UNIVERSITE DE LIEGE
Faculté des Sciences

PROBLÈME DE TOURNÉES DE DISTRIBUTION AVEC FENÊTRES DE TEMPS ET APPLICATION

Mémoire présenté par
Patrick MEYER
pour l'obtention du grade de
licencié en sciences mathématiques

Année académique 1998-1999

Je tiens à remercier M. le Professeur Marc Roubens pour m'avoir donné l'occasion de réaliser ce mémoire.

Je tiens aussi à remercier Mlle Christelle Wynants de l'attention qu'elle a bien voulu porter à ce travail durant son élaboration, et des conseils dont il a bénéficié.

Mes plus vifs remerciements vont à M. Fabien Boniver pour son soutien tout au long de la réalisation de ce mémoire.

Introduction

Ce mémoire traite des problèmes de création et d'amélioration de tournées de distribution. Il a pour motivation principale la résolution d'un problème effectif de livraison de repas scolaires posé par une entreprise du Nord de la France.

Les problèmes de tournées de distribution consistent en l'élaboration de routes servant à visiter des clients à partir d'un ou de plusieurs dépôts. L'objectif est de créer ces routes en vue d'optimiser une fonction économique telle que la minimisation de coûts de transports. Les variantes sont multiples et, selon la complexité du problème réel, diverses contraintes sont introduites. Dans ce travail nous nous intéressons plus particulièrement aux problèmes de tournées de véhicules avec des fenêtres de temps relatives aux clients et au dépôt (on impose le passage du livreur dans des intervalles de temps spécifiés).

Les méthodes retenues pour leur résolution se répartissent en deux classes : les méthodes exactes et les méthodes heuristiques. Contrairement aux méthodes exactes qui fournissent la solution optimale du problème posé, les heuristiques ont pour but d'élaborer des solutions satisfaisantes en des temps raisonnables. Le problème des tournées de distribution étant *NP-complet*, aucune méthode exacte ne garantit un temps de résolution polynomial en la taille du problème. Pour cette raison, les méthodes les plus employées sont les heuristiques. Dans un premier temps, celles développées pour les tournées de véhicules s'inspiraient des méthodes existant pour le problème du voyageur de commerce. Au cours du temps elles ont évolué vers des algorithmes dont l'implémentation devient de plus en plus complexe.

Dans la première partie, nous détaillons une heuristique de recherche avec tabous développée par Taillard *et al.* Nous pensons qu'il est intéressant de présenter l'évolution des approches vers des algorithmes de plus en plus complexes et toujours mieux adaptés au problème que nous désirons traiter. C'est pourquoi dans les Chapitres 1 et 2 nous introduisons les problèmes de tournées de distribution et présentons quelques méthodes de résolution. De plus, dans le Chapitre 3, un aperçu général des heuristiques de recherche avec tabous est donné en préambule à l'algorithme final décrit dans le Chapitre 4.

Dans la seconde partie nous traitons une application dont le but est de réduire la distance parcourue par six camions devant livrer des repas scolaires à une centaine

d'écoles dans la région du Nord-Pas-de-Calais. Nous disposons d'un logiciel dont la version de base nous a été fournie par M. Gendreau et F. Guertin de l'Université de Montréal. Cependant les spécifités de l'application ne nous permettent pas de l'utiliser directement. Nous avons donc développé un modèle rendant compte du problème et modifié en conséquence le programme. Nous avons ainsi obtenu une solution qui améliore d'une manière significative la distance totale parcourue par les camions. En outre, la méthode que nous avons développée permet de traiter des instances de taille supérieure.

Nous terminons par quelques conclusions et nous présentons en annexe les données du problème que nous résolvons ainsi que des résultats comparatifs sous forme de tableaux et de schémas.

Table des matières

Première partie

Partie théorique

Chapitre 1

Aperçu général sur les tournées de distribution

1.1 Introduction

Dans ce chapitre, nous introduisons les tournées de distribution (*VRP*, Vehicle Routing Problem) d'une manière générale.

Le premier point a pour but de décrire le problème classique ainsi que quelques de ses dérivés. Nous y faisons également allusion au *VRP* avec fenêtres de temps, qui sera plus amplement traité dans la suite de ce mémoire.

Dans la Section 1.4 nous présentons deux formulations mathématiques du *VRP* classique. Nous détaillerons les contraintes introduites, tout en soulignant l'existence d'un grand nombre d'autres formulations. Dès ici on peut remarquer que suivant le problème à résoudre, on optera pour une formulation particulière.

Dans la section suivante un cas particulier de *VRP* est traité où le nombre de clients par route est limité.

Lors d'une prochaine étape nous décrivons quelques méthodes de résolution du *VRP* classique. Nous nous concentrerons sur des méthodes dites *heuristiques* tout en remarquant l'existence de méthodes dites *exactes*. Les heuristiques que nous présentons se divisent en deux groupes : les méthodes de construction de routes et celles d'amélioration de routes. Toutes ces considérations ne portent cependant que sur les tournées de véhicules classiques. Certains problèmes dus à des contraintes telles que les fenêtres de temps empêchent parfois le bon déroulement des méthodes que nous introduisons dans ce chapitre. A cette fin, dès le Chapitre 2 nous nous intéressons exclusivement au *VRP* avec fenêtres de temps et aux méthodes de résolution associées.

1.2 Le *VRP* classique et ses dérivés

Une composante importante de nombreuses entreprises est l'élaboration de tournées de distribution servant à effectuer un service quelconque chez des clients. Ce problème apparaît dans un grand nombre de situations pratiques. Les plus fréquents sont la collecte ou la livraison de courrier ou de toute autre marchandise, des tournées de visite à domicile pour des médecins, des tournées de maintenance (systèmes d'alarme) ou des rondes de police. Toutes ces situations, ainsi qu'un grand nombre de problèmes similaires, sont regroupées sous le nom de *tournées de distribution*, ou *VRP, Vehicle Routing Problem*. Dans certains cas, l'action de *livraison* peut représenter à la fois la livraison et la collecte d'un objet, ou alors ni l'un ni l'autre. Ainsi la *demande* des clients et les *véhicules* peuvent prendre toutes sortes de formes et dans certains cas n'être même pas de nature physique.

Le problème de base se posant est donc le suivant : on dispose d'un certain nombre de véhicules pouvant transporter une marchandise ou capables d'effectuer un travail et qui sont stockés dans un dépôt central. Des clients devant être livrés sont reliés par un réseau de routes. Afin de simplifier le discours, nous parlerons dans la suite de *livraison* de clients pour tous les services tels que chargement, déchargement d'une marchandise, travail effectué auprès des clients etc.

On déduira facilement de ces considérations les cas plus particuliers de *VRP*.

Le *VRP* classique consiste alors à trouver des routes (une route par véhicule, chaque route ayant son origine et son extrémité au dépôt) telles que chaque client soit livré une et une seule fois et les coûts de déplacement soient minimaux. La Figure 1.1 représente l'allure générale de la solution d'un problème de tournées de distribution.

Cependant, dans la réalité des contraintes de toutes sortes peuvent se présenter. Des variantes au *VRP* classique sont soumises à différents types de contraintes :

- 1. Chaque véhicule peut livrer plus qu'une route, à condition que le temps passé sur ces routes soit inférieur ou égal à un certain temps T dépendant du temps de livraison total. On peut remarquer ici qu'une contrainte de ce type (ainsi que d'autres contraintes citées ci-après) nécessitent la connaissance des temps de parcours t_{ij} entre les clients.
- 2. Chaque client peut imposer d'être livré dans un certain intervalle de temps (pouvant être différent pour chaque client). On parle alors de *VRP* avec fenêtres de temps. Ce problème étant très courant en pratique, nous le traiterons plus amplement dans la suite de ce mémoire.
- 3. Le problème peut imposer la livraison et la collecte d'une marchandise auprès des clients. On doit alors également tenir compte de la manière dont les marchandises sont entreposées dans les véhicules afin de faciliter leur déchargement par exemple.
- 4. De même qu'en 2., les conducteurs ne pouvant parfois travailler qu'à certaines

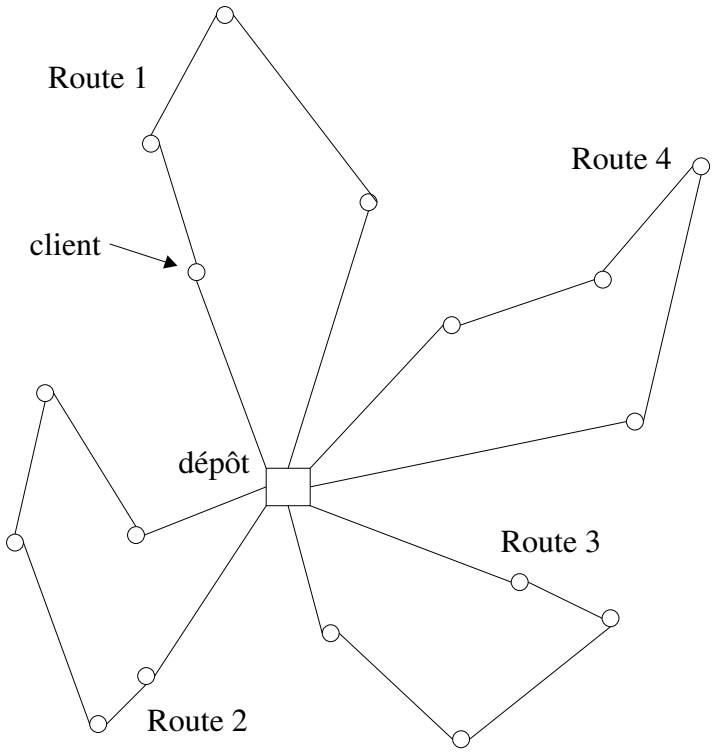

FIG. 1.1: Allure de la solution d'un *VRP*

heures, ce qui se traduit par des fenêtres de temps pour les véhicules.

– 5. Des facteurs demandant un certain temps peuvent intervenir dans le problème. Souvent il s'agit de temps de déchargement ou de chargement en chaque client ou dans le dépôt, ou de temps d'attente au dépôt lorsque le nombre de camions pouvant être chargés ou déchargés est limité.

Les contraintes citées ci-avant ne représentent qu'une petite portion de celles qu'on peut rencontrer en pratique. Cependant elles font partie des données qui sont facilement incorporables dans un grand nombre de méthodes heuristiques de résolution du *VRP*. D'un autre côté, il existe certaines considérations pratiques que l'on retrouve souvent, mais qui ne trouvent pas aussi facilement leur place dans le modèle de base du *VRP*. Nous en discutons dans la section suivante.

1.3 Aspects pratiques du *VRP*

Un certain nombre de données pratiques ne rentrent pas dans la version de base des tournées de distribution. Nous évoquons dans cette section des problèmes tels que les dépôts multiples, les livraisons de différents types de marchandises ainsi que des objectifs autres que la minimisations des distances.

1.3.1 Dépôts multiples

Dans certains cas, une compagnie peut posséder plusieurs dépôts. Il y a alors deux cas de figure différents :

- Si les dépôts sont indépendants, c'est à dire si chaque dépôt possède sa propre flotte de véhicules, alors on considère simplement plusieurs cas de *VRP* à dépôt unique.
- Dans le cas où un dépôt peut dépendre d'un autre dépôt, on ne peut plus considérer plusieurs *VRP* indépendamment. Ceci est le cas si un véhicule peut passer par plusieurs dépôts afin de charger ou de décharger de la marchandise avant de revenir à son point de départ.

1.3.2 Marchandises de type différent

Dans certains problèmes de tournées de véhicules, les camions peuvent transporter différentes marchandises. Dans ces cas, les clients peuvent recevoir des quantités prédéfinies des différentes marchandises. On rencontre ce type de problème lors de la livraison de repas ou de carburant. A côté du *VRP* il s'agit alors de prendre en compte un problème dit de *sac à dos*.

1.3.3 Objectifs différents

Parfois il est impossible de résoudre un *VRP* avec les contraintes et les données dont on dispose. Ce problème peut être du à un manque de véhicules ou à un problème de temps en les clients. En effet, on pourrait supposer que le problème prévoit des périodes de temps pendant lesquelles un certain ensemble de clients doit être livré. Le problème surgit alors si les temps de service et le nombre de clients ne sont pas compatibles avec l'intervalle de temps prévu. Deux solutions existent à ces problèmes :

- utiliser plus de véhicules
- déplacer des clients d'une période de livraison à la prochaine

Dans ces circonstances, l'objectif est de minimiser

- le nombre de véhicules utilisés en plus
- le nombre de clients à déplacer
- la distance totale parcourue (ou les temps de parcours)

En général, on peut imaginer que l'objectif est la minimisation d'une combinaison linéaire des 3 points ci-dessus.

Après ces considérations assez générales sur les différents problèmes qui peuvent apparaître à côté du problème de base, nous passons à deux formulations mathématiques du *VRP* classique.

1.4 Formulations du *VRP* classique

Considérons un graphe $G = (N, E)$, où N est l'ensemble des noeuds et E est l'ensemble des arêtes du graphe. Les clients sont représentés par les noeuds $i = 2, \ldots, n$, le noeud $i = 1$ représentant le dépôt. Les arêtes représentent des routes reliant les clients entre-eux. Les véhicules sont représentés par $k = 1, \ldots, M$.

La demande du client i vaut q_i, le coût du trajet entre i et j vaut c_{ij}. La capacité du véhicule k est de Q_k.

On suppose que les clients et les véhicules sont ordonnés d'après les q_i et les Q_k décroissants, c'est à dire que $q_n \leq q_{n-1} \leq \ldots \leq q_2$ et $Q_M \leq \ldots \leq Q_1$.

Il existe dès lors plusieurs formulations mathématiques du *VRP*. Suivant le type de problème à traiter, on utilisera la formulation la mieux adaptée. Nous en présenterons deux dans ce travail.

1.4.1 Formulation 1

Cette formulation est due à Fisher et Jaikumar [5] [6]. Elle se base principalement sur des variables binaires.

Etant donné les variables

$$x_{ijk} = \begin{cases} 1 & \text{si le véhicule } k \text{ visite le client } j \text{ immédiatement après } i \\ 0 & \text{sinon} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{si le véhicule } k \text{ visite le client } i \\ 0 & \text{sinon} \end{cases}$$

La fonction à minimiser est

$$\sum_{i,j} c_{ij} \sum_{k} x_{ijk} \tag{1.1}$$

sous les contraintes

$$\sum_{k} y_{ik} = \begin{cases} 1 & i = 2, \ldots, n \\ m & i = 1 \end{cases} \tag{1.2}$$

$$\sum_i q_i y_{ik} \leq Q_k \qquad k = 1, \ldots, m \tag{1.3}$$

$$\sum_j x_{ijk} = \sum_j x_{jik} = y_{ik} \qquad \begin{matrix} i = 1, \ldots, n \\ k = 1, \ldots, m \end{matrix} \tag{1.4}$$

$$\sum_{i,j \in S} x_{ijk} \leq |S| - 1 \qquad \forall S \subseteq \{2, \ldots, n\} \qquad k = 1, \ldots, m \tag{1.5}$$

$$y_{ik} \in \{0, 1\} \qquad \begin{matrix} i = 1, \ldots, n \\ k = 1, \ldots, m \end{matrix} \tag{1.6}$$

$$x_{ijk} \in \{0, 1\} \qquad \begin{matrix} i = 1, \ldots, n \\ j = 1, \ldots, n \\ k = 1, \ldots, m \end{matrix} \tag{1.7}$$

La contrainte 1.2 assure que chaque client soit visité une et une seule fois et le dépôt m fois. La contrainte 1.3 sert à ne pas dépasser la capacité maximale de chaque camion k. La contrainte 1.4 est nécessaire pour que chaque camion arrivant chez un client, en reparte également. De plus elle lie également les deux variables. Finalement, la contrainte 1.5 sert à éliminer les sous-tours. En effet, supposons que nous soyons en présence d'un sous-tour composé de deux clients p et q. Alors $x_{pqk} = x_{qpk} = 1$ pour un certain k. Pour $S = \{p, q\}$ on a alors $\sum_{i,j \in S} x_{ijk} = 2 \geq |S|$ (alors que pour tout autre ensemble S l'inégalité de 1.5 est vérifiée).

1.4.2 Formulation 2

Cette formulation est due à Christofides, Mingozzi et Toth [3]. Elle se base sur une récursion dynamique.

Soit $N = \{2, \ldots, n\}$ l'ensemble des clients. Pour tout sous-ensemble T de N, soit $f(k, T)$ le coût minimum des routes nécessaires à satisfaire à la demande des clients de T en utilisant les véhicules $1, \ldots, k$.

Soit $v(T)$ le coût minimum du problème de voyageur de commerce associé à T et au dépôt (on a donc affaire à un *VRP* à un véhicule dans T).

Soit

$$q(T) = \sum_{i \in T} q_i$$

la demande totale des clients de T.

On initialise la récursion avec :

$$\text{pour k=1 : } f(1,T) = v(T) \tag{1.8}$$

Les étapes ultérieures sont alors données par :

$$\text{pour } k \geq 2 : f(k,T) = \min_{S \subset T} f(k-1, T-S) + v(S) \tag{1.9}$$

Sous les contraintes :

$$q(T) - \sum_{h=1}^{k-1} Q_h \leq q(S) \leq Q_k \qquad \text{pour } k = 2, \ldots, m \tag{1.10}$$

$$\frac{1}{m-k} q(N-T) \leq q(S) \leq \frac{1}{k} q(T) \qquad \text{pour } k = 2, \ldots, m-1 \tag{1.11}$$

Les conditions sur les ensembles T s'écrivent :

$$q(N) - \sum_{h=k+1}^{m} Q_h \leq q(T) \leq \sum_{h=1} k Q_h \qquad k = 2, \ldots, m \tag{1.12}$$

Les contraintes sur S et T sont nécessaires afin d'éviter des calculs de f et de v qui mèneraient à des solutions non réalisables.

La contrainte 1.10 a un sens, car pour rappel, les camions sont rangés dans l'ordre décroissant des Q_k. Le membre de droite est une contrainte sur le k^e véhicule, alors que le membre gauche est une contrainte sur les $k-1$ premiers véhicules.

La contrainte 1.11 impose un ordre sur les routes : la charge sur la route k est supérieure à la moyenne des charges des $m-k$ routes restantes.

Dans cette section nous avons identifié routes et camions et dans la suite nous ne ferons plus de distinction entre les deux termes. Passons désormais à un cas particulier où toutes les capacités sont égales et où nous imposons une contrainte sur le nombre de clients par route.

1.5 Cas particulier

Considérons un *VRP* tel que $Q_1 = Q_2 = \ldots = Q_m = Q$ et $q_i + q_{i-1} + q_{i-2} > Q \quad i \in \{4, \ldots, n\}$. Ainsi chaque route comprendra au plus deux clients.

Soit le graphe $G = (N, E)$ où N est l'ensemble des clients $N = \{2, \ldots, n\}$ et E l'ensemble des arêtes tel que $E = \{(i,j) : i,j \in N \text{ et } q_i + q_j \leq Q\}$. Soit $c'_{ij} = c_{1i} + c_{ij} + c_{j1}$ le coût de l'arête (i,j) et $p_i = 2c_{1i}$ le poids de i.

Dans G, l'arête (i, j) représente alors la route $(1, i, j, 1)$ dans le *VRP*. Un sommet libre i représente la route $(1, i, 1)$. On remarque que si G possède s arêtes, alors le nombre de routes de la solution du *VRP* est $n - s$. Ainsi, si on désire avoir exactement m routes, on doit imposer la condition $s = n - m$.

La solution du problème est la solution du *generalized minimum cost matching problem* du graphe G. C'est à dire il s'agit de trouver un sous-ensemble d'arêtes tel que la somme des coûts des arêtes et des poids des sommets libres soit minimale.

Les prochains points qui nous intéresserons concerneront la résolution du problème général de tournées de véhicules. Dès ici nous pouvons évoquer l'existence de méthodes dites exactes. En particulier, une méthode intéressante de *séparation et évaluation* a été développée par Christofides, Mingozzi et Toth [2] [3]. Cependant, cette méthode n'est que peu intéressante d'un point de vue pratique vu qu'elle nécessite en général de longs temps de calculs. De plus, jusqu'à présent, le *VRP* le plus important résolu à l'aide de cette méthode comprenait 53 clients et 8 véhicules. Le problème comprenait à côté des contraintes de capacité des contraintes sur les temps de parcours des camions, des véhicules de capacité différente, des temps de déchargement et quelques fenêtres de temps (larges). Après résolution du problème, le nombre maximum de clients par route était de 15 clients.

A côté de cette méthode exacte, il existe des méthodes heuristiques qui fournissent de bons résultats en des temps raisonnables. Les heuristiques pour ce type de problème peuvent être classées en deux grandes catégories : les heuristiques gloutonnes et les heuristiques d'échange. L'idée conductrice des première méthode est la construction de routes, alors que dans la seconde catégorie il s'agit d'améliorer des routes existantes.

1.6 Méthodes heuristiques gloutonnes pour le *VRP* classique

Il existe de nombreuses heuristiques gloutonnes pour résoudre le *VRP*. On peut les classer en 3 catégories :

- les méthodes constructives
- les méthodes à deux phases
- les méthodes d'optimisation incomplètes

1.6.1 Méthodes constructives

Le premier type d'heuristique contient entre autres un algorithme très connu, cependant un peu simpliste. Il s'agit du *savings algorithm* de Clarke et Wright [4]. Cette méthode ayant été traitée dans un travail précédent, nous ne l'évoquerons pas ici.

Un bon nombre d'autres méthodes constructives existent. La procédure de construction séquentielle de Mole et Jameson utilise un critère dépendant du choix de l'utilisateur. Nous remarquons dès à présent que de nombreuses méthodes dépendent de paramètres définis par l'utilisateur et produisent donc des solutions différentes suivant la valeur de ceux-ci.

Cependant, ces méthodes constructives ne fournissent en général que des solutions assez médiocres. Nous préférons passer à des heuristiques de qualité supérieure.

1.6.2 Méthodes à deux phases

Dans ce type d'heuristiques, il s'agit d'abord de classer les clients dans des ensembles assignés à des véhicules et ensuite il s'agit de déterminer les routes optimales à l'intérieures de ces ensembles. Dans cet exposé nous présenterons deux méthodes nous semblant particulièrement intéressantes.

L'algorithme de balayage

Cet algorithme est dû à Gillett et Miller [8].

On suppose que les clients sont localisés par leurs coordonnées polaires (r_i, θ_i) par rapport à un lient i^* en $\theta_{i^*} = 0$ et tel que le dépôt soit en $r_1 = 0$. On ordonne les clients de manière à ce que $\theta_2 \leq \ldots \leq \theta_n$.

Algorithme 1.1

Phase I

- Etape 1. Choisir un véhicule k non encore utilisé.
- Etape 2. A partir du client i d'angle θ_i minimal, inclure consécutivement les clients $i+1, i+2, \ldots$ dans la route k jusqu'à ce que la capacité du véhicule soit atteinte.
- Etape 3. Si tous les clients ont été "balayés" ou si tous les camions utilisés, commencer la phase II, sinon, retourner à l'étape 1.

Phase II

- Etape 4. Résoudre le TSP pour chaque ensemble de clients créé dans la première phase.

Il est évident que le résultat final dépendra du choix du client i^* ainsi que de la manière dont on choisira l'ordre des véhicules dans la première phase.

La méthode à deux phases de Christofides, Mingozzi et Toth [1]

La première phase de cette heuristique consiste à faire plusieurs essais de regroupement en utilisant un critère d'insertion de coût minimal à l'aide d'un paramètre contrôlé par l'utilisateur. Ce dernier pourrait alors créer différents essais suivant sa valeur.

Algorithme 1.2

Phase I

- Etape 1. (*Essai séquentiel*) Choisir un client non encore visité et le définir comme noyau. Choisir un véhicule k pour livrer la route qui émergera de ce noyau.
- Etape 2. Ajouter des clients non encore visités au noyau suivant un coût d'insertion croissant relatif au noyau, jusqu'à ce que la capacité du véhicule k choisi soit atteinte. Lorsque tous les clients sont répartis ou tous les véhicules utilisés, aller à l'étape 3, sinon répéter à partir de l'étape 1.
- Etape 3. (*Essai parallèle*) Utiliser les noyaux choisis lors de l'essai séquentiel et libérer tous les autres clients de leurs regroupements.
- Etape 4. Pour chaque client, calculer son coût d'insertion dans chaque regroupement (réalisable). Pour un client donné, parmi tous les regroupements, retenir celui avec le meilleur coût d'insertion.
- Etape 5. Parais tous les clients libres, ajouter celui dont le coût d'insertion est minimal à son regroupement (défini par l'étape précédente).
- Etape 6. Répéter l'étape 4 pour tous les clients dont la meilleure insertion est devenue non réalisable. Continuer avec l'étape 5 jusqu'à ce qu'il n'y ait plus d'insertion réalisable.

Phase II

- Etape 7. Pour chacun des deux essais précédents, résoudre un TSP pour chaque regroupement. Garder la meilleure solution comme solution du VRP.

Nous remarquons à nouveau que cette méthode dépend fortement du coût d'insertion et donc de la manière dont l'utilisateur a décidé de mesurer ce coût. Plusieurs solutions peuvent ainsi apparaître suivant ce choix.

1.6.3 Méthode d'optimisation incomplète

Si l'on reprend la méthode de séparation et évaluation évoquée plus tôt, que l'on effectue une première recherche et que l'on applique une heuristique pour le choix du client sur lequel effectuer le branchage, alors la méthode devient une heuristique si l'on s'arrête prématurément.

1.6.4 Remarques sur les résultats numériques

Des tests menés sur douze *VRP* euclidiens dont la taille varie de 50 à 200 clients, ont donné les résultats suivants. En ce qui concerne la qualité de la solution, la méthode d'optimisation incomplète l'a emporté en moyenne sur les autres méthodes présentées (après 92 secondes de calculs sur un CDC 6600). Ensuite viennent la méthode à deux phases de Christofides, Mingozzi et Toth (à 4,5 % de la meilleure solution), l'algorithme de balayage de Gillett et Miller (à 4,8 %) et finalement le *savings algorithm* de Clarke et Wright (à 12,5 %). Quant aux temps de calculs, c'est l'algorithme de Clarke et Wright qui l'emporte (5 à 6 secondes), alors que la méthode d'optimisation incomplète est la plus lente.

1.7 Heuristiques d'échange

A côté des méthodes présentées précédemment, nombreuses sont les heuristiques permettant d'améliorer une solution d'un *VRP* afin d'obtenir un meilleur résultat. Il s'agit de méthodes de recherche locale. La plupart du temps on inclut l'étape de modification dans un processus itératif du type suivant [10] :

Algorithme 1.3

- Etape 1. Générer une solution admissible initiale.
- Etape 2. Modifier la solution courante afin d'obtenir une nouvelle solution admissible améliorée.
- Etape 3. Répéter l'étape 2 jusqu'à ce qu'aucune amélioration ne soit possible.

Pour illustrer ce type de méthode, nous allons présenter une procédure d'échange due à Lin [10]. Dans l'algorithme précédent, elle est appelée à la deuxième étape.

Algorithme r-opt

Nous partons d'une route réalisable (c'est à dire d'un chemin dont l'extrémité et l'origine coïncident). Dans le cas général, dans une première phase r arêtes sont supprimées. Ensuite les noeuds libérés sont reconnectés à l'aide de r arêtes ne faisant pas partie de la route choisie et de manière à ce que la nouvelle route créée soit meilleure que la précédente. Dans ce cas nous parlerons de procédure r-opt, où r est le nombre d'arêtes échangées à chaque itération. Un algorithme r-opt teste toutes les reconnections possibles de r arêtes jusqu'à ce qu'il n'y ait plus d'échange menant à une amélioration de la solution courante. Cette solution sera alors dite r-optimale. En général si r est choisi grand, la solution finale a plus de chance d'être optimale. Cependant, le nombre d'opérations nécessaires à tester r échanges augmente rapidement avec le nombre de clients (noeuds). Dès lors, le plus souvent on se limite à $r = 2$ ou $r = 3$.

Les figures 1.2 et 1.3 illustrent ces considérations pour $r = 4$.

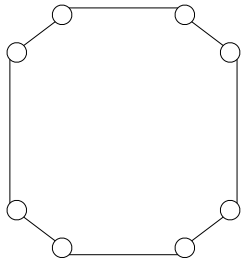

FIG. 1.2: Route courante.

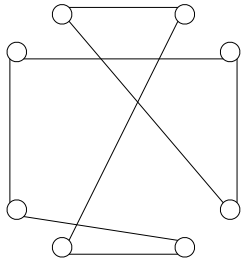

FIG. 1.3: Route après l'échange.

En ce qui concerne la complexité de ce type d'algorithme, nous pouvons retenir les quelques considérations suivantes. Dans une solution comprenant N arêtes, il existe $O(N^r)$ possibilités de choisir r liens pour générer une nouvelle solution. L'ensemble de nouvelles solutions ainsi créé est souvent appelé *voisinage de la solution courante*. Ces solutions sont toutes évaluées et la première solution qui améliore la solution courante sera choisie comme la nouvelle solution courante. Excepté pour quelques cas particuliers, la méthode converge rapidement vers un optimum local. Dans la suite on parlera d'heuristique en $O(N^r)$ lorsque la taille du voisinage de la solution courante sera de l'ordre de $O(N^r)$.

Un autre algorithme de ce type est donné par Or [11]. Il ne considère qu'une partie des échanges de 3-opt.

Algorithme *Or*-opt

Cette procédure ne considère que des échanges qui résulteraient en une insertion d'une suite d'un, deux ou trois noeuds adjacents entre deux autres noeuds. En limitant de cette manière le nombre d'échanges devant être considérés, cette méthode nécessite moins d'étapes de calcul que 3-opt.

Afin de mieux comprendre le fonctionnement de cette méthode, nous nous référons à l'exemple pour 3 noeuds illustré sur les figures 1.4 et 1.5.

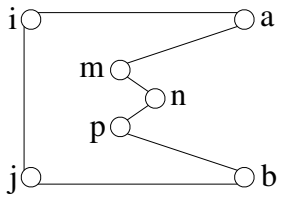

FIG. 1.4: Route courante.

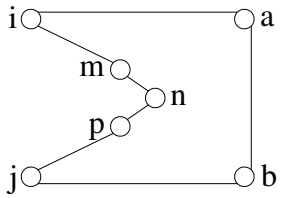

FIG. 1.5: Route après l'échange.

Pour chaque segment de s clients (tout d'abord $s = 3$, puis $s = 2$ et finalement $s = 1$) consécutifs de la route courante, nous vérifions s'il peut être réinséré entre deux autres clients de manière à réduire le coût de la route. Si c'est le cas, nous effectuons les transformations nécessaires. Dans le cas de la Figure 1.4 on considère chaque segment de 3 clients consécutifs m, n, p de la route courante pour l'insertion entre toute paire de clients i et j ne faisant pas partie du segment en question. On effectue alors l'échange si le coût total des arcs à éliminer $((a, m), (p, b), (i, j))$ excède celui des nouveaux arcs $((i, m), (p, j), (a, b))$. Après avoir considéré ainsi tous les segments à 3 clients, on fait de même pour ceux contenant 2 respectivement de 1 client. Lorsque plus d'amélioration n'est possible, on arrête la procédure.

Lorsque le problème à résoudre comprend également des contraintes, l'admissibilité de chaque solution trouvée doit être vérifiée. Ceci est le cas lors de contraintes de capacité sur les camions ou dans le cas de fenêtres de temps imposées par les clients. Ce

dernier problème sera traité plus amplement dans le prochain chapitre. Intuitivement on comprendra que la réalisabilité d'une solution se vérifiera dans le plus mauvais des cas en vérifiant les temps d'arrivée en chaque client. Ce test sera alors en $O(N)$. Cependant plusieurs techniques ont été développées pour réduire ces temps de calcul. Nous nous occuperons de ces heuristiques dans le chapitre suivant.

Chapitre 2

Heuristiques d'échange pour le VRP avec fenêtres de temps

2.1 Introduction.

Le *VRP avec fenêtres de temps* est un dérivé du *VRP* classique. La différence majeure entre ces deux types de problème est l'introduction de fenêtres de temps relatives aux clients à livrer.

Nous parlons de fenêtres de temps *souples*, si nous acceptons une arrivée anticipée ou tardive d'un camion chez un client. Dans le premier cas, nous imposons une période d'attente au camion, dans le second cas une pénalité lui est appliquée (pouvant être propre à chaque client).

Dans le cas ou aucune tolérance n'est acceptée vis-à-vis d'arrivées anticipées ou tardives, nous parlons de fenêtres de temps *rigides*.

Le plus souvent, ce type de problème comprend en outre des temps de déchargement ou de chargement des véhicules en chaque client.

Dans ce chapitre nous positionnons en premier lieu le problème avec fenêtres de temps. Nous y faisons plus particulièrement attention aux différences entre les fenêtres de temps souples et rigides.

Aux sections 2.3 et 2.4 nous introduisons successivement les deux heuristiques d'échange 2-opt* et Or-opt-1. Elles sont mieux adaptées aux fenêtres de temps que les échanges classiques dont elles s'inspirent et qui ont été présentés à la Section 1.7.

Pour finir nous présentons une heuristique d'échange hybride composée des deux méthodes 2-opt* et Or-opt et nous faisons quelques remarques sur la qualité de cette

nouvelle heuristique.

2.2 Position du problème

Soit $G = (V, E)$ un graphe complet, non orienté, avec :

- V l'ensemble des noeuds : $V = (v_0, v_1, \ldots, v_n)$
- E l'ensemble des arêtes : $E = \{(v_i, v_j) : v_i, v_j \in V, i < j\}$
- v_0 le dépôt, $v_i \ \ (i = 1, \ldots, n)$ les clients

A chaque noeud est associé :

- un poids q_i qui représente la demande de ce client $(q_0 = 0)$
- une fenêtre de temps $[e_i, l_i]$ où e_i représente l'arrivée au plus tôt et l_i l'arrivée au plus tard
- un temps de déchargement s_i des véhicules $(s_0 = 0)$

On définit également une matrice des distances $D = (d_{ij})$ définie sur E et satisfaisant à l'inégalité triangulaire. Les temps de parcours seront supposés proportionnels aux distances.

Finalement, on dispose de m véhicules identiques (afin de simplifier le problème) servant à desservir les clients. Leur capacité maximale est de Q .

Dans le cas de fenêtres de temps souples, il s'agit de trouver un ensemble de routes de coût minimal, d'origines et d'extrémités en v_0 tels que :

- chaque véhicule desserve une route
- chaque client soit visité une et une seule fois
- la quantité transportée sur chaque route soit inférieure à Q
- l'heure de départ du dépôt de chaque route soit supérieure ou égale à e_0
- l'heure d'arrivée au dépôt de chaque route soit inférieure ou égale à l_0
- l'heure du début du déchargement chez chaque client soit supérieure ou égale à e_i. Dans le cas où un camion arrive à l'heure $t_i < e_i$ chez le client v_i, on lui impose un temps d'attente de $w_i = (e_i - t_i)$.

La fonction à minimiser sur l'ensemble S des solutions réalisables est alors donnée par :

$$f(s) = \sum_{k=1}^{m} d_k + \sum_{i=1}^{n} \alpha_i (t_i - l_i)^+ \tag{2.1}$$

où $(y)^+$ signifie $\max(0, y)$, d_k est la longueur totale de la route k, α_i est un coefficient de retard associé au client v_i.

Ainsi, si un camion arrive en retard chez un client , $(t_i - l_i)$ est positif et on lui afflige la pénalité α_i proportionnellement à son retard.

Une route doit remplir deux conditions (*rigides*) :

- la charge totale sur la route doit être inférieure ou égale à Q
- chaque véhicule doit se soumettre aux conditions de la fenêtre de temps du dépôt

Par contre, en chaque noeud (client) nous rencontrons des contraintes *souples* :

- si un camion arrive trop tôt, il doit attendre avant de pouvoir décharger
- si un camion arrive trop tard, il est pénalisé pour ce retard

Les pénalités peuvent être ajustées aux exigences des clients : lorsque les délais définis dans la fenêtre de temps d'un certain client doivent être respectés à tout prix, la valeur de α_i sera importante. Souvent, le nombre de véhicules est une variable de décision et on procède à la résolution suivante : On minimise d'abord le nombre de véhicules, puis on minimise la distance totale à parcourir.

Un problème similaire au *VRPSTW* (*VRP* with soft time windows) est celui du *VRPHTW* (*VRP* with hard time windows). Comme souligné lors de l'introduction, le *VRPHTW* ne permet pas de retards.

Cependant il y a 3 avantages à résoudre le *VRPSTW* :

- 1. Le modèle du *VRPHTW* est inclut dans celui du *VRPSTW*. On résout le premier problème à l'aide du deuxième modèle en augmentant les pénalités de retard.
- 2. Le *VRPSTW* est plus approprié aux problèmes rencontrés en pratique.
- 3. Il est plus facile de trouver des solutions réalisables, étant donné que le nombre de contraintes *rigides* est moins important.

De nombreuses méthodes ont été développées pour résoudre le *VRPSTW* en un temps raisonnable. Nous présenterons d'abord des heuristiques basées sur des heuristiques d'échanges classiques et dues à Potvin et Rousseau [12]. Plus tard, nous traiterons une heuristique de type *tabou*.

2.3 Heuristique 2-opt*

Les échanges classiques r-opt présentés à la Section 1.7 ne sont pas très bien adaptés aux problèmes avec fenêtres de temps, car les échanges effectués ne préservent en général pas l'ordre des noeuds au sein d'une route. Ainsi la Figure 2.1 montre un problème typique pouvant survenir dans le cas d'un échange 2-opt. Avant l'échange on suppose

que les clients sont ordonnés dans l'ordre croissant des bornes supérieures des fenêtres de temps. Cependant, après les transformations, rien ne garantit que la solution reste admissible.

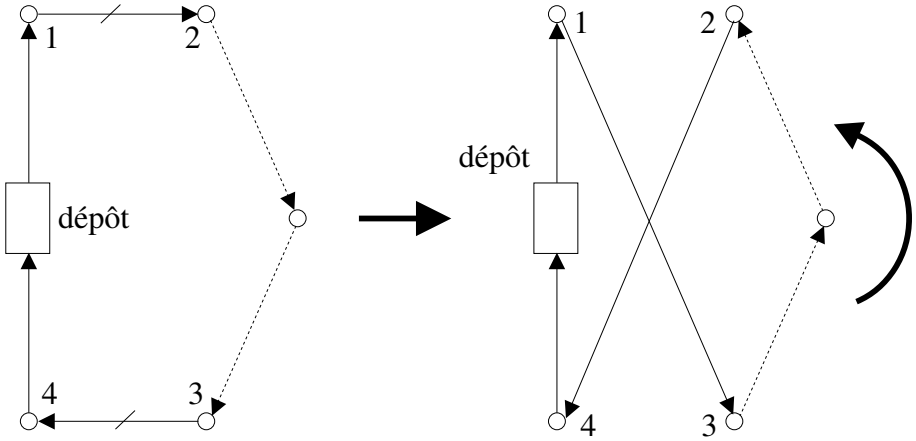

FIG. 2.1: Echange de (1,2),(3,4) contre (1,3),(2,4)

Pour un problème de routes multiples il est possible de voir un échange de deux paires de liens d'une façon différente. En premier lieu on remarque qu'un *VRP* sans contraintes de capacité sur les camions peut facilement être transformé en un problème de voyageur de commerce (*TSP*, Travelling Salesman Problem). A cette fin on crée M copies du dépôt (si M est le nombre de camions disponibles par exemple) et on les relie aux clients de manière à ce que chaque route passe par deux copies du dépôt original. En d'autres termes, après cette transformation, chaque route a son origine et son extrémité en des copies du dépôt original différentes. La Figure 2.2 illustre ceci pour deux routes.

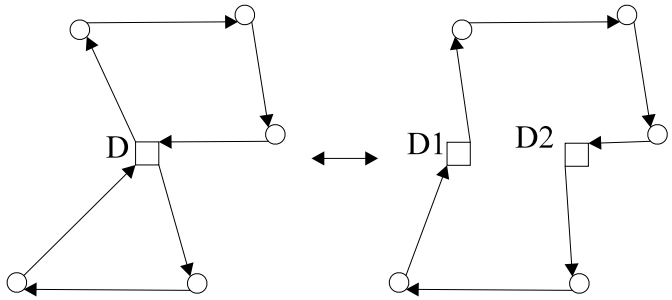

FIG. 2.2: Copie du dépôt

Vu ainsi, la solution de ce *VRP* ressemble à celle du *TSP* ainsi créé. On applique alors l'échange spécial 2-opt* de la façon suivante :

Deux liens sont remplacés par deux nouveaux liens de façon à diviser la tournée en deux sous-tournées. Une condition évidente à remplir est que chaque sous-tour devra contenir au moins une copie du dépôt.

L'avantage de ce procédé devient visible lors de tournées de véhicules avec des fenêtres de temps. Cet échange préserve en effet l'ordre dans les routes. En plus elle introduit les derniers clients d'une route (c'est à dire ceux avec des fenêtres *tardives*) derrière les premiers clients d'une autre route. De cette manière la nouvelle solution a plus de chance d'être réalisable.

La dernière étape consiste alors à recréer une tournée unique en reliant le dernier client de chaque sous-tour à l'autre copie du dépôt. Les figures 2.3 et 2.4 représentent ces deux dernières étapes.

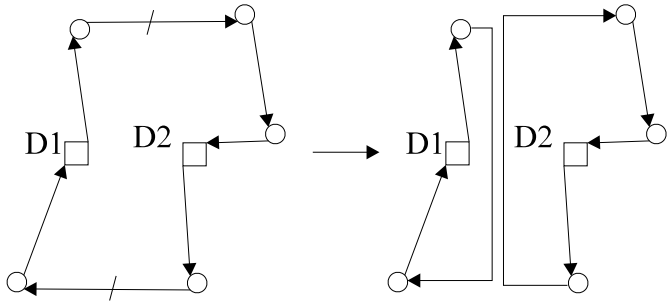

FIG. 2.3: Création de deux sous-tournées.

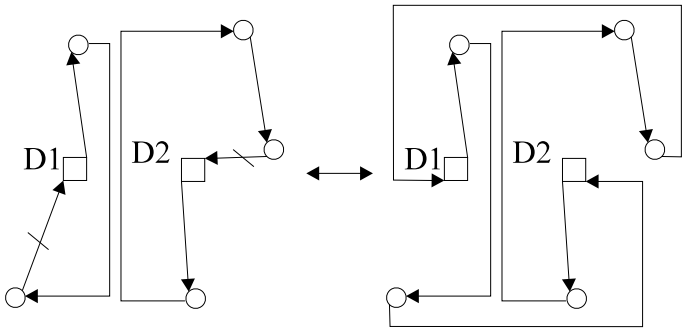

FIG. 2.4: Création d'une tournée unique

On remarque qu'après 2-opt* quatre nouvelles arêtes ont été introduites dans la solution (en incluant celles servant à recréer la tournée unique). Etant donné que les deux derniers liens dépendent des deux premiers (c'est à dire que l'on n'a pas besoin de les choisir à nouveau), cette heuristique est de l'ordre de $O(N^2)$.

D'autre part, cette heuristique n'est utile que lorsque les deux arêtes à échanger sont dans deux routes différentes. Sinon, on crée un sous-tour sans copie du dépôt.

Le plus souvent, cette méthode est complétée par une méthode 2-opt classique. On utilise cette dernière alors lorsque l'échange concerne deux arêtes d'une même route. Ceci peut être bénéfique lorsque les fenêtres de temps ne sont pas trop étroites et qu'un changement dans la route ne crée pas de solution non réalisable.

2.4 Heuristiques *Or*-opt-1 et *Or*-opt et fenêtres de temps

Nous avons déjà introduit précédemment à la Section 1.7 les heuristiques de type *Or*-opt. *Or*-opt-1 considère chaque client l'un après l'autre pour l'insérer dans différents endroits de la route afin d'améliorer cette dernière. *Or*-opt étend ce qui précède en considérant des séquences de deux ou trois clients adjacents.

On remarque que les échanges créés par ces deux méthodes forment un sous-ensemble des échanges de la méthode 3-opt. Comme indiqué ci-avant, cette dernière méthode tend à ne pas préserver l'orientation sur les routes et génère souvent des solutions qui sont non réalisables. Par contre le sous-ensemble d'échanges de *Or*-opt comprendra une majeure partie de solutions admissibles. En effet nous ne déplacerons que des séquences de plusieurs clients proches d'un point de vue temporel et spatial. Après réinsertion dans la route, l'orientation sera d'autre part préservée. De cette manière des solutions de qualité quasi égale à celles de 3-opt sont créées, tout en diminuant considérablement les temps de calculs. D'autre part, contrairement à 2-opt*, ces méthodes sont adaptées pour des échanges inter et intra-routes.

Dans la suite nous présenterons une méthode hybride qui utilise les avantages des méthodes *Or*-opt et 2-opt*.

2.5 Heuristique hybride 2-opt*/*Or*-opt

Par ce qui précède, nous remarquons que les deux méthodes 2-opt* et *Or*-opt explorent des voisinages différents. D'autre part, 2-opt* crée des modifications importantes par des échanges entre deux routes, alors qu'*Or*-opt ne modifie que légèrement la solution. En d'autres termes, les deux méthodes se déplacent différemment dans l'espace des solutions. En rassemblant ces deux approches dans une méthode hybride, nous pourrons donc explorer les deux voisinages afin d'améliorer au mieux la solution finale.

Deux implémentations de cette méthode se présentent :

- 1. Les deux voisinages 2-opt* et *Or*-opt sont explorés l'un après l'autre. Tout d'abord nous appliquons l'heuristique 2-opt* jusqu'à trouver une solution optimale pour cette méthode. Ensuite nous appliquons *Or*-opt à la solution trouvée par la première méthode jusqu'à trouver une solution optimale pour cette méthode. On répète cette procédure jusqu'à ce que la solution devient 2-opt* *et Or*-opt optimale. Une seconde étape consiste à refaire le tout à partir de la solution initiale en inversant l'ordre des deux méthodes. On choisit finalement la meilleure des deux solutions trouvées.
- 2. Les deux voisinages 2-opt* et *Or*-opt sont réunis. En d'autres termes on analyse les deux voisinages afin d'améliorer la solution courante. Dans une première étape,

les deux voisinages sont réunis de manière à considérer en premier lieu tous les échanges 2-opt* avant les échanges *Or*-opt. Dans un second essai il s'agit de repartir de la solution initiale en réunissant les deux voisinages de manière à pouvoir considérer les échanges *Or*-opt avant les échanges 2-opt*. A la fin on choisit la meilleure des deux solutions trouvées.

Une description plus détaillée de cet algorithme peut être trouvée dans Potvin et Rousseau [12].

2.6 Remarques sur la qualité de l'heuristique

Des tests effectués sur les problèmes de Solomon (voir Section 4.4 pour plus de détails) et sur des problèmes créés aléatoirement montrent la supériorité de cette méthode hybride sur la méthode classique 3-opt. D'autre part, l'heuristique hybride est assez facile à implémenter et produit des solutions de bonne qualité en des temps raisonnables. Cependant, il est clair qu'en utilisant des méthodes plus complexes (et demandant plus de temps de calcul), on peut obtenir de meilleurs résultats. Une classe de telles méthodes est la classe des heuristiques de type tabou.

Dans la suite, nous introduirons une telle méthode. Nous commencerons cependant par expliquer brièvement les heuristiques de type tabou et motiverons leur application.

Chapitre 3

Considérations générales sur les heuristiques de type tabou

Les informations rassemblées dans ce chapitre proviennent principalement des article de Gendreau [7], Soriano et Gendreau [14], Hertz et al. [9]. Nous commencerons par un bref apercu historique motivant l'introduction des méthodes de recherche avec tabous.

3.1 Contexte historique

Dès le début de la recherche d'optima par des méthodes d'amélioration locale telles que r-opt ou OR-opt, on s'est aperçu que celles-ci s'arrêtaient dès qu'elles rencontrent un optimum local. Cependant rien ne garantit qu'il n'y a pas de meilleure solution bien au-delà de cet optimum local. D'autre part, la qualité des solutions trouvées par ces méthodes dépend fortement de la qualité des transformations appliquées à la solution courante à chaque étape de l'heuristique.

En 1983, Kirkpatrick, Gelatt et Vecchi publient un article sur le *recuit simulé*, une heuristique capable de passer au-delà des optima locaux. A partir de là, toute une batterie de nouvelles heuristiques ont été développées (appelées *metaheuristiques*) et qui se basent sur des analogies avec des phénomènes naturels (algorithmes génétiques, recuit simulé). La *recherche avec tabous* fait partie de ces méthodes et fait référence à l'intelligence artificielle.

Dans ce chapitre, nous présentons la recherche avec tabous en général. Nous définissons les concepts de *tabou* et de *voisinage* d'une solution. Ensuite nous présentons un exemple pratique avant de passer à un schéma d'algorithme pour une recherche avec tabous. Pour finir nous évoquons quelques améliorations du tabou classique qui sont étudiées plus amplement dans les chapitres suivants.

3.2 Introduction

La recherche avec tabous fait partie des heuristiques dites *metaheuristiques*. En fait il s'agit d'une heuristique qui contrôle une heuristique interne propre au problème qui doit être résolu. Sa spécifité est de garder une trace des recherches déjà effectuées dans un certain nombre de mémoires, le principe de base étant de permettre des transformations (non bénéfiques à première vue) afin de surmonter des optima locaux.

D'une manière plus formelle, on explore l'espace X des solutions d'un problème d'optimisation donné, en se déplaçant d'une solution courante s, $s \in X$, à une nouvelle solution s' située dans le voisinage de s, $s' \in N(s)$. Le choix de cette nouvelle solution s' se fait par l'évaluation d'une certaine fonction objectif f.

L'espace de recherche X n'est autre que l'ensemble de toutes les solutions possibles. Cet espace n'est pas propre au tabou, étant donné qu'on l'utilise dans tous les algorithmes d'amélioration locale. Dans le cas du *VRP* il s'agit de toutes les routes possibles d'un certain problème.

Pour une solution courante s, l'heuristique interne spécifique au problème permet de définir un voisinage $N(S)$. Il s'agit en fait de l'ensemble des solutions obtenues en appliquant une transformation locale à s. Il est clair que $N(s)$ est un sous-ensemble de l'espace de recherche.

Toutefois, contrairement aux méthodes itératives classiques qui s'arrêtent dès qu'il n'y a plus de solution dans le voisinage améliorant f, dans la méthode de recherche avec tabous s' est défini comme la *meilleure* solution parmi les éléments de $N(s)$, sans faire référence à $f(s)$. Ceci permet donc de poursuivre la recherche, même si cela entraîne une dégradation de la fonction objectif.

Le problème qui se pose souvent pour ce type d'heuristiques est celui du cyclage. Par la méthode tabou on peut éviter cet inconvénient en mémorisant un historique de la recherche et en défendant des retours à des solutions précédentes (mouvements tabous). En d'autres termes on garde une trace du cheminement récent effectué à travers l'espace X. Les transformations retenues alors dans la mémoire sont déclarées *tabou*.

Afin de garder une certaine flexibilité, le caractère tabou des transformations ne sera pas maintenu en permanence. Ainsi, en limitant la durée de vie des tabous, on permet à la méthode de remettre en question ses choix passés (une fois que le risque de cyclage a disparu).

Dans la section suivante nous entrons plus en détails dans les tabous, leur gestion ainsi que leur définition. Il est clair que ces concepts jouent un rôle important dans la recherche avec tabous.

3.3 Les tabous

Le tabou est une mémoire court-terme de la recherche. En général on ne retiendra qu'un certain volume fixé à l'avance d'information. Ainsi la mémoire apparaît comme une liste T. Il existe alors plusieurs possibilités de liste :

- Une liste des $|T|$ dernières solutions rencontrées. De cette manière on interdira à la procédure d'y retourner. Ce genre de liste élimine instantanément les cycles de longueur inférieure ou égale à $|T|$. On gère la liste comme une liste *circulaire* en éliminant à chaque itération le tabou le plus ancien pour faire place au plus nouveau. Cependant on évitera souvent cette possibilité, étant donné qu'elle est peu commode et assez chère en mémoire, le volume d'information nécessaire à décrire une solution pouvant s'avérer être très important.
- Une liste des $|T|$ dernières modifications apportées à la solution courante. De cette manière on peut interdire à la procédure d'appliquer la transformation inverse d'une transformation tabou à la solution courante. Souvent on opte pour ce type de tabou. Cependant, il n'évite pas vraiment les risques de cyclages. Ainsi, la procédure pourrait effectuer la suite de transformations suivantes :

  $$(i, j) \rightarrow (j, k) \rightarrow (k, i).$$

  On pourrait imaginer dans le cas d'un *VRP* qu'une transformation du type (i, j) signifie l'échange de l'arête i par l'arête j. On voit donc bien qu'après cette suite de transformations on revient à la solution de départ. Afin d'éviter ceci, un troisième type de liste peut être utilisé.
- Une liste des caractéristiques clés des solutions ou des transformations.

On peut noter que plusieurs listes de tabous peuvent être tenues simultanément afin de conserver différents types d'informations.

Avant de nous lancer dans d'autres considérations, considérons l'exemple suivant d'un *VRP* pour illustrer nos propos :

Soit un problème de tournées de distribution à résoudre à l'aide d'une heuristique interne 2-opt. Comme décrit précédemment, les transformations de base consistent à remplacer deux arêtes $[(i, j), (k, l)]$ par les deux arêtes$[(i, k), (j, l)]$. Dans ce cas nous pourrions envisager les tabous suivants :

- Défendre le cyclage.
- Défendre les transformations inverses $[(i, j), (k, l)] \rightarrow [(i, k), (j, l)]$ pour un certain nombre d'itérations.
- Défendre toute transformation concernant les arêtes (i, k) ou (j, l) pour un certain nombre d'itérations.

Dans d'autres problèmes, des listes de longueur fixe ne peuvent prévenir le cyclage. Certains auteurs ont alors proposé des listes tabou à longueur variable au cours de

l'exécution (Skorin-Kapov, Taillard). La notion d'aléatoire peut aussi être intéressante dans ce domaine. Ainsi, dans certains cas il peut être intéressant de supprimer le statut tabou d'un mouvement après un temps aléatoire. Une autre méthode est de générer à chaque itération un nombre aléatoire qui indiquera jusqu'à quel élément il faudra considérer la liste tabou.

Une remarque importante consiste à souligner la puissance trop importante des tabous dans certains cas. En effet, parfois une transformation intéressante n'est pas tolérée, même si il n'y a aucun danger de cyclage. Dans d'autres situations, les tabous peuvent provoquer la stagnation du processus de recherche. Les *critères d'aspiration* sont la clé de tels problèmes. C'est le sujet de notre prochaine section.

3.4 Critères d'aspiration

Les critères d'aspiration sont des éléments de l'algorithme général permettant de supprimer le statut tabou d'un mouvement dans des situations bien précises.

En effet, lorsque les tabous sont définis en fonction des transformations, ils n'interdisent pas seulement de retourner à la solution précédente, mais à tout un ensemble de solutions dont plusieurs n'ont peut-être pas encore été visitées. Dans certains cas, il faut donc disposer d'un mécanisme permettant de révoquer le statut tabou d'une transformation si elle peut induire une solution intéressante, sans produire de cyclage dans le processus.

Les critères d'aspiration remplissent ainsi un rôle de contrôle. Il en existe une multitude, cependant nous n'en évoquerons que deux. On peut déjà noter ici que dans le cas d'une liste tabou composée de solutions, le concept d'aspiration n'est pas appliquable. En effet, toute annulation de tabou amène la procédure à cycler :

Deux critères d'aspiration assez répandus sont donnés par les deux méthodes suivantes : [14]

- Le statut tabou d'une transformation est révoqué si cela permet d'obtenir une solution qui améliore la meilleure solution rencontrée jusque là. On remarque que c'est un critère sévère et qu'il ne doit pas être vérifié trop souvent pour éviter trop de calculs. En plus il n'ajoute pas beaucoup de flexibilité à la méthode, mais lui évite cependant de commettre des *oublis*.
- On introduit une fonction $A(z)$ déterminant le niveau d'aspiration associé à chaque valeur z de la fonction objectif f. Etant donné une solution courante s avec $f(s) = z$, la valeur de $A(z)$ représente un seuil à atteindre si l'on veut s'assurer de ne pas cycler.

  Plus formellement, considérons le contexte d'une minimisation de f. Considérons

une transformation tabou menant de s à une solution voisine s' de valeur $f(s')$. Si $f(s') \leq A(z)$, alors on peut révoquer le statut tabou de cette transformation car cette solution n'a jamais été visitée auparavant.

Ce genre de fonction s'applique à des problèmes pour lesquels les valeurs de la fonction objectif sont entières. On initialise alors $A(z)$ par $A(z) = z - 1$ pour toutes les valeurs de z. Ensuite, à chaque fois qu'on effectue une transition de s à s' telle que $f(s') \leq A(f(s))$ on pose $A(f(s)) = f(s') - 1$.

Dans la section suivante, nous donnons plus de détails sur le voisinage d'une solution.

3.5 Evaluation du voisinage

Comme défini plus haut, le voisinage d'une solution s est noté $N(s)$. Lors du choix d'une nouvelle solution s' on devrait examiner chacune des solutions contenues dans $N(s)$ et choisir la meilleure non tabou. Autrement dit, on prend

$$s' \in \arg\min\{f(\overline{s}) \text{ tel que } \overline{s} \in N(s) \text{ et } \overline{s} \text{ non tabou}\}.$$

Parfois il est possible de tenir une liste triée des voisins de s et dans ce cas d'effectuer cette évaluation de manière implicite, sans avoir à les énumérer tous. Cependant, le plus souvent ce n'est pas le cas, et il faudrait alors évaluer complètement $N(s)$ à chaque itération. D'un point de vue temps de calcul ceci pose souvent des problèmes. En particulier lorsque le voisinage est très large ou que l'évaluation de chaque solution demande beaucoup de temps de calcul. Une solution souvent utilisée est alors de n'évaluer qu'un sous-ensemble N' de $N(s)$ généré aléatoirement. On choisit alors s' parmi les solutions contenues dans N'.

A côté de la réduction des temps de calcul, cette méthode de nature aléatoire constitue un mécanisme anti-cyclage supplémentaire complétant les listes tabous. On peut alors facilement imaginer d'utiliser des listes plus courtes que lorsque la totalité du voisinage est considérée.

Un désavantage de cette méthode est que la solution idéale pourrait ne se trouver dans aucun des sous-ensembles générés aléatoirement. Plus tard nous verrons des méthodes permettant de raffiner la recherche afin d'éviter ce désagrément. Passons tout d'abord à un schéma de l'algorithme de la recherche avec tabous.

3.6 Algorithme de base de la recherche avec tabous

Supposons que l'on considère un problème d'optimisation consistant à minimiser une fonction f sur un certain domaine X. Désignons par s la solution courante, par f^* la

valeur de la meilleure solution connue et par s^* cette meilleure solution. De plus, soit k un compteur d'itérations. On désigne par $N(s,k)$ l'ensemble des solutions admissibles à partir de s à l'itération k (c'est à dire les solutions non tabou ou dont le statut tabou a été révoqué). Etant donné que le sous-ensemble de solutions de $N(s)$ qui sont sous l'effet d'un tabou dépend de la trajectoire passée de l'algorithme (donc de l'itération k) il est censé de parler de $N(s,k)$.

Le schéma d'algorithme générique comporte deux étapes : l'initialisation et la recherche. La recherche est arrêtée lorsqu'un critère d'arrêt est satisfait. Nous revenons plus en détail sur ce critère après la présentation de l'algorithme.

Algorithme 3.1

Phase I : Initialisation

- Construire (choisir) une solution initiale $s_0 \in X$.
- Poser $s := s_0$, $f^* := f(s_0)$, $s^* := s_0$ et $k := 0$.

Phase II : Tant que le critère d'arrêt n'est pas satisfait, faire :

- $k := k + 1$.
- Choisir $s \in \arg\min_{s' \in N(s,k)}\{f(s')\}$.
- Si $f(s) < f^*$, alors $f^* := f(s)$ et $s^* := s$.
- Mettre à jour les tabous.

3.7 Critère d'arrêt

En théorie, la recherche pourrait se prolonger indéfiniment (excepté le cas où la valeur optimale du problème est connue à l'avance). Cependant en pratique il est clair que la recherche doit s'arrêter à un certain moment. Suivant le problème, plusieurs critères d'arrêts peuvent être choisis :

- arrêt après un certain nombre d'itérations (ou un certain temps) ;
- arrêt après un certain nombre d'itérations durant lesquelles la valeur de la fonction objectif n'a pas été améliorée (critère le plus utilisé) ;
- arrêt lorsque la fonction objectif atteint une certaine valeur satisfaisante.

L'algorithme de base présenté ici produit déjà des résultats impressionnants. Cependant nombreuses sont les améliorations qui peuvent être apportées à cette approche. Ainsi, dans la section suivante nous présentons comment on peut augmenter la puissance et la robustesse des méthodes de recherche avec tabous.

3.8 Améliorations

Dans la description du tabou que nous avons faite jusqu'à présent, l'usage de la mémoire n'a pas été très poussé. Son rôle s'est principalement limité à un contrôle à court terme du déroulement de l'exploration. Dans une version plus poussée de la méthode, elle influence également le processus de recherche à moyen et à long terme. Cela s'effectue à l'aide des deux concepts d'intensification et de diversification que nous étudions plus en détail dans la section suivante.

Ensuite nous nous intéressons à des raffinements tels que la gestion des listes de tabous et l'évaluation du voisinage.

3.8.1 Intensification et diversification

L'idée de base de l'*intensification* est d'explorer plus intensément certaines parties de l'espace de recherche qui semblent prometteuses. Dans ce cas, de temps en temps, le processus de recherche normal est interrompu pour laisser place à une phase d'intensification.

Plus précisément, la procédure analyse les meilleures solutions rencontrées jusqu'à présent pour voir si elles n'ont pas de caractéristiques communes. Dans le cas de l'affirmative, on procède à une modification du processus d'exploration afin de favoriser la présence de ces caractéristiques dans les nouvelles solutions visitées par l'algorithme.

Ensuite, après une durée de temps limitée, le processus normal reprend son cours.

Plusieurs stratégies d'intensification peuvent être imaginées. Nous évoquons ici celles qui sont le plus couramment utilisées :

- La méthode la plus simple consiste à retourner à l'une des meilleures solutions rencontrées jusque-là, puis à reprendre l'exploration à partir de ce point en réduisant la longueur des listes tabous pour un nombre limité d'itérations.
- On peut fixer temporairement certaines portions de la solution courante associées dans le passé à de nombreuses solutions de qualité supérieure.
- Il est possible de remplacer pour un temps limité l'heuristique interne par une autre méthode plus puissante.
- On peut intensifier la recherche en élargissant le voisinage réellement évalué à chaque itération (en augmentant par exemple la taille de l'échantillon $N'(s)$ considéré sous 3.5).

D'autres stratégies plus complexes ont été développées, mais nous ne les évoquons pas ici.

La *diversification de la recherche* consiste à forcer la recherche dans des parties de l'ensemble des solutions qui n'ont pas encore ou peu été visitées. Ceci est souvent nécessaire car dans la plupart des cas le processus de recherche se limite à une portion réduite de l'espace. Ce mécanisme se base sur une mémoire à long terme, la mémoire de fréquence qui enregistre le nombre de fois qu'un élément est apparu dans la solution courante. Plusieurs manières de procéder existent :

- La méthode la plus simple est d'interrompre périodiquement la procédure et de la faire recommencer à partir d'une nouvelle solution générée aléatoirement.
- Une méthode un peu plus évoluée se basant sur la précédente est de repartir d'une solution choisie de manière à se retrouver dans une région non encore visitée.
- Une classe de méthodes est celle de la *diversification continue*. Contrairement aux deux méthodes précédentes, ce type de stratégie de diversification peut être utilisée de façon continue tout au long de l'exploration, sans qu'il y ait besoin d'arrêter le processus.

  On peut ainsi biaiser la fonction d'évaluation en introduisant un terme qui pénalise les transformations effectuées fréquemment ou certaines caractéristiques jugées trop souvent présentes dans les solutions rencontrées jusque-là.

  De la même manière on peut utiliser cette technique dans le but de favoriser des caractéristiques rarement rencontrées.
- Certaines contraintes que doivent satisfaire les solutions peuvent être relaxées. De cette manière on peut franchir ces *obstacles* et se déplacer plus facilement vers d'autres régions. Il est évident que certaines solutions visitées alors peuvent être irréalisables. Afin de se ramener dans le domaine d'origine, on augmente graduellement les pénalités jusqu'à obtenir des solutions réalisables.

On voit donc que les deux concepts présentés ici sont des outils servant à guider intelligemment la recherche de bonnes solutions. On confère de cette manière plus de flexibilité et d'adaptabilité à la recherche avec tabous. D'autre part on augmente généralement très significativement la puissance et la robustesse de la procédure.

3.8.2 Raffinements

Il existe de nombreuses améliorations aux aspects de base introduits plus tôt. Dans cette section nous en abordons brièvement quelques-unes parmi les plus utiles.

Evaluation des listes de tabous

Nous avons vu précédemment que les listes tabous sont gérées d'une manière générale comme des listes circulaires dont la taille est fixée et prédéterminée, mais cette

méthode engendre parfois certaines difficultés. En particulier il n'est pas toujours facile de déterminer une valeur idéale pour la longueur de la liste. On se base donc souvent sur l'expérimentation. On a par exemple remarqué que la taille ne doit ni être trop grande ni trop petite. Dans le premier cas, trop de restrictions font stagner la procédure, alors que dans le second cas le méthode risque de cycler. En général il n'est pas difficile de déterminer un ordre de grandeur et même un intervalle de bonnes valeurs pour la taille de la liste. Cependant le choix d'une valeur précise donnant les meilleurs résultats est très délicat.

On réussit à contourner ce problème et à augmenter en même temps la robustesse des algorithmes en introduisant une gestion *dynamique* plutôt que statique des listes de tabous. Plusieurs versions ont été testées. Suivant le problème à résoudre elles sont plus ou moins efficaces.

- La taille des listes est modifiée à intervalles réguliers et on la fait varier sur 3 ou 4 valeurs distinctes selon une séquence prédéterminée.
- La taille des listes est modifiée périodiquement en tirant la valeur de celle-ci aléatoirement dans un intervalle de bornes fournies à la procédure.

D'autres méthodes plus complexes ont été proposées, mais nous ne nous étendrons pas plus sur ce point.

Evaluation du voisinage

On sait déjà que l'évaluation complète du voisinage d'une solution nécessite parfois de longs temps de calculs. Ainsi, évaluer complètement $N(s)$ ne constitue pas toujours une stratégie d'évaluation raisonnable.

Comme évoqué plus tôt, on peut générer un sous-ensemble $N'(s)$ aléatoirement à partir de $N(s)$. Cependant un problème peut alors apparaître : si $N(s)$ ne contient que peu de solutions de bonne qualité, alors un choix aléatoire de $N'(s)$ n'a que peu de chances de contenir des solutions candidates intéressantes. Au lieu d'améliorer les performances d'une méthode, cette stratégie risque donc plutôt d'être nuisible. En particulier on a remarqué que ceci est le cas pour les problèmes de tournées de véhicules.

Dans le chapitre prochain, lorsque nous traitons en détail une heuristique de type tabou pour le *VRP* avec fenêtres de temps, nous introduisons une manière plus efficace d'évaluer le voisinage d'une solution tout en limitant les temps de calcul.

3.8.3 Mise en oeuvre efficace

On aura compris que la recherche avec tabous se base sur des concepts assez simples. Toutefois une implémentation sur ordinateur n'est pas toujours triviale. En effet nombreuses sont les composantes pouvant intervenir dans la méthode. De plus la calibration des nombreux paramètres employés par un tel algorithme peut être très délicate. Pour qu'un algorithme de ce type soit efficace plusieurs points doivent être respectés.

Il s'agit principalement de procéder à une bonne modélisation du problème. En particulier l'efficacité dépend principalement de l'heuristique interne exploitée par la recherche avec tabous, donc du choix de l'espace des solutions X, du voisinage $N(s)$ ainsi que de la fonction objectif f permettant de choisir la prochaine solution dans $N(s)$.

Le choix de la fonction objectif f doit permettre de départager rapidement et facilement les solutions candidates à la prochaine itération.

Dans la suite nous voyons à l'aide d'un algorithme tabou adapté au *VRP* comment on adapte toutes les considérations précédentes à la pratique. Nous retrouverons la plupart des concepts introduits dans ce chapitre et verrons comment ils ont été modifiés pour augmenter leur efficacité.

Une remarque intéressante concerne l'efficacité de cette méthode. Mathématiquement, il n'existe aucune démonstration prouvant sa convergence. Cependant ceci ne doit pas dénigrer l'efficacité pratique de la méthode qui a déjà souvent été prouvée.

Chapitre 4

Heuristique de type tabou pour le VRP avec fenêtres de temps souples.

Cette heuristique a été développée par Taillard et al [15]. Par rapport aux heuristiques développées précédemment elle fournit en général de meilleures solutions. De part sa complexité, son implémentation n'est cependant pas aisée.

4.1 Introduction

L'heuristique présentée dans ce chapitre est de type tabou et est bien adaptée aux tournées de véhicules avec fenêtres de temps souples. Dans cette optique, des retards vont donc être permis, cependant ils se traduiront par une pénalité dans la fonction objectif. Comme évoqué dans la Section 2.2, le problème similaire avec des fenêtres de temps rigides se déduira facilement de ces développements en augmentant fortement les pénalités en cas de non respect des fenêtres de temps. Lors de la recherche avec tabous, un voisinage de la solution courante est créé à l'aide d'une procédure qui échange des suites de clients consécutifs entre deux routes. Cette recherche exploite également une mémoire adaptative qui contient les routes des meilleures solutions visitées précédemment. De nouveaux points de départ pour la recherche avec tabous sont alors obtenus en combinant des routes de solutions différentes stockées dans la mémoire adaptative.

Ce chapitre est organisé de la façon suivante. En premier lieu nous introduisons une nouvelle structure de voisinage par le biais d'une nouvelle heuristique d'échange. Nous remarquons dans ce point que cette méthode est particulièrement bien adaptée aux problèmes avec des fenêtres de temps. Nous détaillons de plus comment il est possible de générer le voisinage de la solution courante en des temps raisonnables. Ensuite nous présentons les grandes lignes de cet algorithme de recherche avec tabous. Nous analysons

ensuite chaque partie de cet algorithme et notamment le concept de mémoire adaptative. Pour finir nous faisons quelques remarques sur la qualité de cette méthode.

Passons désormais à la présentation d'une nouvelle structure de voisinage.

4.2 Structure de voisinage

Afin de pouvoir décrire le voisinage d'une solution courante, nous devons tout d'abord présenter l'heuristique qui nous permet de créer de nouvelles solutions. Ceci se fait à l'aide d'une heuristique d'échange en croix.

4.2.1 Nouvelle heuristique d'échange

La méthode que nous présentons ici se nomme *CROSS exchange*. Elle se base sur des méthodes d'échange classiques. Pour l'introduire nous considérons deux routes d'une solution d'un *VRP* représentées dans la Figure 4.1.

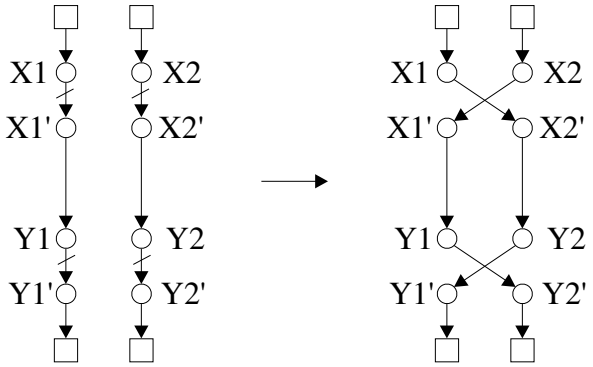

FIG. 4.1: L'échange en croix.

Les carrés représentent le dépôt et les cercles les clients. On a représenté le dépôt plusieurs fois pour faciliter les manipulations sur le graphe. La méthode d'échange en croix consiste à supprimer les arcs $(X1, X1')$ et $(Y1, Y1')$ ainsi que les arcs $(X2, X2')$ et $(Y2, Y2')$. Ensuite on échange le segment $[X1', Y1']$ de la première route contre le segment $[X2', Y2']$ de la seconde route en créant les arcs $(X1, X2')$ et $(Y1, Y2')$ ainsi que les arcs $(X2, X1')$ et $(Y2, Y1')$. Notons que les segments $[X1', Y1']$ et $[X2', Y2']$ peuvent contenir plusieurs clients.

Sur les routes initiales une orientation est définie à partir des fenêtres de temps. L'avantage principal du *CROSS exchange* est que les orientations des deux segments échangés restent les mêmes et par conséquent, les deux routes finales ont des chances de rester réalisables d'un point de vue temporel.

Cette méthode ressemble à de nombreuses méthodes existantes. Ainsi 2-opt et *Or*-opt en sont des cas particuliers. Dans un contexte général la méthode 2-opt requiert $O(n^2)$ étapes de calculs (pour un problème de taille n), étant donné qu'il y a $\frac{n(n-1)}{2}$ moyens de choisir 2 arcs et 2 possibilités de les reconnecter. Dans *Or*-opt, pour $s = 3, 2$ ou 1 on échange deux segments de s clients consécutifs. En pratique ces deux dernières méthodes sont également rapides et nécessitent le même nombre d'opérations.

Nous pouvons facilement imaginer que dans certains cas, l'échange en croix peut créer des routes vides. Mais un désavantage bien plus grand est le nombre d'étapes de calcul que requiert cette méthode. En imposant une condition sur la longueur des segments à choisir ($\leq L$) ; il y a $O(nL)$ possibilités pour ce choix (dans un problème de n clients). Au total le nombre d'étapes est donc de l'ordre de $O(n^2L^2)$.

Cependant, malgré ce désavantage flagrant, cette méthode reste particulièrement intéressante à cause de sa compatibilité avec les fenêtres de temps.

Dans le paragraphe suivant, nous introduisons des procédures de simplification et d'approximation pour réduire le nombre d'échanges et nous présentons comment est généré le voisinage de la solution courante.

4.2.2 Génération du voisinage

Dans la suite nous reprenons les mêmes notations et conventions qu'en 2.2. Etant donné un graphe complet non orienté $G = (V, E)$ avec :

V l'ensemble des noeuds : $V = (v_0, v_1, \ldots, v_n)$,
E l'ensemble des arêtes : $E = \{(v_i, v_j) : v_i, v_j \in V, i < j\}$,
v_0 le dépôt, v_i $(i = 1, \ldots, n)$ les clients,

on associe plusieurs données à chaque noeud :

- un poids q_i représentant la demande de ce client ($q_0 = 0$),
- une fenêtre de temps $[e_i, l_i]$ où e_i représente l'arrivée au plus tôt et l_i l'arrivée au plus tard,
- un temps de déchargement s_i des véhicules ($s_0 = 0$).

Une matrice des distances $D = (d_{ij})$ est également définie sur E. Elle doit satisfaire à l'inégalité triangulaire. Les temps de parcours sont supposés proportionnels aux distances.

La flotte se compose de m véhicules identiques (afin de simplifier le problème) servant à desservir les clients. Leur capacité maximale est égale à Q .

Le but de cette section est tout d'abord d'évaluer les échanges de segments par le *CROSS exchange* en utilisant des approximations. Ensuite une technique sera introduite pour éliminer des mouvements qui n'apportent aucun bénéfice.

Evaluation d'un mouvement

Pour faciliter la suite de notre exposé nous supposons que les pénalités sont toutes égales, c'est à dire, soit $\alpha = \alpha_i \quad i = 1, \ldots, n$. Dans ce cas la fonction objectif 2.1 devient :

$$f(s) = \sum_{k=1}^{m} d_k + \alpha \sum_{i=1}^{n} (t_i - l_i)^+ \qquad s \in S \tag{4.1}$$

La seconde somme de 4.1 représente donc le retard total de la solution. Pour évaluer un échange, nous nous intéressons à la différence Δf entre la valeur de la nouvelle solution et celle de la solution courante. Une amélioration a donc lieu si Δf est négatif. Soit

$$\Delta f = \Delta d + \alpha \Delta l,$$

où Δd et Δl sont les modification de la distance totale et du retard total respectivement. L'évaluation de Δd se fait en un temps constant au cours de la procédure d'échange. En effet, il suffit de soustraire la longueur des segments supprimés de la solution à celle des segments ajoutés à la solution. Par contre, l'évaluation de Δl, c'est à dire des retards totaux, nécessite un temps de calcul non constant. En effet, les conséquences d'un échange de deux segments se font ressentir d'un point de vue temporel sur tous les clients jusqu'à la fin de la route.

Calculons la contribution du retard de la route desservant les clients $X1, X2', Y2, Y1'$ (voir Figure 4.1). Il est évident que la différence du retard total sera obtenue en faisant la somme des différences du retard des deux routes séparément.

Si après introduction du segment $[X2', Y2]$ nous remarquons un retard en $X2'$, alors les clients suivants de la route considérée peuvent être affectés par ce retard. Son effet peut s'étendre au-delà du segment considéré. Afin de déterminer ce retard lors d'un échange, on effectue une approximation sur la partie non concernée directement par l'échange. L'expression de Δl pour cette route est donnée par :

$$\tilde{\Delta} l = \Delta l_{X2'-Y2} + \tilde{\Delta} l_{Y1'-\text{dépôt}} \tag{4.2}$$

où

$$\tilde{\Delta} l_{Y1'-\text{dépôt}} = g_{Y1'}(\Delta b_{Y1'}).$$

Sans entrer immédiatement dans les détails, nous pouvons cependant déjà saisir l'idée intuitive de l'expression de $\tilde{\Delta} l$. La première composante de 4.2 est la différence de retard sur le segment $[X2', Y2]$. On évalue cette grandeur exactement. Par contre le second terme sera approximé par une fonction $g_{Y1'}$.

Dans la suite nous analysons plus en détail les deux termes de cette expression. Dans un premier point nous voyons comment évaluer le retard sur le segment échangé en un temps constant. Ensuite nous détaillons l'approximation faite sur le retard de la partie de la route restante.

Evaluation exacte de $\Delta l_{X2'-Y2}$

Expliquons comment actualiser le temps de début de déchargement de chaque client du segment $[X2', Y2]$ en un temps constant. A l'aide de ce résultat, nous pourrons facilement évaluer $\Delta l_{X2'-Y2}$.

Par l'introduction du nouvel arc reliant $X1$ à $X2'$, le temps de début de déchargement $b_{X2'}$ de $X2'$ est modifié de

$$\Delta b_{X2'} = b^{\mathrm{n}}_{X2'} - b_{X2'}$$

avec

$$b^{\mathrm{n}}_{X2'} = \max\{e_{X2'}, b_{X1} + s_{X1} + t_{X1X2'}\}.$$

où $e_{X2'}$ est le début au plus tôt du déchargement en $X2'$, s_{X1} est la durée du déchargement en $X1$ et $t_{X1X2'}$ est la durée du trajet de $X1$ à $X2'$.

Désormais l'impact de l'échange sur $Y2$ peut être évalué en propageant $\Delta b_{X2'}$ le long du segment $[X2', Y2]$. Ceci est nécessaire, car des temps d'attente en différents endroits peuvent alors absorber totalement ou partiellement $\Delta b_{X2'}$.

Heureusement, l'exploration du voisinage se fait de manière à éviter trop d'étapes de calcul. Le schéma d'algorithme pour générer tout le voisinage est donné ci-après. Nous procédons à des explications dans une étape suivante.

Algorithme 4.1

- Pour $X1$ variant du *dépôt* au *dernier client* :
  Définir $X1'$ comme successeur immédiat de $X1$,
  ...
  - Pour $X2$ variant du *dépôt* au *dernier client* :
    Définir $X2'$ comme successeur immédiat de $X2$,
    ...
    - Pour $Y1$ variant de $X1$ au *dépôt* :
      Définir $Y1'$ comme successeur immédiat de $Y1$,
      ...
      - Pour $Y2$ variant de $X2$ au *dépôt* :
        Définir $Y2'$ comme successeur immédiat de $Y2$,
        ...

Explications : Afin d'illustrer le déroulement de l'algorithme, fixons à un instant donné $X1$, $X2$ et $Y1$, c'est à dire de manière à ce que l'algorithme ne tourne que dans la dernière boucle. Alors le premier segment à échanger ne contiendra que le client $Y2 = X2'$ ($Y2$ étant le successeur immédiat de $X2$ dans la deuxième étape de la dernière boucle). Ensuite, le segment est progressivement étendu par le deuxième successeur de $X2$, le troisième successeur de $X2$, et ainsi de suite.

Ainsi, Δb_{Y2} peut être évalué facilement en un temps constant en exploitant la valeur calculée lors de l'étape précédente.

On arrive donc à effectuer ces calculs en des temps raisonnables en se rappelant ce qui a été fait dans le passé.

En ce qui concerne le deuxième terme de l'expression 4.2, nous allons introduire dans la suite une méthode approximative, qui fournit de bons résultats sans effectuer tous les calculs qui seraient nécessaires à l'évaluation exacte de $\Delta l_{Y1'-\text{dépôt}}$.

Approximation de $\Delta l_{Y1'-\text{dépôt}}$

En utilisant Δb_{Y2}, on calcule facilement $\Delta b_{Y1'}$ en un temps constant à l'aide de l'arête $(Y2, Y1')$. De cette manière, on approche alors $\Delta l_{Y1'-\text{dépôt}}$ par la fonction g de 4.2. Cette fonction dépend du client v_i et se construit de la manière suivante.

En premier lieu, la différence de retard Δl_i sur la route desservant le client i dans la solution courante est évaluée d'une manière exacte pour différentes valeurs de Δb_i (que l'on notera $z_{ij}, j = 1, \ldots, Z$). Ensuite on crée une fonction linéaire par morceaux par interpolation. La figure 4.2 montre ceci pour $Z = 6$.

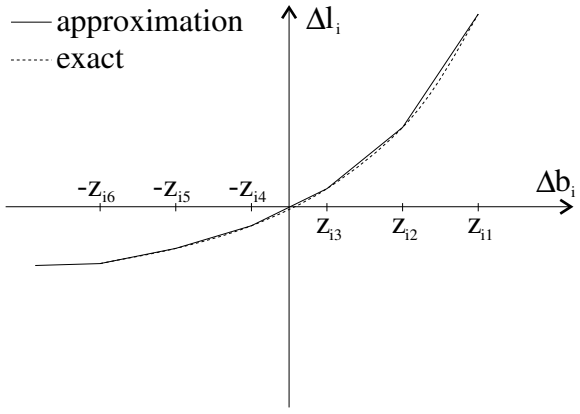

FIG. 4.2: Approximation de g_i en le client v_i

On remarque qu'en augmentant Δb_i du côté positif, c'est à dire en postposant l'instant de début de déchargement dans le temps, on augmente le retard total sur la route.

D'un autre côté, en reculant cet instant, on remarque qu'à un certain moment, l'effet devient minime.

Lors de l'évaluation du voisinage, ces fonctions restent les mêmes. On les évalue de nouveau lorsque le meilleur échange possible est appliqué à la solution courante.

Dans l'exemple de la figure, on a fixé Z à 6. Il est évident qu'un nombre plus important de valeurs nous fournirait une meilleure approximation de g_i. Cependant, des tests ont montré que $Z = 6$ est un bon compromis entre temps de calcul et résultat.

Faisabilité d'un mouvement

Les contraintes de capacité ainsi que les bornes des fenêtres de temps au dépôt sont facilement vérifiées en un temps raisonnable. Dans le second cas, on maintient le *temps de début de déchargement au plus tard* en chaque client de la solution courante. La solution d'un *CROSS*-exchange viole alors la fenêtre de temps du dépôt lorsque le temps de début de déchargement au plus tard de $Y1'$ (ou de $Y2'$) est supérieur à la limite du dépôt.

Dans la suite nous voyons comment on réussit à diminuer encore les calculs en ne considérant pas le voisinage tout entier.

Réduction du voisinage

La taille du voisinage peut être réduite en évitant des mouvements qui ne sont pas susceptibles d'améliorer une solution. A cette fin, on arrête la génération du voisinage au niveau $Y2$ lorsqu'une dégradation (monotone) de la valeur de la fonction objectif est observée sur trois itérations consécutives. On applique la même approche au niveau $X2$. Dans ce cas, la valeur de la fonction objectif associée à $X2$ est la meilleure solution trouvée après itération sur $Y1$ et $Y2$. Il est clair qu'une interruption à ce niveau est beaucoup plus bénéfique du point de vue temps de calcul qu'au niveau $Y2$.

Une interruption au niveau $X2$ a souvent lieu lorsque les segments à échanger couvrent des périodes de temps totalement différentes. Dans ce cas des retards très importants se créeraient. Il semble donc évident qu'il faille éviter de tels mouvements.

Un autre outil important et permettant de gagner en calculs est une structure mémorisant des informations sur les transformations entre les routes d'une solution. Nous la détaillerons dans le point suivant.

Matrice d'approximation

Il est également possible de réduire les temps de calcul en mémorisant les calculs précédents dans une structure appelée *matrice d'approximation*. Il s'agit d'une matrice triangulaire supérieure, dont chaque élément $(i, j), i, j = 1, \ldots, m, i < j$ est associé à une paire de routes i et j. Il contient des informations sur le meilleur échange en croix pour cette paire de routes. En fait il s'agit de la valeur de la nouvelle solution ainsi que des segments introduits ou enlevés des deux routes.

Si un échange entre les routes i' et j' est accepté, nous ne changeons que l'information contenue dans la ligne i' et la colonne j'. Ainsi nous évitons de recalculer des informations sur des paires de routes ne contenant ni i' ni j'.

Jusqu'ici les considérations ne portaient que sur des échanges entre deux routes différentes. Cependant il peut être intéressant d'effectuer des changements à l'intérieur de chaque route. Le point suivant porte sur ce problème.

4.2.3 Echanges intra-routes

Afin d'améliorer une solution, des échanges intra-routes sont également d'une grande importance. A cette fin on agrandit le voisinage *CROSS* en incluant des échanges à l'intérieur de chaque route. Ceux-ci sont similaires aux précédents : on écarte deux arêtes d'une route et on insère le segment libre ainsi créé quelque part dans cette même route.

Maintenant que nous avons décrit la structure du nouveau voisinage, nous pouvons aborder l'algorithme général de la méthode.

4.3 Schéma de l'algorithme général

Afin de mieux cerner les différents concepts qui sont introduits dans la suite, nous avons décidé de présenter en premier lieu un schéma de l'algorithme général avant de passer aux explications détaillées des différentes étapes.

L'algorithme se base principalement sur une recherche avec tabous qui prend ses informations initiales dans une structure appelée mémoire adaptative. Intuitivement il s'agit d'un tableau de routes créé à partir des meilleures solutions. Cette mémoire sert de point de départ pour toute une série d'opérations.

La résolution se fait comme suit :

Algorithme 4.2

- Etape 1 : Initialisation. Construire I différentes solutions en utilisant une heuristique d'insertion stochastique. Ensuite appliquer l'heuristique de recherche tabou à chaque solution et sauver les routes résultantes dans la mémoire adaptative.
- Etape 2 : Tant que le critère d'arrêt n'est pas satisfait, faire :
  - Construire une solution initiale à partir des routes de la mémoire adaptative et définir cette solution comme la solution courante.
  - Itérer W fois :
    - Décomposer la solution courante en C sous-ensembles de routes disjoints.
    - Appliquer une recherche tabou sur chaque sous-ensemble.
    - Reconstruire une solution complète en rassemblant les routes ainsi trouvées et définir cette solution comme la solution courante.
  - Sauver les routes de la solution courante dans la mémoire adaptative.
- Appliquer une procédure de post-optimisation à chaque route de la meilleure solution.

Le lecteur aura noté l'existence des paramètres I, W et C dans cet algorithme. Il parait clair dès à présent que suivant les valeurs allouées à ces paramètres, l'algorithme débouchera sur des solutions différentes. Dès à présent nous pouvons donc noter la difficulté de choisir ces valeurs afin d'obtenir une bonne solution en un temps raisonnable.

Passons désormais aux différentes étapes de cet algorithme général et analysons les concepts qui y ont été introduits.

4.3.1 Initialisation.

Même sans comprendre le fonctionnement exact de la mémoire adaptative à cette étape de l'exposé, le lecteur a cependant déjà une vision globale sur son rôle par les considérations faites avant l'algorithme.

Afin de remplir la mémoire adaptative, on utilise une heuristique d'insertion aléatoire pour construire I solutions initiales. La première étape consiste à initialiser les routes en choisissant au hasard m clients *noyaux*. A ce moment chaque route ne dessert qu'un et un seul client. Les clients restants sont alors choisis de manière aléatoire et insérés progressivement aux endroits minimisant les coûts d'insertion de *Solomon* qui seront détaillés dans la Section 4.5. Cette méthode d'insertion ne produit cependant pas de solutions de haute qualité. Pour les améliorer, on applique l'heuristique tabou présentée ci-après à chaque solution, avant que les routes résultantes soient stockées dans la

mémoire adaptative. Après cette phase, la mémoire contiendra une grande diversité de routes étant donné qu'elles sont extraites de différents types de solutions.

Lors de la seconde étape de l'algorithme, après la création d'une solution initiale par le processus de sélection décrit ci-après à la Section 4.3.3), nous effectuons plusieurs fois une décomposition ainsi qu'une reconstruction de la solution courante. Les raisons de cette étape ainsi que son déroulement sont traités dans le point suivant.

4.3.2 Décomposition/Reconstruction

Afin de réduire le temps de calcul, chaque solution initiale est partitionnée en C sous-ensembles disjoints de routes, chaque sous-problème étant ensuite soumis à une recherche tabou différente. Le nombre de routes de chaque sous-problème est lié à un paramètre *borne supérieure* U. Alors C est le plus petit entier supérieur ou égal à 1 tel que $\frac{m}{C} \leq U$. Ensuite, les meilleures routes trouvées pour chaque sous-problème sont rassemblées afin de créer une nouvelle solution pour le prochaine *Décomposition/Reconstruction* (D/R). Après W *D/R*, la route finale est stockée dans la mémoire adaptative si elle est vide. Sinon, les routes de la plus mauvaise solution sont effacées et remplacées par les nouvelles routes et leur solution (si biensûr la nouvelle solution est meilleure que la plus mauvaise des solutions de la mémoire).

En ce qui concerne la décomposition, il s'agit en premier lieu de rechercher le centre de gravité de chaque route. Ensuite on localise ces points par leurs coordonnées polaires. Par une procédure de balayage, on divise alors l'espace en secteurs comprenant approximativement le même nombre de routes. Afin de permettre au *CROSS*-exchange de s'attaquer sans cesse à de nouvelles paires de routes, on choisit un angle initial de référence différent avant chaque *D/R*.

La mémoire adaptative joue un rôle très important tout au long du déroulement de l'algorithme. Dans le point suivant nous détaillons plus ce concept et expliquons son utilisation lors du processus de sélection de routes.

4.3.3 La mémoire adaptative

La mémoire adaptative est un tableau de routes créé à partir des meilleures solutions rencontrées au cours de la recherche. Sa fonction est de fournir des nouvelles solutions de départ pour la recherche tabou par sélection de routes de cette mémoire.

En premier lieu, elle est partiellement remplie de routes produites lors de la procédure d'initialisation. Les routes d'une solution donnée sont stockées les unes à côté des autres en mémoire et les différentes solutions sont classés suivant la valeur de la fonction objectif

qui leur est associée. On trouvera les routes associées aux meilleures solutions dans les premières positions de la mémoire.

Le processus de sélection lors de la création de nouvelles routes est stochastique et a une préférence pour les meilleures routes. En fait, on associe une probabilité de sélection à chaque route. Lorsqu'une première route est choisie, toutes les routes de la mémoire qui ont un ou plusieurs clients en commun avec la route choisie ne seront plus considérées dans le processus de sélection. Cette procédure est répétée jusqu'au moment où les routes choisies couvrent tous les clients, ou qu'il n'y ait plus de route admissible dans la mémoire. Dans ce second cas, on utilise la procédure d'insertion $I1$ de Solomon (voir 4.5) afin de desservir le reste des clients. Au cas où il n'y aurait pas de moyen d'insérer ces clients (par exemple à cause des contraintes de capacité ou des fenêtres de temps), on les laisse de côté provisoirement. Dans la suite pourtant, on essayera de les réinsérer dans la solution courante après chaque appel de la procédure D/R.

Jusqu'ici la notion de tabou n'a pas encore été utilisée. La recherche avec tabous apparaît en fait après la décomposition de la solution en un certain nombre de sous-ensembles de routes. Nous la détaillons plus dans le point suivant.

4.3.4 Procédure de recherche tabou

La méthode de recherche tabou présentée dans la suite est assez classique et exploite la structure de voisinage présentée précédemment. Comme introduit ci-avant, elle est appliquée à un sous-ensemble de routes créé lors de la procédure D/R. On la résume de la manière suivante :

Algorithme 4.3

- Définir un sous-ensemble de routes comme la solution courante.
- Tant que le critère d'arrêt n'est pas satisfait, faire :
  - Générer le voisinage de la solution courante par des échanges en croix.
  - Choisir la meilleure solution non-tabou dans ce voisinage et la définir comme la solution courante.
  - Si la solution courante est meilleure que la solution globale alors
    - réordonner les clients à l'intérieur de chaque route par l'heuristique d'insertion $I1$ de Solomon. Définir cette solution comme la nouvelle solution courante et comme la meilleure solution globale.
  - Réactualiser la liste tabou.
- Rendre la meilleure solution globale.

Afin de mieux comprendre ce processus, voyons plus en détail les différentes étapes de cet algorithme.

Initialisation La solution initiale est créée en combinant des routes se trouvant dans la mémoire adaptative (voir précédemment). Ensuite, cet ensemble de routes est divisé en sous-ensembles, chaque sous-ensemble devant être traité ensuite par la recherche tabou.

Critère d'arrêt La recherche tabou est arrêtée après un certain nombre d'itérations. Ce nombre est donné par

$$A[1 + \frac{(DR - 1)}{B}],$$

où A et B sont deux paramètres, et DR est la D/R courante, $DR = 1, \ldots, W$. On remarque que le nombre d'itérations augmente avec DR. Comme la solution s'améliore d'une D/R à l'autre, il faut plus d'itérations pour améliorer la solution dans la dernière D/R.

Liste tabou La liste tabou a la longueur T et ses éléments sont indicés de 0 à $T - 1$. Chaque solution est associée à une position de la liste. Cette position est la valeur de la fonction objectif de la solution modulo T. La valeur stockée en cette position est le nombre d'itérations après lesquels la solution perdra son statut tabou. Lorsque l'échange en croix produit une solution de voisinage, la valeur de la fonction objectif associée modulo T donne sa position dans la liste. Lorsque la valeur trouvée en cet endroit est supérieure au numéro de la dernière itération, cet échange est considéré comme tabou.

Moins formellement, il est ainsi possible de filtrer les solutions valables.

Deux solutions se heurtent à la même position de la liste si leur valeurs objectives diffèrent d'un multiple de T. Cependant on choisit T assez grand pour éviter ce genre d'inconvénients. On notera aussi que la valeur de la fonction objectif est entière, car on se sera assuré que les distances réelles ont été transformées en entiers.

Diversification La diversification dynamique est incluse dans la recherche avec tabous en pénalisant des échanges en croix qui sont souvent répétés durant la recherche.

Soit f_e la fréquence d'un échange e, soit $f_{\max}$ la fréquence maximale observée, soit i le numéro de l'itération courante, soit m le nombre de clients et soit n le nombre de routes. Soit x un nombre aléatoire choisi uniformément dans l'intervalle $]0,0;0,5]$ et soit $\Delta_i^{\max}$ la différence maximale entre les valeurs de la fonction objectif de deux solutions consécutives, au cours des i premières itérations. Alors l'échange e est pénalisé de

$$x\Delta_i^{\max}\frac{f_e}{f_{\max}}.$$

La valeur $f_{\max}$ normalise les fréquences, car celles-ci tendent à diminuer pour des voisinages importants et à augmenter pour des voisinages moins importants. Le facteur

$\Delta_i^{\max}$ sert à ajuster la pénalité aux magnitudes des mouvements. On obtient ainsi des pénalités adaptées à la fréquence des différents échanges.

Réordonnement des clients de chaque route. Lorsqu'une meilleure solution globale est trouvée, les clients à l'intérieur de chaque route sont réorganisés. Ce réordonnement est basé sur l'heuristique d'insertion $I1$ de Solomon. Pour une route donnée, on procède de la manière suivante : En premier lieu, tous les clients de la route sont considérés comme non desservis Ensuite le client le plus éloigné du dépôt est choisi comme *noyau* de la route. A cette étape, le camion ne livre que ce client et retourne au dépôt. Le client suivant à être inséré dans cette route est alors celui qui maximise un critère de gain général (dû à Clarke et Wright, mais qui est un cas particulier de $I1$ comme nous le montrons dans 4.5). L'endroit d'insertion est également donné par $I1$. L'heuristique de Solomon est alors répétée R fois avec des paramètres différents et la meilleure solution est choisie à la fin. La route initiale est reprise si cette méthode ne produit pas de meilleure solution ou une solution non réalisable.

Dans la suite nous parlons brièvement de l'efficacité de la méthode de résolution introduite dans ce chapitre.

4.4 Remarques sur la qualité de l'heuristique

Les problèmes tests de Solomon. En général, les algorithmes servant à résoudre ce genre de problèmes sont testés sur des problèmes types créés par Solomon [13]. De cette manière il est possible de comparer les différentes méthodes en termes d'efficacité, de rapidité et de robustesse. Les problèmes de Solomon sont des problèmes euclidiens à 100 clients. Les durées de déplacements correspondent aux distances euclidiennes. Six types de problèmes ont été définis : *C1, R1, RC1, C2, R2, RC2*. Dans les problèmes C les clients sont regroupés en certains endroits du plan alors que dans les problèmes de type R ils sont répartis d'une manière uniforme. D'autre part la fenêtre de temps du dépôt est plus étroite dans les problèmes de type 1 que dans ceux de type 2. De cette manière on impose dans les problèmes 1 que le nombre de clients desservis par une route soit petit. Finalement chaque client dispose d'un temps de déchargement (10 unités de temps pour les problèmes R ou RC, 90 u.d.t. pour les problèmes C).

En général, cette méthode produit des résultats de bonne qualité. Une question se posant est celle de l'efficacité de l'échange en croix. En insérant des heuristiques comme 2-opt* (voir 2.3) ou *Or*-opt dans la méthode à l'endroit de *CROSS exchange* on remarque que cette dernière heuristique coûte cher en temps de calcul. Cependant, après des temps égaux, la qualité des solutions donnée par l'heuristique utilisant l'échange en croix est clairement supérieure.

4.5 Heuristique d'insertion de Solomon

Comme son nom l'indique cette méthode est due à Solomon [13]. En premier lieu nous présentons un Lemme qui permet l'insertion facile d'un client dans une route donnée. Ensuite nous décrivons la méthode d'insertion en question, qui est en fait une heuristique de création de routes.

Nous reprenons une nouvelle fois les notations de 2.2.

Considérons une route réalisable $(v_{i_1}, \ldots, v_{i_m})$ (avec $v_{i_1} = v_{i_m} = v_0$) pour laquelle les temps de début de déchargement b_{i_r} sont connus pour $r = 1, \ldots, m$. On insère le client u entre $v_{i_{p-1}}$ et v_{i_p}. Soit $b^n_{i_p}$ le nouveau temps de début de déchargement chez le client v_{i_p}. Soit w_{i_r} le temps d'attente chez le client v_{i_r} pour $p \leq r \leq m$.

Dans ce cas, l'insertion de u provoque un *avancement* dans l'horaire de v_{i_p} de

$$A_{i_p} = b^n_{i_p} - b_{i_p} \geq 0.$$

D'autre part,

$$A_{i_{r+1}} = \max(0, A_{i_r} - w_{i_{r+1}}), \quad p \leq r \leq m-1.$$

Si $A_{i_p} > 0$ certains des clients v_{i_r} pourraient devenir *non réalisables* du point de vue temporel, c'est à dire que leurs fenêtres de temps ne seraient plus compatibles avec les heures d'arrivées du véhicule. Il faut alors examiner les clients séquentiellement jusqu'à en trouver $(v_{i_r}, r < m)$ tels que

$$A_{i_r} = 0 \text{ ou } v_{i_r} \text{ est non réalisable ou tous les } v_{i_r}, p \leq r \leq m \text{ soient examinés.}$$

On en déduit le résultat suivant.

Lemme 4.1 *Les conditions nécessaires et suffisantes pour la réalisabilité temporelle lors de l'insertion de u entre $v_{i_{p-1}}$ et v_{i_p} $(1 \leq p \leq m)$ dans une route réalisable $(v_{i_0}, \ldots, v_{i_m})$ sont*

$$b_u \leq l_u \text{ et } b_{i_r} + A_{i_r} \leq l_{i_r} \quad p \leq r \leq m.$$

La méthode d'insertion en elle même utilise deux critères d'insertion $c_1(v_i, u, v_j)$ et $c_2(v_i, u, v_j)$ à chaque itération pour introduire le client u entre deux clients adjacents v_i et v_j d'une même route.

Soit $(v_{i_1}, \ldots, v_{i_m})$ (avec $v_{i_1} = v_{i_m} = v_0$ la route courante. Pour chaque client non encore inséré nous calculons le meilleur endroit d'insertion dans la route courante comme suit :

$$c_1(v_i(u), u, v_j(u)) = \min[c_1(v_{i_{p-1}}, u, v_{i_p})], \quad p = 1, \ldots, m.$$

Comme précisé précédemment, cette insertion pourrait créer des incompatibilités avec les fenêtres de temps existantes. Cependant des tests effectués par Solomon [13] ont montré qu'il suffisait d'appliquer le lemme 4.1 au lieu de tester tous clients après l'endroit d'insertion de u.

L'étape suivante consiste à choisir le meilleur client u parmi les clients non encore insérés. On choisit celui pour lequel

$$c_2(v_i(u^*), u^*, v_j(u^*)) = \text{optimum}[c_2(v_i(u), u, v_j(u))],$$

u non encore inséré et réalisable temporellement. Le client u^* est alors inséré entre $v_i(u^*)$ et $v_j(u^*)$. Lorsque tous les clients réalisables temporellement ont été insérés, la méthode démarre une autre route ou s'arrête si tous les clients font partie d'une route.

Considérons à présent les critères d'insertion plus en détail. Solomon [13] propose le critère $I1$ suivant :

$$c_{11}(v_i, u, v_j) = d_{iu} + d_{uj} - \mu d_{ij}, \quad \mu \geq 0;$$
$$c_{12}(v_i, u, v_j) = b_{j_u} - b_j,$$

où b_{j_u} est le nouveau temps de début de déchargement pour le client v_j étant donné que u fait aussi partie de la route.
Alors

$$c_1(v_i, u, v_j) = \alpha_1 c_{11}(v_i, u, v_j) + \alpha_2 c_{12}(v_i, u, v_j), \quad \alpha_1 + \alpha_2 = 1, \alpha_1 \geq 0, \alpha_2 \geq 0;$$

$$c_2(v_i, u, v_j) = \lambda d_{0u} - c_1(v_i, u, v_j), \quad \lambda \geq 0.$$

Pour illustrer ceci considérons comme exemple le cas où $\mu = \alpha_1 = \lambda = 1$ et $\alpha_2 = 0$. Dans ce cas $c_2(v_i, u, v_j)$ est le gain de distance lorsque u est desservi par la même route que v_i et v_j au lieu d'être desservi séparément. Ainsi, le meilleur endroit d'insertion est celui qui minimise la combinaison pondérée de sa *distance* et de son *temps d'insertion*, c'est à dire celui qui minimise une mesure de la distance et du temps supplémentaires nécessaire à la livraison du client. Il est clair que des valeurs différentes de μ et de λ conduisent à des critères d'insertion différents et influencent ainsi le choix du client à insérer et de l'endroit d'insertion.

En ce qui concerne la qualité de cette heuristique d'insertion, des tests sur un certain nombre de problèmes ont prouvé sa supériorité en temps de calcul et en qualité du résultat par rapport à des heuristiques telles que l'algorithme de balayage ou le *savings algorithm* présentés ou évoqués en 1.6.

Deuxième partie

Partie pratique

Chapitre 5

Application

5.1 Introduction

Dans ce chapitre nous traitons le cas concret d'une entreprise effectuant des tournées de véhicules. Il s'agit d'une entreprise de livraison de repas scolaires située dans le Nord de la France près de Lille. Depuis la création de l'entreprise, les tournées se basent sur l'expérience d'une personne. Notre but est d'améliorer la longueur totale de ces routes à l'aide de l'algorithme présenté au Chapitre 4.

Dans un premier point nous présentons le fonctionnement du programme de base utilisant l'algorithme développé dans le Chapitre 4. Nous y détaillons les deux fichiers les plus importants utilisés par le logiciel et modifiables par l'utilisateur : le fichier problème contenant entre autres les coordonnées des clients et le fichier *config* qui, comme son nom l'indique, sert à configurer les paramètres utilisés par le programme. Ensuite, nous exposons le problème ainsi que les différentes contraintes se posant lors des tournées. Le déroulement de nos travaux avant le lancement du programme est décrit dans la Section 5.4. Nous y détaillons plus particulièrement comment nous en sommes arrivés à modifier la version de base du logiciel ainsi que la détermination d'une vitesse moyenne sur des arcs euclidiens créés par le programme. Pour finir nous présentons les résultats et passons aux conclusions directes en découlant.

5.2 Le programme dyn

Le programme que nous utilisons nous a été fourni par M. Gendreau et F. Guertin de l'Université de Montréal ; nous le nommerons *dyn* dans la suite. L'algorithme à la base du logiciel est celui présenté au Chapitre 4. Le programme utilise des données en coordonnées euclidiennes. Nous avons utilisé les problèmes tests de Solomon décrits à la

Section 4.4 afin de mieux comprendre le fonctionnement du programme. Le langage de programmation utilisé est le C++ et les résultats présentés dans la suite ont été obtenus sur un micro-ordinateur de type PC muni d'un processeur Intel Pentium II 266.

Au début de son exécution, le programme charge des informations nécessaires à son initialisation dans deux fichiers : le fichier *config* ainsi que le fichier problème.

Le fichier *config*

Le programme fait appel au fichier *config* en début d'exécution afin de déterminer la valeur des paramètres rencontrés au cours de l'exécution (voir Chapitre 4 pour une description détaillée de ces paramètres). Il contient en particulier la longueur de la liste tabou (300), la pénalité infligée aux camions pour un retard (100), le nombre de décompositions lors de la procédure D/R (3), la longueur maximale des segments à échanger lors de l'échange en croix (5) et le nombre d'appels à la mémoire adaptative (100). Les nombres entre parenthèses sont les valeurs que nous avons utilisées lors de lors de la résolution du problème qui nous intéresse. Ces valeurs correspondent aux valeurs données par les concepteurs du logiciel, excepté pour le nombre de D/R qui valait 1.

Nous remarquons ici qu'en augmentant certains de ces paramètres, on peut espérer trouver de meilleures solutions. En particulier, le nombre de D/R, la longueur maximale des segments à échanger, le nombre d'appels à la mémoire adaptative ainsi que la longueur de la liste tabou jouent un rôle important dans la qualité de la solution finale ainsi que dans le temps d'exécution du programme.

Pour référence, en prenant les valeurs des paramètres indiquées entre parenthèses pour un fichier d'une centaine de clients avec des fenêtres de temps assez larges, l'exécution de *dyn* dure environ une heure sur un micro-ordinateur usuel (PC muni d'un processeur Intel Pentium II, 266).

Un autre fichier important au bon fonctionnement de *dyn* est le fichier problème comportant les données du problème.

Le fichier problème

Dans la suite nous nous référons à la Figure 5.1 qui illustre la structure générale d'un fichier problème. Nous rappelons que *dyn* utilise exclusivement les coordonnées euclidiennes des clients et par conséquent des distances euclidiennes lors de l'élaboration des routes.

La version de base du programme suppose que les distances entre les villes équivalent

aux temps de parcours correspondants.

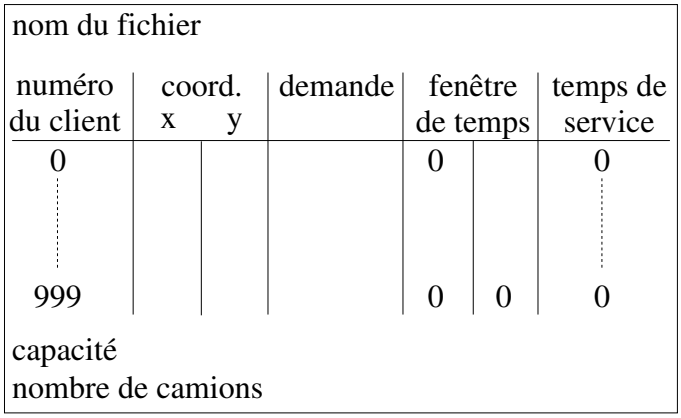

FIG. 5.1: un fichier-problème type

La première colonne contient le numéro du client. Nous supposons que le dépôt porte le numéro 0. Les deuxième et troisième colonnes contiennent les coordonnées euclidiennes alors que dans la quatrième colonne nous introduisons les demandes des clients. La demande du dépôt vaut 0. Les deux colonnes suivantes contiennent les bornes des fenêtres de temps tandis que la dernière détaille le temps nécessaire au service de chaque client.

Durant l'exécution de *dyn*, il est possible d'envoyer la "sortie" vers un fichier mais aussi d'observer graphiquement le déroulement du programme. Grâce à deux fenêtres graphiques, on peut visualiser l'allure de la meilleure solution obtenue jusqu'à là ainsi que les tentatives de modifications des routes courantes. Cependant pour notre cas, le plus intéressant est la sauvegarde des données fournies par *dyn* dans un fichier de texte.

Le programme présente deux inconvénients majeurs : les coordonnées euclidiennes qu'il nécessite ainsi que la lenteur de son exécution.

A priori, le premier point limite donc son domaine d'application. En effet il est impensable que la distance réelle par route entre deux clients corresponde à la distance à vol d'oiseau (c'est à dire euclidienne). Nous résolvons ce problème par une approche statistique du rapport entre les distances réelles et les distances euclidiennes.

En ce qui concerne le temps d'exécution du problème, il est clair que pour un problème où les clients changent journalièrement, la rapidité joue un grand rôle. Dans le cas de livraisons de marchandises, on ne pourrait concevoir un programme dont l'exécution prendrait plusieurs heures, alors que le temps entre l'introduction des commandes dans l'ordinateur et le début des premières livraisons est très limité. Cependant nous verrons plus loin que cette contrainte ne nous affecte pas, étant donné que les tournées ne changent pas au cours de l'année et qu'une révision des clients à livrer ne se fait que rarement.

Au début de l'exécution, le programme crée une matrice d'intertemps entre les villes à partir des coordonnées euclidiennes introduites dans le fichier problème. En d'autres termes, le programme suppose que le temps de parcours entre les villes est proportionnel aux distances euclidiennes correspondantes. A partir de cet instant, *dyn* utilise cette matrice. Nous verrons plus loin les problèmes engendrés par cette manière de procéder.

Avant de détailler notre raisonnement, nous allons introduire le problème à traiter.

5.3 Position du problème

L'entreprise en question doit livrer des repas de cantine à des écoles situées dans le Nord de la France et en Belgique. Elle dispose d'une flotte de 23 camions dont un est utilisé comme camion de dépannage en cas de problème. L'organisation des tournées de distribution se fait sans aucune aide informatique et se base principalement sur l'expérience de la personne responsable de la création des routes.

En tout, l'entreprise livre journalièrement 400 écoles, ce qui correspond à 24000 repas et 5000 km parcourus au total chaque jour.

L'entreprise a une place prépondérante dans le secteur de la restauration scolaire dans la région du Nord-Pas-de-Calais.

Les clients de l'entreprise introduisent leurs commandes la veille de leur production. Ces commandes changent de jour en jour, cependant leur variation est trop minime pour influencer la création des tournées. Cela signifie qu'il suffit que nous nous intéressions à une journée type lors de l'élaborations des tournées.

Les contraintes spécifiques à ce problème sont de deux types : une contrainte de capacité sur les camions et des contraintes de temps propres aux clients.

Dans la suite nous détaillons ces deux types de contraintes.

La contrainte de capacité sur les camions

Les camions sont des camions frigorifiques de petite taille. Leur capacité est approximativement identique : environ 1300 repas par véhicule. Ceci leur permet de circuler sur n'importe quelle route ; ils ne sont en outre pas soumis à des contraintes horaires imposées au passage d'engins plus lourds dans certains lieux publics. Comme il n'est pas rentable de faire partir un camion trop peu chargé, l'entreprise tend à respecter une capacité minimale de 1000 repas par véhicule.

Les contraintes de temps

Les tournées débutent la nuit vers 4h30. On veille à livrer les derniers repas aux alentours de 11h afin de permettre aux différentes cantines de les réchauffer avant l'arrivée des premiers élèves.

Afin de laisser une certaine marge aux livreurs en cas de problème, l'heure limite de livraison est fixée à 10h45.

L'entreprise dispose de clés lui permettant de délivrer la marchandise dans la plupart des écoles sans la présence d'un employé. Dans ce cas, la livraison peut être effectuée à n'importe quelle heure entre 4h30 et 10h45. Cependant, à certains endroits, les clients désirent être livrés entre 8h15 et 10h45 et ne laissent pas de clé à l'entreprise.

Une autre donnée temporelle est celle des temps de déchargement auprès de chaque client. Ces temps varient avec le nombre de repas à livrer. Nous compterons 5 minutes par tranche de 65 repas, selon l'estimation de la personne responsable de l'élaboration des tournées.

Position du problème : résumé

En résumé, le problème d'optimisation que nous traitons ici consiste à minimiser la distance totale parcourue par les camions tout en respectant les contraintes suivantes :

- Chaque client (école) est livré une et une seule fois.
- Chaque route débute et se termine au dépôt.
- Les tournées débutent au plus tôt à 4h30.
- Les camions doivent être rentrés au dépôt avant 12h30.
- Les clients *avec clés* doivent être livrés dans l'intervalle de temps $[4\text{h}30, 10\text{h}45]$.
- Les clients *sans clé* doivent être livrés dans l'intervalle de temps $[8\text{h}15, 10\text{h}45]$.
- La capacité de 1200 repas par camion ne doit pas être dépassée (afin de laisser une marge en cas de journée plus chargée).

Afin d'évaluer nos performances, l'entreprise nous a fourni six routes se situant dans une même région au sud de Lille. L'objectif principal est de diminuer la distance totale parcourue par les six camions. Pour l'heure, les camions partent du dépôt avec une charge proche de la capacité maximale. Nous ne pouvons donc pas espérer diminuer le nombre de routes et par conséquent le nombre de véhicules.

Dans la suite nous détaillons les différentes étapes nécessaires à la résolution du problème. Elles se divisent en deux classes : d'une part la modification du programme

de base et d'autre part l'adaptation du problème concret aux contraintes imposées par *dyn*.

5.4 Déroulement des travaux

Modification du programme de base

Comme nous l'avons déjà expliqué, le logiciel *dyn* utilise des temps de parcours entre deux villes basés sur les distances euclidiennes. Cependant l'application à traiter n'est pas de nature euclidienne. Une solution serait de transformer la matrice des interdistances euclidiennes en une matrice d'interdistances réelles. L'utilisateur devrait donc introduire non plus les coordonnées des clients, mais la distance réelle (ou plutôt les temps de parcours) entre toute paire de clients. Dans ce cas, trois problèmes majeurs se posent.

- Ces temps de parcours varient suivant l'heure à laquelle on les considère (heures creuses, heures de pointe).
- Nous ne connaissons pas les temps de parcours exacts entre toute paire de clients.
- Le nombre de valeurs à introduire dans cette matrice des intertemps serait très important. Pour le problème que nous considérons, le nombre de clients est de l'ordre de cent. En supposant les temps de parcours égaux pour l'aller et le retour, l'utilisateur devrait donc introduire près de 5000 éléments.

Nous ne retiendrons donc pas cette technique trop contraignante. Nous nous décidons donc à utiliser les coordonnées euclidiennes pour localiser les clients.

Il est clair qu'il existe une grande différence entre les distances euclidiennes que *dyn* utilise et les distances réelles entre les villes. Cependant nous remarquons que le réseau routier est bien développé dans cette région. Ainsi, la plupart des villes sont directement reliées aux villes voisines. Cet argument géographique nous permet de penser qu'il existe une relation entre les distances euclidiennes et réelles qui nous permettrait, à partir des distances euclidiennes, de calculer des temps de parcours proches de la réalité. Pour ce faire nous allons estimer une vitesse moyenne des camions sur les arcs euclidiens virtuels créés par le programme.

Nous recueillons les coordonnées des villes par rapport à un point fixé sur une carte du Nord de la France à l'échelle 1 :250000 (1 cm correspond donc à 2500m). Comme le programme de base utilise les coordonnées euclidiennes pour déterminer les temps de parcours entre deux villes (1 unité correspondant à 1 minute), les distances recueillies par transformation des coordonnées correspondent à une vitesse moyenne de 75 km/h. Cette vitesse est cependant trop élevée pour deux raisons évidentes :

- la plupart du temps les camions circulent sur des routes secondaires ou dans des villages,

– les distances euclidiennes entre les villes étant inférieures aux distances réelles, une vitesse moyenne inférieure à la vitesse moyenne réelle doit être utilisée afin d'égaler les temps de parcours sur les deux types d'arcs différents

Nous modifions donc le programme en introduisant un facteur de dilatation aux endroits où les distances euclidiennes sont utilisées. De cette manière, nous pouvons augmenter ou diminuer la vitesse des camions sans modifier les coordonnées introduites ou les bornes des fenêtres de temps. Afin de laisser toute sa flexibilité au programme et de faciliter son emploi à l'utilisateur, la valeur de ce paramètre de vitesse peut être changé dans le fichier de configuration. Il porte le nom de *speed* et représente le rapport de la vitesse moyenne implicitement définie par les coordonnées (ici 75 km/h) à la vitesse moyenne désirée. Il s'agit donc dans la suite de déterminer une vitesse moyenne valable sur l'ensemble des villes visitées par les 6 tournées.

Détermination d'une vitesse moyenne sur les arcs virtuels euclidiens

Afin d'obtenir une vitesse moyenne sur les arcs virtuels euclidiens, nous générons aléatoirement deux échantillons de villes appartenant à l'ensemble de toutes les villes visitées par les six routes.

Pour chaque échantillon, nous recherchons les temps de parcours, les distances réelles ainsi que les distances euclidiennes entre une vingtaine de couples de villes. Pour chaque couple nous calculons alors la vitesse moyenne réelle ainsi que la vitesse moyenne virtuelle. Ensuite nous calculons le rapport entre la vitesse moyenne réelle et la vitesse moyenne virtuelle.

Pour les distances réelles ainsi que les temps de parcours, nous utilisons le logiciel *Microsoft Autoroute Express 98*. Dans ce programme il est possible de changer les vitesses moyennes sur les différents types de routes (autoroute, route nationale, etc.). Afin de rendre compte de la réalité, nous procédons de la manière suivante :

> Nous introduisons les routes originales fournies par l'entreprise ainsi que les temps de déchargement dans le logiciel *Microsoft Autoroute Express 98*. Ensuite nous modifions les vitesses moyennes sur les différents types de routes de manière à obtenir une bonne approximation de la réalité : respect des fenêtres de temps et retour au dépôt avant 12h30.

De cette manière nous arrivons aux résultats repris dans les tableaux 5.1 et 5.2 ((e) et (r) correspondent aux données euclidiennes et réelles respectivement).

Les rapports $\frac{vm(r)}{vm(e)}$ multipliés par mille sont repris dans le Tableau 5.3 pour les deux échantillons.

Echantillon 1					
temps (min)(r)	km (r)	km (e)	v.m. (km/h)(r)	v.m. (km/h)(e)	$\frac{vm(r)}{vm(e)}$
55	87,2	43,86	95,12	47,84	1,9881
56	65,0	38,25	69,64	40,98	1,6993
26	19,2	14,11	44,30	32,56	1,3607
49	51,1	33,49	62,57	41,00	1,5258
58	64,3	39,35	66,51	40,71	1,6338
20	20,2	13,94	60,60	41,82	1,4490
26	27,3	16,06	63,00	37,07	1,6993
13	12,4	6,55	57,23	30,24	1,8921
39	43,2	26,26	66,46	40,40	1,6447
55	61,7	42,84	67,30	46,73	1,4402
25	27,8	14,87	66,72	35,70	1,8689
34	39,2	23,97	69,18	42,30	1,6357
18	15,6	8,67	52,00	28,90	1,7993
31	40,1	23,80	77,61	46,06	1,6849
11	10,5	7,51	57,27	40,98	1,3973
10	11,8	6,93	70,80	41,61	1,7012
23	23,6	16,57	61,56	43,23	1,4238
36	33,1	22,44	55,16	37,40	1,4750
26	36,6	21,33	84,46	49,23	1,7154
15	17,3	10,54	69,20	42,16	1,6414
moyenne			64,46	40,34	1,6048

TAB. 5.1: Données pour l'échantillon 1

Echantillon 2					
km (r)	temps (min)(r)	km (e)	v.m. (km/h)(r)	v.m. (km/h)(e)	$\frac{vm(r)}{vm(e)}$
41,8	38	27,60	66,00	43,57	1,5144
32,1	31	20,24	62,12	39,17	1,5859
89,7	62	47,04	86,80	45,52	1,9068
83	63	52,96	79,04	50,43	1,5672
31,5	32	18,08	59,06	33,90	1,7422
26,9	25	19,44	64,56	46,65	1,3837
86,5	78	53,84	66,53	41,41	1,6066
42,3	35	29,84	72,51	51,15	1,4175
15,2	16	7,62	57,00	28,59	1,9937
78,6	57	46,80	82,73	49,26	1,6794
9,2	10	5,60	55,20	33,60	1,6428
40,7	38	26,08	64,26	41,17	1,5605

suite à la page suivante

suite de la page précédente					
km (r)	temps (min)(r)	km (e)	v.m. (km/h)(r)	v.m. (km/h)(e)	$\frac{vm(r)}{vm(e)}$
37,2	39	24,72	57,23	38,03	1,5048
40,4	39	26,56	62,15	40,86	1,5210
92,9	79	58,24	70,55	44,23	1,5951
13,9	13	7,22	64,15	33,34	1,9241
64	54	45,12	71,11	50,13	1,4184
18	17	11,60	63,52	40,94	1,5517
31,2	29	18,24	64,55	37,73	1,7105
60,5	48	34,88	75,62	43,60	1,7345
moyenne			67,23	41,66	1,6280

TAB. 5.2: Données pour l'échantillon 2

éch. 1	éch. 2
1988	1514
1699	1586
1361	1907
1526	1567
1634	1742
1449	1384
1699	1607
1892	1418
1645	1994
1440	1679
1869	1643
1635	1561
1799	1505
1685	1521
1397	1595
1701	1924
1424	1418
1475	1552
1715	1711
1641	1735

TAB. 5.3: Résumé

A partir des tableaux 5.1 et 5.2, nous pouvons déterminer une vitesse moyenne sur les arcs virtuels euclidiens. Elle vaut $\text{vm(e)} = \frac{\text{vm(e)}_1+\text{vm(e)}_2}{2} = \frac{42,16+40,39}{2} = 41,28(km/h)$. Cependant, afin de vérifier la validité de ce résultat, nous effectuons plusieurs tests sur les deux échantillons. Notre but est de vérifier statistiquement que la variation autour de cette vitesse moyenne est faible et que les vitesses moyennes obtenues à partir de chacun

de ces deux échantillons sont statistiquement égales.

Les développements futurs se structureront comme suit : en premier lieu nous vérifions la normalité des populations dont les échantillons sont extraits. Ensuite, un test d'égalité des variances ainsi que des moyennes nous confirmera l'existence d'un facteur multiplicatif entre les vitesses moyennes euclidiennes et réelles valable pour toute la population composée de la centaine de clients des six routes considérées. Pour finir, nous calculerons les coefficients de variation dans les deux cas. Cet indice de dispersion $V = \frac{\sigma}{m}$ est en effet très utile car il a un caractère relatif et est sans unité.

Validité de la vitesse moyenne euclidienne

Commençons par soumettre les deux échantillons aux tests statistiques suivants :

- test de normalité
- test d'égalité des variances
- test d'égalité des moyennes

En premier lieu, vérifions l'hypothèse de normalité des populations.

Tests de normalité

Grâce au programme *Statlets* (www.statlets.com) nous créons un tableau de fréquences en divisant l'intervalle de valeurs en des classes de longueurs égales et en comptant le nombre d'observations dans chaque intervalle. Les résultats de ces calculs sont repris dans les tableaux 5.4 et 5.5 ainsi que sur la Figure 5.2.

Echantillon 1			
classe	inf.	sup.	fréq.
	<1300.0		0
1	1300.0	1433.33	3
2	1433.33	1566.67	4
3	1566.67	1700.0	7
4	1700.0	1833.33	3
5	1833.33	1966.67	2
6	1966.67	2100.0	1
	>2100.0		0

TAB. 5.4: Echantillon 1 : Tableau des fréquences

Echantillon 2			
classe	inf.	sup.	fréq.
	<1300.0		0
1	1300.0	1433.33	3
2	1433.33	1566.67	5
3	1566.67	1700.0	6
4	1700.0	1833.33	3
5	1833.33	1966.67	2
6	1966.67	2100.0	1
	>2100.0		0

TAB. 5.5: Echantillon 2 : Tableau des fréquences

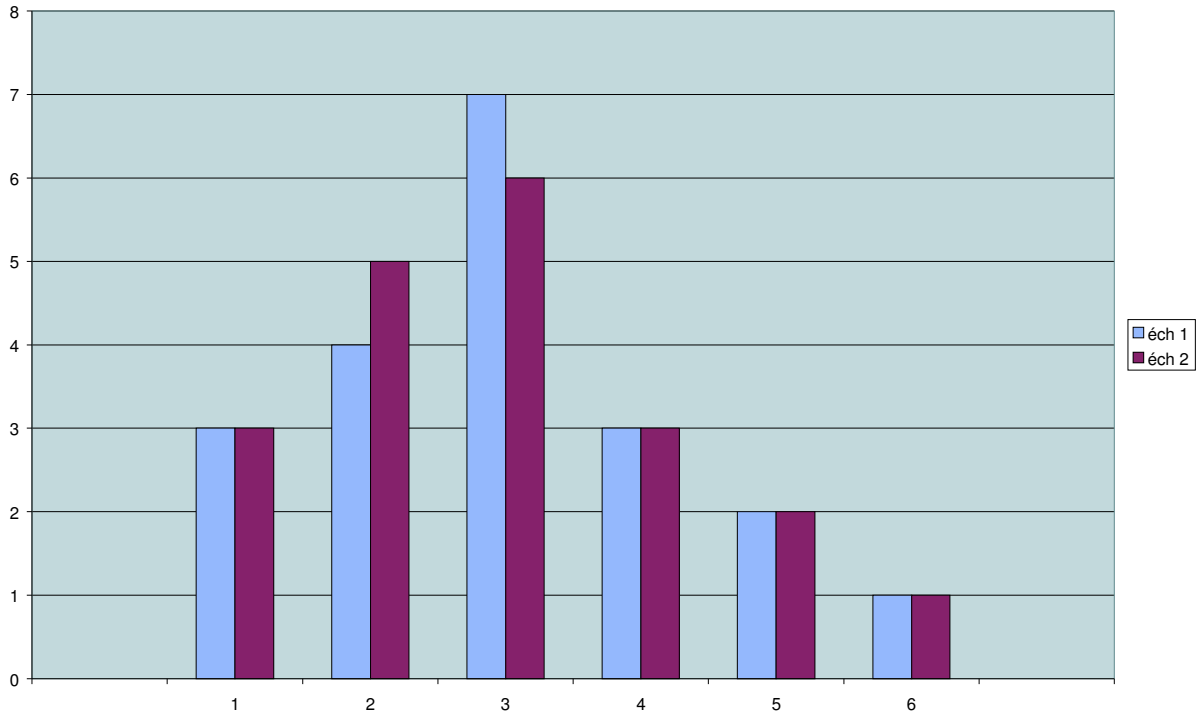

FIG. 5.2: Histogrammes des deux échantillons

Nous obtenons aussi un résumé comparatif des deux distributions dans le tableau 5.6.

Comparaison des deux échantillons		
	éch. 1	éch. 2
taille de l'échantillon	20	20
moyenne	1604.85	1628.15

suite à la page suivante

suite de la page précédente		
	éch. 1	éch. 2
médiane	1638.0	1590.5
écart type	160.21	168.298
minimum	1361.0	1384.0
maximum	1892.0	1994.0
taille de l'intervalle	531.0	610.0
coefficient de kurtosis	-0.589	-0.066
kurtosis standardisé	-0.538	0.059
coefficient d'asymétrie	0.212	0.750
coeff. d'asym. stand.	0.384	1.369

TAB. 5.6: Résumé

Le coefficient d'asymétrie et d'asymétrie standardisé nous renseignent sur le caractère symétrique des données. Dans le cas d'une distribution normale, le coefficient d'asymétrie vaut 0. Dans le cas où le pic de la distribution se situe plus vers la gauche, le coefficient prend des valeurs positives, alors que dans le cas contraire, le coefficient d'asymétrie adopte des valeurs négatives. Le coefficient d'asymétrie standardisé est utilisé afin de déterminer si l'asymétrie observée est consistante avec l'hypothèse de normalité de la distribution. Dans ce cas, l'asymétrie standardisée doit se trouver dans l'intervalle $[-2, 2]$.

Le coefficient de kurtosis et de kurtosis standardisé donnent une idée sur la "distance" entre la distribution en question et une distribution normale. Dans le cas d'une distribution normale, la valeur du coefficient de kurtosis vaut 0. Dans le cas où la distribution admet un pic plus aigu, le coefficient est positif, alors que dans le cas d'un pic plus plat, le coefficient est négatif. Le coefficient de kurtosis standardisé est utilisé afin de déterminer si la valeur du coefficient de kurtosis observé est consistante avec l'hypothèse de normalité. Dans ce cas, le coefficient de kurtosis standardisé doit se trouver dans l'intervalle $[-2, 2]$.

De plus, le programme *Statlets* nous permet d'effectuer un test de Kolmogorov-Smirnov sur les deux échantillons. Le résultat est dans les deux cas qu'au niveau $\alpha = 0,05$ on ne peut pas rejeter l'hypothèse de normalité.

Toutes ces considérations nous mènent à accepter la normalité.

Il nous est donc possible d'effectuer dans la suite des tests sur l'égalité des moyennes et des variances.

Test d'égalité des variances

Afin d'effectuer un test sur l'égalité des moyennes, nous devons tout d'abord vérifier l'égalité des variances des deux populations indépendantes.

Comparaison des variances		
	éch. 1	éch. 2
taille de l'échantillon	20	20
écart type	174.749	168.298
variance	30537.1	28324.3
rapport des variances		1.07812

TAB. 5.7: Variances

Le tableau 5.8 présente le résultat d'un *F-test* :

$$\begin{cases} H_0 & \sigma_1^2 = \sigma_2^2 \\ H_1 & \sigma_1^2 \neq \sigma_2^2 \end{cases}$$

Test sur les variances	
H_0	$\frac{\sigma_1^2}{\sigma_2^2} = 1.0$
H_1	$\frac{\sigma_1^2}{\sigma_2^2} \neq 1.0$
F-statistique	1.07812
P-valeur	0.871482
Ne pas rejeter H_0 pour $\alpha = 0.05$	

TAB. 5.8: F-Test

D'après *Statlets*, étant donné que *P-valeur* ≥ 0.05, au niveau $\alpha = 0,05$ on accepte l'hypothèse H_0, c'est à dire l'égalité des variances. On remarquera ici que ce test dépend de la normalité des deux distributions.

En acceptant l'égalité des variances, nous pouvons passer à la comparaison des moyennes des deux populations.

Test d'égalité des moyennes

Comparaison des moyennes		
	éch. 1	éch. 2
taille de l'échantillon	20	20
suite à la page suivante		

suite de la page précédente		
	éch. 1	éch. 2
moyenne	1633.7	1628.15
différence des moyennes		5.55

TAB. 5.9: Moyennes

Le tableau 5.10 présente le résultat d'un test d'égalité des moyennes (*t-test*) :

$$\begin{cases} H_0 & m_1 = m_2 \\ H_1 & m_1 \neq m_2 \end{cases}$$

Test sur les moyennes	
H_0	$m_1 - m_2 = 0$
H_1	$m_1 - m_2 \neq 0$
t-statistique	0.102304
P-valeur	0.919054
Ne pas rejeter H_0 pour $\alpha = 0.05$	

TAB. 5.10: t-test

Statlets nous fournit le protocole suivant : accepter l'hypothèse d'égalité des moyennes au niveau $\alpha = 0.05$ car *P-valeur* ≥ 0.05.

Coefficient de variation

Nous obtenons $V = 0.1$ pour les deux cas. La variation autour du rapport moyen est donc très faible.

Conclusions

Les considérations précédentes nous permettent d'affirmer que la vitesse moyenne sur les arcs virtuels euclidiens (vm(e) $= 41,28(km/h)$) induit des temps de parcours proches de la réalité.

Nous en déduisons que le facteur *speed* à introduire dans le fichier *config* vaut $\frac{75}{41,28} = 1,82$. Nous choisirons *speed*$= 1,9$.

5.4.1 Les villes à écoles multiples

Un autre problème que nous avons rencontré lors de la résolution du problème est celui des villes à clients multiples. Etant donné que nous ne connaissons pas la localisation

exacte des écoles à l'intérieur des villes, nous nous contentons d'utiliser les coordonnées du centre de la localité. Dans le cas où plusieurs écoles sont présentes dans une même ville, nous rassemblons les différents établissements en un client auquel nous ajoutons 5 minutes au temps de déchargement en guise de déplacement entre deux écoles.

5.4.2 Introduction du fichier problème

Un village i comportant j écoles sera alors défini par les caractéristiques suivantes dans le fichier problème :

ville no.	x	y	demande	borne inf.	borne sup.	service
i	x_i	y_i	$\sum_{k=1}^{j} d_k$	e_i	l_i	$(\sum_{k=1}^{j} \lceil \frac{d_k}{65} \rceil + j) \cdot 5$

où d_k représente la demande de la k^e école du village en question et $\lceil x \rceil$ est le plus petit entier supérieur ou égal à x.

5.5 Résolution du problème et exploitation des résultats

A partir des 6 routes, nous créons un fichier qui comprend 118 villes. La longueur des routes initiales est donnée par le tableau suivant à l'aide de *Microsoft Autoroute Express 98* :

route	1	2	3	4	5	6	
distance	303,5	256,2	295,3	221,2	274,7	300,5	km

TAB. 5.11: Routes initiales

Au total, les 6 camions parcourent une distance de 1651,4 km. La Figure 5.3 en annexe représente les routes euclidiennes virtuelles créées à partir de ces données. Le cercle représente l'emplacement du dépôt. Les flèches sur les routes indiquent le sens de parcours des camions, alors que les flèches à côté des noeuds indiquent les clients sans clé. Les arcs entre le dépôt et les premiers et les derniers clients ne sont pas représentés afin de ne pas alourdir les dessins.

Après 65 minutes de calculs sur un micro-ordinateur de type PC Intel Pentium II 266, le programme affiche un résultat comprenant 6 routes.

Après évaluation des routes par *Microsoft Autoroute Express 98* nous arrivons au tableau suivant :

route	1	2	3	4	5	6	
distance	261,2	242,9	155,0	258,8	223,6	254,1	km

TAB. 5.12: Routes données par dyn

Au total, les 6 routes font 1395,6 km. Trois modifications ont eu lieu concernant les heures de départ des camions :

- Route 3 : départ 5h15
- Route 4 : départ 5h00
- Route 5 : départ 5h00

Les 6 nouvelles tournées sont représentées sur la Figure 5.4 en annexe.

En comparant les deux données, nous arrivons aux constatations suivantes :

- L'amélioration est claire et vaut 255,8 km. Cela équivaut à un gain de distance de 15,5%.
- Comme remarqué plus tôt, nous n'avons pas pu diminuer le nombre de camions utilisés.
- Un autre point important est celui des fenêtres de temps. Elles ne sont violées en aucun endroit. Cela nous montre que le modèle avec ses paramètres arbitraires et expérimentaux se révèle une bonne approximation de la réalité. Notons que c'est la présence des fenêtres de temps parmi les contraintes du problème qui requiert un traitement sophistiqué des temps et vitesses de parcours.
- On remarque également une organisation moins chaotique des différentes routes dans la solution améliorée. On pourrait parler d'un phénomène de *pétales de fleur*. En effet, s'il y avait eu une répartition plus homogène des clients autour du dépôt, on aurait obtenu comme résultat des routes formant une sorte de fleur autour du dépôt central. Ici nous ne retrouvons qu'une partie de cette configuration.
- On vérifie également que les clients sans clés se trouvent à la fin des routes, ce qui indique le bon fonctionnement de la partie du programme qui contrôle la gestion des fenêtres de temps.

Les données du problème ainsi que les résultats fournis sont repris en annexe.

Conclusion

Lors de l'élaboration de ce travail, nous avons tenu à insister sur la diversité des problèmes de tournées de distribution. Nous avons plus particulièrement traité le cas particulier des fenêtres de temps relatives aux clients et au dépôt.

L'intérêt de détailler l'algorithme du Chapitre 4 apparaît désormais clairement à la vue des résultats concluants obtenus dans la deuxième partie de ce travail. Nous remarquons par ailleurs que le modèle que nous avons créé pour ce problème rend bien compte de la réalité vu que les contraintes imposées sont toutes respectées.

Nous avons donc réussi à adapter d'une manière satisfaisante une méthode qui a priori ne pouvait être directement appliquée à notre problème pratique. Il est cependant possible que les résultats obtenus puissent être améliorés après de plus longs temps de calculs ou un changement des paramètres du fichier *config*. D'autre part, malgré l'évidente efficacité de l'algorithme utilisé, une modification plus importante du programme par l'introduction d'une matrice d'interdistances réelles aurait certainement fourni des routes de qualité encore supérieure.

Il est évident que nous avons dû faire plusieurs concessions lors de la création du modèle. C'est probablement la précision qui a le plus souffert de ces adaptations et approximations. Cependant, il ne faut pas oublier qu'il s'agit ici d'un problème concret déjà tâché de nombreuses fluctuations au cours du temps. Le modèle adopté rend donc plutôt compte d'une journée que nous qualifierons de "moyenne", sans obstacles techniques majeurs.

Dans une phase ultérieure, il serait intéressant de prendre en compte les influences des heures creuses et des heures de pointe sur la vitesse moyenne. De même que pour les distances réelles, cet ajout aurait certainement fourni des résultats rendant mieux compte de la réalité. D'un autre côté, en introduisant cette donnée supplémentaire dans le fichier source du logiciel, nous aurions probablement observé des temps de calculs plus importants.

Pour terminer, nous désirons également souligner que les contraintes dues aux fenêtres de temps ont plus d'influence sur les résultats qu'il ne pourrait paraître à première vue. En effet, si nous supprimons ces contraintes en introduisant des fenêtres de temps très larges dans le logiciel, les six routes générées améliorent notre solution de 10 % en terme de distance.

Annexe A : Données numériques

Fichier problème						
ville no.	x	y	demande	borne inf.	borne sup.	service
1	-1.70	24.30	0.00	0.00	480.00	0.00
2	15.70	-7.60	54.00	0.00	375.00	5.00
3	17.30	-9.50	37.00	0.00	375.00	5.00
4	18.60	-10.40	14.00	0.00	375.00	5.00
5	14.20	-18.70	14.00	0.00	375.00	5.00
6	10.40	-18.50	6.00	0.00	375.00	5.00
7	9.90	-19.80	55.00	0.00	375.00	5.00
8	10.10	-17.40	30.00	0.00	375.00	5.00
9	10.10	-12.70	100.00	0.00	375.00	20.00
10	11.60	-14.70	28.00	0.00	375.00	5.00
11	35.60	-22.90	41.00	0.00	375.00	5.00
12	39.50	-25.30	105.00	0.00	375.00	10.00
13	42.90	-13.00	35.00	225.00	375.00	5.00
14	38.90	-13.30	61.00	225.00	375.00	5.00
15	29.50	-10.20	18.00	0.00	375.00	5.00
16	21.30	-10.10	25.00	0.00	375.00	5.00
17	16.30	-6.80	220.00	0.00	375.00	25.00
18	-7.90	-7.40	14.00	0.00	375.00	5.00
19	-9.20	-6.50	60.00	0.00	375.00	5.00
20	-6.30	-8.20	49.00	0.00	375.00	5.00
21	-4.60	-9.40	21.00	0.00	375.00	5.00
22	1.70	-10.00	25.00	0.00	375.00	5.00
23	-1.30	-11.20	9.00	0.00	375.00	5.00
24	-5.10	-6.20	13.00	0.00	375.00	5.00
25	-5.30	-4.30	117.00	0.00	375.00	20.00
26	-3.60	-4.00	20.00	0.00	375.00	5.00

suite à la page suivante

suite de la page précédente

ville no.	x	y	demande	borne inf.	borne sup.	service
27	-6.70	-1.40	74.00	0.00	375.00	10.00
28	-1.10	-1.80	270.00	0.00	375.00	25.00
29	3.60	-6.10	26.00	0.00	375.00	5.00
30	8.80	-6.50	115.00	0.00	375.00	20.00
31	11.10	-7.20	118.00	0.00	375.00	10.00
32	12.00	-7.10	53.00	0.00	375.00	5.00
33	12.80	-2.60	32.00	0.00	375.00	5.00
34	13.50	-2.60	176.00	0.00	375.00	25.00
35	10.70	-3.10	50.00	225.00	375.00	5.00
36	8.70	-2.50	13.00	225.00	375.00	5.00
37	2.50	-3.60	25.00	225.00	375.00	5.00
38	-10.50	-5.60	90.00	0.00	375.00	10.00
39	-18.50	-4.50	63.00	0.00	375.00	5.00
40	-20.40	-3.40	15.00	0.00	375.00	5.00
41	-22.00	-3.30	26.00	0.00	375.00	5.00
42	-20.40	-8.50	20.00	0.00	375.00	5.00
43	-19.00	-9.30	45.00	0.00	375.00	5.00
44	-17.60	-8.40	20.00	0.00	375.00	5.00
45	-16.50	-8.00	60.00	0.00	375.00	15.00
46	-13.40	-11.20	17.00	0.00	375.00	5.00
47	-11.40	-10.80	19.00	0.00	375.00	5.00
48	-9.00	-10.90	15.00	0.00	375.00	5.00
49	-10.30	-7.20	23.00	0.00	375.00	5.00
50	-0.30	-19.80	75.00	0.00	375.00	10.00
51	-13.50	-16.80	14.00	0.00	375.00	5.00
52	-13.20	-13.60	10.00	0.00	375.00	5.00
53	-17.00	-13.50	15.00	0.00	375.00	5.00
54	-16.50	-11.40	30.00	0.00	375.00	5.00
55	-14.80	-3.30	28.00	0.00	375.00	5.00
56	-12.20	1.70	124.00	225.00	375.00	15.00
57	-9.50	-8.90	28.00	0.00	375.00	5.00
58	-11.90	-8.40	9.00	225.00	375.00	5.00
59	-2.50	20.90	67.00	225.00	375.00	15.00
60	17.80	-2.40	106.00	0.00	375.00	10.00
61	4.60	-10.10	63.00	0.00	375.00	5.00
62	4.20	-12.20	33.00	0.00	375.00	5.00
63	-6.70	-13.30	35.00	0.00	375.00	5.00

suite à la page suivante

suite de la page précédente						
ville no.	x	y	demande	borne inf.	borne sup.	service
64	-9.50	-13.30	39.00	0.00	375.00	5.00
65	-10.30	-16.70	12.00	0.00	375.00	5.00
66	-6.90	-19.90	20.00	0.00	375.00	5.00
67	-9.60	-21.10	24.00	0.00	375.00	5.00
68	-12.00	-23.50	27.00	0.00	375.00	5.00
69	-12.60	-24.30	25.00	0.00	375.00	5.00
70	-3.90	-14.60	28.00	0.00	375.00	5.00
71	-3.00	-23.00	6.00	0.00	375.00	5.00
72	14.90	-19.10	222.00	225.00	375.00	30.00
73	-26.90	-9.90	260.00	0.00	375.00	35.00
74	-22.60	-8.90	20.00	0.00	375.00	5.00
75	-22.20	-9.50	17.00	0.00	375.00	5.00
76	-18.10	-9.70	20.00	0.00	375.00	5.00
77	-19.40	-13.20	25.00	0.00	375.00	5.00
78	-20.00	-17.70	30.00	0.00	375.00	5.00
79	-18.40	-22.40	40.00	0.00	375.00	5.00
80	-21.90	-26.30	124.00	0.00	375.00	10.00
81	-23.50	-28.20	27.00	0.00	375.00	5.00
82	-28.70	-27.10	22.00	0.00	375.00	5.00
83	-25.30	-12.20	171.00	0.00	375.00	20.00
84	-30.00	-6.70	94.00	0.00	375.00	10.00
85	-31.20	-8.20	50.00	0.00	375.00	5.00
86	-27.20	-12.50	30.00	0.00	375.00	5.00
87	-31.10	-14.80	48.00	0.00	375.00	5.00
88	-27.20	-11.00	224.00	225.00	375.00	45.00
89	13.80	-0.60	70.00	0.00	375.00	10.00
90	15.40	-1.30	50.00	0.00	375.00	5.00
91	16.00	-3.70	70.00	225.00	375.00	10.00
92	19.20	3.10	35.00	225.00	375.00	5.00
93	16.10	4.80	35.00	0.00	375.00	5.00
94	14.90	6.60	49.00	0.00	375.00	5.00
95	12.60	7.30	37.00	0.00	375.00	15.00
96	13.50	7.80	16.00	0.00	375.00	5.00
97	15.80	9.00	26.00	0.00	375.00	15.00
98	16.10	6.10	68.00	0.00	375.00	15.00
99	18.60	5.10	105.00	0.00	375.00	10.00
100	19.90	4.60	235.00	0.00	375.00	20.00

suite à la page suivante

suite de la page précédente						
ville no.	x	y	demande	borne inf.	borne sup.	service
101	24.20	-0.30	127.00	0.00	375.00	10.00
102	21.80	-1.20	18.00	0.00	375.00	5.00
103	20.00	-4.30	30.00	0.00	375.00	5.00
104	20.80	-6.20	18.00	0.00	375.00	5.00
105	6.90	5.50	107.00	0.00	375.00	10.00
106	5.60	6.80	8.00	0.00	375.00	5.00
107	6.30	8.10	24.00	0.00	375.00	5.00
108	3.10	8.20	38.00	225.00	375.00	5.00
109	6.30	8.10	29.00	225.00	375.00	5.00
110	3.20	6.10	37.00	225.00	375.00	5.00
111	0.80	8.70	56.00	225.00	375.00	15.00
112	-25.70	-16.60	23.00	0.00	375.00	5.00
113	-6.20	-30.90	60.00	0.00	375.00	5.00
114	-4.20	-12.10	21.00	0.00	375.00	5.00
115	46.50	-21.40	29.00	0.00	375.00	5.00
116	60.30	-25.10	8.00	0.00	375.00	5.00
117	47.70	-13.10	9.00	0.00	375.00	5.00
118	44.50	-14.40	17.00	225.00	375.00	5.00
999	0.00	0.00	0.00	0.00	0.00	0.00

TAB. 5.13: Fichier Problème

Annexe B : Résultats

Routes originales							
no	8	9	11	12	13	14	
	1	1	1	1	1	1	
	17	89	39	18	73	60	
	2	90	40	19	74	61	
	3	91	41	20	75	62	
	4	92	42	21	76	63	
	5	93	43	114	77	64	
	6	94	44	22	78	65	
	7	95	45	23	79	66	
	8	96	46	24	80	67	
	9	97	47	25	81	68	
	10	98	48	26	82	69	
	11	99	49	27	112	113	
	12	100	50	28	73	70	
	115	101	51	29	83	71	
	116	102	52	30	73	72	
	117	103	53	31	84	1	
	118	104	54	32	85		
	13	105	55	33	86		
	14	106	56	34	87		
	15	107	57	35	88		
	16	108	58	36	?		
	1	109	59	37	1		
		110	1	?			
		111		1			
		1					
Total	303,5	256,2	295,3	221,2	274,7	300,5	km

TAB. 5.14: Routes originales

Routes obtenues par dyn							
no	1	2	3	4	5	6	
	1	1	1	1	1	1	
	55	27	107	97	95	49	
	39	25	89	102	96	57	
	40	26	34	15	94	48	
	41	24	17	11	98	64	
	84	19	2	12	93	65	
	87	18	32	115	90	67	
	74	20	31	116	33	113	
	75	21	30	117	9	68	
	42	114	35	12	10	69	
	43	63	36	13	8	79	
	11	66	105	14	6	80	
	76	71	106	101	7	81	
	44	50	109	100	5	82	
	45	70	108	1	72	112	
	54	23	111		3	83	
	53	22	59		4	86	
	77	62	1		16	88	
	78	61			104	73	
	51	29			103	1	
	52	37			91		
	46	28			60		
	47	110			92		
	58	1			99		
	38				1		
	56						
	1						
Total	261,2	242,9	155,0	258,8	223,6	254,1	km

TAB. 5.15: Routes obtenues par dyn

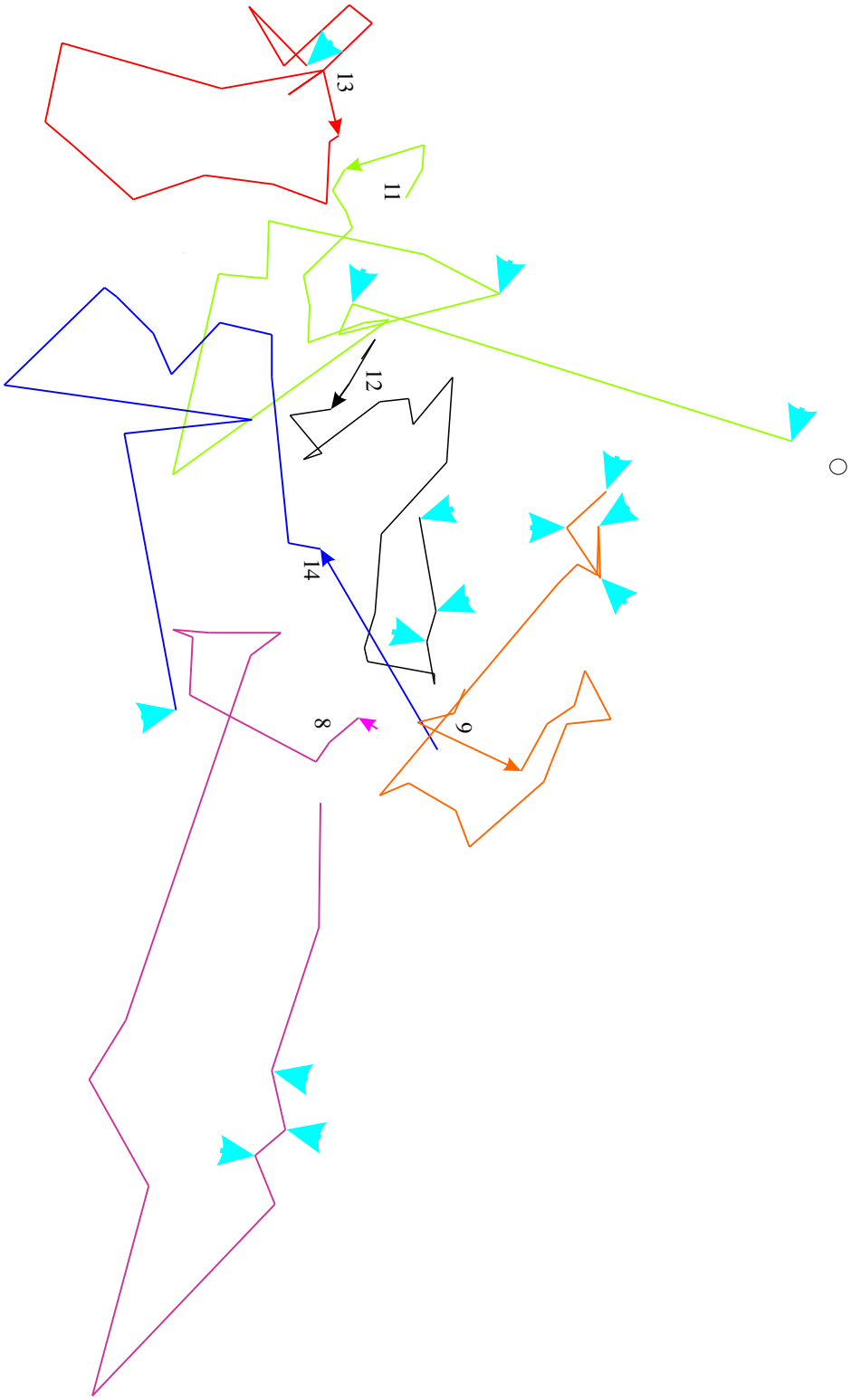

FIG. 5.3: Les 6 routes originales de l'entreprise

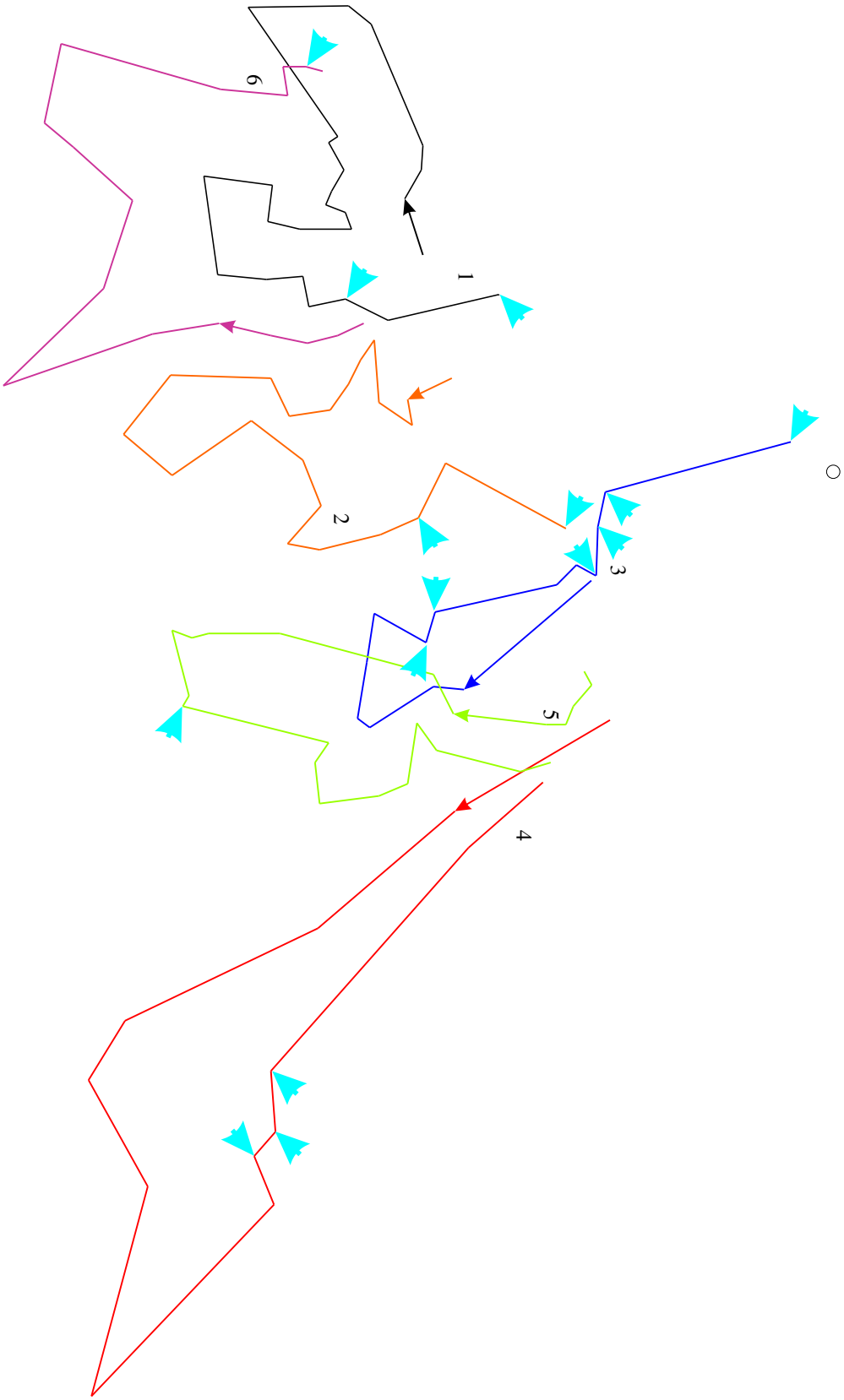

FIG. 5.4: Les 6 routes créées par dyn

Bibliographie

[1] N. Christofides, A. Mingozzi, P. Toth (1979). The vehicle routing problem. N. Christofides, A. Mingozzi, P. Toth, C. Sandi (eds.). *Combinatorial Optimization*, Wiley, Chichester, 315-338.

[2] N. Christofides, A. Mingozzi, P. Toth (1980). Exact algorithms for the vehicle routing problem based on spanning tree and shortest path relaxations. *Math. Programming 20*, 255-282.

[3] N. Christofides, A. Mingozzi, P. Toth (1981). State-space relaxation procedures for the computation of bounds to the routing problems. *Networks 11*, 145-164.

[4] G. Clarke, J.W. Wright (1964) Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res. 12*, 568-581.

[5] M.L. Fisher, R. Jaikumar (1978). A Decomposition Algorithm for Large-Scale Vehicle Routing, *Report 78-11-05*, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia.

[6] M.L. Fisher, R. Jaikumar (1981). A generalized assignment heuristic for vehicle routing. *Networks 11*, 109-124.

[7] Michel Gendreau (1998) An introduction to tabu search. *INFORMS - SCRO/CORS Meeting Montreal*, April 1998

[8] B.E. Gillett, L.R. Miller (1974). A heuristic algorithm for the vehicle dispatch problem. *Oper. Res. 22*, 340-349.

[9] A. Hertz, E. Taillard, D.de Werra (1992) "Tabu Search" dans *Local Search in Combinatorial Optimization*, J.K. Lenstra (Ed.), Dép. math. EPFL, Suisse.

[10] S.Lin (1965) Computer solutions of the travelling salesman problem. *Bell System Tech. J. 44*, 2245-2269.

[11] I. Or (1976) Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking, *Ph.D. thesis*, Northwestern University, Evanston, IL.

[12] J.Y. Potvin, J.M. Rousseau (1995) An exchange heuristic for routing problems with time windows. *Oper. Res. 46*, 1433-1446.

[13] M.M. Solomon (1987) Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Oper. Res. 35*, 254-265.

[14] P. Soriano, Michel Gendreau (1994) Fondements et applications des méthodes de recherche avec tabous. Centre de recherche sur les transports, Université de Montréal, *Publication #969*.

[15] E. Taillard, P. Badeau, M. Gendreau, F.Guertin, J.Y. Potvin (1997) A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science Vol. 31, No. 2*, 170-186.