolution du problème en tenant compte divers contraintes réelles liées au domaine maritime comme les fenêtres de temps. L'heuristique proposée a permis de résoudre des instances comportant jusqu'à 60 périodes.

2.5 Conclusion

Le problème du voyageur de commerce est à l'origine de nombreux problèmes qui apparaissent dans l'activité de transport. Dans ce chapitre, nous avons fait un courte revue de la littérature sur ce problème, en mettant l'accent sur sa formulation et quelques une des principales méthodes efficaces développées pour le résoudre.

Une extension de ce problème est le problème d'élaboration de tournées de véhicules. Nous avons suivi une démarche similaire, en présentant quelques formulations et variantes de ce problèmes ainsi que quelques unes des méthodes les plus connues et les plus efficaces pour le résoudre. Finalement, nous avons considéré les problèmes de tournées avec gestion de stock. Ici encore ce

problème se décline sous différentes variantes dans la littérature. Nous avons listé les caractéristiques principales de ce problème, et nous avons présenté une formulation très générique de cette catégorie de problème. Nous avons enfin présenté quelques variantes étudiées dans la littérature.

La suite de ce mémoire est centrée sur nos contributions. Nous abordons dans le chapitre suivant la résolution d'un problème de tournées avec gestion de stock.

Chapitre 3

Recherche à voisinage variable pour un problème de tournées avec gestion de stock multi-périodes

3.1 Introduction

Comme nous l'avons vu dans le chapitre 2 de cette thèse, le problème de routage avec gestion de stock (IRP) se présente sous plusieurs variantes dans la littérature. Dans ce chapitre, nous nous intéressons à un cas particulier parmi ces variantes de l'IRP, et nous proposons une Recherche à Voisinage Variable (RVV) pour le résoudre. La RVV, plus connue sous la dénomination anglaise "*Variable Neighborhood Search"* (VNS), est une métaheuristique proposée par Mladenovic et Hansen à la fin des années 90 [94]. Cette approche, sur laquelle le chapitre 1 de cette thèse est centré, peut être grossièrement décomposée en deux grandes phases : une phase de descente qui permet de trouver un optimum local du problème traité, et une deuxième phase dite de perturbation qui permet d'en échapper (cette phase est appelée en anglais "*Shaking*"). Sans être redondant par rapport au chapitre 1, nous pouvons rappeler que la recherche à voisinage variable a été appliquée à de nombreux problèmes d'optimisation combinatoire et différentes variantes de l'approche ont été proposées depuis son apparition. Ce succès s'explique en partie par sa simplicité de mise en œuvre (en général), et surtout par son efficacité. Rappelons que l'idée de base de la méthode est l'utilisation de plusieurs structures de voisinages au sein de l'algorithme, de manière à pouvoir explorer efficacement les différentes parties de l'espace de recherche associé au problème.

Ce procédé permet ainsi de contourner le problème majeur des méthodes de recherche locale, à savoir une convergence prématurée vers un optimum local.

Comme nous le verrons plus précisément dans la suite de ce chapitre, nous proposons une RVV en deux phases pour la résolution d'une variante de problème de routage avec gestion de stock multi produits. Ce problème consiste à satisfaire les demandes en différents produits d'une usine d'assemblage sur un horizon donné. Ces produits doivent être récupérés chez un ensemble de fournisseurs géographiquement dispersés, en utilisant une flotte homogène de véhicules. L'objectif du problème consiste à minimiser trois coûts étroitement liés : le coût de stockage des produits sur tout l'horizon ; le coût de transport induit par les routes empruntées par les véhicules et le coût d'utilisation des véhicules. Dans une première phase, la RVV est utilisée pour trouver une solution initiale réalisable pour ce problème en ne minimisant que les coûts de transport. Dans une deuxième phase, nous utilisons la RVV pour améliorer la solution initiale et en intégrant les coûts de transports, de stockage et d'utilisation des véhicules. Afin de minimiser le coût de stockage lors de l'exploration des voisinages, nous proposons un programme linéaire qui détermine les quantités de produits à collecter chez l'ensemble des fournisseurs lors d'un mouvement. Nous proposons aussi une heuristique efficace qui permet de résoudre ce programme linéaire très rapidement.

La suite de ce chapitre est organisée comme suit. Nous commençons par présenter de façon plus précise le problème de routage avec gestion de stock considéré dans cette étude dans la section 3.2. Nous présentons également dans cette section une formulation mathématique du problème. Nous abordons ensuite la méthode de recherche à voisinage variable à deux phases que nous proposons pour résoudre ce problème dans la section 3.3. Cette section est principalement composée de la description des différentes structures de voisinages utilisées, et d'une description des deux phases de notre approche. Nous présentons dans la section 3.4 les résultats expérimentaux obtenus sur des instances disponibles dans la littérature, de manière à valider notre proposition. Nous terminons ce chapitre par une conclusion dans la section 3.5.

3.2 Présentation du problème

Comme nous l'avons vu dans le chapitre 2 de la thèse, plusieurs critères peuvent être utilisés pour classifier les problèmes de routage avec gestion de stock (livraison/collecte, demande déterministe/stochastique, flotte homogène/hétérogène de véhicules, etc.). Nous passons donc en revue ces différents critères, en décrivant précisément la variante de l'IRP qui nous concerne dans ce travail.

Le problème traité est un problème de collecte sur un horizon fini composé de plusieurs périodes. Une usine d'assemblage doit collecter un ensemble de produits fabriqués et mis à disposition par un ensemble de fournisseurs géographiquement dispersés. Une particularité du problème est le fait que chaque fournisseur fabrique exactement un produit unique. La demande de l'usine est connue et est donc déterministe. Pour collecter les produits, une flotte homogène et supposée infinie de véhicules avec une capacité limitée est disponible. Tous les véhicules sont initialement localisés à un dépôt. Un fournisseur peut être visité plusieurs fois (i.e., par plusieurs véhicules) durant une même période (ce qui correspond à la terminologie anglaise "*split pick-up*"). A chaque période, les véhicules utilisés quittent le dépôt, visitent les fournisseurs qui leur sont affectés, et effectuent la livraison à l'usine d'assemblage. Ils doivent nécessairement retourner au dépôt à la fin de la période. La problématique est centrée sur le routage et la gestion de stock au niveau de l'usine d'assemblage. Il faut donc s'assurer que l'usine ait toujours les quantités de produits nécessaires pour pouvoir continuer son activité. Par contre, nous ne nous intéressons pas dans ce problème à la gestion des stocks chez les fournisseurs. Cela implique qu'il est supposé que les produits sont toujours disponibles en quantité suffisantes chez les fournisseurs.

De façon plus formelle, nous considérons un horizon décisionnel T composé de τ périodes successives. Pour chaque période de temps $t \in T = \{1, ..., \tau\}$, l'usine d'assemblage a une demande d_{it} associée à chaque produit $i \in N = \{1, ..., n\}$ et nous supposons que la quantité nécessaire du produit i est disponible chez le fournisseur i. Pour assurer le ramassage de ces produits, une flotte (infinie) M de véhicules est disponible. Comme nous le verrons dans la suite, le nombre de véhicules utilisés en pratique sera implici-

tement minimisé lors de la résolution du problème (l'utilisation d'un véhicule ayant un impact économique). Nous désignons par m une borne supérieure sur le nombre de véhicules nécessaires à la collecte des produits. Chaque véhicule $v \in M = \{1, ..., m\}$ a une capacité C et est localisé initialement au dépôt. Si nous représentons la situation à l'aide d'un graphe, nous pouvons utiliser les conventions suivantes : le dépôt correspond au sommet d'indice 0, les véhicules effectuent des visites chez les n fournisseurs associés aux indices $1, 2, \ldots, n$, et ils assurent la livraison vers l'usine d'assemblage qui est représentée par un sommet d'indice $n + 1$. La figure 3.1 illustre le problème traité.

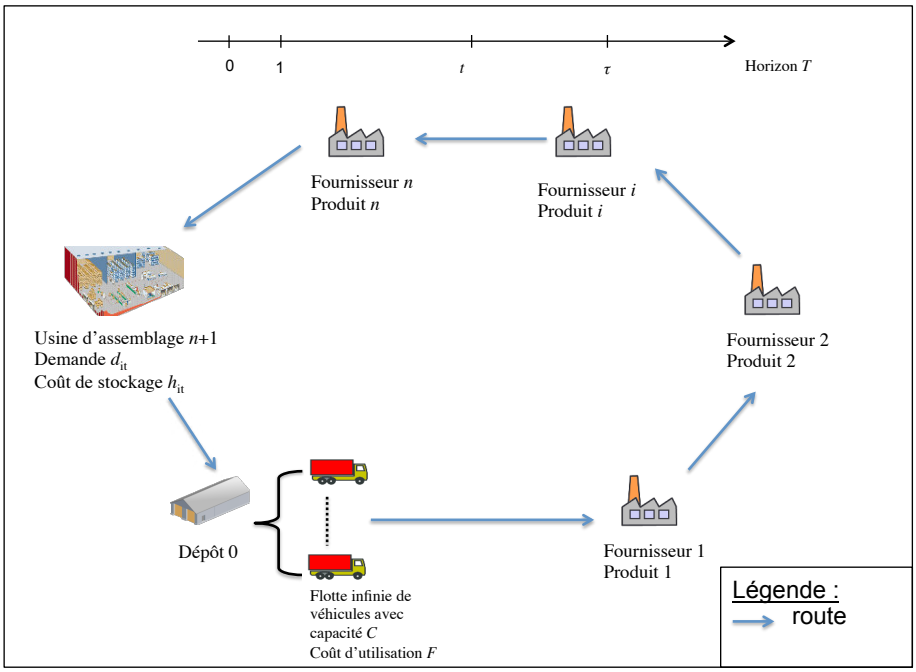

FIGURE 3.1 – Illustration du problème.

Même si le problème peut être considéré comme un problème multi-objectifs, la fonction à optimiser est ici ramenée à un seul objectif, en pondérant trois coûts :

– Les coûts de stockage. A chaque produit i est associé un coût de stockage unitaire (par période) à l'usine d'assemblage. Ce coût est noté h_i pour le produit i, il est fixe et déterministe, mais il varie selon le

produit à stocker. Il n'y a pas de coût de stockage chez les fournisseurs.

- Les coûts de déplacements des véhicules. Dans ce problème, nous connaissons le coût c_{ij} (fixe et déterministe) pour aller du sommet i au sommet j, et cela entre les différents nœuds du réseau (dépôt, fournisseurs, usine d'assemblage) avec $i, j \in \{0, \ldots, n+1\}$.
- Les coûts d'utilisation des véhicules. A chaque fois qu'un véhicule est utilisé à une période donnée, un coût fixe F est à prendre en compte dans le calcul des coûts. De plus, un coût unitaire, noté V, est associé au transport des produits.

Le problème peut être modélisé sous la forme d'un programme linéaire en nombres entiers mixtes. Nous reprenons ci-dessous une formulation proposée dans [98]. Nous introduisons les variables suivantes :

- a_{it} : la quantité de produit ramassé chez le fournisseur i durant la période t.
- I_{it} : la quantité de produit i à stocker à l'usine à la fin de la période t. I_{i0} est le stock initial (donnée du problème).
- q_{ijt} : la quantité de produit i transportée sur l'arc (i, j) durant la période t.
- x_{ijt} : le nombre de fois où l'arc (i, j) est visité par l'ensemble des véhicules durant la période t.

Le problème se formule alors comme suit :

$$\min \; z = \sum_{i \in N} h_i \Big(\sum_{t \in T} I_{it}\Big) + V\Big(\sum_{j \in N} \sum_{\substack{i \in N \cup \{0\} \\ i \neq j}} c_{ij} \Big(\sum_{t \in T} x_{ijt}\Big) + \sum_{i \in N} c_{i,n+1} \sum_{t \in T} x_{i,n+1,t}\Big) + (F + V c_{n+1,0}) \sum_{i \in N} \sum_{t \in T} x_{0it} \tag{3.1}$$

$$\text{s.c.} \quad I_{it} = I_{i,t-1} + a_{it} - d_{it}, \quad \forall i \in N, \forall t \in T, \tag{3.2}$$

$$\sum_{\substack{i \in N \cup \{0\} \\ i \neq j}} q_{ijt} + a_{jt} = \sum_{\substack{i \in N \cup \{n+1\} \\ i \neq j}} q_{ijt}, \quad \forall j \in N, \forall t \in T \tag{3.3}$$

$$\sum_{i \in N} q_{i,n+1,t} = \sum_{i \in N} a_{it}, \quad \forall t \in T, \tag{3.4}$$

$$\sum_{\substack{i\in N\cup\{0\}\\ i\neq j}} x_{ijt} = \sum_{\substack{i\in N\cup\{n+1\}\\ i\neq j}} x_{jit}, \quad \forall j \in N, t \in T \tag{3.5}$$

$$\sum_{j\in N} x_{0jt} = \sum_{j\in N} x_{j,n+1,t}, \quad \forall t \in T \tag{3.6}$$

$$q_{ijt} \leq C x_{ijt}, \quad \forall i \in N, \forall j \in N \cup \{n+1\}, i \neq j, \forall t \in T \tag{3.7}$$

$$I_{it} \geq 0, \quad \forall i \in N, \forall t \in T \tag{3.8}$$

$$a_{it} \geq 0, \quad \forall i \in N, \quad \forall t \in T \tag{3.9}$$

$$x_{i,j,t} \geq 0, \quad \text{entier}, \quad \forall i, j \in N \cup \{0, n+1\}, \forall t \in T \tag{3.10}$$

$$x_{i,0,t} = x_{n+1,i,t} = 0, \quad \forall i \in N \cup \{0, n+1\}, \forall t \in T \tag{3.11}$$

$$q_{ijt} \geq 0, \quad \forall i \in N, j \in N \cup \{n+1\}, \forall t \in T \tag{3.12}$$

$$q_{0it} = 0, \quad \forall i \in N, \forall t \in T. \tag{3.13}$$

La fonction objective (3.1) cherche à minimiser à la fois le coût de stockage au niveau de l'usine d'assemblage et les coûts de transport. Cette formulation montre bien la présence des trois coûts évoqués précédemment. Plus précisément, le premier terme permet de déterminer la somme des coûts de stockage. Le deuxième terme donne la somme des coûts des trajets des véhicules, en considérant d'abord les coûts de transport entre le dépôt et les fournisseurs, puis entre les fournisseurs, et enfin entre les fournisseurs et l'usine d'assemblage. Finalement, le dernier terme calcule la somme des coûts fixes d'utilisation des véhicules (plus les coûts de déplacement des véhicules de l'usine vers le dépôt). La contrainte (3.2) assure la cohérence du stock pour chaque produit au niveau de l'usine d'assemblage sur l'horizon de planification. La contrainte (3.3) assure qu'à une période donnée la quantité de produits qui arrive à un sommet plus celle qui en sort est égale à la somme des quantités qui passent par ce sommet (cette contrainte permet aussi d'éliminer les sous-tours). La contrainte (3.4) assure que la quantité totale transportée par l'ensemble des véhicules vers l'usine est égale à la somme des quantités collectées chez les fournisseurs, et cela à chaque période. Les contraintes (3.5) et (3.6) assurent que le nombre de véhicules quittant le dépôt et visitant les fournisseurs est égal au nombre de véhicules arrivant à l'usine, tandis que la contrainte (3.7) garantit que la capacité des véhicules soit respectée.

La contrainte (3.8) oblige que la demande de l'usine soit satisfaite à chaque période (il n'est pas possible d'autoriser une rupture de stock des produits nécessaires à l'assemblage). Les contraintes (3.9), (3.10) et (3.12) assurent la positivité des variables, alors que (3.11) évite qu'il y ait un chemin direct partant d'un fournisseur vers le dépôt, du dépôt vers l'usine, ou de l'usine vers l'un des fournisseurs. Finalement, (3.13) oblige à ne pas transporter de produit entre le dépôt et les autres sommets du graphe.

Nous abordons dans la section suivante l'approche de résolution que nous proposons pour ce problème.

3.3 Recherche à voisinage variable en deux phases

Comme nous l'avons vu dans le chapitre 1 de la thèse, la recherche à voisinage variable est une des métaheuristiques les plus connues et les plus utilisées de nos jours. L'utilisation de plusieurs structures de voisinages au sein d'une ou plusieurs méthodes de recherche locale permet de mettre en place une exploration efficace de l'espace de recherche du problème traité. Le changement systématique de voisinage permet en particulier d'éviter une convergence prématurée vers un optimum local.

Nous proposons dans cette section deux approches de recherche à voisinage variable pour résoudre le problème d'IRP multi-produits décrit dans la section précédente [91, 92]. Ces deux approches sont décomposées en deux phases, et elles partagent la même première phase. Celle-ci consiste à trouver une première solution réalisable minimisant uniquement le coût de transport. La deuxième phase vise à améliorer cette solution initiale en prenant en compte tous les coûts définis dans la fonction objective du problème. La première approche proposée est centrée autour du développement d'un algorithme de Descente à Voisinage Variable (DVV) (*"Variable Neighborhood Descent"* selon la terminologie anglaise), alors que la seconde approche repose sur un algorithme de type RVV. Nous commençons par aborder les différentes structures de voisinages utilisées pour traiter le problème.

3.3.1 Structures de voisinages

La première chose à faire avant de concevoir les structures de voisinages est de définir la représentation d'une solution qui est mise en œuvre. Dans le cas de notre problème, une solution doit regrouper toutes les informations nécessaires à l'élaboration des tournées effectuées par les véhicules avec, en particulier : la liste des fournisseurs visités (et l'ordre dans lequel ils sont visités), la quantité de produit collectée, la période correspondante.

Notons S une solution du problème. Celle-ci est composée d'une séquence de τ éléments, où chaque élément S_t représente l'ensemble des routes associées à la période $t \in T$. Chaque route (implicitement associée à un véhicule donné) contient une séquence de fournisseurs à visiter (dans l'ordre de visite). Pour pouvoir construire le planning des tournées, chaque fournisseur apparaissant dans une route est caractérisé par deux éléments : son indice (le numéro du sommet associé au fournisseur dans le graphe) et la quantité de produit à ramasser (pour la période considérée). Nous obtenons ainsi une décomposition d'une solution S en plusieurs "*niveaux*", comme suit :

- $S = \{S_1, ..., S_\tau\}$: une solution pour le problème sur l'horizon T de planification. Cette solution est composée de τ éléments S_t.
- $S_t = \{S_{t_1}, .., S_{t_{m(t)}}\}$: les routes construites pour la période $t \in T$, où $m(t)$ est le nombre de véhicules utilisés durant la période t. Nous notons $M(t)$ l'ensemble des véhicules utilisés durant la période t. La notation S_{t_1} désigne donc la route du premier véhicule utilisé à la période t.
- $S_{t_v} =< S_{t_{v,1}}, ..., S_{t_{v,n(t,v)}} >$: la séquence des fournisseurs visités par le véhicule v durant la période t, où $n(t, v)$ est le nombre de fournisseurs visités par le véhicule v durant cette période t. Nous notons également par $N(t, v) = \{1, ..., n(t, v)\}$ l'ensemble des fournisseurs visités par le véhicule v durant la période t. S_{t_v} est donc la description de la tournée du véhicule v à la période t, $S_{t_{v,1}}$ faisant référence au premier fournisseur visité par le véhicule v dans sa tournée de la période t.
- $S_{t_{v,j}} = (S^{ind}_{t_{v,j}}, S^{q}_{t_{v,j}})$: un couple formé de l'indice du $j^{ème}$ fournisseur visité par le véhicule v (noté $S^{ind}_{t_{v,j}}$) et de la quantité à collecter chez ce fournisseur (notée $S^{q}_{t_{v,j}}$).

La représentation de la solution que nous proposons met en évidence la forte

relation entre IRP et problème de tournées de véhicules. La manipulation d'une solution va donc reposer sur des mouvements de base au sein d'une tournée (ou route). Nous introduisons trois mouvements élémentaires au sein d'une route : la suppression d'un fournisseur donné (noté *Drop*), l'ajout d'un fournisseur à une certaine position (noté *Add*) et l'échange d'un fournisseur par un autre (noté *Change*). Les trois mouvements peuvent être décrits comme suit :

$Drop(S_{t_v}, k) = S'_{t_v}$: suppression de l'élément à la position k de la route S_{t_v}.
S'_{t_v} vérifie : $\forall p \in N^-(t,v), S'_{t_{v,p}} = S_{t_{v,p}}$ si $p < k$,
ou $S'_{t_{v,p}} = S_{t_{v,p+1}}$, si $p \geq k$,
avec $N^-(t,v) = N(t,v) - \{n(t,v)\}$.

$Add(S_{t_v}, k, j, S^q_{t_{v,j}}) = S'_{t_v}$: ajout du fournisseur j avec la quantité de produit q à ramasser à la position k dans la route S_{t_v}.
$S'_{t_{v,j}}$ vérifie : $\forall j \in N^+(t,v), S'_{t_{v,j}} = S_{t_{v,j}}$ si $j < k$,
ou $S'_{t_{v,j}} = S_{t_{v,j-1}}$ si $j > k$,
ou $S'_{t_{v,k}} = (i, S^q_{t_{v,i}})$ si $j = k$,
avec $N^+(t,v) = N(t,v) + \{n(t,v)\}$.

$Change(S_{t_v}, k, j, q) = S'_{t_v}$: remplace le fournisseur à la position k (avec sa quantité) par le fournisseur j avec la quantité q à ramasser dans la route S_{t_v}.
$S'_{t_{v,j}}$ vérifie : $\forall p \in N(t,v) - \{j\}, S'_{t_{v,p}} = S_{t_{v,p}}$
et $S'_{t_{v,k}} = (j,q)$.

Nous proposons l'utilisation de sept structures de voisinage qui reposent sur ces mouvements simples. Ces différentes structures seront utilisées dans nos algorithmes de RVV.

Voisinage de type déplacement d'un fournisseur

Ce premier voisinage consiste à déplacer un fournisseur présent dans une route, soit depuis sa position actuelle vers une autre position au sein de la même route, soit dans une autre route de la même période. La figure 3.2 illustre ce voisinage : le fournisseur 6 est ici déplacé de la route (2) vers la route (1).

$\mathcal{V}_1(S) = \{S' \mid \forall t \in T, \forall v, v' \in M(t), \forall j \in N(t,v), \forall k \in N(t,v')$, soit $(j^*, q^*) = S_{t_{v,j}}, S'_{t_v} = Drop(S_{t_v}, j), S'_{t_{v'}} = Add(S_{t_{v'}}, k, j^*, q^*)\}$.

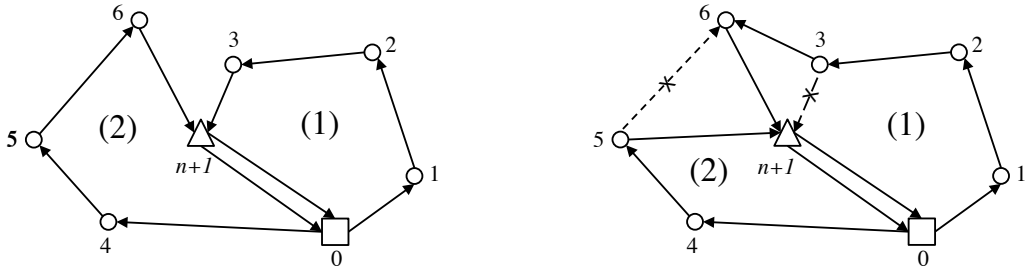

FIGURE 3.2 – Illustration du voisinage $\mathcal{V}_1(S)$

Voisinage de type échange de fournisseurs

Un voisin d'une solution S en considérant ce second voisinage est obtenu en échangeant un fournisseur d'une route donnée par un autre fournisseur appartenant à une autre route. Les deux routes doivent être planifiées pour la même période. Dans l'exemple de la figure 3.3, le fournisseur 3 de la route (1) est échangé avec le fournisseur 6 de la route (2).

$$\mathcal{V}_2(S) = \{S' \mid \forall t \in T, \forall v, v' \in M(t), \forall k \in N(t,v), \forall k' \in N(t,v'), S'_{t_v} = Change(S_{t_v}, k, S^{ind}_{t_{v',k'}}, S^{q}_{t_{v',k'}}), S'_{t_{v'}} = Change(S_{t_{v'}}, k', S^{ind}_{t_{v,k}}, S^{q}_{t_{v,k}}),$$

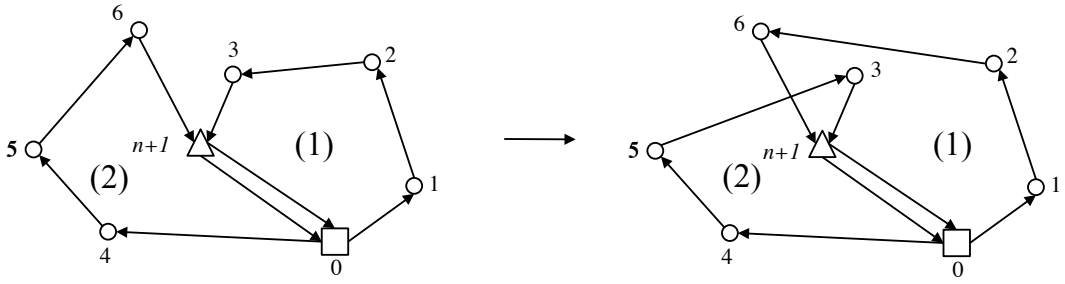

FIGURE 3.3 – Illustration du voisinage $\mathcal{V}_2(S)$

Voisinage de type suppression d'un fournisseur

Une solution S' voisine de la solution courante S appartenant à ce voisinage est obtenue en supprimant un fournisseur d'une des routes (d'une des périodes) de la solution S'. la figure 3.4 illustre un cas ou le fournisseur 2 est supprimé de la route (1).

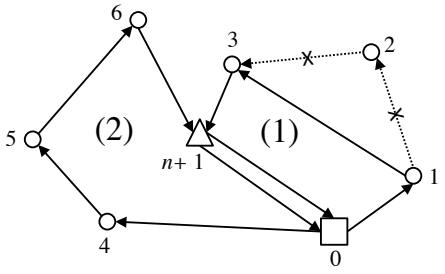

FIGURE 3.4 – Illustration du voisinage $\mathcal{V}_3(S)$

$$\mathcal{V}_3(S) = \{S' \mid \forall t \in T, \forall v \in M(t), \forall k \in N(t,v), S'_{t_v} = Drop(S_{t_v}, k)\}.$$

Voisinage de type insertion d'un fournisseur

Lorsque nous explorons ce voisinage, nous considérons toutes les solutions voisines de la solution S obtenues en insérant (si cela est possible) n'importe quel fournisseur dans une des routes de S. La figure 3.5 illustre un cas où le fournisseur $i \in N$ est inséré dans la route (1) entre les fournisseurs 2 et 3.

$$\mathcal{V}_4(S) = \{S' \mid \forall t \in T, \forall v \in M(t), \forall k \in N(t,v), \forall k' \in N,$$
$$S'_{t_v} = Add(S_{t_v}, k, k', q')\}.$$

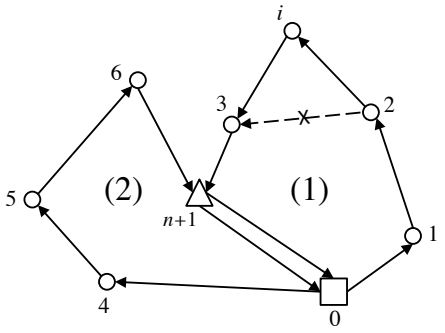

FIGURE 3.5 – Illustration du voisinage $\mathcal{V}_4(S)$

Voisinage de type changement de période d'un fournisseur

Ce voisinage comprend l'ensemble des solutions obtenues en supprimant un fournisseur d'une des routes d'une des périodes de la solution courante, et en le réinsérant dans n'importe quelle autre route de n'importe quelle autre période.

Dans l'illustration de la figure 3.6, le fournisseur 5 est supprimé d'une route de la période t et est inséré dans une nouvelle route de la période t'.

$\mathcal{V}_5(S) = \{S' \mid \forall t, t' \in T, \forall v \in M(t), \forall v' \in M(t'), \forall k \in N(t,v),$
$\forall k' \in N(t',v')$, soit $(k^*, q^*) = (S^{ind}_{t_{v,k}}, S^{q}_{t_{v,k}}), S'_{t_v} = Drop(S_{t_v}, k),$
$S'_{t'_{v'}} = Add(S_{t'_{v'}}, k', k^*, q^*)\}$.

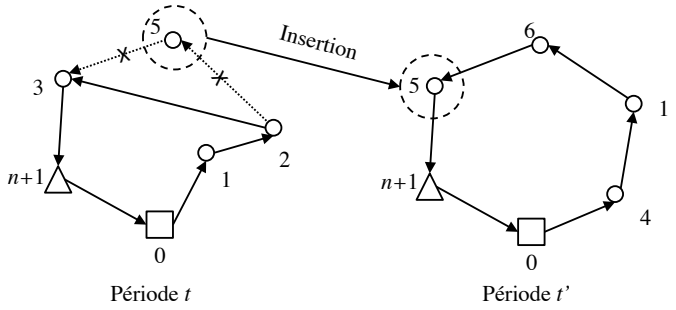

FIGURE 3.6 – Illustration du voisinage $\mathcal{V}_5(S)$

Voisinage de type remplacement d'un fournisseur

Dans ce voisinage les solutions sont obtenues en remplaçant un fournisseur d'une des routes de la solution courante par un autre fournisseur de N. Dans la figure 3.7, le fournisseur 2 est supprimé de la route (1) et il est remplacé par le fournisseur $i \in N$.

$\mathcal{V}_6(S) = \{S' \mid \forall t \in T, \forall v \in M(t), \forall i \in N(t,v), \forall i' \in N,$
$S'_{t_v} = Add(Drop(S_{t_v}, i), i, i', q')\}$.

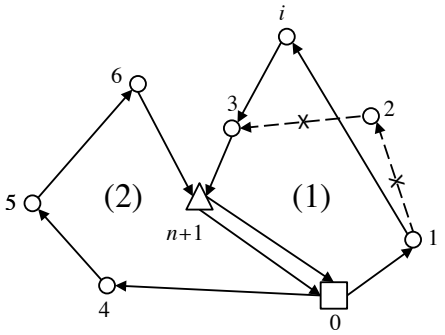

FIGURE 3.7 – Illustration du voisinage $\mathcal{V}_6(S)$

Voisinage de type échange périodique de fournisseurs

Ce voisinage comprend toutes les solutions obtenues à partir de la solution courante S en permutant deux fournisseurs appartenant à deux routes de deux périodes différentes.

La Figure 3.8 illustre un cas où le fournisseur 5 appartenant à une route de la période t est permuté avec le fournisseur 7 d'une route de la période t'.

$\mathcal{V}_7(S) = \{S' \mid \forall t, t' \in T, \forall v \in M(t), \forall v' \in M(t'), \forall k \in N(t,v), \forall k' \in N(t',v'), S'_{t_v} = Change(S_{t_v}, k, S_{t'_{v',k'}}), S'_{t'_{v'}} = Change(S_{t'_{v'}}, k', S_{t_{v,k}})\}.$

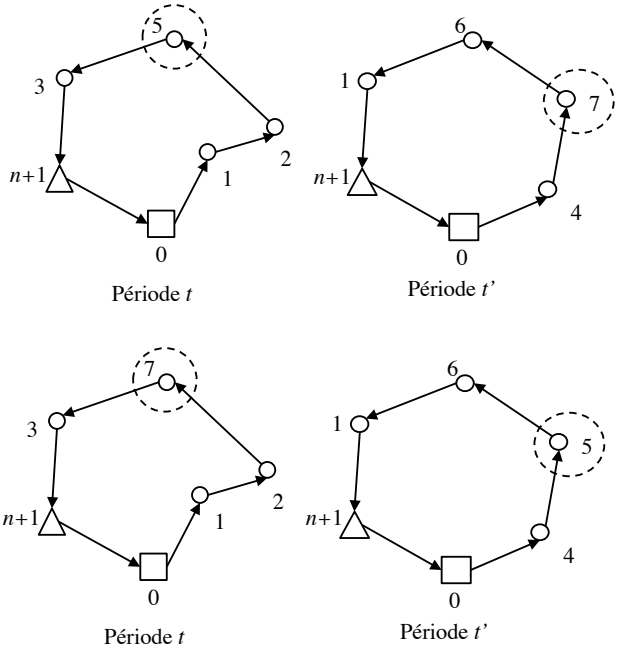

FIGURE 3.8 – Illustration du voisinage $\mathcal{V}_7(S)$

Nous pouvons maintenant décrire plus précisément la première phase de notre approche.

3.3.2 Première phase : Construction d'une solution initiale

Dans la première phase de l'approche, nous ne considérons pas les coûts de stockage, et nous ne prenons en compte que les coûts liés au transport des produits. Cela signifie que notre problème se ramène à un problème de tournées de véhicules multi-périodes. Dans ce cas, nous cherchons à construire pour chaque période un ensemble de tournées permettant de ramasser exactement la quantité de produit nécessaire chez chaque fournisseur (forcément disponible par hypothèse) pour pouvoir assurer la production au niveau de l'usine d'assemblage. Cela permet d'avoir un coût de stockage nul au niveau de l'usine d'assemblage. Pour cela, nous commençons par générer une solution à l'aide d'un algorithme glouton, en introduisant l'aléatoire. Cet algorithme construit un ensemble de tournées satisfaisant exactement la demande de l'usine et visitant tous les fournisseurs, cela pour chaque période. Cette solution sert alors de point de départ pour notre algorithme de recherche à voisinage variable.

Comme nous l'avons dit dans le chapitre 1, la RVV comporte une phase de perturbation qui permet en général d'éviter à la recherche de rester piégée dans les optima locaux. Plusieurs méthodes de perturbation existent dans la littérature. Dans notre cas, nous définissons la perturbation comme l'utilisation successive (h fois) d'un même voisinage $\mathcal{V}_k(S)$. Plus formellement, en notant par $\mathcal{V}_k^{[h]}(S)$ le voisinage obtenu après l'application successive de h mouvements induits par le voisinage $\mathcal{V}_k$ à partir de la solution S, nous obtenons la définition récursive de $\mathcal{V}_k^{[h]}(S)$:

$$\begin{cases} \mathcal{V}_k^{[1]}(S) = \mathcal{V}_k(S) \\ \mathcal{V}_k^{[h+1]}(S) = \cup_{S' \in \mathcal{V}_k(S)} \mathcal{V}_k^{[h]}(S') \end{cases} \tag{3.14}$$

La RVV mise en place dans la première phase repose sur l'utilisation des voisinages $\mathcal{V}_1(S)$ et $\mathcal{V}_2(S)$. Ces deux voisinages ne considèrent en effet que des modifications *mineures* de la solution courante S, en n'impactant qu'une seule période. Il s'agit donc de deux voisinages de taille raisonnable qui peuvent être explorés assez rapidement. De plus, l'objectif de la première

phase étant de trouver une première solution réalisable d'assez bonne qualité rapidement, nous utilisons une stratégie d'exploration du type *premier voisin améliorant*. Nous notons dans la suite $\mathrm{RL}_1(S, \mathcal{V}_k)$ pour faire référence à la recherche locale utilisant la solution S comme solution initiale et $\mathcal{V}_k$ comme structure de voisinage.

Cette approche $\mathrm{RL}_1(S, \mathcal{V}_k)$ est utilisée au sein d'une seconde méthode de recherche locale, de type descente, dans laquelle nous appliquons successivement $\mathrm{RL}_1(S, \mathcal{V}_1)$ et $\mathrm{RL}_1(S, \mathcal{V}_2)$, tant qu'il est possible d'améliorer la solution courante. Cette fonction, appelée RL_2, prend en entrée une solution initiale S, ainsi que deux voisinages $\mathcal{V}'$ et $\mathcal{V}''$ (en pratique $\mathcal{V}_1$ et $\mathcal{V}_2$), et retourne la solution améliorée. Elle est décrite dans l'algorithme 3.1. Dans cet algorithme et dans la suite, nous notons $f(S)$ pour faire référence à la valeur de la solution S.

Algorithme 3.1 : Recherche Locale RL_2

Fonction $\mathrm{RL}_2(S, \mathcal{V}', \mathcal{V}'')$
répéter
  $S' \leftarrow \mathrm{RL}_1(S, \mathcal{V}')$;
  $S'' \leftarrow \mathrm{RL}_1(S', \mathcal{V}'')$;
  $\Delta = f(S'') - f(S)$;
  si $\Delta < 0$ **alors**
    $S \leftarrow S''$;
jusqu'à $\Delta \geq 0$;
retourner S ;

Nous présentons dans l'algorithme 3.2 le déroulement de la recherche à voisinage variable qui est utilisée pour obtenir une première solution du problème. Cet algorithme, noté RVV_1, prend en entrée une solution initiale S, k_{max} qui définit le nombre maximum de perturbations à appliquer sur la solution courante et $iter_{max}$ le nombre d'itérations de l'algorithme. $\mathcal{V}'$ et $\mathcal{V}''$ font référence aux deux voisinages utilisés dans la recherche locale RL_2.

La solution retournée par l'algorithme RVV_1 servira de point de départ à la seconde phase.

Algorithme 3.2 : Recherche à Voisinage Variable : première phase

Fonction $\mathrm{RVV}_1(S,\, k_{max},\, iter_{max},\, \mathcal{V}',\, \mathcal{V}'')$
pour $iter = 1$ *à* $iter_{max}$ **faire**
  $k \leftarrow 1$;
  tant que $k \leq k_{max}$ **faire**
    soit $S' \in \mathcal{V}^{[k]}(S)$;
    $S'' \leftarrow \mathrm{RL}_2(S', \mathcal{V}', \mathcal{V}'')$;
    si $f(S'') < f(S)$ **alors**
      $S \leftarrow S''$;
      $k \leftarrow 1$;
    sinon
      $k \leftarrow k + 1$;
retourner S ;

3.3.3 Deuxième phase : phase d'amélioration

Pour améliorer la solution générée précédemment, nous proposons une recherche à voisinage variable qui utilise plusieurs structures de voisinages entraînant des modifications des quantités de produits à ramasser chez les différents fournisseurs. Il s'agit en effet d'impacter à la fois les coûts de transport et les coûts de stockage au niveau de l'usine d'assemblage. Le fait de modifier les quantités de produits à ramasser requiert la mise en place d'un processus de calcul de ces quantités. Nous proposons deux approches pour cela : la première repose sur un programme linéaire, alors que la seconde correspond à une heuristique. Ces deux méthodes sont présentées dans la suite.

Programme linéaire pour la gestion de stock

Dans cette section nous proposons un programme linéaire permettant de déterminer exactement la quantité de produits à ramasser chez les fournisseurs, lorsque les tournées des véhicules sont déterminées. Ce modèle sera utilisé lors de l'évaluation des voisins de la solution courante. Nous introduisons les notations suivantes :

- $M(t)$: l'ensemble des véhicules utilisés pendant la période t.
- $M(t, i)$: l'ensemble des véhicules visitant le fournisseur i pendant la

période t.

- B : une très grande constante représentant une pénalité si une rupture de stock apparaît pour un produit.

Le programme linéaire que nous proposons utilise les variables suivantes :

- I_{it} : le niveau de stock du produit i à l'usine d'assemblage à la période t.
- q_{ivt} : la quantité de produit à collecter chez le fournisseur i à la période t par le véhicule v.
- r_{it} : la quantité de produit i manquante (si une rupture de stock apparaît) pendant la période t.

Le programme linéaire se formule alors comme suit :

$$\min \quad z = \sum_{i \in N} h_i \sum_{t \in T} I_{it} + B \sum_{i \in N} \sum_{t \in T} r_{it} \tag{3.15}$$

$$\text{s.c.} \quad I_{it} = I_{i,t-1} + \sum_{v \in M(t,i)} q_{ivt} - d_{it} + r_{it}, \quad \forall i \in N, \forall t \in T, \tag{3.16}$$

$$\sum_{i \in N(t,v)} q_{ivt} \leq C, \quad \forall t \in T, \forall v \in M(t), \tag{3.17}$$

$$I_{it} \geq 0, \quad \forall i \in N, \forall t \in T, \tag{3.18}$$

$$q_{ivt} \geq 0, \quad \forall t \in T, \forall v \in M(t), \forall i \in N(t,v), \tag{3.19}$$

$$r_{it} \geq 0, \quad \forall i \in N, \forall t \in T. \tag{3.20}$$

La fonction objective (3.15) minimise le coût total de stockage ainsi que le coût associé aux éventuelles ruptures de stock. L'utilisation de la constante B permet de limiter la possibilité d'avoir une rupture de stock. La contrainte (3.16) assure la cohérence des stocks des produits à l'usine d'assemblage sur chaque période de l'horizon, en fonction du stock de la période précédente, de la demande à la période courante, de la quantité de produit ramassée et de l'éventuelle quantité manquante. La contrainte (3.17) garantie que la capacité de chaque véhicule n'est pas excédée, les dernières contraintes (3.18)-(3.20) étant les contraintes de positivité des variables.

L'avantage du programme linéaire proposé est le fait de pouvoir déterminer exactement les quantités de produits à collecter à chaque période tout en

minimisant le coût du stockage, dès lors que les tournées sont déterminées. Cependant, il est clair que la résolution de ce programme linéaire peut être coûteuse dès lors que la taille de l'instance considérée augmente. De plus, dans la recherche à voisinage variable, les tournées changent lors de l'exploration des voisinages des solutions, ce qui implique que pour être précis dans la détermination des quantités à collecter nous aurons besoin de résoudre ce programme linéaire des milliers de fois. Une évaluation exacte des voisins de la solution courante dans un algorithme de type RVV n'est de ce fait pas envisageable en pratique. A partir de cette constatation, nous proposons dans la section suivante une heuristique efficace pour déterminer les quantités de produits à ramasser tout en minimisant le coût de stockage.

Heuristique pour la gestion de stock

A la différence du modèle précédent, l'heuristique que nous proposons ne peut garantir l'optimalité de la solution obtenue. En contre partie, nous cherchons à définir un processus rapide et permettant tout de même d'avoir une bonne estimation des quantités à ramasser. Pour cela, nous utilisons une stratégie de type *retour arrière*, en partant de la période courante de l'horizon et en revenant jusqu'à la première.

Pour chaque période, nous devons déterminer la quantité de produit à collecter chez chaque fournisseur. Nous devons alors prendre en compte deux critères importants : l'ordre dans lequel nous traitons les fournisseurs et l'ordre dans lequel nous considérons les véhicules visitant un fournisseur donné (puisqu'il est possible qu'un fournisseur soit visité par plusieurs véhicules durant la même période).

Pour le premier point, nous calculons le coût de stockage associé au fournisseur $i \in N$ juste avant de le visiter à la période $t \in T$. Plus précisément, ce coût, noté α_{it} est déterminé par la formule suivante :

$$\alpha_{it} = h_i(t - \max\{\max\{t' < t \ : \ \text{si le fournisseur } i \text{ est visité à la période } t'\}, 0\}) \tag{3.21}$$

Cette valeur α_{it} définit une priorité pour le fournisseur i, obtenu à partir du coût de stockage du produit i à l'usine d'assemblage et de la dernière visite chez ce fournisseur. Plus la valeur α_{it} est grande, plus le fournisseur i sera

considéré comme prioritaire au sein de l'heuristique. Une fois qu'un fournisseur est sélectionné, il nous reste à déterminer l'ordre dans lequel traiter les véhicules visitant ce dernier durant la période t considérée. Pour cela, nous estimons dans un premier temps le coût associé au fait pour un véhicule v de ne pas rapporter à l'usine une unité de produit parmi celles prévues initialement. Cette estimation, notée h^*_{vt} est obtenue comme suit :

$$h^*_{vt} = \frac{\sum_{i \in N(t,v)} (d_{it} + I_{i,t+1}) \alpha_{it}}{\sum_{i \in N(t,v)} (d_{it} + I_{i,t+1})}. \tag{3.22}$$

Nous utilisons cette valeur pour ordonner les véhicules, en calculant la valeur β_{vt} associée à chaque véhicule v et chaque période t comme suit :

$$\beta_{vt} = h^*_{vt} \sum_{i \in N(t,v)} max(d_{it} + I_{i,t+1} - C, 0) \tag{3.23}$$

Finalement, les véhicules sont triés dans l'ordre croissant des valeurs β_{vt}, ce qui nous permet de favoriser le remplissage des véhicules utilisés.

L'heuristique de gestion des stocks (HGS) repose donc sur deux critères de tri, et elle est appelée lors de l'évaluation du voisinage de la solution courante.

Elle est présentée dans l'algorithme 3.3.

Descente à voisinage variable

La phase d'amélioration repose sur un algorithme de descente à voisinage variable au sein duquel nous utilisons le modèle mathématique ou l'heuristique présentés précédemment pour déterminer les nouvelles quantités de produits à ramasser chez les fournisseurs lorsqu'un mouvement est appliqué à la solution courante. Nous avons effectué quelques expériences numériques préliminaires pour essayer de déterminer les voisinages pertinents à explorer, et surtout l'ordre dans lequel ils seront utilisés. A partir des résultats observés, nous avons décidé d'explorer les voisinages dans l'ordre suivant au sein de l'algorithme de DVV : $\mathcal{V}_{DVV} = \{\mathcal{V}_3, \mathcal{V}_1, \mathcal{V}_4, \mathcal{V}_1, \mathcal{V}_6, \mathcal{V}_1, \mathcal{V}_5, \mathcal{V}_1, \mathcal{V}_7, \mathcal{V}_1\}$

Une fois les voisinages choisis et l'ordre de parcours de ces voisinages,

Algorithme 3.3 : Heuristique de gestion de stock

Fonction HGS(S)
$I_{i\tau} \leftarrow 0$, pour chaque $i \in N$;
pour chaque *période* t *de* τ *à* 1 **faire**
  Soit $N(t)$ l'ensemble des fournisseurs visités à la période t ;
  pour chaque $i \in N(t)$ **faire**
    Calculer α_{it} selon (3.21) ;
  Réordonner $N(t)$ selon les valeurs de α_{it} décroissantes ;
  $C_{vt} \leftarrow C$, pour chaque $v \in S_t$;
  pour chaque $i \in N(t)$ **faire**
    Soit $M(i,t)$ l'ensemble des véhicules visitant le fournisseur i ;
    pour chaque $v \in M(i,t)$ **faire**
      Calculer β_{vt} selon (3.23) ;
    $E \leftarrow \emptyset$;
    tant que $E \neq M(i,t)$ **faire**
      $v \leftarrow argmin\{\beta_{v't} : v' \in M(i,t) - E\}$;
      $q_{ivt} \leftarrow \min\{I_{it} + d_{it} - \sum_{v' \in E} q_{iv't}, C_{vt}\}$;
      $C_{vt} \leftarrow C_{vt} - q_{ivt}$;
      $E \leftarrow E + \{v\}$;
    $I_{i,t-1} \leftarrow I_{it} + d_{it} - \sum_{v \in M(i,t)} q_{ivt}$;
  pour chaque $v \in N - M(i,t)$ **faire**
    $I_{i,t-1} \leftarrow I_{it} + d_{it}$;
retourner S ;

la stratégie d'exploration est un autre critère important et ayant un impact sur l'efficacité de la méthode. Deux stratégies classiques sont utilisées : le meilleur voisin ou le premier voisin améliorant. Le choix de la stratégie peut reposer, en particulier, sur la taille du voisinage à explorer. Dans notre cas, nous optons pour une exploration complète du voisinage $\mathcal{V}_3$, alors que nous utilisons une stratégie du premier voisin améliorant pour les autres voisinages. La méthode proposée est décrite plus précisément dans l'algorithme 3.4. Dans cet algorithme, nous supposons que c'est l'heuristique de gestion des stocks HGS qui est utilisée (il suffirait de remplacer l'appel à cette heuristique par un appel au modèle pour obtenir la solution alternative). La notation $\mathcal{V}_{[k]}$ fait référence au $k^{\text{ème}}$ voisinage de l'ensemble $\mathcal{V}_{DVV}$.

Algorithme 3.4 : Descente à voisinage variable

Fonction DVV(S, $\mathcal{V}_{DVV}$)
$k \leftarrow 1$;
tant que $k \leq |\mathcal{V}_{DVV}|$ **faire**
  Soit $\mathcal{V}^+ = \{S' : S'' \in \mathcal{V}_{[k]}(S), S' = HGS(S'')\}$;
  $S' \leftarrow RL_1(S, \mathcal{V}^+)$;
  si $f(S') < f(S)$ **alors**
    $S \leftarrow S'$;
    $k \leftarrow 1$;
  sinon
    $k \leftarrow k + 1$;
retourner S;

Recherche à voisinage variable générale

Nous pouvons renforcer l'approche précédente en intégrant la DVV au sein d'une Recherche à Voisinage Variable Générale (RVVG). La RVVG repose, comme la recherche à voisinage variable, sur une phase de perturbation et une phase de recherche locale. La descente à voisinage variable peut ainsi être utilisée comme méthode de recherche locale. Les expériences numériques préliminaires nous ont permis de restreindre le nombre de voisinages à utiliser dans la phase de recherche locale. En fait, nous conservons les voisinages : de déplacement ($\mathcal{V}_1$), de suppression ($\mathcal{V}_3$), d'insertion ($\mathcal{V}_4$) et de remplacement

($\mathcal{V}_6$) d'un fournisseur. L'ordre d'application des voisinages au sein de la méthode de recherche locale est donné par : $\mathcal{V}_{RVVG} = \{\mathcal{V}_3, \mathcal{V}_1, \mathcal{V}_4, \mathcal{V}_1, \mathcal{V}_6, \mathcal{V}_1\}$. Pour la phase de perturbation, nous utilisons les voisinages $\mathcal{V}_3$ et $\mathcal{V}_4$. En pratique, le choix du voisinage est aléatoire, et les deux voisinages ont la même probabilité d'être choisi. La perturbation est réalisée également de façon aléatoire pour la suppression d'un fournisseur (dans le cas de $\mathcal{V}_3$) ou l'insertion d'un fournisseur (pour $\mathcal{V}_4$).

La recherche à voisinage variable générale est décrite dans l'algorithme 3.5. Dans cet algorithme, k_{max} correspond au nombre maximum de perturbations consécutives à appliquer à la solution courante (sur un principe similaire à l'algorithme de RVV de la première phase, voir l'algorithme 3.2). Le paramètre CPU_{max} est le temps maximum alloué à l'algorithme.

Algorithme 3.5 : Recherche à voisinage variable générale

```
Fonction RVVG(S, V_RVVG, k_max, CPU_max)
répéter
    k ← 1 ;
    tant que k ≤ k_max faire
        Soit S' ∈ (V_3 ∪ V_4)^[k](S) ;
        S'' ← DVV(S', V_RVVG) ;
        si f(S'') < f(S) alors
            S ← S'' ;
            k ← 1 ;
        sinon
            k ← k + 1 ;
jusqu'à CPU < CPU_max ;
retourner S ;
```

3.4 Expérimentations et résultats

Dans cette section nous présentons les résultats obtenus avec les approches proposées et décrites précédemment. Tous les algorithmes ont été implémentés avec le langage C++. La machine utilisée est équipée d'un processeur Pentium IV 3.4 GHz et dispose de 4 Go de mémoire RAM. Nous avons observé rapidement que le modèle mathématique pour la gestion des stocks

n'était pas utilisable en pratique, en particulier parce qu'il doit être résolu un grand nombre de fois lors de l'évaluation du voisinage des solutions rencontrées pendant la recherche. Nous présentons donc les résultats obtenus avec l'heuristique de gestion de stocks (HGS) présentée dans l'algorithme 3.3. Les résultats reportés ont été obtenus sur un ensemble d'instances existantes et décrites ci-dessous.

3.4.1 Caractéristiques des instances et paramétrage

Le problème étudié dans ce chapitre a été étudié dans Moin et al. [98]. Les auteurs ont généré un ensemble de 14 instances pour valider leur approche, basée sur un algorithme génétique. Ces 14 instances sont caractérisées entre autre par le nombre de fournisseurs et un nombre de périodes, et ont été générées à partir de 4 instances proposées initialement par Lee et al. [82]. Nous pouvons distinguer 3 catégories d'instances : les petites instances comportant 12 fournisseurs, les instances de taille moyenne (avec 20 ou 50 fournisseurs) et les instances de grande taille (avec 98 fournisseurs). Les caractéristiques des instances sont données de façon plus précise dans le tableau 3.1. Dans ce tableau et dans la suite nous utilisons la même notation que dans [98] pour nommer une instance : la valeur qui suit "S" correspond au nombre de fournisseurs, et la valeur de "y" fait référence au nombre de périodes. Notons finalement que les instances sont téléchargeables sur le site web du *Centre for Logistics and Heuristic Optimisation* : `www.kent.ac.uk/kbs/research/research-centres/clho`.

Type d'instances	S12Ty	S20Ty	S50Ty	S98Ty
Coût fixe	20	20	20	200
Coût de voyage	1	1	1	50
Capacité des véhicules	10	10	10	400
Coût de stockage	[3, 27]	[3, 27]	[1, 9]	[1, 44]
Demande	[1, 4]	[1, 4]	[0, 9]	[0.04, 393.33]

TABLEAU 3.1 – Caractéristiques des instances de la littérature.

Notre approche en deux phases nécessitent le réglage de quelques paramètres. La première phase, qui repose sur un algorithme de RVV s'arrête dès

qu'il n'est plus possible d'améliorer la solution courante ou que le nombre maximum d'itérations est atteint. Les deux paramètres de la méthode sont k_{max} et $iter_{max}$. Les valeurs utilisées pour ces paramètres sont $k_{max} = 10$ et $iter_{max} = 200 \times T$.

La deuxième phase repose sur l'algorithme de descente à voisinage variable ou l'algorithme de RVV générale. Dans le cas de la DVV, le critère d'arrêt est la non amélioration de la solution courante. Cet algorithme n'a pas d'autre paramètre (les voisinages utilisés étant décrits dans la section correspondante). Pour la recherche à voisinage variable générale, nous définissons une probabilité de perturbation pour choisir entre la suppression (voisinage $\mathcal{V}_3$) ou l'insertion (voisinage $\mathcal{V}_3$) d'un fournisseur. Les expériences numériques préliminaires nous ont guidés vers le choix de l'intervalle $[0; 0, 2]$ pour la suppression et $[0, 3; 1]$ pour l'insertion. Le paramètre k_{max} est fixé à $4 \times T$. Finalement, le critère d'arrêt de l'algorithme est un temps maximum de calcul, fixé à 2000 secondes.

3.4.2 Étude comparative

Pour évaluer le comportement et la performance des approches proposées, nous avons exécuté 10 fois chaque approche sur chaque instance de la littérature.

Nous reportons dans le tableau 3.2 les résultats obtenus par les deux variantes proposées dans ce chapitre : l'algorithme reposant sur la descente à voisinage variable dans la seconde phase (noté 2p-VND) et celui utilisant la recherche à voisinage variable générale dans la seconde phase (noté 2p-VNS). Pour avoir une idée plus précise de la performance de ces approches, nous reportons dans la colonne "GA" les résultats reportés dans [98] et obtenus avec un algorithme génétique sur ces instances. Il est important de noter que pour cet algorithme nous reprenons les meilleurs résultats obtenus par les différentes variantes proposées par les auteurs. Dans le tableau 3.2, la première colonne, "Inst", reporte le nom de l'instance, alors que la colonne (n, T) donne le nombre de fournisseurs et le nombre de périodes définis dans l'instance. Les valeurs reportées dans les colonnes f^* et $\#v$ correspondent, respectivement, au coût associé à la meilleure solution sur les 10 exécutions

et au nombre de véhicules utilisés dans cette solution. Les valeurs en gras dans ce tableau correspondent aux meilleurs coûts obtenus.

Inst	(n, τ)	GA		2p-VND		2p-VNS	
		f^*	$\#v$	f^*	$\#v$	f^*	$\#v$
S12T5	(12, 5)	2096,75	14	**1961,71**	14	**1961,71**	14
S12T10	(12, 10)	4333,27	29	**4002,85**	29	**4002,85**	29
S12T14	(12, 14)	6115,19	41	**5635,77**	41	**5635,77**	41
S20T5	(20, 5)	3143,39	21	3045,08	23	**2861,26**	21
S20T10	(20, 10)	6499,40	43	6098,15	46	**5944,72**	43
S20T14	(20, 14)	9208,43	61	8589,36	65	**8422,02**	63
S20T21	(20, 21)	13948,41	92	12944,08	98	**12700,27**	94
S50T5	(50, 5)	5618,09	45	5144,23	46	**5084,00**	45
S50T10	(50, 10)	11642,00	95	10787,88	99	**10694,60**	96
S50T14	(50, 14)	16987,00	135	15399,82	139	**15384,96**	137
S50T21	(50, 21)	26448,77	209	23500,94	214	**23497,40**	211
S98T5	(98, 5)	561168,21	57	575790,06	59	**557287,97**	57
S98T10	(98, 10)	1124797,57	113	1160921,87	119	**1119767,72**	114
S98T14	(98, 14)	1571652,32	159	**1567904,75**	160	1577951,04	161
Moyenne	**-**	240261,34	79,57	242980,47	82,29	239371,16	80,43

TABLEAU 3.2 – Comparaison entre les approches proposées et l'algorithme génétique de Moin et al. (2010) [98].

Les résultats reportés dans le Tableau 3.2 montrent que, globalement, les deux approches proposées obtiennent de meilleures solutions (en termes de coût) que celles obtenues par l'algorithme génétique. De façon plus précise, la méthode 2p-VND améliore 12 valeurs par rapport à celles fournies dans la littérature, ne parvenant pas à faire mieux pour les instances S98T5 et S98T10. De son côté, la méthode 2p-VNS obtient 13 meilleures solutions sur les 14 instances, avec uniquement l'instance S98T14 pour laquelle elle ne parvient pas à améliorer la valeur de la solution de l'algorithme génétique. Si nous nous intéressons au nombre de véhicules utilisés, nous pouvons remarquer quelques déviations entre les 3 approches. Il est intéressant d'observer qu'il arrive parfois d'obtenir une meilleure solution (en termes de coût) en utilisant un ou deux véhicules supplémentaires. Cette situation peut s'expliquer si les tournées correspondantes n'engendrent pas un coût de transport trop important par rapport au coût de stockage des produits.

De manière à compléter notre analyse des résultats, nous reportons dans le Tableau 3.3 quelques éléments de comparaison complémentaires : la valeur

de la borne inférieure donnée dans Moin et al. [98] (colonne "BI"), puis la déviation observée entre la solution fournie par les différentes approches et cette borne. La borne inférieure a été obtenue en utilisant le solveur `CPLEX` d'IBM-ILOG pour résoudre le modèle mathématique présenté dans la section 3.2. En notant $f(S)$ le coût de la solution S fournie par un algorithme donné, la déviation par rapport à la borne inférieure est obtenue par le calcul suivant :

$$\Delta(\%) = ((f(S) - LB)/LB) \times 100. \tag{3.24}$$

Nous donnons dans le tableau 3.3 les déviations minimale (min), maximale (max) et en moyenne (moy) pour chacune des deux approches ($\Delta_{2p-VND}(\%)$ et $\Delta_{2p-VNS}(\%)$), et la déviation associée à la solution de l'algorithme génétique ($\Delta f^*_{GA}(\%)$).

Notons que pour les deux dernières instances il n'y a pas de borne inférieure reportée dans [98], le logiciel n'ayant pu en fournir en 1 heure de temps de calcul. Dans ce cas, les valeurs reportées correspondent à la déviation entre la valeur de la solution fournie par un algorithme et la meilleure valeur obtenue (f_{Best}) par l'ensemble des trois approches :

$$\Delta(\%) = ((f(S) - f_{Best})/f_{Best}) \times 100. \tag{3.25}$$

Les valeurs du tableau 3.3 confirment le bon comportement des deux approches proposées. La dernière ligne montre que la déviation moyenne est toujours inférieure à celle de l'algorithme génétique, en considérant la meilleure, la moins bonne ou la moyenne des solutions retournées par nos méthodes. Les valeurs moyennes ne reflétant pas toujours la réalité pratique, nous pouvons aussi observer que la déviation maximale des deux algorithmes 2p-VND et 2p-VNS est inférieure à celle de l'algorithme génétique dans 11 cas sur les 14 instances. En fait, seuls les résultats obtenus sur les trois dernières instances sont un peu moins à l'avantage de nos deux algorithmes.

Un autre point de comparaison des différentes approches est le temps de calcul. Nous reportons dans le tableau 3.4 le temps d'exécution moyen (colonne t) et le temps d'exécution nécessaire pour obtenir la meilleure solution par les algorithmes 2p-VND et 2p-VNS (colonne t^*). La colonne t^*_{GA} reporte le temps de calcul de la meilleure version de l'algorithme génétique.

Inst.	BI	$\Delta f^*_{GA}(\%)$	$\Delta_{2p-VND}(\%)$			$\Delta_{2p-VNS}(\%)$		
			min	moy	max	min	moy	max
S12T5	1650	27,07	18,89	18,89	18,89	18,89	18,89	18,89
S12T10	3218	34,65	24,39	24,39	24,39	24,39	24,39	24,39
S12T14	4709	29,86	19,68	19,68	19,68	19,68	19,68	19,68
S20T5	2607	20,57	16,80	16,80	16,80	9,75	11,34	13,35
S20T10	5227	24,34	16,67	17,40	18,32	13,73	14,23	14,71
S20T14	7181	28,23	19,61	20,15	20,94	17,28	17,77	18,34
S20T21	10717	30,15	20,78	21,40	22,26	18,51	19,34	19,89
S50T5	4547	23,55	13,13	14,30	15,22	11,81	12,64	13,45
S50T10	9289	25,33	16,14	16,87	17,38	15,13	16,08	16,72
S50T14	13193	28,75	16,73	17,38	17,86	16,61	17,15	17,76
S50T21	20185	31,03	16,43	17,02	17,60	16,41	16,94	17,50
S98T5	544036	3,14	5,84	7,36	7,58	2,44	3,55	4,31
S98T10	-	0,45	3,68	4,32	4,50	0,00	1,25	3,62
S98T14	-	0,24	0,00	3,85	4,50	0,64	2,13	3,90
Moyenne (%)	-	21,95	14,91	15,70	16,14	13,23	13,96	14,75

TABLEAU 3.3 – Déviation par rapport à la borne inférieure.

Inst	(n, T)	t^*_{GA} [a]	2p-VND		2p-VNS	
			t	t^*	t	t^*
S12T5	(12, 5)	58,48	0,08	0,10	0,10	0,10
S12T10	(12, 10)	111,23	0,21	0,20	0,20	0,20
S12T14	(12, 14)	120,31	0,34	0,30	0,30	0,30
S20T5	(20, 5)	31,81	0,37	0,40	25,30	27,20
S12T10	(20, 10)	126,03	3,12	2,74	102,30	98,34
S20T14	(20, 14)	360,33	9,63	8,45	89,50	113,36
S20T21	(20, 21)	255,83	20,85	17,79	222,70	185,99
S50T5	(50, 5)	133,40	4,97	3,55	112,40	93,03
S50T10	(50, 10)	226,01	34,38	27,68	75,30	183,79
S50T14	(50, 14)	328,07	64,90	55,58	289,92	212,07
S50T21	(50, 21)	496,72	196,31	140,51	373,20	370,40
S98T5	(98, 5)	476,77	100,14	76,50	1263,80	799,60
S98T10	(98, 10)	1307,26	467,61	330,39	1768,30	1128,34
S98T14	(98, 14)	1589,71	5793,01	1353,51	1210,30	1354,80
Moyenne	-	401.57	478.28	144.11	395.26	326.26

TABLEAU 3.4 – Comparaison des temps de calcul.

a. L'algorithme a été exécuté sur une machine avec un processeur à 2,8 GHz et 4 Gb de RAM

Les résultats reportés dans le tableau 3.4 montrent que les deux approches proposées rivalisent aussi en termes de temps de calcul avec l'algorithme génétique. En particulier, les temps de calcul observés pour les petites et

moyennes instances sont nettement inférieurs, en particulier pour l'approche 2p-VND. Cette méthode est logiquement moins coûteuse que 2p-VNS (en général), la différence entre les deux méthodes augmentant sensiblement avec la taille des instances. Nous pouvons aussi observer que la différence entre les deux valeurs t et t^* n'est finalement pas très importante, en général, pour 2p-VND et 2p-VNS. Cette différence a toutefois tendance à augmenter pour les instances de plus grande taille.

Tous ces résultats démontrent que l'approche proposée permet d'obtenir de meilleurs résultats en moyenne (en considérant le coût de la solution et le temps de calcul) que l'algorithme génétique précédemment conçu pour résoudre ce problème.

Finalement, notons que le modèle mathématique proposé dans la section 3.3.3 peut remplacer l'heuristique de gestion des stocks. Nous avons testé cette éventualité dans l'approche 2p-VND, mais les résultats obtenus ont confirmés l'impact trop important du modèle sur le temps de calcul de la méthode.

3.5 Conclusion

Dans ce chapitre, nous avons étudié un problème spécifique de routage avec gestion de stock. Il consiste à ramasser différents produits chez un ensemble de fournisseurs afin d'assurer la demande d'une usine d'assemblage, et ainsi de permettre une production continue du produit fini. L'objectif du problème est de minimiser les coûts associés au transport et au stockage. Afin de trouver la meilleure solution possible en respectant un compromis entre ces objectifs, nous avons proposé deux méthodes basées sur la recherche à voisinage variable.

Dans les deux cas, l'algorithme de résolution travaille en deux phases. La première repose également sur un algorithme de recherche à voisinage variable, dans le but de trouver une solution initiale réalisable et dans laquelle les coûts de stockage ne sont pas considérés. L'idée consiste à se ramener à la résolution d'un problème d'élaboration de tournées de véhicules, en s'assurant que la demande de l'usine d'assemblage soit respectée exactement pour chaque période de l'horizon de planification.

Pour la seconde phase, nous avons proposé deux algorithmes. Le premier est une descente à voisinage variable et le second une méthode de recherche à voisinage variable générale. Cette fois les algorithmes prennent en compte tous les coûts du problème. Les différents voisinages définis et utilisés dans ces algorithmes permettent une modification implicite des quantités à transporter (par déplacement, suppression, insertion d'un fournisseur dans une tournée). Nous avons ensuite proposé un modèle mathématique permettant de définir exactement les nouvelles quantités à ramasser chez les fournisseurs suite à ces modifications de tournées. L'utilisation pratique de ce modèle étant fortement contrainte par la taille des instances, nous avons également développé une heuristique de gestion des stocks. Cette heuristique peut être utilisée lors de l'évaluation du voisinage d'une solution donnée. Nous avons finalement évalué et validé les deux méthodes sur un ensemble de 14 instances disponibles dans la littérature. Les résultats ont montré qu'elles étaient toutes les deux capables de faire mieux qu'un algorithme génétique proposé précédemment pour résoudre ce problème, aussi bien au niveau du coût de la solution finale que du temps de calcul requis.

Les résultats obtenus par les deux approches de résolution sont donc encourageants. Nous avons cependant remarqué que les plus grandes instances leur posaient plus de problèmes. Nous avons donc continué à travailler sur ce problème, en restant focalisé sur la recherche à voisinage variable, mais en essayant d'apporter quelques modifications sur la représentation d'une solution, en particulier sur la gestion des quantités de produit à ramasser, et sur les voisinages utilisés. Un des objectifs est de pouvoir permettre une mise en œuvre plus simple de mouvements entre des périodes différentes. Nous abordons ce travail dans le chapitre suivant.

Chapitre 4

Améliorations basées sur la gestion des quantités de produit

4.1 Introduction

Nous avons abordé dans le chapitre précédent un problème particulier d'élaboration de tournées avec gestion de stock multi-produits. Devant la difficulté de ce genre de problème, il est généralement difficile de concevoir et mettre en œuvre une méthode exacte efficace pour résoudre des instances comportant un nombre significatif de clients (ou fournisseurs dans notre cas). Nous nous sommes donc naturellement orientés vers le développement d'une métaheuristique, et avons opté pour la recherche à voisinage variable. L'efficacité de l'approche décrite dans le chapitre précédent repose, en plus des structures de voisinages dédiées au problème de tournées avec gestion de stock, d'une heuristique efficace de gestion des stocks utilisée pour estimer la quantité de produit à ramasser chez chacun des fournisseurs à chaque période de l'horizon de planification.

Dans ce chapitre, nous considérons le même problème, et nous proposons une nouvelle recherche à voisinage variable dans le but d'améliorer les résultats obtenus précédemment. Dans cette approche, nous déployons de nouvelles structures de voisinage qui s'appuient en particulier sur des mouvements de quantités de produits entre les périodes et entre les tournées. La mise en œuvre de ces voisinages passe par une gestion différente de la quantité de produit à ramasser sans avoir recours à l'heuristique de gestion de stock. Les résultats expérimentaux montrent que notre nouvel algorithme converge vers des solutions *presque* optimales plus rapidement.

Le reste de ce chapitre est organisé comme suit. La Section 4.2 décrit la

représentation de la solution utilisée dans la RVV, et introduit la règle de ramassage des quantités de produit permettant d'introduire des mouvements de quantités. Les structures de voisinage sont décrites dans la Section 4.3. L'approche de résolution est présentée et synthétisée dans la Section 4.4, alors que la Section 4.5 est consacrée aux expérimentations numériques menées sur les instances disponibles. Nous terminons ce chapitre par une conclusion et quelques perspectives de recherche dans la Section 4.6.

4.2 Représentation d'une solution et gestion des quantités à collecter

L'efficacité d'une métaheuristique et en particulier de la recherche à voisinage variable dépend fortement de la représentation de la solution du problème considéré. Nous avons opté pour une représentation proche de celle décrite dans le chapitre 3. Comme décrit dans la suite, la gestion des quantités de produits à ramasser sera traitée différemment dans cette nouvelle approche. Pour une lecture autonome de ce chapitre, nous reprenons l'ensemble des notations utilisées. Une solution S du problème est décomposée en un ensemble de τ sous–solutions (ou solutions partielles) associées aux τ périodes de l'horizon de planification, i.e.

$$S = \{S_1, ..., S_\tau\}.$$

Chacun des τ éléments de S doit fournir les informations nécessaires à la construction d'une solution du problème original. Il s'agit en particulier dans notre cas de décrire les tournées qui doivent être effectuées à chaque période. En notant $m(t)$ le nombre de véhicules utilisés durant la période $t \in T$ (une route étant associée à un véhicule), nous obtenons :

$$S_t = \{S_{t_1}, ..., S_{t_{m(t)}}\}$$

Pour obtenir une description complète des tournées, il reste à préciser la suite des fournisseurs visités dans chaque tournée de la période considérée. Ainsi, pour une période $t \in T$ et un véhicule $v \in \{1, \ldots, m(t)\}$, et en supposant

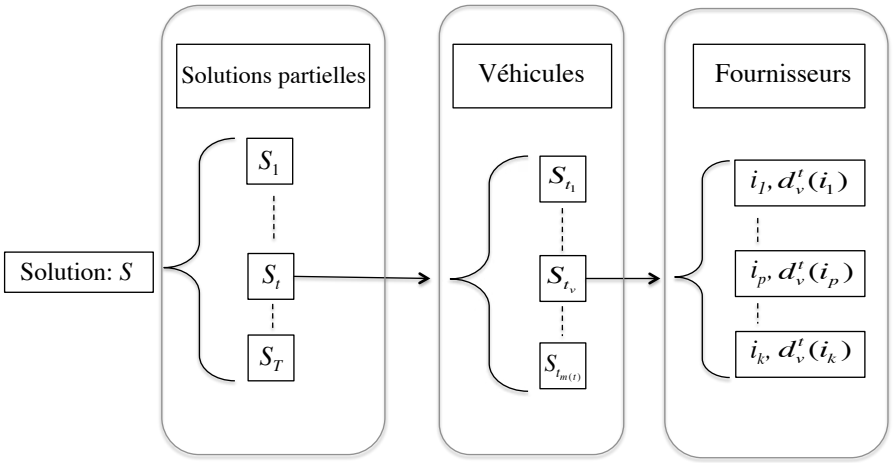

FIGURE 4.1 – Représentation d'une solution S

que le véhicule visite k fournisseurs, nous notons la tournée S_{t_v} par :

$$S_{t_v} = \{i_1, i_2, \ldots, i_k\}$$

Il est évident que cette description ne permet pas une représentation totale de la solution. Il manque en effet un élément important : la quantité de produit à ramasser chez chaque fournisseur. Cette quantité est notée $d_v^t(i)$ pour la tournée de la période t par le véhicule v chez le fournisseur i. La figure 4.1 illustre la représentation utilisée.

Le calcul des valeurs $d_v^t(i)$ est fondamentale dans l'approche de résolution. Lee et al. [82] ainsi que Moin et al. [98] manipulent ces quantités comme des variables continues. Nous avons également procédé de cette façon dans le chapitre précédent. Plus précisément pour une certaine période t, la quantité de produit collectée chez un fournisseur i est une fraction de la demande de l'usine d'assemblage. L'approche que nous proposons dans ce chapitre est de considérer ces quantités comme des variables entières : si un véhicule visite un fournisseur i à la période p, alors il collecte (au moins) toute la demande de l'usine d'assemblage pour ce produit à cette période. Ainsi, nous introduisons la variable discrète y_{ip} définie comme suit :

$$y_{ih} = \begin{cases} 1, \text{si le fournisseur } i \text{ est visité à la période } p \\ 0, \text{sinon.} \end{cases}$$

La quantité prélevée par le véhicule v chez le fournisseur $i \in S_{t_v}$ à la période t est alors définie par :

$$d_v^t(i) = \sum_{p=t}^{T} d_{ip} y_{ip}$$

Cette définition de la quantité à prélever implique que le véhicule v associé à la tournée peut collecter des quantités du produit i pour des périodes postérieures à t. En effet avec cette procédure de ramassage, à chaque période $p \in \{t, \ldots, \tau\}$ le véhicule collecte une quantité q égale à toute la quantité demandée par l'usine d'assemblage à cette période $q = d_{ip}$ ou rien du tout $q = 0$. Cette définition nous assure donc de ne manipuler que des quantités discrètes de produits dans les solutions.

Notons que certaines sous–solutions S_t d'une solution S peuvent être vides (i.e., elles ne contiennent aucune tournée). Cela s'explique par le fait que les quantités de produits nécessaires à une période t donnée pour l'usine d'assemblage aient pu être collectées à des périodes antérieures. Nous illustrons cette situation dans l'exemple suivant.

Exemple : Considérons un exemple avec trois fournisseurs, un horizon de planification décomposé en trois périodes $T = \{1, 2, 3\}$, et un seul véhicule disponible. L'usine d'assemblage a une demande d_{it} pour chaque fournisseur $i \in \{1, 2, 3\}$ et à chaque période $t \in T$. La figure 4.2 montre une situation possible où il n'y a pas de tournée pour la période 3.

Dans la figure 4.2, le véhicule visite les trois fournisseurs lors de la première période, la solution partielle correspondante est alors $S_{1_1} = \{1, 2, 3\}$. Supposons que durant cette tournée le véhicule collecte la quantité de produit associée à la demande $d_{1,1}$ (i.e., $d_1^1(1) = d_{1,1}$) pour le premier fournisseur, toute la quantité de produit requise chez le fournisseur 2 (i.e., $d_1^1(2) = d_{2,1} + d_{2,2} + d_{2,3}$), et la quantité de produit demandée pour la première période chez le fournisseur 3 (i.e., $d_1^1(3) = d_{3,1}$). Supposons ensuite que lors de la se-

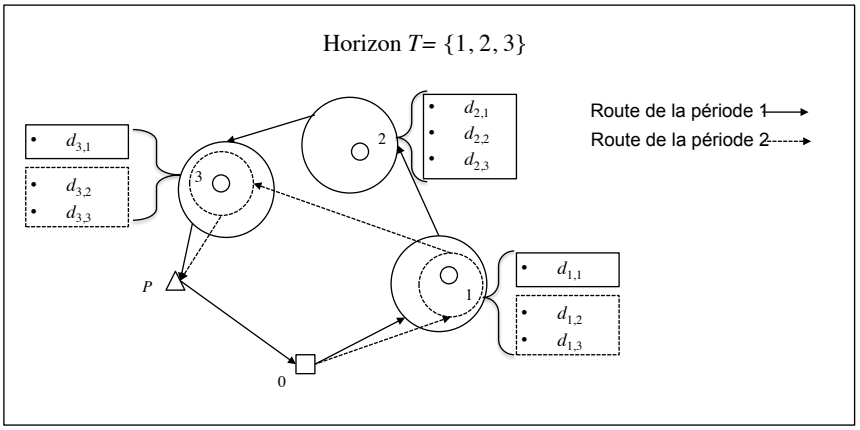

FIGURE 4.2 – Illustration d'une période sans tournée dans une solution.

conde période, le véhicule ramasse les quantités demandée pour les périodes 2 et 3 chez les fournisseurs 1 et 3. Dans ce cas nous avons $S_{2_1} = \{1, 3\}$ et $d_1^2(1) = d_{1,2} + d_{1,3}$ et $d_1^2(3) = d_{3,2} + d_{3,3}$. Ainsi, dans l'exemple toutes les quantités de produits ont alors été ramassées, et qu'il est donc inutile d'effectuer une tournée lors de la dernière période : $S_{3_1} = \varnothing$.

Dans la méthode de résolution basée sur la RVV en deux phases présentée dans le chapitre précédent, les mouvements utilisés concernaient directement les fournisseurs (suppression, ajout, permutation) présents dans une tournée ou dans une période. Dans ce chapitre, les mouvements modifient les quantités ramassées d'une manière discrète (par période) sans avoir recours à une heuristique ni à un modèle mathématique. Nous décrivons ces mouvements et les structures de voisinages associées dans la section suivante.

4.3 Structures de voisinages

Dans cette section nous définissons six voisinages basés sur les mouvements des quantités entre des tournées ou entre des périodes. Ces voisinages peuvent être regroupés en deux catégories : les mouvements de type insertion et les mouvements de type échange.

4.3.1 Voisinages de type insertion ($\mathcal{V}_1$)

Trois voisinages de type insertion sont proposés, où les mouvements correspondants concernent, selon le cas, une seule tournée, deux tournées de la même période ou deux tournées de périodes différentes.

Insertion dans la même tournée (déplacement) $\mathcal{V}_{1,1}$

Ce voisinage consiste à déplacer une quantité à ramasser au sein de la même tournée. Autrement dit, il s'agit de modifier l'ordre de visite des fournisseurs par un véhicule pour une période donnée. Il s'agit d'un mouvement classique dans les problèmes d'élaboration de tournées, parfois référencé comme un mouvement 1–0. Ce mouvement est illustré dans la figure 4.3. Les exemples présentés dans cette figure et dans les figures suivantes sont fournis sous la forme des tournées et de tableaux qui synthétisent les informations relatives à ces tournées. En particulier, la ligne *Route* indique l'ordre de visite des fournisseurs, la ligne *Quantités* reprend les quantités collectées chez chaque fournisseur, et la ligne *Périodes* mentionnent la période associée à la demande qui est collectée. Pour illustrer cela, nous pouvons par exemple voir dans la figure 4.3 qu'après le mouvement le véhicule visite les fournisseurs 1, 2,..., 6. Chez le fournisseur 1 il ramasse la demande correspondant aux périodes 1 et 3. Nous pouvons constater que le mouvement associé au voisinage $\mathcal{V}_{1,1}$ ne modifie que le coût de transport de la solution.

Insertion entre deux tournées de la même période $\mathcal{V}_{1,2}$

Ce voisinage concerne deux tournées de la même période. Le mouvement associé consiste à supprimer une quantité déjà planifiée dans une tournée et à l'insérer dans une autre tournée de la même période. Deux cas doivent être distingués :

- Premier cas : le fournisseur est déjà présent dans la tournée où l'insertion est effectuée. Dans ce cas, l'insertion revient à fusionner les deux quantités qui seront ramassées chez le fournisseur. La tournée (i.e., l'ordre de visite des fournisseurs) n'est pas modifiée. La figure 4.4 illustre ce cas. Dans cet exemple, la quantité de produit collectée chez le fournisseur 3 par le véhicule 2 (route b) est transférée dans la tournée

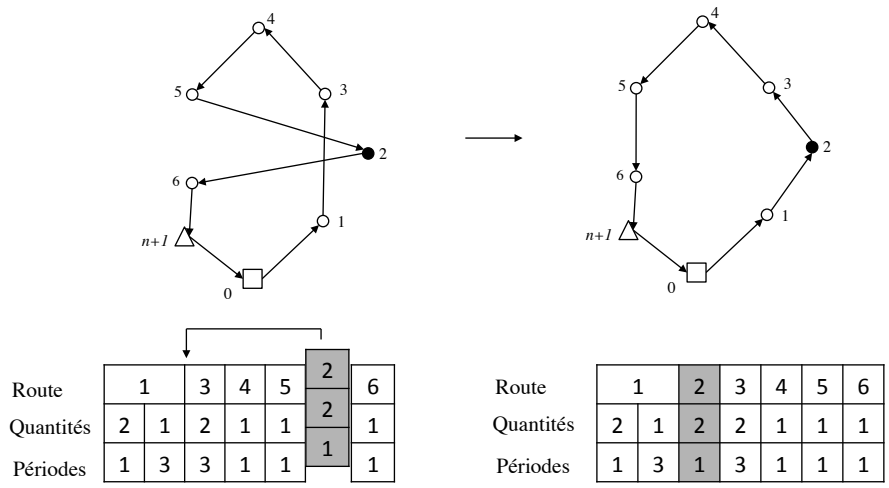

FIGURE 4.3 – Voisinage $\mathcal{V}_{1,1}$: insertion dans la même tournée (déplacement)

du véhicule 1. Après ce mouvement, le fournisseur 3 n'apparaît plus dans la tournée du véhicule 2.

– Deuxième cas : le fournisseur n'existe pas dans la tournée où se fait l'insertion. Dans ce cas, l'insertion de la quantité à ramasser revient à l'insertion d'un nouveau fournisseur dans la tournée, ce qui entraine une modification de cette dernière. Ce mouvement est illustré dans la figure 4.5.

Insertion entre deux tournées de périodes différentes $\mathcal{V}_{1,3}$

Ce voisinage est similaire au voisinage $\mathcal{V}_{1,2}$ mais il est plus grand car nous considérons deux tournées de deux périodes différentes. Le principe du mouvement reste cependant similaire au cas précédent : une fusion des quantités à ramasser est faite si le fournisseur est déjà présent dans la tournée d'insertion, alors qu'une insertion du fournisseur est nécessaire si celui-ci n'est pas déjà présent dans la tournée. En fait, le voisinage $\mathcal{V}_{1,2}$ est un cas particulier du voisinage $\mathcal{V}_{1,3}$.

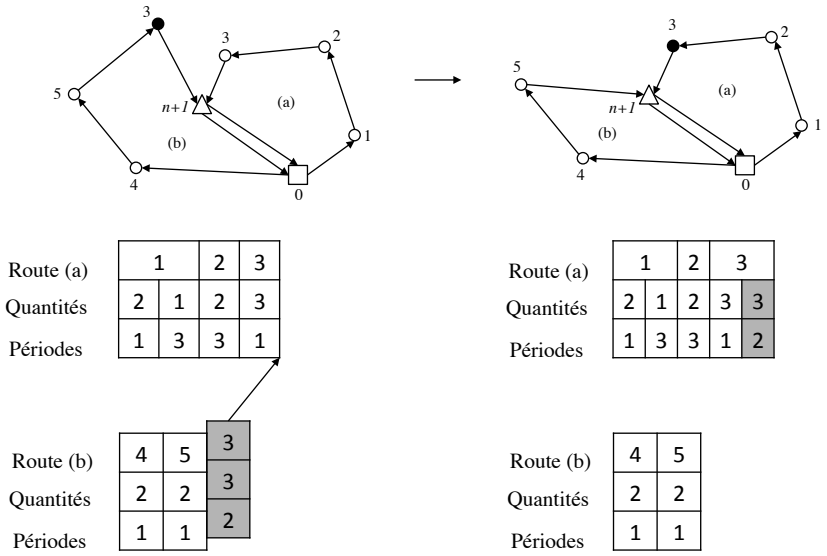

FIGURE 4.4 – Voisinage $\mathcal{V}_{1,2}$, cas 1 : Fournisseur existant (jointure des quantités)

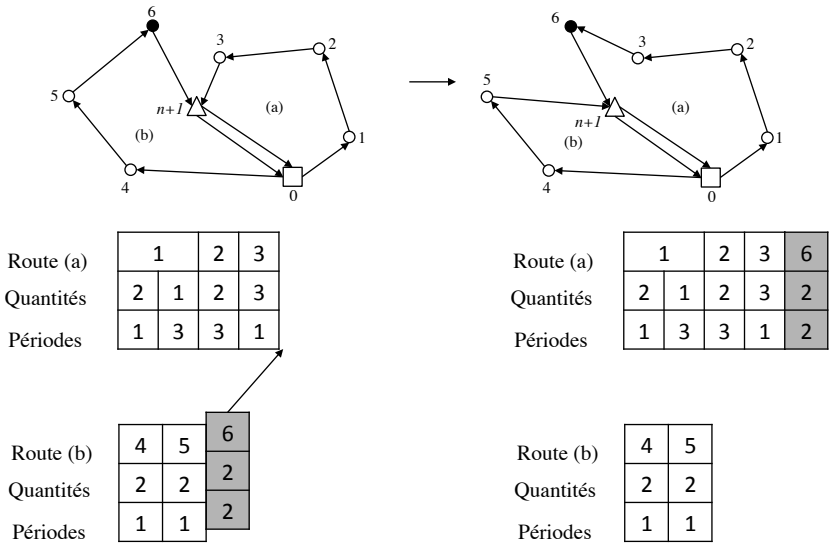

FIGURE 4.5 – Voisinage $\mathcal{V}_{1,2}$, cas 2 : fournisseur inexistant (Changement de forme)

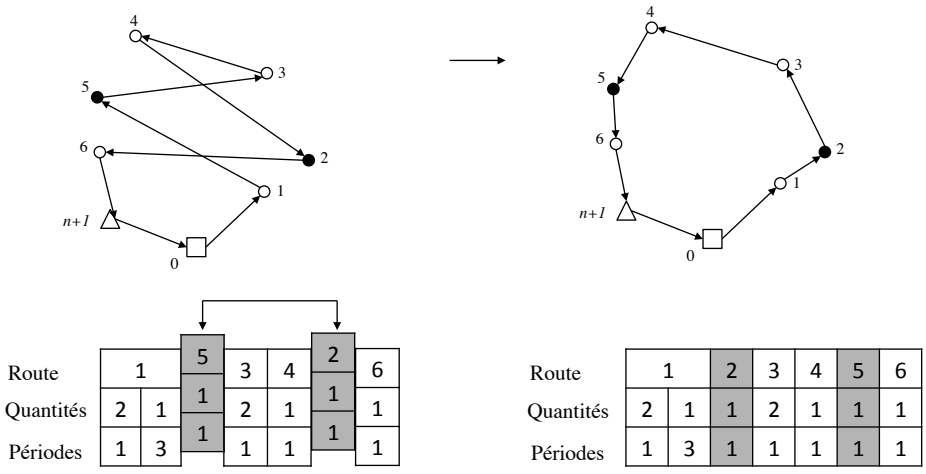

FIGURE 4.6 – Voisinage $\mathcal{V}_{2,1}$: échange dans la même tournée

4.3.2 Voisinages de type échange ($\mathcal{V}_2$)

Les voisinages de type échange reviennent à permuter deux quantités. De façon similaire aux voisinages d'insertion, ils sont définis au sein de la même tournée, entre deux tournées de la même période ou entre deux tournées de deux périodes différentes.

Échange dans la même tournée $\mathcal{V}_{2,1}$

Ce voisinage revient à échanger deux quantités à ramasser de deux fournisseurs visités dans la même tournée. Il est donc équivalent au voisinage classique d'échange dans les problèmes d'élaboration de tournées, et le mouvement 2-opt peut ainsi être un cas particulier de ce mouvement. La figure 4.6 présente un exemple.

Dans cet exemple, la position des fournisseurs 5 et 2 est inversée, ce qui correspond bien au mouvement classique utilisé dans le problème d'élaboration de tournées.

Échange entre deux tournées de la même période $\mathcal{V}_{2,2}$

Ce voisinage concerne l'échange de deux quantités entre deux tournées de la même période. Cela revient à permuter deux quantités à ramasser entre

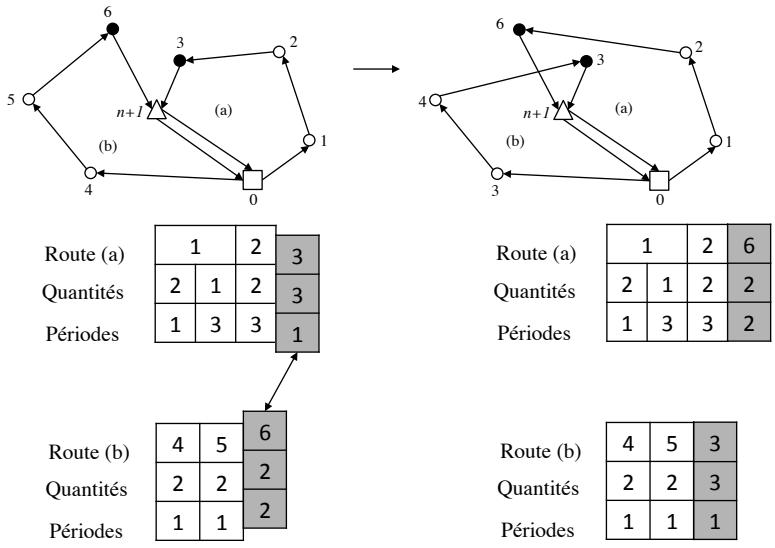

FIGURE 4.7 – Voisinage $\mathcal{V}_{2,2}$, cas 1 : échange entre deux tournées de la même période

deux tournées. Comme dans le cas du voisinage $\mathcal{V}_{1,2}$, deux cas peuvent se produire :

- Si chacun des deux fournisseurs à permuter n'est présent que dans une des deux tournées, alors le mouvement correspond à un échange de fournisseurs (voisinage souvent référencé échange 1-1 dans la littérature). Ce cas est illustré dans la figure 4.7.
- Si au moins un des deux fournisseurs est déjà présent dans les deux tournées impactées par le mouvement, alors les quantités à échanger doivent en plus être fusionnées. La figure 4.8 illustre ce cas avec les fournisseurs 3 et 5. Après le mouvement, le fournisseur 3 est visité seulement par la seconde route et la quantité collectée par le véhicule chez ce fournisseur est égale à $d_v^t(3) = d_{3,1} + d_{3,2}$.

Échange entre deux tournées de périodes différentes $\mathcal{V}_{2,3}$

Comme dans le cas des voisinages d'insertion, ce voisinage est une généralisation du cas précédent au cas plus général impliquant deux périodes distinctes. Le principe reste tout à fait similaire lors de l'échange des four-

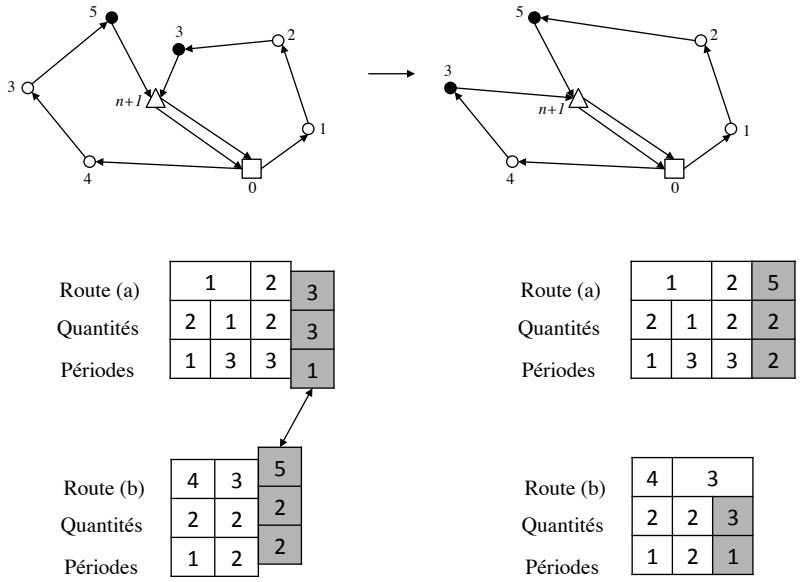

FIGURE 4.8 – Voisinage $\mathcal{V}_{2,2}$, cas 2 : échange entre deux tournées de la même période

nisseurs (jointure des quantités ou échange simple des fournisseurs).

4.4 Approche de résolution

Nous présentons dans cette section notre approche de résolution basée sur la RVV. Plus précisément, nous définissons une approche de recherche à voisinage variable générale qui utilise une descente à voisinage variable comme recherche locale [93].

4.4.1 Phase de descente

La descente à voisinage variable (DVV) est une version déterministe de la RVV. De manière à décrire précisément notre approche, nous commençons par définir un algorithme de recherche locale noté $RLV(S, \mathcal{V}_{i,1}, \mathcal{V}_{i,2}, \mathcal{V}_{i,3})$, où S est une solution initiale, et $\mathcal{V}_{i,1}$, $\mathcal{V}_{i,2}$, et $\mathcal{V}_{i,3}$ sont trois structures de voisinages parmi les six décrites dans la section 4.3 avec $i \in \{1, 2\}$. Cet algorithme RLV est une méthode de descente que nous utilisons au sein de notre DVV.

En fait, dans l'algorithme RLV, nous explorons consécutivement les voisinages $\mathcal{V}_{i,1}$, $\mathcal{V}_{i,2}$, et $\mathcal{V}_{i,3}$ autour de la solution courante jusqu'à ce qu'il n'y ait plus d'amélioration, pour un $i \in \{1, 2\}$. Une description de la méthode est donnée dans l'algorithme 4.1. Dans cet algorithme, la notation *RL* désigne l'exploration du voisinage donné de la solution courante par une méthode de descente.

Algorithme 4.1 : Recherche Locale par Voisinage

Fonction RLV(S, $\mathcal{V}_{i,1}$, $\mathcal{V}_{i,2}$,$\mathcal{V}_{i,3}$)
répéter
  $S^1 \leftarrow \mathrm{RL}(S, \mathcal{V}_{i,1})$;
  $S^2 \leftarrow \mathrm{RL}(S^1, \mathcal{V}_{i,2})$;
  $S^3 \leftarrow \mathrm{RL}(S^2, \mathcal{V}_{i,3})$;
  $\Delta = f(S^3) - f(S)$;
  si $\Delta < 0$ **alors**
    $S \leftarrow S^3$;
jusqu'à $\Delta \geq 0$
retourner S

Nous pouvons maintenant décrire notre algorithme de DVV. Celui-ci prend en entrée une solution initiale S, et les deux ensembles de structures de voisinages décrites précédemment, notées $\mathcal{V}_1 = \{\mathcal{V}_{1,1}, \mathcal{V}_{1,2}, \mathcal{V}_{1,3}\}$ pour les voisinages d'insertion, et $\mathcal{V}_2 = \{\mathcal{V}_{2,1}, \mathcal{V}_{2,2}, \mathcal{V}_{2,3}\}$ pour les voisinages d'échange. Cet algorithme peut être vu comme une variante de la DVV classique. En effet, l'algorithme RLV décrit juste au dessus est une méthode déterministe utilisée comme sous-programme par notre DVV pour explorer à la fois les voisinages d'insertion et d'échange. L'algorithme de descente à voisinage variable s'arrête dès lors qu'il n'y a plus d'amélioration possible. Cette méthode est résumée dans l'algorithme 4.2.

4.4.2 Recherche à voisinage variable générale

Après avoir défini précisément la phase de descente, nous pouvons maintenant décrire notre algorithme de RVV Générale (RVVG). Cet algorithme utilise l'algorithme DVV précédent comme sous-programme, ce qui le différencie d'une approche classique de type RVV. Un élément important de la

Algorithme 4.2 : Descente à Voisinage Variable basée sur RLV

Fonction DVV(S, $\mathcal{V}_{set_1}$, $\mathcal{V}_{set_1}$)
répéter
  $S^1 \leftarrow \text{RLV}(S, \mathcal{V}_{1,1}, \mathcal{V}_{1,2}, \mathcal{V}_{1,3})$;
  $S^2 \leftarrow \text{RLV}(S^1, \mathcal{V}_{2,1}, \mathcal{V}_{2,2}, \mathcal{V}_{2,3})$;
  $\Delta = f(S^2) - f(S)$;
  si $\Delta < 0$ **alors**
    $S \leftarrow S^2$;
jusqu'à $\Delta \geq 0$
retourner S ;

RVVG est la phase de perturbation qui doit permettre de sortir des optima locaux durant la recherche. Dans notre cas, nous avons opté pour l'utilisation des voisinages d'insertion $V_{1,q}$ pour $q \in \{1, 2, 3\}$. Plus précisément, le voisinage utilisé pour la phase de perturbation est choisi aléatoirement parmi les trois voisinages de l'ensemble $\mathcal{V}_1$. Ensuite, nous appliquons la phase de perturbation en répétant k mouvements consécutifs dans le voisinage choisi (avec $1 \leq k \leq k_{max}$, où k_{max} est un paramètre). Cette utilisation répétée d'un voisinage donné dans la phase de perturbation est également une différence par rapport à un algorithme de RVV classique.

Nous décrivons notre RVVG dans l'algorithme 4.3. Cet algorithme prend en entrée une solution initiale, notée S. Celle-ci est générée aléatoirement. Deux autres paramètres sont en entrée :

- Le paramètre k_{max} détermine le nombre maximal de mouvements à effectuer à partir de la solution courante dans la phase de perturbation.
- Le paramètre CPU_{max} qui correspond à la condition d'arrêt de l'algorithme.

Dans l'algorithme 4.3, la notation $\mathcal{V}^{[k]}$ dénote k utilisations consécutives du voisinage $\mathcal{V}$. De plus, la notation $r\,[a, b]$ correspond à la génération d'un nombre entier aléatoirement dans l'intervalle $[a, b]$ selon une distribution uniforme.

Algorithme 4.3 : Recherche à Voisinage Variable Générale

Fonction GVNS(S, k_{max}, CPU_{max})
soit $\mathcal{V}_{set_1} = \{\mathcal{V}_{1,1}, \mathcal{V}_{1,2}, \mathcal{V}_{1,3}\}$;
soit $\mathcal{V}_{set_2} = \{\mathcal{V}_{2,1}, \mathcal{V}_{2,2}, \mathcal{V}_{2,3}\}$;
répéter
  $k \leftarrow 1$;
  tant que $k \leq k_{max}$ **faire**
    $i \leftarrow r\,[1, 3]$;
    $S^1 \leftarrow \mathcal{V}_{1,i}^{[k]}(S)$;
    $S^2 \leftarrow DVV(S^1, \mathcal{V}_{set_1}, \mathcal{V}_{set_2})$;
    $k \leftarrow k + 1$;
    si $f(S^2) < f(S)$ **alors**
      $S \leftarrow S^2$;
      $k \leftarrow 1$;
jusqu'à $CPU \leq CPU_{max}$
retourner S;

4.5 Résultats numériques

Pour valider cette approche, nous considérons les 14 instances proposées dans [98] et déjà utilisées dans le chapitre 3. Tous nos algorithmes sont implémentés en langage C++ et sont testés avec le même environnement que l'approche du chapitre précédent, à savoir une machine avec un processeur Pentium IV d'une fréquence de 3.4Ghz et 4Go de RAM.

Au niveau des paramètres de l'algorithme, nous avons fixé le temps maximum d'exécution de notre RVVG à 1000 secondes. La valeur de k_{max} est fixée à $4 \times T$. Finalement, l'algorithme est exécuté 10 fois sur chaque instance.

Nous reportons dans le Tableau 4.1 une synthèse des résultats obtenus. Les valeurs reportées dans ce tableau sont les résultats moyens. Nous donnons pour chaque instance : ses caractéristiques principales (nombre de fournisseurs et nombre de périodes) ; les résultats obtenus par les deux approches proposées dans le chapitre 3, et notées 2p-VND et 2p-VNS ; les résultats obtenus avec notre nouvelle approche. Pour être complet, la notation f^* désigne la valeur de la solution finale obtenue par un algorithme, alors que la valeur $\#v$ désigne le nombre de véhicules utilisés par cette solution. Notons que

nous ne reportons pas dans ce tableau les résultats de Moin et al. [98] car nous avons vu dans le chapitre précédent que les approches 2p-VND et 2p-VNS obtenaient de meilleurs résultats. Les résultats reportés dans le tableau

Inst	(n, τ)	2p-VND			2p-VNS			GVNS		
		f^*	$\#v$	Temps	f^*	$\#v$	Temps	f^*	$\#v$	Temps
S12T5	(12, 5)	1961,71	14	0,08	1961,71	14	0,10	**1961,71**	14	0,10
S12T10	(12, 10)	4002,85	29	0,21	4002,85	29	0,20	**4002,85**	29	0,20
S12T14	(12, 14)	5635,77	41	0,34	5635,77	41	0,30	**5635,77**	41	0,30
S20T5	(20, 5)	3045,08	23	0,34	**2861,26**	21	25,30	2880,65	21	11,33
S20T10	(20, 10)	6098,15	46	3,12	5944,72	43	102,30	**5919,38**	46	81,54
S20T14	(20, 14)	8589,36	65	9,63	8422,02	63	89,50	**8349,86**	61	75,92
S20T21	(20, 21)	12944,08	98	20,85	12700,27	94	222,70	**12627,87**	95	197,28
S50T5	(50, 5)	5144,23	46	4,97	5084,00	45	112,40	**5026,55**	45	48,49
S50T10	(50, 10)	10787,88	99	34,38	10694,60	96	75,30	**10544,43**	97	97,11
S50T14	(50, 14)	15399,82	139	64,90	15384,96	137	289,92	**15116,20**	136	249,06
S50T21	(50, 21)	23500,94	214	196,31	23497,40	211	373,20	**23331,83**	210	347,13
S98T5	(98, 5)	575790,06	59	100,14	**557287,97**	57	1768,30	557901,25	57	998,20
S98T10	(98, 10)	1160921,87	119	467,61	1119767,72	114	1263,80	**1114725,28**	114	985,67
S98T14	(98, 14)	1567904,75	160	5793,01	1577951,04	161	1210,30	**1562861,43**	158	986,83
Moyenne	-	242980,47	82,29	478,28	239371,16	80,42	395,26	237920,36	80,28	291,36

TABLEAU 4.1 – Résultats obtenus par la RVVG par rapport à l'existant

4.1 nous conduisent aux observations et conclusions suivantes :

- Le nouvel algorithme basé sur la recherche à voisinage variable obtient de meilleures solutions finales pour 12 et 13 instances (sur les 14 instances disponibles) par rapport aux approches 2p-VND et 2p-VNS, respectivement.
- Le temps de calcul nécessaire pour converger vers la meilleure solution est en général plus faible pour la nouvelle approche que pour les algorithmes 2p-VND et 2p-VNS. Nous remarquons ainsi que la limite fixée à 1000 secondes n'est, en général, pas un obstacle pour notre RVVG pour obtenir de bonnes solutions. La valeur moyenne de ces temps de calcul est donc également inférieure à celles de 2p-VNS et 2p-VND.
- Si nous regardons le nombre de véhicules utilisés, nous observons des valeurs sensiblement similaires entre l'algorithme 2p-VNS et la nouvelle RVVG. Rappelons que dans le problème considéré de routage et gestion de stock, nous ne minimisons pas explicitement le nombre de véhicules utilisés. Cela justifie que certaines solutions puissent correspondre à un coût total plus faible, même en utilisant plus de véhicules.

Afin de compléter l'analyse des résultats, et étant donné que notre approche

est exécutée 10 fois sur chaque instance, nous reportons dans le tableau 4.2 quelques informations complémentaires sur la déviation observée entre la valeur de la solution obtenue par un algorithme et la meilleure solution obtenue par l'ensemble des algorithmes que nous comparons. Nous considérons ici les deux algorithmes les plus performants sur l'ensemble des 14 instances, à savoir l'approche 2p-VNS du chapitre 3 et la RVVG générale présentée dans ce chapitre. Pour chaque instance, nous reportons la déviation minimale (colonne *min*) qui correspond donc au meilleur résultat obtenu par l'algorithme, la déviation moyenne (colonne *moy*) et la déviation maximale (colonne *max*). Pour un algorithme A, la déviation Δ_A (en %) est calculée selon la formule :

$$\Delta_A = ((f_A - f^*)/f^*) \times 100,$$

où f_A (respectivement, f^*) est la valeur de la solution finale retournée par l'algorithme A (resp., la valeur de la meilleure solution). Le tableau 4.2 montre

Inst	Δ_{2p-VNS}			Δ_{GVNS}		
	min	*moy*	*max*	*min*	*moy*	*max*
S12T5	0	0	0	0	0	0
S12T10	0	0	0	0	0	0
S12T14	0	0	0	0	0	0
S20T5	0	1,45	3,28	0,67	0,92	1,18
S12T10	0,43	0,87	1,29	0	0,29	0,61
S20T14	0,86	1,28	1,77	0	0,29	0,74
S20T21	0,57	1,28	1,75	0	0,24	0,66
S50T5	1,14	1,90	2,63	0	0,35	1,03
S50T10	1,42	2,26	2,82	0	0,69	1,22
S50T14	1,78	2,25	2,78	0	0,57	1,29
S50T21	0,71	1,16	1,66	0	0,61	1,49
S98T5	0	1,09	1,83	0,11	0,31	0,60
S98T10	0,45	1,71	4,09	0	0,27	0,51
S98T14	0,97	2,46	4,24	0,00	0,33	0,69
Moyenne	0,60	1,26	2,01	0,06	0,35	0,72

TABLEAU 4.2 – Déviations observées entre les deux algorithmes les plus performants

que la nouvelle approche RVVG obtient des résultats assez homogènes et de

bonne qualité. L'intervalle des valeurs moyennes obtenues, [0,06% ; 0,72%], est nettement inférieur à celui de 2p-VNS ([0,60% ; 2,01%]). En fait, les résultats obtenus pour l'instance S20T5 sont ceux qui sont les plus mauvais pour la RVVG par rapport à 2p-VNS. Nous pouvons toutefois remarquer que l'écart moyen obtenu par l'algorithme 2p-VNS pour cette instance n'est pas si bon et même supérieur à celui de la RVVG (1,45% contre 0,92%), ce qui relativise la contre performance de notre approche dans ce cas. Au final, les valeurs moyennes montrent bien qu'en général, la recherche à voisinage variable générale peut être considérée comme la meilleure approche actuelle pour ce problème, puisqu'elle domine les autres approches sur trois critères importants : la meilleure valeur obtenue, le temps nécessaire pour converger vers la meilleure solution, et la deviation moyenne observée.

4.6 Conclusion

Nous nous sommes intéressés dans ce chapitre à la proposition d'une nouvelle approche basée sur la recherche à voisinage variable pour traiter un problème d'élaboration de tournées avec prise en compte de la gestion des stocks chez le client. Ce problème, introduit dans le chapitre 3 de la thèse, est une variante multi-produits et multi-périodiques de l'IRP. Les résultats obtenus par l'approche décrite dans le chapitre 3 nous ont laissé penser qu'il était possible de faire mieux. Cette observation était en particulier liée à l'utilisation d'une heuristique pour la gestion des stocks, qui entraînait un coût non négligeable dans l'évaluation des différents voisinages proposés. Dans ce chapitre, nous avons donc adapté la représentation d'une solution du problème de manière à inclure implicitement la gestion des stocks dans les mouvements associés aux voisinages. Cette transformation a été possible en intégrant et en imposant une gestion des quantités à collecter à l'aide de variables discrètes. Il n'est ainsi plus possible de prélever chez un fournisseur une fraction de la demande de l'usine d'assemblage pour la période considérée, mais la politique mise en œuvre est une politique du "tout ou rien". Ce changement fondamental dans la gestion des stocks nous a permis de proposer de nouvelles structures de voisinages, que nous avons utilisées au sein d'une descente à voisinage variable et d'un nouvel algorithme de recherche à voisinage variable

générale.

Les résultats obtenus sur l'ensemble de 14 instances disponibles du problème montrent le bon comportement de notre approche. En effet, les solutions obtenues permettent un gain non négligeable par rapport aux autres approches existantes pour résoudre ce problème (l'algorithme génétique de Moin et al. [98] et les algorithmes proposés dans le chapitre 3), lorsque nous considérons le coût total combinant les coûts de transport et de stockage. De plus, ce gain n'est pas dû à une augmentation de la complexité temporelle de l'algorithme, et le compromis entre la qualité de la solution et le temps d'exécution requis est clairement un indicateur positif de la performance de notre approche.

Au niveau des perspectives, nous envisageons deux pistes principales. Nous pouvons tout d'abord continuer à travailler sur le même problème. Même si les résultats obtenus par nos différents algorithmes ont permis d'obtenir des solutions de bonne qualité sur les instances existantes, les solutions optimales de ces instances restent inconnues. Il serait donc intéressant de pouvoir trouver les solutions optimales, en combinant par exemple nos algorithmes avec des techniques de la programmation mathématique au sein d'approches hybrides. Même s'il n'est pas évident à première vue de pouvoir développer une méthode exacte pour ce problème à partir de nos algorithmes, il semble possible dans un premier temps d'essayer d'accélérer la convergence vers des solutions *presque* optimales. Il serait également intéressant de considérer notre approche sur une ou plusieurs autres variantes de l'IRP. En effet, la recherche à voisinage variable est une métaheuristique très souple pour le développement d'algorithmes efficaces. Il serait intéressant d'analyser quels éléments de nos algorithmes peuvent être réutilisés de façon efficace pour résoudre d'autres problèmes.

Chapitre 5

Recherche à voisinage variable pour le problème du voyageur de commerce avec tirant d'eau

5.1 Introduction

Nous avons abordé le problème du voyageur de commerce dans le chapitre 2 de ce mémoire. De nombreuses variantes et extensions ont été définies et peuvent être listées dans la littérature. Nous pouvons également trouver un grand nombre de méthodes exactes ou approchées pour résoudre ces problèmes. Dans ce chapitre, nous nous intéressons plus précisément à une variante proposée récemment dans Rakke et al. [51] dans le contexte du transport maritime. Ce problème est connu sous la terminologie anglaise "*Traveling Salesman Problem with Draft Limits*" (TSPDL), ce que nous pouvons traduire par "*problème du voyageur de commerce avec tirant d'eau*". Il s'agit de définir la meilleure route possible pour un navire contenant un ensemble de marchandises à livrer à un ensemble de ports, en s'assurant de respecter les contraintes liées au niveau de la mer dans chacun des ports. En fait, le tirant d'eau correspond à la hauteur de la partie immergée du navire, et cette hauteur varie en fonction de sa charge. Il faut donc s'assurer que le bateau puisse visiter les différents ports en fonction de son tirant d'eau.

Il est facile d'observer que ce problème est un problème NP-difficile, puisqu'il est dérivé du problème du voyageur de commerce [51]. Ce problème ayant été proposé récemment, nous ne pouvons recenser que peu de travaux lui étant consacrés. En fait, nous pouvons lister deux approches exactes. Dans [51], les auteurs ont proposé deux formulations mathématiques du problème et ont proposé un algorithme de type *Branch & Cut* (B&C) pour le résoudre.

Ils ont introduit des inégalités valides afin de réduire le nombre de nœuds dans l'arbre de recherche et ainsi réduire le temps de résolution nécessaire. Dans Battara et al. [11] les auteurs ont proposé trois formulations pour le problème. Ils ont ensuite développé un algorithme de type *Branch & Cut & Price* (B&C&P) pour le résoudre. Cette approche a permis de résoudre toutes les instances de la littérature de façon exacte. Cependant, les temps de calcul nécessaires pour arriver à ces résultats sont souvent importants, en particulier pour les instances de grande taille. Cela nous incite à proposer une métaheuristique pour résoudre ce problème en cherchant à obtenir des solutions presque optimales en un temps limité.

La suite de ce chapitre est organisée comme suit. Dans la section 5.2 nous présentons de façon plus précise le problème traité, et nous en donnons une formulation mathématique. Puis, nous présentons dans la section 5.3 notre algorithme, qui est une recherche à voisinage variable. Nous terminons avec les résultats numériques reportés dans la section 5.4 et les conclusions et perspectives de ce travail regroupées dans la section 5.5.

5.2 Description du problème

Le problème du voyageur de commerce avec tirant d'eau (TSPDL) peut être défini par un graphe orienté $G = (N, A)$ où $N = \{0, 1, \ldots, n\}$ représente l'ensemble des sommets, c'est-à-dire ici l'ensemble des ports à visiter, et $A = \{(i, j)|i, j \in N, i \neq j\}$ représente l'ensemble des arcs (i, j) entre les couples de sommets i et j. Le sommet 0 dans N est par convention le point de départ du navire. Nous notons dans la suite $N' = N\backslash 0$ l'ensemble des ports, sans prendre en considération le port de départ. Un coût de déplacement (strictement positif) est associé à chaque arc $(i, j) \in A$, et ce coût est noté c_{ij}. La demande de chaque port est connue et déterministe. En fait cette demande, notée d_i pour le port $i \in N'$, est entière et positive. La demande du port initial est supposée nulle. Nous connaissons également le tirant d'eau limite de chaque port, noté L_i. Pour savoir si un bateau respecte la contrainte du tirant d'eau maximum il est nécessaire de connaître le tirant d'eau du navire à l'entrée du port. Nous notons l_i cette valeur. Dans le TSPDL le tirant d'eau est en fait assimilé à la somme des charges présentes sur le

navire. Il est donc variable en fonction de l'avancement dans la route du navire. L'objectif du problème est de déterminer un cycle hamiltonien qui permette de minimiser le coût total de la tournée, et qui respecte la contrainte de tirant d'eau. Pour cela, il faut nécessairement avoir : $l_i \leq L_i, \forall i \in N'$.

Comme nous l'avons indiqué dans l'introduction, plusieurs formulations mathématiques ont été proposées pour ce problème. En particulier, nous pouvons considérer une première modélisation reportée dans [51] qui repose sur la modélisation de Gavish et Graves [47] pour le problème du voyageur de commerce asymétrique. Cette formulation repose sur la définition des variables suivantes :

- x_{ij} : variable égale à 1 si l'arc (i, j) est utilisé dans une solution optimale, 0 sinon.
- y_{ij} : variable qui représente la charge du bateau lors de la traversée de l'arc (i, j).

Le TSPDL peut alors être modélisé de la façon suivante :

$$\min \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{5.1}$$

$$\text{s.c.} \quad \sum_{i\in N, i\neq j} x_{ij} = 1 \quad \forall j \in N \tag{5.2}$$

$$\sum_{j\in N, j\neq i} x_{ij} = 1 \quad \forall i \in N \tag{5.3}$$

$$\sum_{i\in N} y_{ij} - \sum_{i\in N} y_{ji} = d_j \quad \forall j \in N' \tag{5.4}$$

$$\sum_{i\in N'} y_{0i} = \sum_{i\in N'} d_i \tag{5.5}$$

$$\sum_{i\in N'} y_{i0} = 0 \tag{5.6}$$

$$0 \leq y_{ij} \leq L_j x_{ij} \quad \forall (i, j) \in A \tag{5.7}$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \tag{5.8}$$

La fonction objective (5.1) minimise la somme des coûts des arcs visités par une tournée. Les contraintes (5.2) et (5.3) sont les contraintes de degré

des sommets. Les contraintes (5.4) permettent d'éviter la création de sous-tours, et elles assurent que la demande de chaque port est satisfaite. Les contraintes (5.5) et (5.6) assurent que le bateau parte du dépôt avec la totalité des demandes et qu'il revienne au port de départ, vide. La contrainte (5.7) assure le respect de la contrainte de tirant d'eau, alors que la contrainte (5.8) est la contrainte sur les variables binaires.

Dans [51], les auteurs ont proposé plusieurs inégalités valides pour renforcer le modèle précédent. Ces inégalités valides sont utilisées au sein d'un algorithme de B&C. En particulier, ils génèrent des contraintes d'élimination de sous-tours, ainsi que des contraintes imposant des bornes inférieures et supérieures sur les variables du modèle. D'autres inégalités valides sont obtenues à partir des contraintes sur le tirant d'eau et à partir de la demande des différents ports. Les expériences numériques reportées dans l'article montrent que les modèles et techniques proposés permettent de résoudre une partie non négligeable des instances dérivées de la TSP-Lib (198 instances sur les 240 en fixant le temps maximum de résolution à 10000 secondes).

5.3 Recherche à voisinage variable générale pour le TSPDL

Nous avons opté pour la recherche à voisinage variable pour résoudre le TSPDL [127]. Cette méthode a déjà fait ses preuves pour résoudre de nombreux problèmes de tournées, et nous l'avons également utilisée dans les chapitres précédents. Nous commençons cette partie par une description de la méthode utilisée pour générer une solution initiale réalisable du problème. Nous abordons ensuite les différentes structures de voisinages utilisées, et la description d'un algorithme de descente à voisinage variable. Nous terminons cette partie par la description de la recherche à voisinage variable générale que nous proposons.

5.3.1 Solution initiale

Il est facile d'observer qu'une solution du problème du voyageur de commerce n'est pas nécessairement une solution réalisable du TSPDL, en raison

de la contrainte du tirant d'eau. Comme pour le TSP, une solution du TSPDL est une permutation des ports, de manière à représenter l'ordre de visite des différents sommets du graphe. En fait, Rakke et al. ont même montré dans [51] que toute instance du TSPDL n'admet pas nécessairement une solution réalisable.

Proposition : Supposons que tous les ports soient triés dans l'ordre décroissant des niveaux d'eaux, de sorte à avoir : $L_i \geq L_{i+1}, \forall i \in 1, \ldots, n-1$. Alors, une instance du TSPDL admet une solution réalisable si et seulement si $l_i \leq L_i$ pour chaque $i \in N'$. Si la solution $\pi = (0, \ldots, n)$ n'est pas réalisable alors, aucune autre permutation des ports n'est réalisable.

La preuve de la proposition est disponible dans [51].

Nous pouvons utiliser cette propriété du problème pour générer facilement une solution réalisable (et en même temps s'assurer que l'instance est faisable). Il suffit pour cela de trier l'ensemble des ports dans l'ordre décroissant des niveaux d'eaux, et de considérer cette permutation comme étant une solution initiale du problème.

5.3.2 Structures de voisinages et vérification de la faisabilité

Le fait de définir une solution du TSPDL comme une permutation des ports nous permet d'utiliser toute structure de voisinage applicable pour le TSP à notre problème. Il faut seulement ajouter la prise en compte de la contrainte de tirant d'eau pour s'assurer de la faisabilité de la solution. Cette tâche peut être effectuée à l'aide d'une fonction de *vérification* qui nous permet d'exclure du voisinage considéré toute solution ne satisfaisant pas cette contrainte.

Différents mouvements ont été définis et appliqués au cas du TSP. Parmi les plus répandus figurent le mouvement 2-opt et le OR-opt. Nous définissons plusieurs structures de voisinages reposant sur ces deux mouvements de base.

La première famille de structure de voisinages est associée au mouvement 2-opt. Ce mouvement consiste à supprimer deux arcs de la solution courante, puis à reconnecter les sous-tours obtenus pour former un nouveau circuit. Plus formellement, en notant $\pi = (0, \pi_1, \ldots, \pi_i, \ldots, \pi_j, \ldots, \pi_n)$ la solution cou-

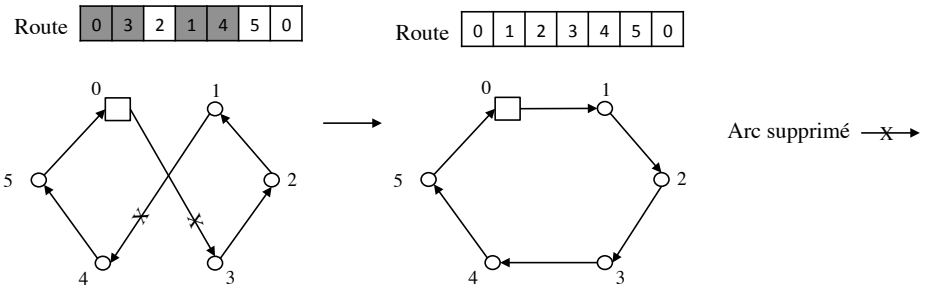

FIGURE 5.1 – Illustration du 2-opt

rante, une solution voisine est $\pi' = (0, \pi_1, \ldots, \pi_i, \pi_j, \ldots, \pi_{i+1}, \pi_{j+1}, \ldots, \pi_n)$. Nous illustrons le mouvement 2-opt dans la figure 5.1. Dans cette figure, les arcs $(0, 3)$ et $(1, 4)$ sont supprimés, puis les arcs $(0, 1)$ et $(3, 4)$ sont ajoutés pour reconstruire un circuit. Nous pouvons observer qu'au niveau de la représentation de la solution ce mouvement revient à permuter deux sommets.

Nous utilisons également une variante de l'algorithme 2-opt appelée 1-opt. C'est en fait un cas particulier du 2-opt lorsque nous avons $\pi_j = \pi_{i+2}$ dans la solution courante. Dans ce cas, la solution modifiée est obtenue en supprimant les arcs $(\pi_i, \pi - i + 1)$ et (π_j, π_{j+1}), en ajoutant les arcs (π_i, π_j) et (π_{i+1}, π_{j+1}), comme dans le cas du 2-opt, et enfin en inversant l'arc (π_{i+1}, π_j) qui devient donc (π_j, π_{i+1}). Formellement, ce mouvement se résume comme suit : soit $\pi = (0, \pi_1, \ldots, \pi_i, \pi_{i+1}, \pi_j, \pi_{j+1}, \ldots, \pi_n)$, alors $\pi' = (0, \pi_1, \ldots, \pi_i, \pi_j, \pi_{i+1}, \pi_{j+1}, \ldots, \pi_n)$.

La deuxième famille de structure de voisinages repose sur les mouvements induits par l'utilisation de trois variantes de l'algorithme OR-opt. Celui-ci consiste à déplacer une suite de sommets du graphe visités successivement dans la solution courante vers une autre position. La suite de sommets à déplacer est appelée chaîne. Si cette chaîne contient k sommets (k ports dans notre cas), l'algorithme sera appelé OR-opt-k. Comme nous travaillons dans un cas orienté, nous pouvons observer que le mouvement de la chaîne peut être fait vers l'avant ou vers l'arrière dans la route. Ceci nous donne donc deux variantes de ce mouvement que nous noterons : OR-opt-k avant et OR-opt-k arrière. Si $k = 2$, ce mouvement peut être introduit plus formellement comme suit : si la solution courante est $\pi =$

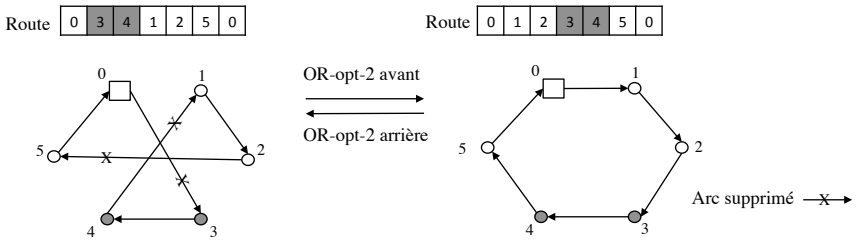

FIGURE 5.2 – Illustration du OR-opt-2

$(0, \pi_1, \ldots, \pi_i, \pi_{i+1}, \pi_j, \pi_{j+1}, \pi_k, \pi_{k+1}, \ldots, \pi_n)$, alors une solution voisine sera de la forme $\pi' = (0, \pi_1, \ldots, \pi_i, \pi_{j+1}, \pi_k, \pi_{i+1}, \pi_j, \pi_{k+1}, \ldots, \pi_n)$. La figure 5.2 illustre ce mouvement d'une chaîne contenant 2 ports (les ports 3 et 4) vers l'arrière et vers l'avant.

À partir de ces deux familles, nous définissons l'ensemble des voisinages qui seront utilisés dans notre approche comme suit :

- $\mathcal{V}_1$: mouvement 1-opt.
- $\mathcal{V}_2$: mouvement 2-opt.
- $\mathcal{V}_3$: mouvement OR-opt-3 arrière.
- $\mathcal{V}_4$: mouvement OR-opt-3 avant.
- $\mathcal{V}_5$: mouvement OR-opt-2 arrière.
- $\mathcal{V}_6$: mouvement OR-opt-2 avant.
- $\mathcal{V}_7$: mouvement OR-opt-1 arrière.
- $\mathcal{V}_8$: mouvement OR-opt-1 avant.

Nous pouvons remarquer que les voisinages $\mathcal{V}_7$ et $\mathcal{V}_8$ correspondent à un simple mouvement d'insertion d'un port dans une route. Rappelons également qu'une permutation est réalisable pour le TSPDL si la contrainte de tirant d'eau est satisfaite. Nous proposons une fonction de vérification efficace pour tester la faisabilité des solutions voisines à moindre coût, car il n'est pas nécessaire de vérifier cette contrainte pour tous les sommets visités d'une solution voisine.

Considérons tout d'abord le cas du voisinage 2-opt. Soit une solution courante $\pi = (0, \pi_1, \ldots, \pi_i, \ldots, \pi_j, \ldots, \pi_n)$. L'application de ce mouvement sur les arcs (π_i, π_{i+1}) et (π_j, π_{j+1}) conduit à la solution voisine $\pi' = (0, \pi_1, \ldots,$

$\pi_i, \pi_j, \ldots, \pi_{i+1}, \pi_{j+1}, \ldots, \pi_n)$. En comparant ces deux solutions voisines, nous observons que l'ordre de visite des ports est inchangé du port initial 0 jusqu'au port π_i. Cela implique que la contrainte du tirant d'eau est respectée de la position 1 jusqu'à la position i. De plus, nous remarquons que la charge du bateau est la même sur la portion allant du port π_{j+1} au port π_n, ce qui implique que la contrainte du tirant d'eau est respectée de la position $j+1$ jusqu'à la position n. Autrement dit, nous avons $l'_k = l_k$ pour $k \in \{1, \ldots, i\} \cup \{j+1, \ldots, n\}$. Ainsi nous savons que l'évaluation du mouvement 2-opt ne peut affecter la faisabilité de la solution que sur la partie impactée de la solution, entre les positions $i+1$ et j.

Considérons maintenant le voisinage OR-opt-k avant. Soit la solution courante $\pi = (0, \pi_1, \ldots, \pi_i, \ldots, \pi_{i+k-1}, \ldots, \pi_l, \ldots, \pi_n)$. Supposons que nous appliquions un mouvement OR-opt-k sur cette solution. Ce mouvement consiste à déplacer k ports consécutifs $\pi_i, \ldots, \pi_{i+k-1}$ après un certain port, π_l. La solution résultante de ce mouvement sera alors de la forme $\pi' = (0, \pi_1, \ldots, \pi_{i-1}, \pi_{i+k}, \ldots, \pi_l, \pi_i, \ldots, \pi_{i+k-1}, \pi_{l+1}, \ldots, \pi_n)$. Nous observons que, comme dans le cas précédent, la première partie de la position 1 à la position $i-1$ reste inchangée, et donc que la contrainte de tirant d'eau reste réalisable pour cette partie. De même, la partie finale de la position $l+1$ à la position n est inchangée également. Donc, la contrainte reste réalisable de la position $i+k$ à la position l. Ainsi, il suffit de vérifier la faisabilité de la solution voisine juste pour les positions entre i et $i+k-1$.

Supposons maintenant que nous appliquions le mouvement OR-opt-k arrière sur la solution courante. Nous pouvons appliquer un raisonnement similaire. En effet, à partir de la solution courante $\pi = (0, \pi_1, \ldots, \pi_l, \ldots, \pi_i, \ldots, \pi_{i+k-1}, \ldots, \pi_n)$, il s'agit ici d'avancer une suite de k ports consécutifs $\pi_i, \ldots, \pi_{i+k-1}$ après un certain port, π_l. La solution résultante de ce mouvement sera alors sous la forme $\pi' = (0, \pi_1, \ldots, \pi_l, \pi_i, \ldots, \pi_{i+k-1}, \pi_{l+1}, \ldots, \pi_{i-1}, \pi_{i+k}, \ldots, \pi_n)$. Nous voyons à nouveau que la première partie de la solution reste inchangée et donc satisfait la contrainte entre le port initial 0 et le port π_l. De même, la partie finale de π_{i+k} à π_n est inchangée. Donc, la partie allant de π_{l+1} à π_{i-1} ne peut pas non plus violer la contrainte car la charge de la

solution voisine n'est pas modifiée par rapport à la solution courante. Ainsi, il suffit donc de vérifier la faisabilité de la solution voisine pour les positions entre i et $i + k - 1$.

5.3.3 Descente à voisinage variable

Les différentes structures de voisinages définies précédemment sont utilisées au sein de plusieurs algorithmes basés sur la recherche à voisinage variable. Dans cette section, nous décrivons en particulier deux algorithmes de descente à voisinage variable pour notre problème.

Dans le premier algorithme, noté `VND-1`, nous utilisons l'ensemble des 8 voisinages décrits dans la section précédente, dans l'ordre dans lequel nous les avons présentés : 1-opt ($\mathcal{V}_1$), 2-opt ($\mathcal{V}_2$), OR-opt-3 arrière ($\mathcal{V}_3$), OR-opt-3 avant ($\mathcal{V}_4$), OR-opt-2 arrière ($\mathcal{V}_5$), OR-opt-2 avant ($\mathcal{V}_6$), OR-opt-1 arrière ($\mathcal{V}_7$) et OR-opt-1 avant ($\mathcal{V}_8$). En notant π une solution du problème et $f(\pi)$ son coût, nous pouvons donner une description algorithmique de la méthode dans l'algorithme 5.1.

Algorithme 5.1 : Algorithme VND-1

```
Fonction VND-1(π)
k ← 1;
répéter
    π' ← RL(π, V_k);
    si f(π') < f(π) alors
        π ← π';
        k ← 1;
    sinon
        k ← k + 1;
jusqu'à k > 8;
retourner π;
```

Dans cet algorithme, $\mathrm{RL}(\pi, \mathcal{V}_k)$ désigne l'appel à une méthode de recherche locale au sein de laquelle nous explorons le voisinage $\mathcal{V}_k$ de la solution courante π, pour k donné.

Le deuxième algorithme de descente à voisinage variable, `VND-2`, repose sur l'utilisation de 6 voisinages : 1-opt ($\mathcal{V}_1$), OR-opt-2 arrière ($\mathcal{V}_5$), OR-opt-2

avant ($\mathcal{V}_6$), OR-opt-1 arrière ($\mathcal{V}_7$), OR-opt-1 avant ($\mathcal{V}_8$) et 2-opt ($\mathcal{V}_2$), dans cet ordre. L'algorithme 5.2 résume le principe de la méthode.

Algorithme 5.2 : Algorithme VND-2

```
Fonction VND-2(π)
Soit V' =< V_1, V_5, V_6, V_7, V_2 >;
répéter
    k ← 1;
    améliore ← faux;
    tant que k < 6 faire
        π' ← RL(π, V'_k);
        si f(π') < f(π) alors
            π ← π';
            améliore ← vrai;
        sinon
            k ← k + 1;
jusqu'à améliore = faux;
retourner π;
```

Les deux algorithme `VND-1` et `VND-2` sont similaires, même s'ils utilisent différentes structures de voisinages. En pratique, ils se différencient également par la méthode utilisée pour l'exploration des voisinages. En fait, nous utilisons la stratégie d'exploration du premier voisin améliorant dans l'algorithme `VND-1`. Cette stratégie permet de ne pas trop alourdir le processus, puisque l'algorithme utilise les 8 structures de voisinages que nous avons définies. A chaque fois qu'un voisin améliorant est trouvé dans un voisinage, le processus recommence avec le premier voisinage. L'algorithme s'arrête lorsqu'il n'est plus possible de trouver un voisin améliorant de la solution courante et que tous les voisinages ont été explorés. La stratégie d'exploration utilisée dans l'algorithme `VND-2` est différente. Ici, l'algorithme considère le même voisinage tant qu'il est possible de trouver un voisin améliorant dans ce voisinage. Le processus recommence avec le premier voisinage lorsque tous les voisinages ont été utilisés et qu'il y a eut au moins une amélioration.

5.3.4 Recherche à voisinage variable générale

Pour résoudre efficacement le TSPDL, nous proposons deux méthodes de recherche à voisinage variable générale. La première, notée `GVNS-1`, utilise l'algorithme `VND-1` comme procédé d'intensification (de recherche locale), alors que la seconde, `GVNS-2`, utilise l'algorithme `VND-2`. Les deux méthodes sont décrites dans l'algorithme 5.3.

Les deux algorithmes ont trois paramètres en entrée : k_{max} qui représentent le nombre maximum de perturbations successives à appliquer à la solution courante au sein de l'algorithme, t_{max} qui correspond au temps total alloué pour la recherche, et π qui est une solution initiale réalisable du problème (générée selon la procédure décrite dans la section 5.3.1). Dans les deux cas, l'algorithme va répéter la perturbation de la solution courante k fois, avant d'appeler la descente à voisinage variable pour essayer d'améliorer la solution obtenue. Si cela est possible alors le processus est relancé avec $k = 1$, et l'ensemble de la méthode est répété tant que le temps de calcul n'a pas atteint la limite fixée (notée t_{max}).

Un élément important dans le développement de la recherche à voisinage variable générale est la phase de perturbation. Dans le cas du TSPDL, nous avons décidé d'utiliser le même algorithme de perturbation pour les deux approches, `GVNS-1` et `GVNS-2`. La perturbation utilise le voisinage OR-opt-1 et consiste à appliquer k mouvements consécutifs à partir de la solution courante π. En fait, la méthode consiste à choisir aléatoirement un port dans la solution courante, et à le déplacer avant ou après un autre port choisi également aléatoirement dans la solution courante, selon la position des deux ports. Ce procédé est résumé dans l'algorithme 5.4. Dans cet algorithme, la notation $\mathcal{V}_k(\pi, i, j)$ fait référence à l'utilisation du voisinage $\mathcal{V}_k$ sur la solution courante π en utilisant les ports aux positions i et j. Si le port i se situe avant le port j dans la solution courante, alors nous choisissons une solution π' aléatoirement en appliquant le mouvement OR-opt-1 avant pour déplacer de façon aléatoire le port i après le port j. Nous appliquons réciproquement le mouvement OR-opt-1 arrière si le port i se trouve après le port j initialement.

Algorithme 5.3 : Algorithme de recherche à voisinage variable générale pour le TSPDL

Fonction `GVNS-m(`k_{max}`,` t_{max} π`)`
// $m \in \{1, 2\}$ selon la méthode
répéter
  $k \leftarrow 1$;
  tant que $k \leq k_{max}$ **faire**
    $\pi' \leftarrow$ `Perturber`(π, k) ;
    $\pi'' \leftarrow$ `VND-`$m(\pi')$;
    $k \leftarrow k + 1$;
    si $f(\pi'') < f(\pi)$ **alors**
      $\pi \leftarrow \pi''$;
      $k \leftarrow 1$;
jusqu'à `Cpu` $> t_{max}$
retourner π ;

Algorithme 5.4 : Procédure de perturbation

Fonction `Perturber(`π`,` k`)`
pour *i de 1 à k* **faire**
  Soient (i, j) deux indices choisis aléatoirement dans π ;
  si $i < j$ **alors**
    Choisir aléatoirement $\pi' \in \mathcal{V}_8(\pi, i, j)$;
  sinon
    Choisir aléatoirement $\pi' \in \mathcal{V}_7(\pi, i, j)$;
  $\pi \leftarrow \pi'$;
retourner π ;

5.4 Expériences numériques

Pour évaluer les deux algorithmes proposés dans ce chapitre, nous avons utilisé les 240 instances proposées par Rakke et al. [51]. Ces instances ont été générées à partir de 8 instances initialement proposées pour le TSP : burma14, ulysses16, ulysses22, fri26, bayg29, gr17, gr22 et gr48 [113]. Les auteurs ont en fait utilisé plusieurs pourcentages de ports affectés par les contraintes de tirant d'eau : 10%, 25% et 50%, et ils ont généré 10 nouvelles instances pour chacun de ces pourcentages à partir des 8 instances du TSP. Chaque instance est caractérisée par le nombre de ports, la demande de chaque port, le niveau d'eau dans chaque port, et la matrice des coûts c_{ij}. Tous les instances sont disponibles sur Internet : `http://jgr.no/tspdl`.

Comme nous le verrons un peu plus loin, les résultats obtenus par nos algorithmes sur ces instances sont, en général, de très bonne qualité. En particulier, le temps d'exécution nécessaire pour converger vers une solution optimale peut être considéré comme faible. Devant cette constatation, nous avons décidé de générer 60 nouvelles instances de plus grande taille, en nous basant également sur des instances proposées pour le TSP (KroA100, KroA200 et pcb442). Pour ces nouvelles instances, nous avons fixé le pourcentage de ports affectés par la contrainte de tirant d'eau à 50 et 75%. Ces nouvelles instances sont accessibles sur l'adresse suivante : `http://www.mi.sanu.ac.rs/~nenad/TSPDL/`.

Quelques tests préliminaires nous ont permis de fixer les paramètres de nos deux algorithmes de recherche à voisinage variable générale :

- $k_{max} = 30$ (nombre maximal de la perturbations successives).
- $t_{max} = 100$ secondes.

Le fait de fixer la valeur de t_{max} à 100 secondes traduit notre volonté d'évaluer l'efficacité de nos métaheuristiques en termes de faciliter à converger vers les solutions optimales, ou très proches des solutions optimales, en un temps raisonnable.

Les expériences numériques que nous avons réalisées ont montré que les deux approches, `GVNS-1` et `GVNS-2`, visitent une solution optimale en moins d'une seconde pour l'ensemble des instances générées à partir de burma14, ulysses16, ulysses22, fri26, bayg29, gr17 et gr21, ce qui représente 210 des 240

Inst	B&C&P		GVNS-1		GVNS-2	
	v^*	CPU	$\overline{v}$	CPU	$\overline{v}$	CPU
Burma_14_10	3386,70	0,52	3386,70	0,00	3386,70	0,00
Burma_14_25	3596,80	0,37	3596,80	0,00	3596,80	0,00
Burma_14_50	3862,30	0,35	3862,30	0,00	3862,30	0,00
Ulysse_16_10	6868,20	50,34	6868,20	0,00	6868,20	0,00
Ulysse_16_25	7165,40	18,58	7165,40	0,00	7165,40	0,00
Ulysse_16_50	7590,30	4,60	7590,30	0,00	7590,30	0,00
Ulysse_22_10	7087,60	32,02	7087,60	0,04	7087,60	0,00
Ulysse_22_25	7508,70	24,61	7508,70	0,01	7508,70	0,00
Ulysse_22_50	8425,60	26,55	8425,60	0,04	8425,60	0,03
fri26_25	963,80	12,37	963,80	0,02	963,80	0,00
fri26_50	1104,70	16,43	1104,70	0,06	1104,70	0,05
fri26_10	1178,70	16,26	1178,70	0,01	1178,70	0,01
bayg29_10	1713,60	11,99	1713,60	0,00	1713,60	0,02
bayg29_25	1792,60	11,05	1792,60	0,14	1792,60	0,02
bayg29_50	2091,00	13,38	2091,00	0,01	2091,00	0,05
gr17_10	2150,30	25,12	2150,30	0,00	2150,30	0,00
gr17_25	2237,70	11,11	2237,70	0,00	2237,70	0,00
gr17_50	2710,30	5,54	2710,30	0,00	2710,30	0,00
gr21_10	2833,60	11,63	2833,60	0,00	2833,60	0,01
gr21_25	2962,60	11,44	2962,60	0,00	2962,60	0,00
gr21_50	3738,10	10,32	3738,10	0,01	3738,10	0,00
Moyenne	3855,65	14,98	3855,65	0,02	3855,65	0,01

TABLEAU 5.1 – Resumé des résultats obtenus sur les instances de petite taille

instances. Le tableau 5.1 donne une synthèse des résultats obtenus sur ces instances. Dans ce tableau, nous reportons pour chaque classe d'instances la moyenne des valeurs optimales obtenues par la méthode de B&C&P et reportées dans Battara et al. [11] dans la colonne v^*, suivi de la moyenne des temps d'exécution dans la colonne CPU. Viennent ensuite les résultats moyens de nos deux approches, avec pour chacune la moyenne des valeurs des solutions obtenues (colonne $\overline{v}$) et la moyenne des temps d'exécution.

Nous présentons ensuite les résultats obtenus sur les 30 instances dérivées de gr48 dans le tableau 5.2. Nous reportons pour chaque instance la valeur

optimale obtenue par l'algorithme de B&C&P [11] dans la colonne v^*. Nous donnons ensuite la valeur de la solution finale obtenue par `GVNS-1` et `GVNS-2`, ainsi que le temps nécessaire pour converger vers cette solution (nous ne considérons qu'une ne seule exécution par instance).

Les tableaux 5.1 et 5.2 nous permettent de tirer les conclusions suivantes :

- L'algorithme `GVNS-1` visite pratiquement toutes les solutions optimales sur l'ensemble des 240 instances. En fait, il n'obtient pas de solution optimale uniquement pour deux instances : gr48_25_2 et gr48_25_6 (valeurs en gras dans le tableau 5.2). Nous pouvons remarquer que même pour ces deux instances le coût de la solution finale est très proche du coût optimal.
- Le temps de calcul nécessaire à l'algorithme `GVNS-1` pour converger vers la solution finale est très intéressant, puisque généralement en dessous de 10 secondes. Seules quelques instances font exception, en particulier l'instance gr48_25_7 pour laquelle l'algorithme n'a obtenu la meilleure solution qu'à la fin du temps de calcul alloué.
- L'algorithme `GVNS-2` est capable d'obtenir toutes les valeurs optimales, sur l'ensemble des 240 instances. Le temps de calcul moyen (environ 8,5 secondes) est légèrement supérieur à celui nécessaire à `GVNS1`, mais reste tout de même très raisonnable en comparaison avec le temps de calcul de la méthode exacte (un peu plus de 800 secondes en moyenne).

Nous nous intéressons aux résultats obtenus par nos approches sur les nouvelles instances de plus grande taille dans les tableaux 5.3 à 5.5. Pour évaluer la qualité des solutions obtenues par nos algorithmes de recherche à voisinage variable générale, nous utilisons une borne inférieure. Cette borne correspond à la valeur optimale de l'instance considérée lorsque nous oublions les contraintes de tirant d'eau, ce qui revient à résoudre l'instance de TSP correspondante. Cette valeur est reportée dans les colonnes $\underline{v}$ des tableaux 5.3 à 5.5. Nous reportons ensuite les résultats obtenus par `GVNS-1` et `GVNS-2`, avec à chaque fois la valeur de la borne supérieure obtenue ($\overline{v}$) et le temps nécessaire pour converger vers cette borne en secondes (CPU). Les deux dernières colonnes donnent la déviation observée entre la solution de chaque algorithme et la valeur de la borne inférieure, calculée pour l'algorithme A par : $\Delta - A = ((\overline{v}_A - \underline{v})/\underline{v}) \times 100$, où $\overline{v}_A$ désigne la valeur de la solution

Inst.	B&C&P		GVNS-1		GVNS-2	
	v^*	CPU	$\overline{v}$	CPU	$\overline{v}$	CPU
gr48_10_1	5046	813,29	5046	0,01	5046	0,00
gr48_10_2	5542	341,27	5542	0,01	5542	0,10
gr48_10_3	5300	3314,02	5300	0,03	5300	0,06
gr48_10_4	5293	7057,36	5293	1,32	5293	0,45
gr48_10_5	5679	496,29	5679	0,21	5679	30,27
gr48_10_6	5610	41,04	5610	0,39	5610	0,42
gr48_10_7	5063	22,79	5063	0,00	5063	1,20
gr48_10_8	5103	188,77	5103	0,01	5103	1,15
gr48_10_9	5153	892,87	5153	0,00	5153	0,17
gr48_10_10	5055	359,47	5055	0,00	5055	0,01
gr48_25_1	5524	1073,43	5524	0,46	5524	0,07
gr48_25_2	5895	361,90	**5901**	0,35	5895	0,03
gr48_25_3	5754	47,85	5754	0,04	5754	0,56
gr48_25_4	5588	126,05	5588	2,34	5588	45,87
gr48_25_5	6159	1793,96	6159	7,62	6159	4,55
gr48_25_6	5760	3053,47	**5777**	10,73	5760	23,32
gr48_25_7	5955	172,08	5955	101,04	5955	41,29
gr48_25_8	5562	1070,86	5562	2,29	5562	0,43
gr48_25_9	5792	184,24	5792	2,46	5792	6,41
gr48_25_10	6014	730,82	6014	5,08	6014	13,97
gr48_50_1	6096	157,47	6096	0,59	6096	0,64
gr48_50_2	6629	21,44	6629	31,53	6629	12,30
gr48_50_3	5896	132,45	5896	3,94	5896	13,79
gr48_50_4	6404	20,75	6404	0,07	6404	0,49
gr48_50_5	6617	19,41	6617	0,06	6617	0,03
gr48_50_6	8533	66,09	8533	5,46	8533	0,48
gr48_50_7	6166	1520,10	6166	0,60	6166	19,59
gr48_50_8	6535	20,93	6535	0,18	6535	27,76
gr48_50_9	7150	42,73	7150	0,25	7150	7,92
gr48_50_10	6331	112,13	6331	17,04	6331	0,04
Moyenne	5906,80	808,51	5907,57	6,66	5906,80	8,45

TABLEAU 5.2 – Comparaison entre B&C&P [11], `GVNS1` et `GVNS2` sur les instances avec 48 ports

Inst.	$\underline{v}$	GVNS-1		GVNS-2		Déviation	
		$\overline{v}$	CPU	$\overline{v}$	CPU	Δ_{GVNS-1}	Δ_{GVNS-2}
kroA100_50_1	21282	21305,00	7,07	21282	78,60	0,11	0,00
kroA100_50_2	21282	21315,00	1,33	21282	2,93	0,16	0,00
kroA100_50_3	21282	21305,00	8,50	21282	27,14	0,11	0,00
kroA100_50_4	21282	21359,00	34,41	21343	3,74	0,36	0,29
kroA100_50_5	21282	21305,00	20,92	21295	75,09	0,11	0,06
kroA100_50_6	21282	21369,00	7,28	21282	14,73	0,41	0,00
kroA100_50_7	21282	21282,00	3,70	21305	13,45	0,00	0,11
kroA100_50_8	21282	21343,00	14,08	21282	12,54	0,29	0,00
kroA100_50_9	21282	21391,00	67,23	21305	18,84	0,51	0,11
kroA100_50_10	21282	21282,00	4,65	21282	31,53	0,00	0,00
Moyenne	21282	21325,60	16,92	21294	27,86	0,20	0,06
kroA100_75_1	21282	21415,00	7,31	21305	0,51	0,62	0,11
kroA100_75_2	21282	21295,00	43,80	21282	13,31	0,06	0,00
kroA100_75_3	21282	21282,00	54,89	21305	90,55	0,00	0,11
kroA100_75_4	21282	21282,00	37,60	21305	8,77	0,00	0,11
kroA100_75_5	21282	21305,00	25,64	21343	27,09	0,11	0,29
kroA100_75_6	21282	21685,00	62,45	21305	12,78	1,89	0,11
kroA100_75_7	21282	21305,00	17,25	21343	49,67	0,11	0,29
kroA100_75_8	21282	21282,00	8,64	21282	41,47	0,00	0,00
kroA100_75_9	21282	21305,00	2,84	21282	66,59	0,11	0,00
kroA100_75_10	21282	21438,00	33,72	21282	16,00	0,73	0,00
Moyenne	21282	21359,40	29,41	21303,4	32,67	0,36	0,10

TABLEAU 5.3 – Comparaison entre la borne inférieure et les valeurs de GVNS-1 et GVNS-2 sur les instances avec 100 ports

retournée par l'algorithme A.

Les résultats reportés dans les tableaux 5.3 à 5.5 confirment les conclusions tirées des résultats observés sur les instances de la littérature. En effet, nous pouvons remarquer que l'algorithme GVNS-2 obtient, en moyenne, des solutions finales de meilleure qualité que l'algorithme GVNS-1, avec en contre partie un temps d'exécution un peu supérieur. D'autres éléments peuvent également être observés, en particulier :

– Il arrive que la déviation des solutions retournées par nos algorithmes soit nulle pour des instances comportant 100 ports (5 instances pour

Inst.	$\underline{v}$	GVNS-1		GVNS-2		Déviation	
		$\overline{v}$	CPU	$\overline{v}$	CPU	Δ_{GVNS-1}	Δ_{GVNS-2}
kroA200_50_1	29368	30344	38,75	31017	80,63	3,32	5,61
kroA200_50_2	29368	30524	17,99	30398	89,13	3,94	3,51
kroA200_50_3	29368	30509	86,98	29911	98,43	3,89	1,85
kroA200_50_4	29368	31009	52,88	30691	78,19	5,59	4,50
kroA200_50_5	29368	30863	75,51	30338	97,89	5,09	3,30
kroA200_50_6	29368	30501	75,61	31052	87,17	3,86	5,73
kroA200_50_7	29368	31555	82,10	30114	60,41	7,45	2,54
kroA200_50_8	29368	31040	95,98	30132	51,55	5,69	2,60
kroA200_50_9	29368	30420	55,65	30749	95,12	3,58	4,70
kroA200_50_10	29368	31928	78,52	32250	36,61	8,72	9,81
Moyenne	29368	30869,3	66,00	30665,2	77,51	5,11	4,42
kroA200_75_1	29368	30623	84,94	31017	65,74	4,27	5,61
kroA200_75_2	29368	31759	79,13	30806	91,90	8,14	4,90
kroA200_75_3	29368	30615	61,16	31530	96,31	4,25	7,36
kroA200_75_4	29368	30740	93,36	29942	96,19	4,67	1,95
kroA200_75_5	29368	37038	99,28	29990	99,54	26,12	2,12
kroA200_75_6	29368	32160	83,22	31647	26,38	9,51	7,76
kroA200_75_7	29368	32046	90,24	31829	99,97	9,12	8,38
kroA200_75_8	29368	31041	68,58	31511	75,68	5,70	7,30
kroA200_75_9	29368	30855	97,73	30690	91,92	5,06	4,50
kroA200_75_10	29368	33538	36,10	29999	26,88	14,20	2,15
Moyenne	29368	32041,5	79,37	30896,1	77,05	9,10	5,20

TABLEAU 5.4 – Comparaison entre la borne inférieure et les valeurs de GVNS-1 et GVNS-2 sur les instances avec 200 ports

Inst.	$\underline{v}$	GVNS-1		GVNS-2		Déviation	
		$\overline{v}$	CPU	$\overline{v}$	CPU	Δ_{GVNS-1}	Δ_{GVNS-2}
pcb442_50_1	50778	60284	87,83	59296	98,34	18,72	16,77
pcb442_50_2	50778	64902	78,41	60434	65,75	27,82	19,02
pcb442_50_3	50778	56681	72,85	57584	23,12	11,63	13,40
pcb442_50_4	50778	65330	84,57	58998	19,78	28,66	16,19
pcb442_50_5	50778	66865	68,94	57686	73,98	31,68	13,60
pcb442_50_6	50778	58928	84,35	59957	67,66	16,05	18,08
pcb442_50_7	50778	61700	76,58	64290	100,25	21,51	26,61
pcb442_50_8	50778	58836	44,24	60568	105,12	15,87	19,28
pcb442_50_9	50778	58723	80,51	63027	77,50	15,65	24,12
pcb442_50_10	50778	59454	96,79	56743	95,96	17,09	11,75
Moyenne	50778	61170,3	77,51	59858,3	72,74	20,47	17,88
pcb442_75_1	50778	66107	54,38	64711	42,93	30,19	27,44
pcb442_75_2	50778	66617	99,14	61801	84,91	31,19	21,71
pcb442_75_3	50778	57315	40,13	57425	96,51	12,87	13,09
pcb442_75_4	50778	61812	90,45	58668	68,31	21,73	15,54
pcb442_75_5	50778	66161	58,95	59135	95,70	30,29	16,46
pcb442_75_6	50778	65259	93,02	58394	93,61	28,52	15,00
pcb442_75_7	50778	66731	85,74	65940	85,46	31,42	29,86
pcb442_75_8	50778	64541	98,83	56605	85,21	27,10	11,48
pcb442_75_9	50778	60348	71,31	68825	87,74	18,85	35,54
pcb442_75_10	50778	64006	31,26	58597	93,25	26,05	15,40
Moyenne	50778	63889,7	72,32	61010,1	83,36	25,82	20,15

TABLEAU 5.5 – Comparaison entre la borne inférieure et les valeurs de GVNS-1 et GVNS-2 sur les instances avec 442 ports

GVNS-1 et 10 instances pour GVNS-2). Cela signifie donc que notre algorithme a obtenu une solution optimale dans ce cas.
– La déviation moyenne a tendance à augmenter avec le nombre de ports, et avec le pourcentage de ports affectés par la contrainte de tirant d'eau lorsque le nombre de ports reste fixe. Cela peut s'expliquer par la complexité de l'instance d'une part, et aussi par la qualité de la borne inférieure d'autre part.

Les résultats numériques présentés dans cette section nous permettent de conclure que nos deux algorithmes fournissent des solutions de très bonne qualité en un temps de calcul tout à fait raisonnable. Les solutions obtenues sur les instances de la littérature sont généralement les solutions optimales (238 sur 240 instances pour l'algorithme GVNS-2). Même si nos algorithmes ne permettent pas de prouver l'optimalité de la solution retournée, nous pouvons dire que leur comportement est très encourageant sur les instances de la littérature. Les résultats obtenus sur les nouvelles instances de plus grande taille sont également très encourageants. La déviation observée par rapport à une borne inférieure reste en effet raisonnable, malgré une contrainte forte sur le temps de calcul des algorithmes.

5.5 Conclusion

Dans ce chapitre, nous avons présenté deux approches basées sur la recherche à voisinage variable générale pour la résolution d'une variante du problème de voyageur de commerce. Cette variante se place dans le contexte du transport maritime et l'objectif est de trouver la meilleure route possible pour un navire contenant un ensemble de marchandises à livrer à un ensemble de ports, en s'assurant de respecter les contraintes liées au niveau de la mer dans chacun de ces ports (contraintes dites de tirant d'eau). Ce problème a été introduit récemment dans la littérature [51].

Les deux approches proposées dans ce chapitre utilisent deux méthodes différentes pour l'exploration de l'espace de recherche : la première variante utilise l'ensemble des voisinages définis pour ce problème, et met en œuvre une stratégie du premier voisin améliorant, alors que la seconde explore une partie seulement des voisinages, en restant dans le même voisinage tant qu'il est pos-

sible d'améliorer la solution courante. Ces deux variantes ont été testées sur l'ensemble des instances disponibles de la littérature. Les résultats obtenus sur ces instances montrent l'efficacité des deux algorithmes qui convergent, en pratique, vers pratiquement toutes les solutions optimales en un temps restreint à 100 secondes. Nous avons également évalué le comportement de nos méthodes sur de nouvelles instances de plus grande taille. Les résultats confirment le bon comportement général de nos algorithmes.

Ce travail n'entre pas tout à fait dans le même cadre que les travaux des chapitres 3 et 4 précédents, et peut être considéré pour nous comme une première base dans le contexte du transport maritime. Il pourrait être intéressant de s'intéresser à des aspects intégrant la gestion de stock dans ce domaine (de nombreux travaux de la littérature existent sur des problématiques d'IRP dans le maritime), voir d'étudier d'autres problèmes intégrant également plusieurs modes de transport (problématique multimodale).

Conclusions et perspectives

L'activité du transport est l'une des activités les plus importantes dans la chaîne logistique. De nos jours, il est devenu indispensable pour les entreprises de trouver une bonne organisation de cette activité, de sorte à optimiser sa productivité. De nombreux travaux et efforts ont été menés dans l'étude et la proposition de solutions efficaces pour traiter des problèmes qui apparaissent dans ce secteur. Dans cette thèse, nous avons proposé et validé des approches basées sur la recherche à voisinage variable pour deux problèmes qui apparaissent dans l'activité de transport. Le premier problème combine le routage avec la gestion de stock, et le deuxième est une variante du problème de voyageur de commerce qui apparaît dans le transport maritime lorsque des contraintes additionnelles dites de tirant d'eau sont prises en compte pour pouvoir accéder aux différents ports.

La première partie de ce mémoire est consacrée à l'introduction de quelques définitions préliminaires qui tournent autour de l'optimisation. Nous avons ensuite présenté le schéma de base de la métaheuristique appelée recherche à voisinage variable, ainsi que quelques unes des différentes variantes développées à partir de ce schéma initial. Nous avons ensuite abordé les définitions et quelques méthodes de résolution parmi les plus efficaces pour trois grandes classes de problèmes de transport : le problème du voyageur de commerce, le problème d'élaboration de tournées de véhicules, et le problème de routage avec gestion de stock. Cette étude nous a permis d'avoir une idée assez générale sur les techniques de résolution existantes pour traiter ces problèmes NP–Difficiles.

Notre première contribution est présentée dans le chapitre 3 de ce mémoire. Nous avons considéré une variante du problème de transport avec gestion de stock. Il s'agit d'un problème de ramassage de produits à partir d'un ensemble de fournisseurs, ces produits devant être acheminés vers une usine d'assemblage. La gestion de stock concerne uniquement l'usine, l'objectif étant d'assurer la continuité de la production. Nous avons proposé et développé dans un premier temps une approche en deux phases basée sur la recherche à voisinage variable pour résoudre ce problème. Dans la première

phase nous ne considérons que les coûts associés aux tournées de véhicules, et omettons donc les coûts de stockage. Cela permet de simplifier le problème en se ramenant à une problème de tournées, et nous sert pour obtenir une solution initiale de bonne qualité, dans le sens où les routes des différents véhicules sont optimisées. Dans la deuxième phase, nous traitons le problème dans sa globalité en introduisant les coûts de stockage dans une optimisation globale. Afin de garantir l'exploration de solutions de bonne qualité, nous avons également intégré une heuristique pour mieux évaluer les voisins de la solution courante dans la recherche à voisinage variable. Cette heuristique détermine les "bonnes" quantités à ramasser chez les fournisseurs et à acheminer vers l'usine. La recherche à voisinage variable en deux phases a été testée sur un ensemble de jeux de données de la littérature et elle a montré son efficacité.

Nous avons traité le même problème dans le chapitre 4 de ce mémoire. Nous avons proposé une deuxième approche basée sur la recherche à voisinage variable. En fait, notre motivation vient de la gestion des quantités de produits manipulées au sein de l'algorithme. Dans notre première approche, l'heuristique de gestion de stocks a l'avantage de déterminer de façon assez précise les quantités à ramasser. Elle présente cependant l'inconvénient d'alourdir assez fortement le processus d'évaluation de chaque voisinage au sein de la RVV. L'idée était donc ici d'introduire des mouvements discrets de quantités entre les périodes et entre les tournées de véhicules. Nous avons ainsi proposé de nouvelles structures de voisinage pour ce problème, et ces structures ont été incluses dans un nouvel algorithme de recherche à voisinage variable. Une série de tests a été effectué, sur le même jeux de données que précédemment, et l'étude comparative a montré l'efficacité de notre approche.

Dans la dernière partie de ce mémoire, nous avons étudié le problème du voyageur de commerce avec tirant d'eau. Il s'agit d'une nouvelle variante proposée récemment dans le domaine du transport maritime. Elle consiste à trouver un cycle hamiltonien de coût minimum pour un navire visitant un ensemble de ports, en respectant la contrainte de tirant d'eau pour chaque port. Dans la littérature, ce problème était résolu uniquement par des approches exactes issues de la programmation mathématique. D'où la motivation de proposer des approches heuristiques efficaces pour résoudre plus rapidement

les instances existantes, et pouvoir résoudre de plus grandes instances. Nous avons proposé deux approches basées sur la RVV et combiné deux stratégies pour l'exploration de l'espace de recherche. Les résultats sur des jeux de données de la littérature montrent la convergence de nos algorithmes vers les solutions optimales dans un temps restreint à 100 secondes. Afin d'évaluer le comportement de nos approches dans des conditions plus difficiles, nous avons généré de nouvelles instances pour ce problème comprenant jusqu'à 442 ports. Les méthodes exactes existantes ne peuvent actuellement pas résoudre ces instances en temps raisonnable. Nous avons donc évalué la qualité des solutions obtenues par nos méthodes grâce à une borne inférieure, et les résultats obtenus confirment la bon fonctionnement de nos approches.

Nous pouvons envisager quelques perspectives aux travaux présentés dans ce mémoire. En particulier, nous soulignons les quatre grands points suivants :

- Les idées et les algorithmes développés dans cette thèse peuvent, certainement, être étendus et/ou adaptés à d'autres problèmes de routage avec gestion de stock, voir à d'autres problèmes traités par la communauté de la recherche opérationnelle. Il nous paraît par exemple envisageable de considérer des problèmes de livraison avec prise en compte de la gestion de stock, et d'adapter les algorithmes proposés dans les chapitres 3 et 4.
- Même si les résultats obtenus par nos algorithmes sont globalement satisfaisants, il reste toujours envisageable d'essayer de les améliorer. Pour cela, une piste de réflexion serait l'hybridation de nos méthodes avec des approches issues de la programmation mathématique. Nous avons par exemple proposé un modèle mathématique dans le chapitre 3 pour déterminer de façon optimale les quantités de produits à ramasser chez les fournisseurs. Nous avons vu que l'utilisation de ce modèle entraînait un surcoût en terme de temps de calcul. Cela se justifie facilement par le nombre d'appels éffectués lors de l'exploration des solutions voisines. Il serait intéressant d'explorer l'utilisation de la programmation mathématique pour gérer d'autres aspects du problème.
- Il nous semble intéressant d'envisager une extension du problème du voyageur de commerce avec tirant d'eau vers une variante où nous considérons en plus la gestion de stock dans les ports. De façon plus

générale, ce type de problématique peut aussi être étendu pour optimiser un processus d'acheminement de marchandises plus complexes de la chaîne logistique.

– Finalement, nous avons proposé différentes approches pour l'exploration de l'espace de recherche avec la descente à voisinage variable pour résoudre le TSPDL. Nous pensons qu'il serait intéressant d'étudier de façon empirique l'efficacité et l'impact de différentes approches de l'exploration de l'espace de recherche au sein de cet algorithme.

Bibliographie

[1] Y. Adulyasak, J.-F. Cordeau, and R. Jans. Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. *INFORMS Journal on Computing*, 26(1) :103–120, 2014.

[2] D. T. Allah, J. Renaud, and F. F. Boctor. Le problème d'approvisionnement des stations d'essence. *Journal européen des systèmes automatisés*, 34(1) :11–33, 2000.

[3] S. Anily and A. Federgruen. One warehouse multiple retailer systems with vehicle routing costs. *Management Science*, 36(1) :92–114, 1990.

[4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde tsp solver, 2006.

[5] D. L. Applegate. *The traveling salesman problem : a computational study*. Princeton University Press, 2006.

[6] D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, and K. Helsgaun. Certification of an optimal tsp tour through 85,900 cities. *Operations Research Letters*, 37(1) :11–15, 2009.

[7] C. Archetti, L. Bertazzi, G. Laporte, and M. G. Speranza. A branch-and-cut algorithm for a vendor-managed inventory-routing problem. *Transportation Science*, 41(3) :382–391, 2007.

[8] R. Baldacci, M. Battarra, and D. Vigo. Routing a heterogeneous fleet of vehicles. In *The vehicle routing problem : latest advances and new challenges*, pages 3–27. Springer, 2008.

[9] M. L. Balinski and R. E. Quandt. On an integer program for a delivery problem. *Operations Research*, 12(2) :300–304, 1964.

[10] J. F. Bard, L. Huang, P. Jaillet, and M. Dror. A decomposition approach to the inventory routing problem with satellite facilities. *Transportation science*, 32(2) :189–203, 1998.

[11] M. Battarra, A. A. Pessoa, A. Subramanian, and E. Uchoa. Exact algorithms for the traveling salesman problem with draft limits. *European Journal of Operational Research*, 235(1) :115–128, 2014.

[12] W. J. Bell, L. M. Dalberto, M. L. Fisher, A. J. Greenfield, R. Jaikumar, P. Kedia, R. G. Mack, and P. J. Prutzman. Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer. *Interfaces*, 13(6) :4–23, 1983.

[13] J. Berger and M. Barkaoui. A hybrid genetic algorithm for the capacitated vehicle routing problem. In *Genetic and Evolutionary Computation ? GECCO 2003*, pages 646–656. Springer, 2003.

[14] L. Bertazzi and M. G. Speranza. Continuous and discrete shipping strategies for the single link problem. *Transportation Science*, 36(3) :314–325, 2002.

[15] J. Bramel and D. Simchi-Levi. A location based heuristic for general routing problems. *Operations research*, 43(4) :649–660, 1995.

[16] J. Brimberg, D. Urošević, and N. Mladenović. Variable neighborhood search for the vertex weighted k-cardinality tree problem. *European Journal of Operational Research*, 171(1) :74–84, 2006.

[17] A. Campbell, L. Clarke, A. Kleywegt, and M. Savelsbergh. The inventory routing problem. In *Fleet management and logistics*, pages 95–113. Springer, 1998.

[18] A. M. Campbell and M. W. Savelsbergh. A decomposition approach for the inventory-routing problem. *Transportation Science*, 38(4) :488–502, 2004.

[19] V. Černỳ. Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1) :41–51, 1985.

[20] P. Chen, H.-k. Huang, and X.-Y. Dong. Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. *Expert Systems with Applications*, 37(2) :1620–1627, 2010.

[21] T. W. Chien, A. Balakrishnan, and R. T. Wong. An integrated inventory allocation and vehicle routing problem. *Transportation Science*, 23(2) :67–76, 1989.

[22] M. Christiansen and B. Nygreen. A method for solving ship routing problemswith inventory constraints. *Annals of Operations Research*, 81 :357–378, 1998.

[23] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[24] N. Christofides. *Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*, chapter Vehicle routing. Wiley, 1985.

[25] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20(1) :255–282, 1981.

[26] G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4) :568–581, 1964.

[27] L. C. Coelho, J.-F. Cordeau, and G. Laporte. Thirty years of inventory routing. *Transportation Science*, 48(1) :1–19, 2014.

[28] L. C. Coelho and G. Laporte. A branch-and-cut algorithm for the multi-product multi-vehicle inventory-routing problem. *International Journal of Production Research*, 51(23-24) :7156–7169, 2013.

[29] L. C. Coelho and G. Laporte. The exact solution of several classes of inventory-routing problems. *Computers & Operations Research*, 40(2) :558–565, 2013.

[30] J.-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2) :105–119, 1997.

[31] J.-F. Cordeau, G. Laporte, and A. Mercier. Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows. *Journal of the Operational Research Society*, 55(5) :542–546, 2004.

[32] J.-F. Cordeau, G. Laporte, A. Mercier, et al. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8) :928–936, 2001.

[33] G. Cornuéjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical programming*, 33(1) :1–27, 1985.

[34] M.-C. Costa, F.-R. Monclar, and M. Zrikem. Variable neighborhood decomposition search for the optimization of power plant cable layout. *Journal of Intelligent Manufacturing*, 13(5) :353–365, 2002.

[35] P. Côté, T. Wong, and R. Sabourin. A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem. In *Practice and Theory of Automated Timetabling V*, pages 294–312. Springer, 2005.

[36] G. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6) :791–812, 1958.

[37] G. Dantzig, D. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2 :393–410, 1954.

[38] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1) :80–91, 1959.

[39] M. C. de Souza and P. Martins. Skewed vns enclosing second order algorithm for the degree constrained minimum spanning tree problem. *European Journal of Operational Research*, 191(3) :677–690, 2008.

[40] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14(4) :545–565, 1984.

[41] M. Dror and M. Ball. Inventory/routing : Reduction from an annual to a short-period problem. *Naval Research Logistics (NRL)*, 34(6) :891–905, 1987.

[42] W. L. Eastman. *Linear programming with pattern constraints : a thesis.* PhD thesis, Harvard University, 1958.

[43] G. Erdoğan, J.-F. Cordeau, and G. Laporte. The attractive traveling salesman problem. *European Journal of Operational Research*, 203(1) :59–69, 2010.

[44] A. Federgruen and P. Zipkin. A combined vehicle routing and inventory allocation problem. *Operations Research*, 32(5) :1019–1037, 1984.

[45] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2) :109–124, 1981.

[46] P. M. Francis, K. R. Smilowitz, and M. Tzur. The period vehicle routing problem and its extensions. In *The vehicle routing problem : latest advances and new challenges*, pages 73–102. Springer, 2008.

[47] B. Gavish and S. C. Graves. The travelling salesman problem and related problems. Massachusetts Institute of Technology, Operations Research Center, Working Paper OR 078-78, 1978.

[48] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10) :1276–1290, 1994.

[49] M. Gendreau, J.-Y. Potvin, O. Bräumlaysy, G. Hasle, and A. Løkketangen. *Metaheuristics for the vehicle routing problem and its extensions : A categorized bibliography.* Springer, 2008.

[50] B. E. Gillett and L. R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, 22(2) :340–349, 1974.

[51] J. Glomvik Rakke, M. Christiansen, K. Fagerholt, and G. Laporte. The traveling salesman problem with draft limits. *Computers & Operations Research*, 39(9) :2161–2167, 2012.

[52] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5) :533–549, 1986.

[53] B. Golden, A. Assad, and R. Dahl. Analysis of a large scale vehicle routing problem with an inventory component. *Large Scale Systems*, 7(2-3) :181–190, 1984.

[54] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, pages 160–168. Lawrence Erlbaum, New Jersey, 1985.

[55] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations*, volume 12 of *Combinatorial Optimization*. Springer, 2007.

[56] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on numerical methods in combinatorial optimization, Capri, Italy*, pages 70–145, 1986.

[57] P. Hansen, J. Brimberg, D. Uroševic, and N. Mladenovic. Primal-dual variable neighborhood search for the simple plant-location problem. *INFORMS Journal on Computing*, 19(4) :552–564, 2007.

[58] P. Hansen, J. Brimberg, D. Urošević, and N. Mladenović. Solving large p-median clustering problems by primal–dual variable neighborhood search. *Data Mining and Knowledge Discovery*, 19(3) :351–375, 2009.

[59] P. Hansen and N. Mladenović. *An introduction to variable neighborhood search*. Springer, 1999.

[60] P. Hansen and N. Mladenović. First vs. best improvement : An empirical study. *Discrete Applied Mathematics*, 154(5) :802–817, 2006.

[61] P. Hansen, N. Mladenović, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4) :335–350, 2001.

[62] K. Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1) :106–130, 2000.

[63] R. Holmes and R. Parker. A vehicle scheduling procedure based upon savings and a solution perturbation scheme. *Journal of the Operational Research Society*, 27(1) :83–92, 1976.

[64] H. Housroum. *Une approche génétique pour la résolution du problème VRPTW dynamique*. PhD thesis, Artois, 2005.

[65] B. Hu and G. R. Raidl. Variable neighborhood descent with self-adaptive neighborhood-ordering. In *Proceedings of the 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics*. Citeseer, 2006.

[66] S.-H. Huang and P.-C. Lin. A modified ant colony optimization algorithm for multi-item inventory routing problems with demand uncertainty. *Transportation Research Part E : Logistics and Transportation Review*, 46(5) :598–611, 2010.

[67] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, 39(2) :206–232, 2005.

[68] A. Ilić, D. Urošević, J. Brimberg, and N. Mladenović. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. *European Journal of Operational Research*, 206(2) :289–300, 2010.

[69] B. Jarboui, H. Derbel, S. Hanafi, and N. Mladenović. Variable neighborhood search for location routing. *Computers & Operations Research*, 40(1) :47–57, 2013.

[70] D. S. Johnson. Local optimization and the traveling salesman problem. In *Automata, languages and programming*, pages 446–461. Springer, 1990.

[71] D. S. Johnson and L. A. McGeoch. The traveling salesman problem : A case study in local optimization. *Local search in combinatorial optimization*, 1 :215–310, 1997.

[72] N. Jozefowiez, F. Semet, and E.-G. Talbi. Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2) :293–309, 2008.

[73] S. Kirkpatrick. Optimization by simulated annealing : Quantitative studies. *Journal of statistical physics*, 34(5-6) :975–986, 1984.

[74] J. Knox and F. Glover. Comparative testing of traveling salesman heuristics derived from tabu search, genetic algorithms, and simulated annealing. Technical report, University of Colorado, 1989.

[75] J. Kytöjoki, T. Nuortio, O. Bräysy, and M. Gendreau. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34(9) :2743–2757, 2007.

[76] G. Laporte. The traveling salesman problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2) :231–247, 1992.

[77] G. Laporte. The vehicle routing problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3) :345–358, 1992.

[78] G. Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics (NRL)*, 54(8) :811–819, 2007.

[79] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, 7(4-5) :285–300, 2000.

[80] G. Laporte, H. Mercure, and Y. Nobert. An exact algorithm for the asymmetrical capacitated vehicle routing problem. *Networks*, 16(1) :33–46, 1986.

[81] J. Lazić, S. Hanafi, N. Mladenović, and D. Urošević. Variable neighbourhood decomposition search for 0–1 mixed integer programs. *Computers & Operations Research*, 37(6) :1055–1067, 2010.

[82] C. Lee, Y. A. Bozer, and C. White III. A heuristic approach and properties of optimal solutions to the dynamic inventory routing problem. *Working Paper, Toronto, Ontario, Canada*, 2003.

[83] M. A. Lejeune. A variable neighborhood decomposition search method for supply chain management planning problems. *European Journal of Operational Research*, 175(2) :959–976, 2006.

[84] A. N. Letchford, S. D. Nasiri, and D. O. Theis. Compact formulations of the steiner traveling salesman problem and related problems. *European Journal of Operational Research*, 228(1) :83–92, 2013.

[85] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2) :498–516, 1973.

[86] W. M, D. P, and H. M. Un modèle et un algorithme de résolution exacte pour les problèmes des tournées de véhicules multi-périodiques : Une application à la distribution de gaz industriels. In *Deuxième congrés international franco- québécois de génie industriel*. Albi, France, 1997.

[87] R. Macedo, C. Alves, J. Valério de Carvalho, F. Clautiaux, and S. Hanafi. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. *European Journal of Operational Research*, 214(3) :536–545, 2011.

[88] M. Malek, M. Guruswamy, M. Pandya, and H. Owens. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, 21(1) :59–84, 1989.

[89] S. Michel and F. Vanderbeck. Optimisation des tournées de véhicules combinées à la gestion de stock. *Université Bordeaux*, 1, 2006.

[90] D. Miller and J. Pekny. Exact solutions of large asymmetric traveling salesman problems. *Science*, 251 :754–761, 1991.

[91] A. Mjirda, B. Jarboui, R. Macedo, and S. Hanafi. A variable neighborhood search for the multi-product inventory routing problem. *Electronic Notes in Discrete Mathematics*, 39 :91–98, 2012.

[92] A. Mjirda, B. Jarboui, R. Macedo, S. Hanafi, and N. Mladenović. A two phase variable neighborhood search for the multi-product inventory routing problem. *Computers & Operations Research*, 52 :291–299, 2014.

[93] A. Mjirda, B. Jarboui, J. Mladenović, C. Wilbaut, and S. Hanafi. A general variable neighbourhood search for the multi-product inventory routing problem. *IMA Journal of Management Mathematics*, page dpu020, 2014.

[94] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097–1100, 1997.

[95] N. Mladenović, F. Plastria, and D. Urošević. Reformulation descent applied to circle packing problems. *Computers & Operations Research*, 32(9) :2419–2434, 2005.

[96] N. Mladenovic, R. Todosijevic, and D. Urosevic. An efficient general variable neighborhood search for large travelling salesman problem with time windows. *The Yugoslav Journal of Operations Research ISSN : 0354-0243 EISSN : 2334-6043*, 23(1), 2012.

[97] N. Mladenović, D. Urošević, S. Hanafi, A. Ilić, et al. A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, 220(1) :270–285, 2012.

[98] N. Moin, S. Salhi, and N. Aziz. An efficient hybrid genetic algorithm for the multi-product multi-period inventory routing problem. *International Journal of Production Economics*, 133(1) :334–343, 2011.

[99] R. Mole and S. Jameson. A sequential route-building algorithm employing a generalised savings criterion. *Operational Research Quarterly*, 27(2).

[100] J. Norback and R. Love. Geometric approaches to solving the traveling salesman problem. *Management Science*, 23 :1208–1223, 1977.

[101] I. Osman and S. Ahmadi. Guided construction search metaheuristics for the capacitated p-median problem with single source constraint. *Journal of the Operational Research Society*, 58(1) :100–114, 2007.

[102] H. Paessens. The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, 34(3) :336–344, 1988.

[103] E. G. Pardo, N. Mladenović, J. J. Pantrigo, and A. Duarte. Variable formulation search for the cutwidth minimization problem. *Applied Soft Computing*, 13(5) :2242–2252, 2013.

[104] J. A. Persson and M. Göthe-Lundgren. Shipment planning at oil refineries using column generation and valid inequalities. *European Journal of Operational Research*, 163(3) :631–652, 2005.

[105] S. Pirkwieser and G. R. Raidl. Multiple variable neighborhood search enriched with ilp techniques for the periodic vehicle routing problem with time windows. In *Hybrid Metaheuristics*, pages 45–59. Springer, 2009.

[106] S. Pirkwieser and G. R. Raidl. Matheuristics for the periodic vehicle routing problem with time windows. *Proceedings of matheuristics*, pages 28–30, 2010.

[107] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8) :2403–2435, 2007.

[108] D. Popović, M. Vidović, and G. Radivojević. Variable neighborhood search heuristic for the inventory routing problem in fuel delivery. *Expert Systems with Applications*, 39(18) :13390–13398, 2012.

[109] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12) :1985–2002, 2004.

[110] R. Qu, Y. Xu, and G. Kendall. A variable neighborhood descent search algorithm for delay-constrained least-cost multicast routing. In *Learning and Intelligent Optimization*, pages 15–29. Springer, 2009.

[111] B. Raa and E.-H. Aghezzaf. A practical solution approach for the cyclic inventory routing problem. *European Journal of Operational Research*, 192(2) :429–441, 2009.

[112] C. Rego, D. Gamboa, F. Glover, and C. Osterman. Traveling salesman problem heuristics : leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3) :427–441, 2011.

[113] G. Reinelt. Tsplib - a traveling salesman problem library. *ORSA journal on computing*, 3(4) :376–384, 1991.

[114] G. Reinelt. *The traveling salesman : computational solutions for TSP applications*. Springer-Verlag, 1994.

[115] J. Renaud, F. F. Boctor, and G. Laporte. An improved petal heuristic for the vehicle routeing problem. *Journal of the Operational Research Society*, 47 :329–336, 1996.

[116] D. J. Rosenkrantz, R. Stearns, and P. Lewis II. An analysis of several heuristics for the traveling salesman problem". *SIAM Journal on Computing*, 6 :563–581, 1977.

[117] D. M. Ryan, C. Hjorring, and F. Glover. Extensions of the petal method for vehicle routing. *Journal of the Operational Research Society*, 44 :289–296, 1993.

[118] S. Salhi, A. Imran, and N. A. Wassan. The multi-depot vehicle routing problem with heterogeneous vehicle fleet : Formulation and a variable neighborhood search implementation. *Computers & Operations Research*, DOI : 10.1016/j.cor.2013.05.011, 2013.

[119] M. Savelsbergh and J.-H. Song. Inventory routing with continuous moves. *Computers & Operations research*, 34(6) :1744–1763, 2007.

[120] M. Savelsbergh and J.-H. Song. An optimization algorithm for the inventory routing problem with continuous moves. *Computers & Operations research*, 35(7) :2266–2282, 2008.

[121] Z. Sevkli and F. E. Sevilgen. A hybrid particle swarm optimization algorithm for function optimization. In *Applications of Evolutionary Computing*, pages 585–595. Springer, 2008.

[122] J.-H. Song and K. C. Furman. A maritime inventory routing problem : Practical approach. *Computers & Operations Research*, 40(3) :657–665, 2013.

[123] É. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8) :661–673, 1993.

[124] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, 31(2) :170–186, 1997.

[125] K. C. Tan, L. H. Lee, Q. Zhu, and K. Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial intelligence in Engineering*, 15(3) :281–295, 2001.

[126] C. D. Tarantilis and C. T. Kiranoudis. Boneroute : an adaptive memory-based method for effective fleet management. *Annals of Operations Research*, 115(1-4) :227–241, 2002.

[127] R. Todosijević, A. Mjirda, M. Mladenović, S. Hanafi, and B. Gendron. A general variable neighborhood search variants for the travelling salesman problem with draft limits. *Optimization Letters*, pages 1–10.

[128] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4) :333–346, 2003.

[129] D. Urošević, J. Brimberg, and N. Mladenović. Variable neighborhood decomposition search for the edge weighted k-cardinality tree problem. *Computers & Operations Research*, 31(8) :1205–1213, 2004.

[130] VeRoLog. Swap-body vehicle routing problem, 2014.

[131] I. R. Webb and R. C. Larson. Period and phase of customer replenishment : A new approach to the strategic inventory/routing problem. *European Journal of Operational Research*, 85(1) :132–148, 1995.

[132] Y. Xiao, I. Kaku, Q. Zhao, and R. Zhang. A reduced variable neighborhood search algorithm for uncapacitated multilevel lot-sizing problems. *European Journal of Operational Research*, 214(2) :223–231, 2011.