Approche polyèdrale du problème de tournées de véhicules

Philippe Augerat

Thèse

présentée par

Philippe AUGERAT

Pour obtenir le titre de

Docteur de l'Institut National Polytechnique de Grenoble

(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

Spécialité : Mathématiques Appliquées

Formation Doctorale : Recherche Opérationnelle

Approche polyédrale du Problème de Tournées de Véhicules

soutenue le lundi 12 juin 1995 devant le jury suivant :

MM	William Cunningham	Président
	Gérard Cornuejols	Rapporteur
	Jean Fonlupt	Rapporteur
	Olivier Goldschmidt	Examinateur
	Denis Naddef	Directeur

Thèse préparée au sein du laboratoire ARTEMIS-IMAG

Résumé

Dans ce mémoire, nous présentons une méthode de résolution du Problème de Tournées de Véhicules grâce à une approche polyédrale. Un état de l'art est fait sur la connaissance du polyèdre correspondant aux solutions de ce problème et de nouvelles inégalités valides (et induisant des facettes) sont présentées pour ce polyèdre. Nous décrivons ensuite des heuristiques pour la séparation des contraintes les plus importantes ainsi qu'un algorithme de «Branchement et Coupe» qui nous permet d'améliorer les résultats connus pour la résolution exacte du problème de tournées.

Mots clefs : routage, problème de tournées, branchement et coupe, approche polyédrale, facettes.

Abstract

This thesis deals with the vehicle routing problem which is the problem of designing optimal routes of vehicles from a depot to a set of customers. Up to now, only heuristic methods have been used in practice. In this work, we focus on the polyhedral approach of the problem, it means on an exact method based on the polyhedral representation of the convex hull of feasible solutions. More precisely, we present a "Branch and Cut" algorithm for the classical vehicle routing problem with any unsplitable demands and identical vehicles located in a same depot.

The originality of our work appears in three points : i) the discovery of new valid inequalities ; ii) separation methods for these inequalities ; iii) a "Branch and Cut" algorithm which combines the use of these methods with new enumeration strategies. This algorithm allows us to solve many instances from the literature, some of them which have never been solved to optimality before.

Key words : vehicle routing, branch anf cut, polyhedral approach, facets.

Remerciements

Je commencerai par remercier M. William Cuningham d'avoir accepté de présider ce jury.

Ma reconnaissance va ensuite aux rapporteurs, M. Gérard Cornuejols et M. Jean Fonlupt, ainsi qu'à M. Olivier Goldschmidt. Tous trois ont consacré beaucoup de temps à lire mon document. Je les remercie ici pour leurs remarques constructives.

Je tiens plus particulièrement à remercier mon directeur de thèse, M. Denis Naddef, à la fois de ses conseils pendant la thèse, de ses remarques au moment de la rédaction finale et des occasions qu'il m'a donné de travailler avec des chercheurs de qualité : M. J.M. Belenguer, M. E. Benavent, M. A. Corberan, M. Y. Pochet, M. G. Rinaldi. Je dois beaucoup à ces personnes ainsi qu'à M. D. Bouali, M. V. Campos, M. J.M. Clochard, M. E. Mota, M. L. Wolsey pour leurs idées et commentaires.

Il aurait été impossible de mener de front ce travail et celui d'administrateur informatique sans la gentillesse et la tolérance des membres du laboratoire ARTEMIS. Je leur en suis très reconnaissant.

Je voudrais enfin remercier mes amis, mes parents et plus particulièrement ma sœur Catherine pour le soutien qu'ils m'ont apporté tout au long de cette thèse.

Table des matières

Introduction

Dans cette thèse, nous étudions une approche polyédrale du Problème de Tournées de Véhicules. Plus précisément, nous présentons un algorithme de «Branchement et Coupe» pour résoudre le problème classique de tournées, c'est-à-dire avec des demandes quelconques et non découpables, des véhicules identiques localisés en un même dépôt.

Dans le premier chapitre, nous présentons le domaine d'application de cette étude, le problème classique de tournées de véhicules et ses variantes. Nous décrivons les méthodes de résolution (heuristiques et exactes) les plus récentes. En particulier, nous introduisons l'approche polyédrale des problèmes d'optimisation combinatoire.

Ce type d'approche implique d'abord des recherches théoriques sur le polytope enveloppe convexe des solutions du problème de tournées. Ceci est traité au chapitre 2. Nous rappelons les inéquations linéaires valides connues pour la description de ce polytope. Ces inéquations sont majoritairement dérivées d'un problème voisin, celui du voyageur de commerce. Nous nous sommes plutôt concentrés sur la recherche d'inéquations plus spécifiquement liées à la capacité des véhicules. De nouvelles inéquations valides sont présentées ainsi que leur propriétés faciales.

La seconde partie de notre travail est algorithmique. Nous avons écrit un algorithme pour la résolution exacte du problème de tournées. Celui-ci est basé sur un programme de «Branchement et Coupe» et utilise bien sûr les résultats théoriques décrits auparavant.

Le chapitre 3 présente des algorithmes de séparation pour des inéquations valides du polytope. Nous étudions leur efficacité dans le cadre de notre algorithme.

Dans le chapitre 4, nous décrivons plus en détail l'algorithme de «Branchement et Coupe». La gestion du programme linéaire y apparaît comme un point important pour que l'algorithme soit rapide. Nous présentons surtout de nouvelles stratégies de branchement qui améliorent de façon notable la phase

d'énumération de l'algorithme. Les résultats numériques que nous avons obtenus montrent l'intérêt des algorithmes que nous avons écrits et des inéquations introduites. Ils nous permettent d'améliorer les bornes inférieures jusqu'ici obtenues par les méthodes exactes et de résoudre de nombreux problèmes.

Chapitre 1

Présentation du problème, état de l'art

1.1 Domaine d'application

Le problème de tournées de véhicules est dérivé d'un problème plus connu, celui du voyageur de commerce. Considérons un représentant habitant une ville donnée et qui doit visiter un ensemble de clients lors de sa tournée journalière. Il désire bien évidemment optimiser certains critères en particulier minimiser la longueur du trajet total effectué. On définit ainsi le problème du voyageur de commerce (PVC). Si plusieurs représentants d'une même agence doivent se partager un ensemble de clients, le problème devient alors un «multi-voyageurs de commerce» (mPVC). Enfin, si des contraintes annexes leur sont imposées, par exemple, si le nombre de clients qu'un représentant peut visiter est limité, pour une raison quelconque, on parle alors de problème de tournées de véhicules (PTV).

Nous présentons d'abord quelques applications pratiques du problème de tournées. L'application principale est la distribution, ou la collecte de biens depuis un ou plusieurs dépôts, à des clients dispersés géographiquement. La livraison de colis, la collecte du lait, la collecte d'argent (distributeurs) sont les exemples les plus connus. Ils font apparaître une contrainte de capacité sur les véhicules utilisés lors des tournées. Le problème de tournées de véhicules apparaît aussi dans la distribution de services : visites à domicile de médecins, de représentants de commerce, d'agents de maintenance. Dans ces derniers

cas, la notion de véhicule et celle de capacité peuvent être remplacées par d'autres. Par exemple, notre représentant de commerce voudra planifier ses visites hebdomadaires. Supposons qu'il travaille cinq jours par semaine, qu'il rentre chez lui tous les soirs et que la durée et/ou l'heure de ses visites est fixée ou estimée. Il doit alors résoudre l'équivalent d'un problème de tournées à cinq véhicules avec des contraintes annexes de temps.

Un autre domaine d'application est l'atelier de production, et par exemple le problème de machines parallèles avec changements d'outils. Considérons des produits différents devant être fabriqués sur plusieurs machines identiques. Chaque produit nécessite un changement d'outils sur la machine. Le temps du changement d'outils dépend du produit actuellement sur la machine et du produit à installer. Le problème de la minimisation de la durée totale d'utilisation des machines pour une journée de travail est équivalent à un problème de tournées de véhicules. Une tournée correspondra à la liste des opérations sur une machine. Notons que dans ce cas, la longueur d'une tournée est limitée, ce qui introduit une nouvelle variante du problème.

1.2 Problème académique et variantes

Vu le nombre de paramètres et contraintes annexes dans les exemples donnés à la section précédente, on imagine mal pouvoir formuler le PTV de manière générale et exhaustive. Nous définissons donc un PTV de base qui correspond au problème académique étudié le plus largement. Nous présenterons ensuite les autres variantes étudiées dans la littérature.

Un ensemble de k véhicules pouvant transporter un volume C de marchandises doivent livrer n commandes de biens à partir d'un dépôt unique. Chaque commande i a un volume d_i $(i = 1, \ldots, n)$ non séparable et on connaît les distances l_{ij} entre les lieux où doivent être livrées les commandes i et j $(i, j = 0, \ldots, n)$, le lieu indexé 0 représentant le dépôt. Les véhicules effectuent des tournées T_i $(i = 1, \ldots, k)$, qui partent du dépôt et qui y reviennent. Le problème est de minimiser la somme des longueurs des tournées. Ce problème a été très largement étudié dans la littérature (voir les états de l'art de Christophides [Chr85], Laporte et Norbert [LN87] et Laporte [Lap92]).

La figure (1.1) décrit les données d'un problème de la littérature (dans ce problème le dépôt est le sommet numéro 1). Les chiffres associés aux

sommets sont les numéros et les demandes $i(d_i)$. Les distances l_{ij} sont les distances euclidiennes entre les clients. Les arêtes du dessin correspondent à une solution réalisable du problème.

Du fait de l'extrême simplification de ce modèle par rapport aux problèmes réels, certains auteurs, Bodin et al. [BG81], Christofides [Chr85] Desrochers et al. [DLS90] ont tenté de définir une classification des problèmes de tournées à partir des paramètres suivants, dont la combinaison permet de mieux approcher les problèmes réels :

1. distances entre les villes :
   (a) ces distances sont ou ne sont pas euclidiennes ;
   (b) la distance pour aller d'une ville A à une autre ville B est la même que pour aller de la ville B à la ville A (Problème Symétrique) ;
   (c) ces distances sont différentes (Problème Assymétrique) ;
2. nombre de dépôts :
   (a) un dépôt (le PTV Classique) ;
   (b) plus d'un dépôt ;
3. taille de la flotte de véhicules :
   (a) un véhicule (le PVC) ;
   (b) plus d'un véhicule (le PTV) ;
4. type de la flotte de véhicules :
   (a) homogène : tous les véhicules sont identiques (le PTV Classique) ;
   (b) hétérogène : au moins un véhicule est différent ;
5. nature de la demande des villes :
   (a) déterministe (le PTV Classique) ;
   (b) stochastique ;

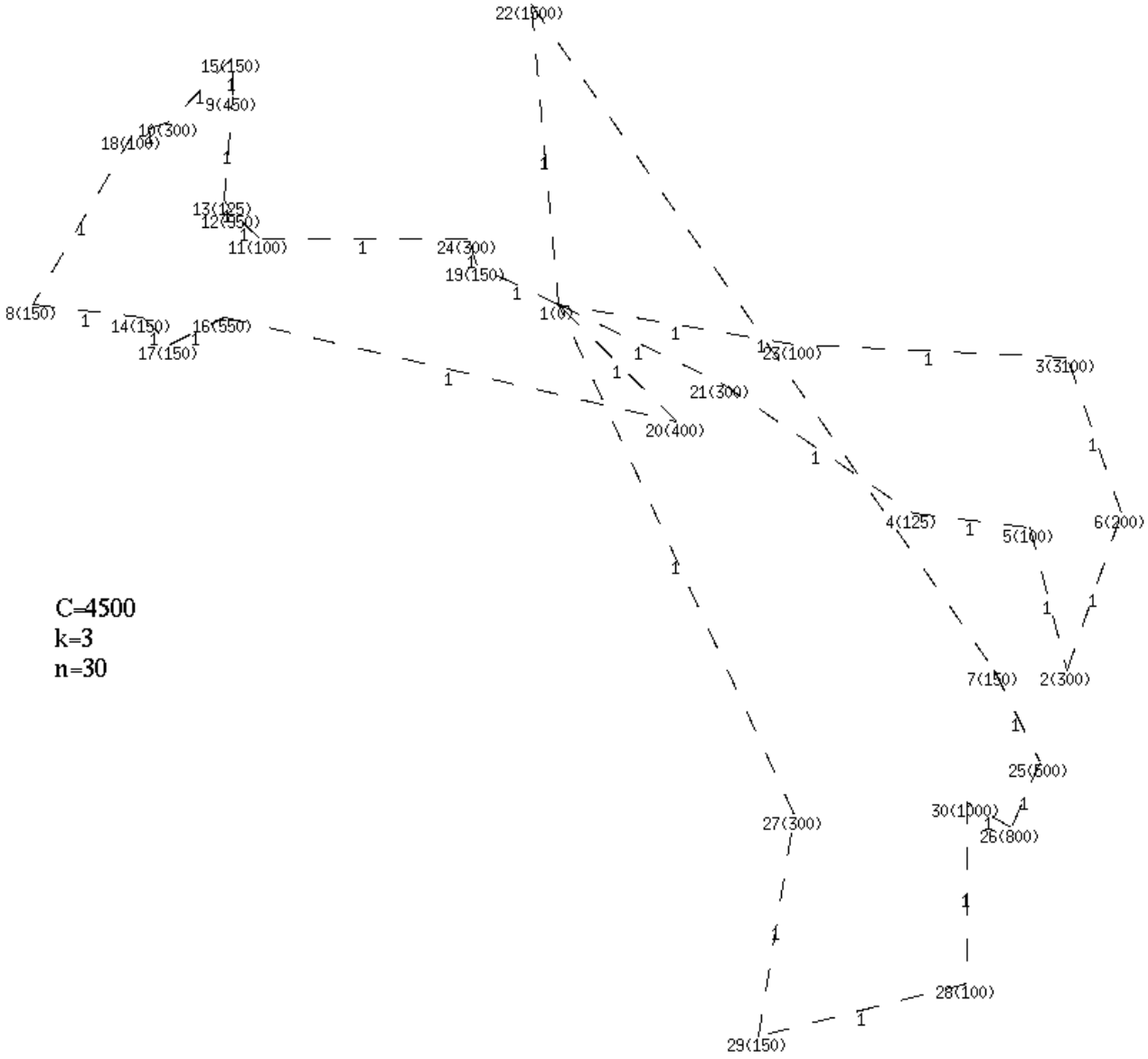


FIG. 1.1 - *Une instance de problème de tournées de véhicules*

6. restrictions de la capacité des véhicules :
   (a) données et égales pour tous les véhicules (le PTV Classique) ;
   (b) données et différentes pour au moins un véhicule ;
   (c) pas de restriction de capacité (le mPVC) ;
7. longueur maximale d'une tournée :
   (a) donnée et identique pour toutes les tournées ;
   (b) donnée et non identique pour toutes les tournées ;
   (c) non donnée (le PTV Classique) ;
8. temps maximal d'une tournée :
   (a) donné et identique pour toutes les tournées ;
   (b) donné et non identique pour toutes les tournées ;
   (c) non donné (le PTV Classique) ;
9. type de services :
   (a) un type (le PTV Classique) ;
   (b) plusieurs mais unique pour un véhicule donné ;
   (c) plusieurs pour un même véhicule ;
10. coûts :
    (a) seulement dans les tournées (le PTV Classique) ;
    (b) dans les tournées et des opérations fixes (exemple : achat de véhicules) ;
11. objectif :
    (a) minimiser les coûts inclus dans les tournées (le PTV Classique) ;
    (b) minimiser la somme des coûts fixes et variables ;
    (c) minimiser le nombre de véhicules achetés ;
12. contraintes horaires de service dans une ville :
    (a) spécifié et préfixé ;

(b) spécifié par un intervalle de temps dans chaque ville (fenêtres horaires) ;

(c) non spécifié (le PTV Classique) ;

13. nature des tournées :

(a) un véhicule ne peut servir que dans une tournée seulement (le PTV Classique) ;

(b) un véhicule peut servir dans plus d'une tournée (ici, le temps total des tournées d'un véhicule est généralement limité).

Pour une liste de références sur la résolution de toutes ces variantes, le lecteur se reportera aux «états de l'art» déjà cités et à Christofides et Mingozzi [CM90].

1.3 Complexité du problème

Le problème de tournées de véhicules est un problème NP-difficile comme l'immense majorité des problèmes de routage (cf Lenstra et Rinnooy Kan [LK81]). Le problème du voyageur de commerce peut se transformer en un problème de tournées de véhicules à un seul véhicule sans contraintes annexes. Le problème de partition d'un ensemble de nombres entiers peut lui aussi se transformer en un problème de tournées, où tous les clients sont situés au même endroit. Ces deux problèmes sont bien entendus NP-difficiles (cf Karp [Kar72] , Garey et Johnson [GJ79]).

Notons que si la capacité des véhicules est supérieure à la somme des demandes, le PTV se réduit à un mPVC. Inversement, on peut imaginer qu'il n'y ait qu'une affectation possible des clients aux véhicules satisfaisant les contraintes de capacité. On pourrait alors séparer la résolution en deux phases. D'abord, on résout un problème de bin-packing (i.e. le problème de ranger des éléments de taille donnée dans des boîtes de taille donnée, cf [MT90a]). Dans notre cas, les éléments sont les commandes et les boîtes sont les véhicules. Ensuite, on résout k problèmes de voyageur de commerce. Le PTV est donc souvent vu comme l'intersection de deux problèmes NP-difficiles. C'est la raison pour laquelle la majorité des méthodes de résolution ont d'abord été des heuristiques. Ce n'est qu'à partir des années 80 qu'ont été présentées des méthodes exactes. Nous présentons ces deux types d'approches dans les prochaines sections.

1.4 Classification des heuristiques

Nous reprenons ici une classification établie par Christofides [Chr85] et revue par Taillard [Tai93]. Il existe quatre types de méthodes heuristiques

- les méthodes constructives qui élaborent les tournées des véhicules en ajoutant l'une après l'autre toutes les commandes (cf Clarke et Wright [CW64], Desrochers et Verhoog [DV89], Dror et Levy [DL86], Gaskel [Gas67], Mole et Jamerson [MJ76], Nelson et al. [NNGW85], Passens [Pas88], Yellow [Yel70]) ;

- les méthodes qui procèdent en deux phases, groupement des villes puis résolution de problèmes de voyageur de commerce, ou bien résolution d'un problème de voyageur de commerce puis décomposition en tournées admissibles (cf Christofides [Chr85], Fisher et Jaikumar [FJ81], Gillet et Miller [GM74], Haimovich et Rinnoy Kan [HK85], Wren et Holliday [WH72]) ;

- celles basées sur une énumération implicite partielle (Christophides [Chr85])

- les méthodes d'amélioration par recherches locales (ou méthodes itératives) (Gendreau et al. [GAG92], Osman [Osm93], Pureza et França [PF91], Taillard [Tai93], Hilquebran et al.[HASG94], Noon et al. [NJR94]).

Parmi ces méthodes nous détaillerons uniquement l'algorithme que nous utilisons dans la phase d'initialisation de notre algorithme de «Branchement et Coupe». Cet algorithme est basé sur la méthode tabou et entre donc dans la quatrième catégorie, d'où sont issus les meilleurs algorithmes connus à ce jour.

1.5 Un algorithme basé sur la méthode tabou

L'heuristique que nous utilisons pour obtenir une borne supérieure de la solution optimale est basée sur un schéma de recherche itérative. Une itération du programme consiste à choisir une solution réalisable non dans le voisinage de la solution courante. La solution finale est bien sûr la meilleure solution «visitée» pendant l'algorithme. Nous définissons maintenant les notions de voisinage d'une solution et de solution tabou. Une solution s' est

dans le voisinage $N(s)$ de la solution s si s' peut être construite à partir de s grâce à une des deux transformations suivantes : la réaffectation d'un client à une tournée différente ou l'échange de deux clients entre les deux tournées correspondantes dans s.

Le choix du voisin dans $N(s)$ s'effectue dans l'ordre suivant. On explore d'abord les voisins obtenus par réaffectation d'un client, puis ceux obtenus par échange de clients. Si un voisin conduit à une amélioration de la fonction objective, on le choisit et on arête l'exploration, sinon on choisit le voisin qui détériore le moins la solution. Les solutions visitées sont sauvegardées et déclarées tabou de manière à ne pas visiter plusieurs fois une solution (en fait ce sont certaines transformations qui sont déclarées tabou). La procédure est arrêtée, si aucune amélioration de la solution n'est apparue pendant un nombre fixé d'itérations.

Lorsque l'on évalue un voisin, on effectue la mise à jour des routes avec la procédure classique d'insertion au mieux. Pour cette raison, la qualité des routes peut se détériorer, c'est-à-dire que la route correspondant à un ensemble de clients ne sera pas une solution optimale du PVC correspondant. Cependant, à intervalles réguliers, on appliquera individuellement à toutes les routes de la solution courante, l'algorithme du voyageur de commerce de Lin et Kernighan [LK73]. On mettra à jour en cas de succès la solution courante.

Une description formelle et précise de l'algorithme utilisé est la suivante.

```
procédure tabou() ;
/* initialisation */

Trouver une solution réalisable s ;
MI=0 ; /* meilleure itération */
MO=s ; /* meilleure solution */
MV=longueur(s) /* meilleure fonction objective */
OC=MO ; /* solution courante */
nbiter=0 ; /* itération courante */

/* méthode tabou */

tant que (nbiter - MI ≤ mbmax) faire
     nbiter=nbiter+1 ;
```

```
        recherche_meilleur_voisin(SC) ;
        mise_à_jour_liste_tabou ;
        si (longueur(OC) < MV) alors
            MV=longueur(OC) ;
            MO=OC ;
            MI=nbiter ;
        fin (si)
    fin (tant que)
    fin (procédure)
```

Les différents paramètres du programme (solution initiale, taille et implémentation de la liste tabou) sont expliqués par Campos et Mota [CM94] qui ont réalisé l'implémentation de l'algorithme que nous utilisons. La comparaison de différentes implémentations de recherches itératives est présentée par Taillard [Tai93].

1.6 Les méthodes exactes

Nous présentons maintenant des méthodes exactes de résolution du PTV. On peut trouver dans Laporte [Lap92] un état de l'art de ces méthodes. Il n'est pas facile de les classifier, car il y a de nombreuses imbrications entre elles. En particulier, on peut retrouver pour une même formulation plusieurs approches, et inversement, une même approche appliquée à des formulations différentes. On peut formuler le PTV de la manière suivante :

1. comme un problème de k-arbre avec contraintes ;
2. comme un problème de partition ;
3. comme un problème de flot de véhicules ;
4. comme un programme linéaire entier ;
5. comme un problème de flot de commodités ;
6. comme un problème d'affectation généralisé.

Les techniques de résolution employées sont

a. les relaxations avec pénalités ;

b. la génération de colonnes ;

c. la programmation dynamique ;

d. la méthode «Branchement et Coupe».

Nous allons décrire maintenant quatres méthodes qui couvrent la majorité des formulations et techniques de résolution. Nous laissons de côté, les formulations 4 (cf [AC91] [DL91]) et 5 (cf [WMGS57], [FCG84], [DKL86]), car aucun résultat numérique n'est à notre connaissance disponible dans le cas du PTV classique. La formulation 5 est cependant intéressante pour son adaptabilité aux problèmes réels. La formulation 6 (Fisher and Jaikumar [FJ78]) ne sera pas traitée non plus, car Fisher a introduit depuis une méthode plus performante.

Les quatres approches que nous allons décrire utilisent les formulations 1 à 3 et les techniques a à d. Nous donnons maintenant quelques notations et définitions. On définit un graphe complet $G = (V, E)$ avec $n + 1$ sommets (correspondant aux clients et au dépôt) et m arêtes. On notera V_0 l'ensemble des sommets qui sont des clients. Le terme tournée et les notations k (nombre de véhicules), C (capacité), d_i (demande du client i), l_{ij} (longueur du trajet entre i et j, $i < j$) sont ceux de la définition du problème académique de tournées. Les termes route, tour et tournée sont équivalents. La demande (ou charge) d'une tournée est la somme des demandes des clients visités dans la tournée. Enfin, la demande $d(S)$ d'un ensemble de clients S est la somme des demandes des clients.

1.6.1 Problème de k-arbre avec pénalités

Fisher [Fis94a] propose une relaxation du PTV basée sur la notion de k-arbre (un ensemble de $n + k$ arêtes induisant un graphe connexe). Toute solution du PTV est alors un k-arbre. La figure (1.2) montre un exemple de 2-arbre qui n'est pas une solution de PTV. Dans cet exemple, la capacité des deux véhicules disponibles est de 10 unités et les demandes sont indiquées à côté des sommets. On voit que des contraintes de capacité et de degré propres au PTV ne sont pas satisfaites par ce k-arbre. Le PTV est en fait équivalent à un problème de k-arbre avec des contraintes annexes de degré et de capacité.

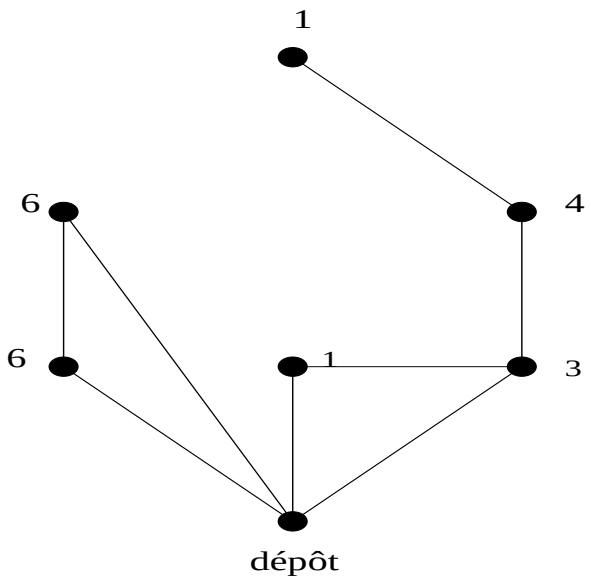


FIG. 1.2 - *Une solution du problème de k-arbre, C=10*

La méthode de Fisher s'inspire de celle de Held et Karp [HK71] pour le voyageur de commerce. Ces derniers calculent un 1-arbre de longueur minimale (un arbre de poids minimal sur tous les sommets moins un, plus les deux arêtes les plus petites adjacentes au sommet restant). Des pénalités sont introduites dans le vecteur distance (l_{ij}) comme indiqué plus loin si des sommets ont un degré différent de 2 dans le 1-arbre. Les contraintes de degré sont donc dualisées pour obtenir un problème lagrangien et le résultat de ce problème est une borne inférieure utilisable dans un algorithme de «Branchement et Evaluation» («Branch and Bound» en anglais). Fisher, quant à lui, modélise le problème de tournées comme la recherche d'un k-arbre minimum avec $2k$ arêtes incidentes au dépôt. Si on compare ce problème avec celui du 1-arbre, des contraintes de capacité apparaissent en plus de celles de degré. Fisher donne dans [Fis94b] un algorithme polynômial pour rechercher un k-arbre minimum avec le degré d'un des sommets fixé, et utilise la méthode de sous-gradient détaillée dans [Fis81] pour obtenir une borne inférieure de la solution optimale du problème.

Soit $S \subseteq V$, on définit $\lceil d(S)/C \rceil$ comme le plus petit entier supérieur à $d(S)/C$ et $\delta(S)$ comme l'ensemble des arêtes ayant exactement une extrémité

dans S. Pour une arête $e \in E$, on dénote x_e la variable qui prend la valeur 1 si l'arête e est utilisée dans la solution et la valeur 0 sinon. On note x le vecteur de dimension m correspondant à ces variables. Soit $X = \{x \ : \ x$ est le vecteur d'incidence d'un k-arbre de degré $2k$ au dépôt$\}$, la solution du problème de k-arbre avec contraintes (KAC) est alors

$$Z(KAC) = \min_{x \in X} \sum_{e \in E} l_e \ x_e \tag{1.1}$$

t.q.

$$\sum_{e \in \delta(\{i\})} x_e \ = \ 2, \qquad \forall i \in V_0 \tag{1.2}$$

$$\sum_{e \in \delta(S)} x_e \ \geq 2 \ \lceil d(S)/C \rceil, \qquad \forall S \subseteq V_0 \ : \ |S| \geq 2 \tag{1.3}$$

On définit $u_i, \ i \in V$ et $v_S \geq 0$ pour $S \subseteq V, \ |S| \geq 2$ comme les multiplicateurs lagrangiens respectivement pour les contraintes de degré (1.2) et de capacité (1.3), on a alors la relaxation lagrangienne suivante :

$$Z_D(u,v) = \min_{x \in X} \sum_{e \in E} l'_e \ x_e \ + \ 2 \sum_{1}^{n} u_i \ + \ 2 \sum_{S \subseteq V_0} v_S \ \lceil d(S)/C \rceil \tag{1.4}$$

avec $u_0 = 0$ et pour $e = (ij) \ : \ l'_e = l_e \ - \ u_i \ - \ u_j - \sum_{S \subset V \ : \ e \in \delta(S)} \ v_S$

La borne inférieure définie par $Z_D = \max_{u,v \geq 0} \ Z_D(u,v)$ est approximée grâce à la méthode de sous-gradient alors que $Z_D(u,v)$ est obtenue grâce à l'algorithme décrit dans [Fis94b]. A la première itération, on calcule un k-arbre minimal avec les coûts initiaux, c'est-à-dire avec $u_i = v_S = 0$, pour tour $i \in V, \ S \subset V$. On identifie ensuite les contraintes de degré et de capacités violées et on les ajoute dans la fonction objective, et ainsi de suite. Cette méthode permet à Fisher de résoudre de manière optimale des problèmes réels de taille importante (en particulier une instance avec 100 clients).

1.6.2 Problème de partition avec génération de colonnes

Cette formulation a été introduite par Balinski et Quandt [BQ64] et reprise par plusieurs auteurs [Orl76], [DSD84], [AMS89]. Le principe est de formuler le problème de tournées comme un problème de partition et de résoudre

ce dernier par la méthode de génération de colonnes. On note $R = (1, \ldots, r)$ l'ensemble indexé de toutes les tournées et $R_i \subset R$ le sous-ensemble des tournées contenant le sommet i. De même, on note respectivement c_t et R_t le coût et l'ensemble des sommets visités par la tournée t. Notons que le calcul de c_t est déjà un problème difficile (de PVC). On peut représenter une solution du PTV par un vecteur x de dimension r dont la t-ième composante x_t vaut 1 si la tournée d'indice t est utilisée dans la solution et 0 sinon. Le problème de partition (SP) se formule alors

$$\min Z(SP) \ = \sum_{t \in R} c_t \ x_t \tag{1.5}$$

t.q.

$$\sum_{t \in R_i} x_t \ = \ 1, \qquad i = 1, \ldots, n \tag{1.6}$$

$$\sum_{t \in R} x_t \ = \ k \tag{1.7}$$

$$x_t \in 0, 1, \ t \in R \tag{1.8}$$

Notons SPL le problème obtenu en relaxant les contraintes d'intégrité de SP. Le nombre de tournées r est trop grand pour que SPL soit résolu directement, même pour n petit. La méthode de génération de colonnes est une méthode itérative. A chaque itération, on résout un problème $SPL(\bar{R})$ obtenu à partir de SPL en remplaçant R par un sous-ensemble de tournées $\bar{R}$. La solution optimale de $SPL(\bar{R})$ est solution optimale de SPL si pour toute tournée $t \notin \bar{R}$, le coût réduit de la variable x_t relativement à la solution optimale de $SPL(\bar{R})$ est négatif. S'il existe une variable (colonne) dont le coût est négatif, on l'ajoute à $\bar{R}$. Le choix des colonnes de SP à générer est donc le problème crucial, le second étant l'obtention d'une solution entière. Ce second problème est en général résolu avec un algorithme de «Branchement et Evaluation».

Agarwal et al. [AMS89] considèrent une solution optimale du problème $SPL(\bar{R})$. On note B la base correspondante, et respectivement c_B et $u = c_B.B^{-1}$, le vecteur coût des variables de base et le vecteur des variables duales. Le sous-problème consistant à chercher une colonne entrante se modélise comme un problème de voyageur de commerce avec récompense (PVCR, «Price Collecting Travelling Salesman Problem» en anglais). Ce nouveau problème est de trouver une colonne avec un coût réduit négatif (voir minimal),

ou de montrer qu'il n'existe pas une telle colonne. Une colonne peut se définir comme un vecteur y de taille n dont la i-ème composante y_i vaut 1 si le sommet i est inclus dans la tournée correspondante. Une colonne «entrante» y sera la solution optimale du PVCR suivant

$$\min Z(PVCR) = c_y - \sum_{i=1}^{n} u_i y_i \tag{1.9}$$

t.q.

$$\sum_{i=1}^{n} d_i y_i \leq C \tag{1.10}$$

$$y_i = 0 \ ou \ 1 \ (i = 1, \ldots, n) \tag{1.11}$$

où c_y est le coût d'un tour optimal sur les sommets de la tournée induite par y.

Ce sous-problème peut être résolu par approche polyédrale (cf Balas [Bal89], [Bal93]) en rajoutant des variables de flot de façon à écrire c_y comme une formule linéaire.

Agarwal choisit plutôt de construire une fonction linéaire $f(y)$ qui est une borne inférieure de c_y, ce qui lui permet d'obtenir une borne inférieure de la solution du PVCR. Il utilise cette borne dans un arbre d'énumération implicite pour obtenir une solution optimale de PVCR. A un nœud donné de cet arbre, soit S l'ensemble des sommets i tel que y_i soit fixé à 1. On note $f(S)$ une borne inférieure de la solution du PVC sur S, $m(S)$ le nombre maximum de sommets pouvant être ajoutés à S pour former une route réalisable. Pour $i \in V_0 \setminus S$, on définit $q_i(S) = \min_{j,h \in S}(l_{ij} + l_{ih} - l_{jh})$. Pour toute tournée y contenant S, on a alors $c_y \geq f(y) = f(S) + \sum_{i \in V_0 \setminus S}(q_i(S)/m(S))y_i$. A chaque nœud de l'arbre, on doit donc résoudre un problème de sac à dos (deux en fait car $m(S)$ est aussi le résultat d'un problème de sac à dos). Agarwal et al. arrivent à résoudre des instances de petite taille (autour de 20 villes).

1.6.3 Problème dual de partition

Mingozzi et al. [MCH94] résolvent le problème dual de la relaxation de (SP) de manière heuristique. On note DSP ce problème, qui se formule de la manière suivante :

$$\max Z(DSP) = \sum_{i=1}^{n} u_i \; + \; k \; u_0 \tag{1.12}$$

t.q.

$$\sum_{i \in R_t} u_i \; \leq \; c_t, \; t \in R \tag{1.13}$$

$$u_i \; non \; restreint, \; i = 0, \ldots, n \tag{1.14}$$

Plusieurs heuristiques sont utilisées pour résoudre le problème (DSP). Soit $(\bar{u}_i^1), i = 0, \ldots, n$ la solution de la première heuristique. La seconde heuristique va s'appliquer au problème modifié

$$\max Z(DSP) = \sum_{i=1}^{n} u_i \; + \; k \; u_0 \tag{1.15}$$

t.q.

$$\sum_{i \in R_t} u_i \; \leq \; c_t \; - \sum_{i \in R_t} \bar{u}_i^1, \; t \in R \tag{1.16}$$

$$u_i \; non \; restreint, \; i = 0, \ldots, n \tag{1.17}$$

et ainsi de suite. La solution finale, s'il y a r heuristiques, sera la somme des solutions des r sous-problèmes. Mingozzi et al. présentent trois heuristiques.

La première heuristique utilise le fait qu'une solution du PTV est un k-arbre, c'est-à-dire utilise des arguments décrits dans une section antérieure. La recherche d'un k-arbre minimum est par contre modélisée comme un problème de transport, ce qui permet de transformer la solution du problème en une solution du DSP.

Dans la seconde heuristique, on calcule des tournées de longueur minimale passant par chaque sommet ce qui permet de définir une solution du DSP comme une certaine combinaison réalisable de tournées minimales. Soit $R_{i,q}$ l'ensemble des tournées de charge q visitant le client i. Soit $\psi_i(q)$, $i = 1, \ldots, n$; $q = d_i, \ldots, C$, une borne inférieure de la longueur de la plus courte tournée de $R_{i,q}$. Les $n+1$ variables, $b_0 = 0$ et $b_i = d_i \min_{d_i \leq q \leq C}\{\psi_i(q)\}$, $i = 1, \ldots, n$, forment une solution réalisable du DSP. Les $\psi_i(q)$ sont calculés en

utilisant la programmation dynamique (cf [CMT81]). On sélectionne ensuite pour chaque sommet i , la tournée minimale parmi celles obtenues en calculant les $\psi_i(q)$. Il est alors possible de calculer le degré d'un sommet dans cette liste de tournées (en fait un degré pondéré), et d'utiliser une méthode de sous-gradient (dans l'esprit de celle utilisée par Fisher) pour obtenir une solution qui soit aussi réalisable pour (SP).

Dans la troisième heuristique, on considère une liste F de tournées telle que toutes les tournées hors de cette liste soient plus longues que la plus longue de F. Le problème (SP) est alors équivalent à un problème de partition sur cet ensemble réduit de tournées avec des variables correctives

$$\min Z(RSP) = \sum_{t \in F} c_t \; x_t \; + \; \sum_{i \in V} L_i y_i \tag{1.18}$$

t.q.

$$\sum_{t \in F_i} x_t \; + \; y_i \; = \; 1, \qquad i = 1, \ldots, n \tag{1.19}$$

$$\sum_{t \in F} x_t \; + \; y_0 \; = \; k \tag{1.20}$$

$$x_t \geq 0, \; t \in F \tag{1.21}$$

$$y_i \geq 0, \; i \in V \tag{1.22}$$

où $F_i = R_i \cap F$ and $L_i = d_i \max\{c_t, \; t \in F\}/Q, i = 1, \ldots, n$. Le calcul de F est l'élément le plus complexe de l'heuristique. Il est décrit dans [Bal94]. La combinaison des trois heuristiques présentées permet dans tous les cas d'obtenir une borne inférieure de la solution de (SP) utilisable dans un programme de «Branchement et Evaluation». Mingozzi et al. arrivent à résoudre des problèmes jusqu'à 50 clients.

1.6.4 Approche polyédrale

L'approche polyédrale des problèmes d'optimisation doit son succès grandissant à ses bons résultats quand elle est appliquée au problème du voyageur de commerce. Padberg et Rinaldi [PR91], Grötschel et Holland [GH91] et récemment Applegate et al. [ABCC94] ont résolu des instances de PVC de très grande taille (plusieurs milliers de sommets). Ces résultats sont dus à une bonne connaissance du polyèdre enveloppe convexe des solutions du PVC ainsi qu'au développement d'algorithmes performants autant pour l'analyse des solutions des relaxations linéaires du PVC que pour l'utilisation des bornes obtenues dans des procédures d'énumérations.

Nous présentons ici la méthode de manière générale et son application au PVC, ceci en suivant la méthode et la terminologie de Grötschel et Padberg [GP85] et Padberg et Rinaldi [PR91]. La formulation et la résolution du PTV est bien sûr détaillée dans les chapitres suivants. Cette comparaison PVC-PTV va en particulier nous permettre d'introduire les grands axes de la recherche que nous avons réalisée.

Considérons un graphe complet à n sommets $K_n = (V, E)$. Soit T un cycle hamiltonien (ou tour) dans ce graphe, on définit le vecteur d'incidence $x^T \in \mathbb{R}^E$ par ses composantes $x_e^T = 1$ si $e \in T$, 0 autrement. Nous appelons $STSP(n)$ l'enveloppe convexe des vecteurs d'incidence de tous les tours de K_n et A la matrice d'incidence sommet-arête de K_n. Les équations $Ax = 2$ sont les équations de degré. Le polytope $STSP(n)$ peut alors être décrit par une famille finie de contraintes L_n :

$STSP(n) = \{x \in \mathbb{R}^E \mid Ax = 2,\ \alpha x \leq \alpha_0,\ \textit{pour tout}\ (\alpha, \alpha_0) \in L_n\}$

La solution optimale du PVC peut alors être obtenue par résolution du programme linéaire suivant

$$\min\ Z(PVC(L_n)) = lx \tag{1.23}$$

t.q.

$$Ax = 2 \tag{1.24}$$

$$\alpha x \leq \alpha_0\ \textit{pour tout}\ (\alpha, \alpha_0) \in L_n \tag{1.25}$$

Il faut noter que notre connaissance de L_n n'est que très partielle et que le nombre de contraintes dans L_n est de toute façon trop important pour les lister de manière explicite dans un programme linéaire.

Soit $L \subset L_n$ et $\bar{x}$ la solution de la relaxation du programme linéaire PVC(L) obtenu en remplaçant L_n par L. Dans un algorithme de coupe, on résout à chaque itération le programme linéaire. Si la solution $\bar{x}$ est entière alors $\bar{x}$ est la solution optimale du PVC. Sinon, on recherche une ou des contraintes de L_n violée par $\bar{x}$ et on l'ajoute au programme linéaire. On itère cette phase «résolution-séparation» jusqu'à ce qu'on obtienne une solution entière ou que l'on ne trouve pas de contrainte violée. Dans ce dernier cas, la solution du programme linéaire est une borne inférieure de la solution optimale. Une telle borne peut être calculée à chaque nœud d'un algorithme d'énumération implicite. On parle alors d'algorithme de «Branchement et Coupe». Le terme «branchement» désigne la séparation en sous-problèmes dans l'arbre d'énumération. Le terme «coupe» désigne la phase «résolution-séparation» exécutée à chaque nœud de cet arbre. L'originalité supplémentaire de l'algorithme de «Branchement et Coupe» est de pouvoir ajouter à tous les nœuds de l'arbre d'énumération, des contraintes de L_n qui sont aussi valides pour les autres nœuds.

La première étape du travail est donc d'obtenir une description partielle suffisante de L_n. De nombreux travaux sur la structure faciale du polytope du PVC ont été réalisés. Plusieurs classes de facettes autres que les contraintes d'élimination de sous-tour ont été découvertes. Les plus connues sont les contraintes de peigne (Chvátal [Chv73], Grötschel et Padberg [GP79]), ou les contraintes de chemins (Cornuejols et al. [CFN85]). Naddef [Nad90] donne un état de l'art des classes de contraintes de manches et de dents qui sont les diverses extensions des contraintes de peignes.

L'approche polyédrale du PTV a logiquement été menée d'abord dans le prolongement de celle sur le PVC. Laporte et Norbert [LN87], Cornuejols et Harche [CH93], Araque [Ara89] ont montré que la validité des contraintes connues pour le PVC était conservée pour le PTV. Les problèmes résultants des demandes et des capacités conduisent en fait à généraliser les contraintes du PVC. Les contraintes de capacité [LN87] sont une généralisation des contraintes de sous-tours. Les contraintes de peigne (plus généralement de chemin) sont généralisables sous un grand nombre de formes en fonction d'une part des demandes des divers éléments du peigne, et aussi de la position du dépôt relativement à ces éléments. Araque et al. [AHM90], en étudiant le PTV pour des commandes unitaires, introduisent de nouvelles

familles de contraintes tout comme Harche et Rinaldi [HR91]. Nous en introduirons aussi de nouvelles. Ces contraintes sont héritées moins du PVC que des problèmes de bin-packing et de sac à dos. Rinaldi [Rin] présente par ailleurs des contraintes valides pour le mPVC. La synthèse de ces résultats fait l'objet d'une partie du chapitre 2 de cette thèse.

Le second point expliquant le succès de l'approche polyédrale du PVC est la qualité des algorithmes de séparation, c'est-à-dire de l'analyse des solutions fractionnaires de PVC(L). Les contraintes les plus utilisées peuvent être séparées par des algorithmes exacts (sous-tour, 2-couplage) [GP85] ou des heuristiques rapides (peignes) [PR90], [CN94]. L'importance de ces contraintes pour le PTV est souvent faible. Les contraintes obtenues par généralisation et adaptation au problème de capacité ont plus de poids. Malheureusement, aucun algorithme exact n'est connu pour les séparer. Les premiers efforts dans ce sens ont été faits par Araque et al. [AKMP94] et Harche et Rinaldi [HR91]. Nous présentons au chapitre 3 des algorithmes plus efficaces pour le problème de séparation des contraintes spécifiques au PTV.

Enfin, un des obstacles majeurs à la résolution des problèmes est la gestion du nombre important de variables. Certaines spécificités du PTV font que ce problème est souvent moins bien «conditionné» que le PVC. Deux exemples le font comprendre très clairement. La solution optimale d'un PVC n'incluera que rarement des arêtes de grande longueur, c'est-à-dire qu'un sommet sera a priori relié à un de ses voisins les moins éloignés. Dans le cas du PTV, une contrainte de capacité peut amener à construire des tournées complexes, par exemple à avoir des tournées avec des arêtes longues ou même des arêtes qui se croisent alors que ceci est impossible dans le cas du PVC avec distances euclidiennes. La principale conséquence est la difficulté d'éliminer des arêtes a priori. C'est cette élimination (présélection en fait) qui permet de traiter des problèmes de PVC de très grande taille. Le deuxième argument s'appuie sur l'expérience numérique lors de la phase d'énumération implicite. Rinaldi [HR91] a conclu de ses tests que la séparation en sous-problèmes en fixant la valeur d'une variable était moins efficace dans le cas du PTV que pour le PVC. Ceci a donc aussi guidé notre recherche, outre le fait d'obtenir de meilleures bornes vers le test d'alternatives à la séparation sur la valeur des variables. Ceci sera discuté au chapitre 4.

Chapitre 2

Étude polyédrale du PTV

2.1 Formulation du problème

Nous présentons maintenant la terminologie et la formulation utilisées dans notre travail. Cette dernière entre dans la catégorie des formulations comme un problème de flot de véhicules. Il s'agit d'une dérivation de celle largement utilisée pour le PVC, c'est-à-dire la formulation comme un «flot de véhicules à deux index». Soit $G = (V, E)$ un graphe complet non orienté contenant $n + 1$ sommets numérotés $0, 1, \ldots, n$. Le sommet 0 correspond au dépôt et les autres sommets correspondent aux clients. L'ensemble des clients sera noté V_0 (ainsi $V = V_0 \cup \{0\}$). A chaque client $i \in V_0$, est associée une demande d_i. A chaque arête $e \in E$, est associée un coût l_e qui correspond à la distance entre les deux extrémités de l'arête e. Soit k le nombre fixé de véhicules disponibles au dépôt et C leur capacité. Pour un sous-ensemble F d'arêtes de E, $G(F)$ définit le sous-graphe $(V(F), F)$ induit par F, où $V(F)$ est l'ensemble de sommets incidents à au moins une arête de F .

Une **route** est définie comme un ensemble F non vide d'arêtes de E tel que le sous-graphe induit $G(F)$ est un cycle élémentaire contenant le dépôt 0 (i.e. $0 \in V(F)$, $G(F)$ est connexe et le degré de chaque sommet de $V(F)$ dans $G(F)$ est 2) et tel que la demande totale des clients dans $V_0(F) = V(F) \setminus \{0\}$ ne dépasse pas la capacité C. Une telle route représente le trajet d'un véhicule partant du dépôt, délivrant la commande des clients dans $V(F)$ (en circulant le long des arêtes de F) et retournant au dépôt. La longueur d'une route est la somme des longueurs des arêtes la définissant. Notons que si $|V(F)| = 2$ pour une route F, cette route contient deux fois la même arête et sa longueur

est deux fois celle de l'arête. Dans les autres cas (i.e. $|V(F)| > 2$), une arête ne peut apparaître qu'une seule fois dans une route. Une **k-route** est définie comme un sous-ensemble R d'arêtes de E qui peut être partitionné en k routes $R_1, R_2, \ldots, R_k$ et tel que chaque client $i \in V_0$ appartient à exactement un des ensembles $V(R_j)$, $1 \leq j \leq k$. La longueur d'une k-route est la somme des longueurs des k routes la définissant. Chaque k-route définit une solution réalisable du PTV. Le problème d'optimisation que nous traitons est celui de trouver une k-route de longueur minimale. La figure (2.1) représente une 3-route dans une instance de PTV avec 7 clients. Les chiffres indiqués à côté des sommets sont les demandes.

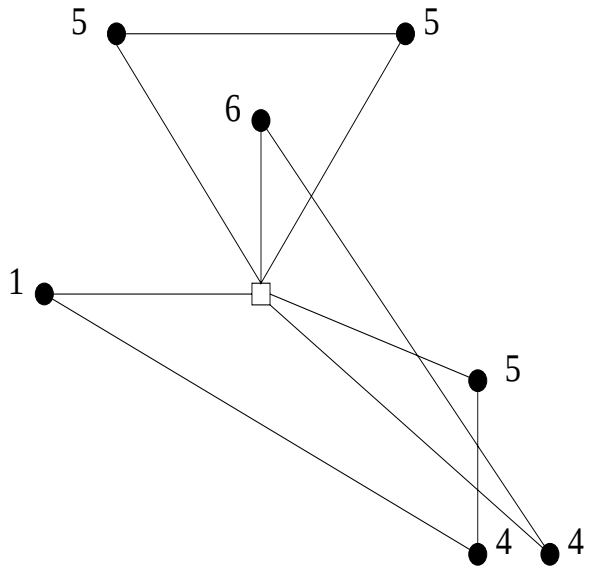


FIG. 2.1 - *Une solution d'un problème avec n=7, k=3, C=10*

Pour formuler le PTV comme un programme en nombres entiers, nous associons une variable x_e à chaque arête de E qui représente le nombre de fois où l'arête e est utilisée dans la solution (k-route). Quelquefois, il est plus pratique d'écrire x_{ij} au lieu de x_e, où e est l'arête entre les sommets i et j, $i < j$. C'est de cette écriture qu'est tiré le terme «deux index». Nous utiliserons aussi les notations suivantes. Pour des sous-ensembles de sommets $S, S' \subseteq V$, l'ensemble $\delta(S)$ (**cocycle** de S) est l'ensemble des arêtes avec une extrémité dans S et l'autre dans $V \setminus S$, $(\delta(S) = \{e = (i,j) \in E \ : \ i \in S, \ j \in V \setminus S\})$.

L'ensemble $(S : S')$ (**coupe** entre S et S') est l'ensemble des arêtes avec une extrémité dans S et l'autre dans S' ($(S : S') = \{e = (i,j) \in E \ : \ i \in S, \ j \in S'\}$). L'ensemble $\gamma(S)$ est l'ensemble des arêtes dont les deux extrémités sont dans S ($\gamma(S) = \{e = (i,j) \in E \ : \ i,j \in S\}$). La quantité $d(S)$ désigne la demande totale dans l'ensemble S ($d(S) = \sum_{i \in S} d_i$). Pour chaque sous-ensemble U d'arêtes, $x(U)$ est la somme des valeurs x_e pour toutes les arêtes $e \in U$. Maintenant, nous pouvons formuler le PTV comme le programme en nombres entiers suivant :

$$\min Z(PTV) = \sum_{e \in E} l_e \ x_e \tag{2.1}$$

$$x(\delta(\{0\})) = 2k \tag{2.2}$$

$$x(\delta(\{i\})) = 2 \qquad \forall i \in V_0 \tag{2.3}$$

$$x(\delta(S)) \geq \ 2 \left\lceil \frac{d(S)}{C} \right\rceil \qquad \forall S \subseteq V_0, \ S \neq \emptyset \tag{2.4}$$

$$0 \leq x_e \leq 1 \qquad \forall e \in \gamma(V_0) \tag{2.5}$$

$$0 \leq x_e \leq 2 \qquad \forall e \in \delta(\{0\}) \tag{2.6}$$

$$x_e \text{ entier} \qquad \forall e \in E. \tag{2.7}$$

La fonction objectif à minimiser, définie en (2.1) est la longueur d'une k-route. La contrainte (2.2) est la contrainte de degré du dépôt imposant que k véhicules quittent et reviennent au dépôt. Les contraintes (2.3) sont les contraintes de degré de chaque sommet client. Elles imposent que chaque véhicule soit servi par exactement un véhicule. Les contraintes (2.4) sont les contraintes de capacité. On définit $\lceil \alpha \rceil$ comme le plus petit entier supérieur ou égal à α. Ces contraintes expriment que pour un sous-ensemble donné de sommets S, au moins $\lceil d(S)/C \rceil$ véhicules sont nécessaires pour satisfaire la demande dans S. Comme le dépôt est hors de S, chaque véhicule doit entrer et ressortir de S. De plus, comme pour les contraintes d'élimination de sous-tour dans le cas du PVC, ces contraintes sont nécessaires pour obtenir une solution connexe. Les contraintes (2.5 - 2.7) imposent l'intégrité et les bornes des variables. Une variable x_e liant directement un client au dépôt (2.5) peut prendre une valeur de 2 quand cette arête e est la seule dans la route. Nous appellerons X^{PTV} l'ensemble des solutions du (PTV), i.e. les vecteurs satisfaisant (2.2 - 2.7). L'enveloppe convexe des vecteurs de X^{PTV} sera appelée $conv(X^{PTV})$ et aussi (par abus de langage) le polytope PTV.

Cette formulation a été utilisée par Laporte et al. [LND85], [LN87] puis

par Harche et Rinaldi [HR91], Cornuejols et Harche [CH93] et Achuttan et al. [ACH]. Son avantage est d'avoir un petit nombre de variables (exactement $n(n+1)/2$).

Campos et al. [CCM91] ont analysé la dimension du polyèdre PTV dans certains cas particuliers, comme celui où toutes les commandes valent 1. Dans le cas général, le calcul de la dimension du polytope est un problème NP-difficile (ce calcul contient un problème de bin packing). Cela rend difficile les démonstrations de facettes pour le PTV. A cause de cela, notre démarche, et la suite de ce chapitre ont été organisées de la manière suivante : trouver des contraintes valides pour PTV, puis choisir un polyèdre lié à PTV, si possible de pleine dimension et pour lequel il est plus facile de décider si une contrainte valide induit une facette de ce polyèdre.

2.2 Contraintes connues

2.2.1 Contraintes de capacité

Les contraintes de capacité (2.4), aussi appelées contraintes de sous-tour généralisées sont suffisantes pour obtenir une formulation entière du PTV mais ne sont pas, en général, des facettes du PTV. Des versions plus fortes de ces contraintes ont été présentées par Laporte et al. [LMN87] puis Cornuejols et Harche [CH93].

Notons $r(S)$ le nombre minimum de véhicules nécessaires pour satisfaire la demande dans un ensemble S de clients. Ce calcul est fait avec les données suivantes, la capacité des véhicules C et les demandes individuelles d_i pour $i \in S$. La valeur $r(S)$ est la solution d'un problème de bin packing, c'est-à-dire le nombre minimum de boîtes de taille C nécessaires pour ranger $|S|$ objets respectivement de taille $d_i,\ i \in S$. Ceci donne une contrainte de capacité renforcée

$$x(\delta(S)) \geq 2\, r(S) \qquad \forall S \subseteq V_0,\ S \neq \emptyset \tag{2.8}$$

La contrainte (2.8) est encore une contrainte locale au sens où les clients qui ne sont pas dans S et le nombre de véhicules k sont ignorés dans le calcul de $r(S)$. Une contrainte globale peut être construite en calculant, pour chaque sous-ensemble de clients, le nombre minimum de véhicules nécessaires pour

servir S parmi toutes les solutions de X^{PTV}. Notons cette quantité $R(S)$. Nous donnons maintenant une formulation mathématique de $R(S)$.

Une **k-partition** $\pi = (P_1, P_2, \ldots, P_k)$ est une partition des sommets de V_0 en k sous-ensembles telle que chaque sous-ensemble satisfait les contraintes $d(P_j) \leq C$ pour $1 \leq j \leq k$. Soit Π l'ensemble de toutes les k-partitions. Pour une k-partition donnée $\pi = (P_1, P_2, \ldots, P_k)$ et un ensemble de sommet S , on note $\beta_\pi(S)$ le nombre de membres de cette partition contenant au moins un sommet de S, c'est-à-dire

$$\begin{aligned} \beta_\pi(S) \quad = \quad & |\{j \in \{1, \ldots, k\} \ : \ P_j \cap S \neq \emptyset\}| \\ & \forall S \subseteq V_0 \ ; \ \pi = (P_1, P_2, \ldots, P_k) \in \Pi. \end{aligned} \tag{2.9}$$

En associant chaque k-partition à une affectation réalisable des clients aux véhicules, on obtient pour $S \subseteq V_0$

$$R(S) = \min_{\pi \in \Pi} (\beta_\pi(S)) \tag{2.10}$$

et les contraintes de capacité globales

$$x(\delta(S)) \geq \ 2 \ R(S) \qquad \forall S \subseteq V_0, \ S \neq \emptyset \tag{2.11}$$

Il est clair au vu de la présentation précédente que

$$R(S) \geq r(S) \geq \left\lceil \frac{d(S)}{C} \right\rceil \quad \forall S \subseteq V_0. \tag{2.12}$$

De plus, Cornuejols et Harche [CH93] présentent une instance du PTV où les contraintes de (2.12) sont strictes (voir la figure 2.2). Notons aussi que si le nombre de véhicules est illimité, on a $r(S) = R(S)$. D'un point de vue pratique, le calcul de $R(S)$ est plus difficile que celui de $r(S)$. Lorsque le nombre de clients dans S est petit, $r(S)$ peut être calculé de manière exacte (cf Martello et Toth [MT90b]) et pour des ensembles de plus grande taille une borne inférieure non triviale peut être calculée (et substituée à r(S) dans (2.8)) en un temps raisonnable (cf Martello et Toth [MT90a]). Si les calculs de $r(S)$ ou d'une borne inférieure deviennent trop coûteux, alors on utilisera la contrainte (2.4)

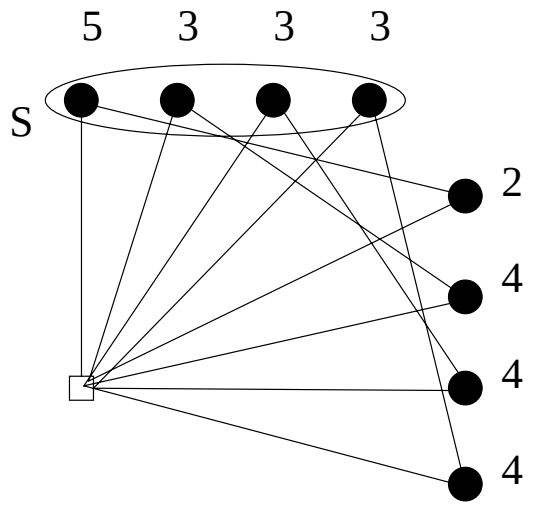


FIG. 2.2 - *Solution réalisable pour un problème, où* $k = 4$ *et* $C = 7$. *L'ensemble* S *satisfait* $R(S) > r(S) > \left\lceil \frac{d(S)}{C} \right\rceil$

2.2.2 Contraintes de peigne

Cette famille contient toutes les contraintes traduisant un problème de parité. Les contraintes de peignes ont été introduites par Chvátal [Chv73] et Grötschel et Padberg [GP79] pour le problème du voyageur de commerce. Elles jouent un rôle très important dans la résolution du PVC par «Branchement et Coupe». Un peigne est défini par un manche H et des dents $T_1, \ldots, T_s$ ayant les propriétés suivantes

$$\begin{array}{lll} (i) & H, T_1, T_2, \ldots, T_s \subseteq V & \\ (ii) & T_j \setminus H \neq \emptyset & \forall\ 1 \leq j \leq s \\ (iii) & T_j \cap H \neq \emptyset & \forall\ 1 \leq j \leq s \\ (iv) & T_i \cap T_j = \emptyset & \forall\ 1 \leq i < j \leq s \\ (v) & s \geq 3 \text{ et impair} & \end{array} \tag{2.13}$$

On peut définir alors la contrainte suivante

$$x(\gamma(H)) + \sum_{j=1}^{s} x(\gamma(T_j)) \leq |H| + \sum_{j=1}^{s} |T_j| - (3s+1)/2 \tag{2.14}$$

La figure (2.3) représente un support de peigne avec 3 dents.

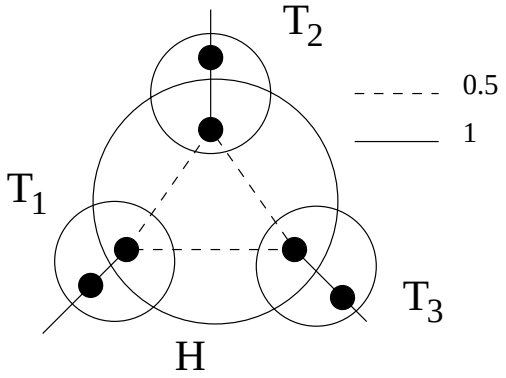


FIG. 2.3 - *Un support classique (et une violation) de peigne à 3 dents*

Dans le cas du PVC, pour chaque ensemble $S \subset V_0$ de sommets et chaque solution x satisfaisant les contraintes de degré , l'équation $2x(\gamma(S)) = 2|S| - x(\delta(S))$ est toujours satisfaite. Il s'agit en fait de la somme des contraintes de degré sur tous les sommets de S. Avec ce résultat, les contraintes de peigne peuvent être écrites de façon équivalente bien que moins standard

$$x(\delta(H)) \geq (s+1) \; - \; \sum_{j=1}^{s} \Big[x(\delta(T_j)) - 2 \Big] \tag{2.15}$$

Cette écriture a un intérêt double. Elle ne prend pas en compte le nombre d'éléments dans les différents ensembles. Elle s'interprète facilement en terme de cocycle. Si la contrainte de sous-tour $x(\delta(T_j)) \geq 2$ est serrée pour tout $i = 1, \ldots, s$ alors la valeur du cocycle $\delta(H)$ est au moins $s+1$. Cette borne peut être diminuée de deux unités si une contrainte de sous-tour n'est pas serrée et ainsi de suite. Nous utiliserons de nouveau cette écriture dans les sections suivantes.

Pour le cas du PTV et en tenant compte du degré du dépôt, Cornuejols et Harche [CH93] ont prouvé que les contraintes de peigne (2.15) restent valides indépendamment de la place du dépôt dans le manche et/ou dans une dent.

Lorsque le dépôt appartient à une dent mais pas au manche (i.e. sans perte de généralité, quand $0 \in T_1 \setminus H$), Cornuejols et Harche [CH93] prouvent que

la contrainte de peigne

$$x(\delta(H)) \geq (s+1) \; - \; \Big[x(\delta(T_1)) - 2p\Big] - \sum_{j=2}^{s} \Big[x(\delta(T_j)) - 2\Big] \qquad (2.16)$$

est valide pour $p = R(V \setminus T_1)$.

D'autre part, lorsque le dépôt est à l'extérieur du support du peigne, i.e. $H, T_1, T_2, \ldots, T_s \subseteq V_0$, Laporte et Norbert [LN84] et Araque [Ara90b] présentent des généralisations de la contrainte de peigne prenant en compte l'aspect «capacité» du problème. La première contrainte, due à Laporte et Norbert, prend en compte les capacités à l'intérieur de chaque dent T_j à travers la solution de bin packing $r(T_j)$. Cette contrainte peut être écrite dans une forme différente, mais au moins aussi forte.

$$x(\delta(H)) \geq (s+1) \; - \sum_{j=1}^{s} \Big[x(\delta(T_j)) - 2r(T_j)\Big] \qquad (2.17)$$

où toutes les dents T_j, $1 \leq j \leq s$, doivent satisfaire : $r(T_j \setminus H) + r(T_j \cap H) > r(T_j)$.

Araque considère le cas spécial des demandes unitaires, $d_i = 1$ pour $i \in V_0$. Sa contrainte de peigne prend en compte la demande de chaque dent T_j ainsi que la demande de l'union de deux dents. Les dents doivent satisfaire la condition $\lceil d(T_j)/C \rceil = \lceil d(T_j \setminus H)/C \rceil$ ce qui implique $\lceil d(T_j \cap H)/C \rceil \leq 1$. On distinguera deux types de dents. Une dent T est dite large si pour toute autre dent $\tilde{T}$, $\lceil d(T \cup \tilde{T})/C \rceil = \lceil d(T)/C \rceil + \lceil d(\tilde{T})/C \rceil$. Une dent T est dite petite si pour toute autre dent non large $\tilde{T}$ du peigne, $\lceil d(T \cup \tilde{T})/C \rceil = \lceil d(T)/C \rceil + \lceil d(\tilde{T})/C \rceil \; - 1$. La contrainte de peigne d'Araque n'est définie que lorsqu'on peut classer toutes les dents dans l'une ou l'autre des catégories. La condition (v) dans la liste de propriétés (2.13) du peigne est remplacée par $s = s_s + s_l$ (où s_s est le nombre de petites dents et s_l est le nombre de grandes dents) avec s_s impair ou nul. La contrainte de peigne d'Araque peut être écrite

$$x(\delta(H)) \geq 2\left\lfloor \frac{s_s+1}{2} \right\rfloor \; + 2s_l \; - \sum_{j=1}^{s} \left[x(\delta(T_j)) - 2\left\lceil \frac{d(T_j)}{C} \right\rceil\right] \qquad (2.18)$$

2.2.3 Contraintes de capacité généralisée

Harche et Rinaldi [HR91] généralisent les contraintes de capacité (2.11) en considérant en même temps plusieurs sous-ensembles de sommets. Soit $\Omega = \{S_1, S_2, \ldots, S_s\}$ une famille d'ensembles disjoints de V_0. On définit

$$\Re(\Omega) = \min_{\pi \in \Pi} \left(\sum_{j=1}^{s} \beta_\pi(S_j) \right) \tag{2.19}$$

où $\beta_\pi(S_j)$ est défini dans (2.9)). Le nombre $\Re(\Omega)$ est le nombre minimum de fois, parmi toutes les k-partitions, où un véhicule visite un ensemble S_j, $1 \leq j \leq s$. Ceci donne directement la contrainte de capacité généralisée

$$\sum_{j=1}^{s} x(\delta(S_j)) \geq 2\ \Re(\Omega) \tag{2.20}$$

La contrainte (2.20) est au moins aussi forte que la somme des contraintes (2.11) sur tous les sous-ensembles S_j, $1 \leq j \leq s$ car

$$\Re(\Omega) \geq \sum_{j=1}^{s} \left(\min_{\pi \in \Pi}\ \beta_\pi(S_j) \right) = \sum_{j=1}^{s} R(S_j) \tag{2.21}$$

et la contrainte (2.20) est plus forte que la somme des contraintes (2.11) si les $x(\delta(S_j))$, $1 \leq j \leq s$, ne prennent pas leur valeur minimum $2R(S_j)$ en même temps. C'est le cas dans la figure (2.4), où la demande des trois sous-ensembles est 6, alors que l'on dispose de deux véhicules de capacité 10.

2.2.4 Contraintes d'étoiles

Araque et al. [AHM90] ont introduit diverses contraintes valides pour le problème de tournées de véhicules lorsque les commandes des clients sont toutes égales à l'unité. Ces contraintes restent en général valides pour le cas de demandes non unitaires. Nous présentons ici celle que nous utilisons dans notre algorithme de «Branchement et Coupe».

Soit v un sommet de V_0 et $\Omega = \{T_i \ : \ i = 1, \ldots, s\}$ une famille de sous-ensembles de V_0 satisfaisant les conditions suivantes

$$\begin{array}{lll} (i) & d(T_i) \leq C & \forall i \\ (ii) & d(T_i \cup T_j) > C & \forall i, j\ i \neq j \\ (iii) & T_i \cap T_j = \{v\} & \forall i, j\ i \neq j \end{array}$$

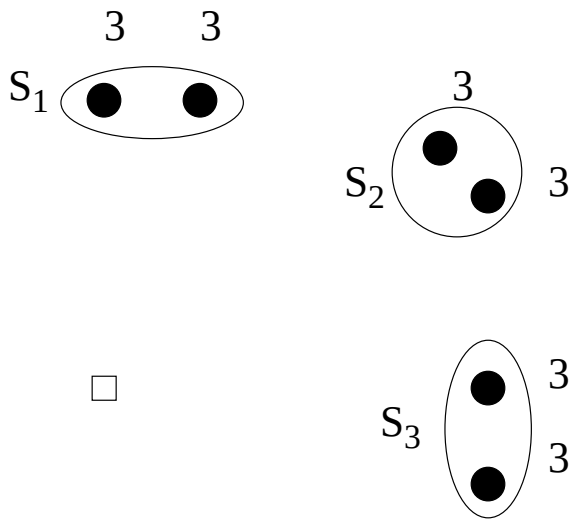


FIG. 2.4 - *Cas où la contrainte* $\Re(\Omega) \geq \sum_{j=1}^{s} R(S_j)$ *est stricte. On a* $C = 10$, $k = 2$ *et* $s = 3$.

alors la contrainte d'étoile suivante

$$\sum_{i=1}^{s} x(\delta(T_i)) \;\geq\; 4\ s \;-2 \tag{2.22}$$

est valide pour le PTV.

La figure (2.5) montre un support de contrainte d'étoile.

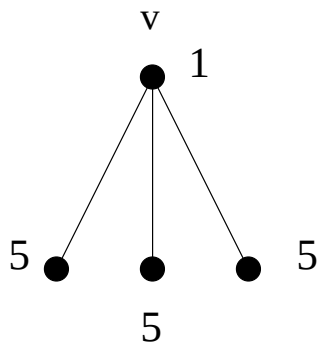


FIG. 2.5 - *Support d'une contrainte d'étoile, C=10*

Nous présentons plus loin une généralisation de cette contrainte.

2.3 Nouvelles contraintes

Les résultats contenus dans cette section sont le fruit d'une collaboration avec Yves Pochet. L'essentiel des résultats de (2.3.3) sont dus à Yves Pochet, et inversement pour (2.3.2).

2.3.1 Contraintes de boîte

Nous étudions maintenant le cas où Ω est une famille d'ensembles tous disjoints, sauf un ensemble incluant tous les autres. Nous présentons d'abord le cas où les sous-ensembles disjoints ont chacun une demande totale inférieure à la capacité. Soient $H, T_1, \ldots, T_s$ des sous-ensembles de V_0 ayant les propriétés suivantes

$$\begin{array}{lll} (i) & d(T_j) \leq C & \forall\ 1 \leq j \leq s \\ (ii) & T_j \subset H & \forall\ 1 \leq j \leq s \\ (iii) & T_i \cap T_j = \emptyset & \forall\ 1 \leq i < j \leq s \\ (iv) & s \geq 1 & \end{array} \tag{2.23}$$

Nous définissons maintenant un problème de bin packing sur H. Soit I l'ensemble des objets de ce problème. Chaque ensemble $T_j,\ 1 \leq j \leq s$ définit un objet de taille $d(T_j)$ dans I. Chaque client $i \in H \setminus (\cup_{j=1}^{s} T_j)$ définit un objet de taille d_i dans I. La taille des boîtes est égale à la capacité d'un véhicule C. Soit $bp(H|T_1, \ldots, T_s)$ le nombre minimum de boîtes nécessaires pour ranger les objets de I.

Proposition 2.3.1 *La contrainte*

$$x(\delta(H)) \geq\ 2\ bp(H|T_1, \ldots, T_s)\ -\ \sum_{j=1}^{s} \Big[x(\delta(T_j)) - 2\Big] \tag{2.24}$$

est valide pour le polyèdre PTV.

Cette contrainte se lit de la manière suivante. Si tous les cocycles $x(\delta(T_j))$ sont serrés (i.e. égaux à la valeur minimum possible), la quantité $2bp(H|T_1, \ldots, T_s)$ est une borne inférieure du cocycle $x(\delta(H))$. Si un des cocycles (par exemple

$x(\delta(T_1)))$ est lâche, on peut diminuer de $x(\delta(T_1)) - 2$ la borne inférieure et ainsi de suite.

Un exemple de point fractionnaire violant une contrainte de ce type peut être trouvé dans la section (3.3) sur la séparation de ces contraintes.

Preuve :

Nous appellerons coupure d'un objet t dans I, l'opération suivante : on retire t de I et on ajoute à I deux nouveaux objets dont la somme des tailles est égale à la taille de t. Une propriété du problème de bin packing est qu'on économise au plus une boîte en effectuant une opération de coupure. En effet, supposons que la solution du problème de bin packing après une coupure soit p. On peux alors construire une solution du problème de bin packing initial avec seulement $p+1$ boîtes. Il suffit pour cela d'ajouter à la solution avec p boîtes, une boîte où on placera les 2 parties de l'objet coupé. On a donc $p + 1 \geq bp(H|T_1, \ldots, T_s)$. Comme l'ensemble d'objets obtenus après une coupure satisfait encore les conditions (2.23), une récurrence simple permet de voir que si on fait q coupures, on économisera au plus q boîtes. En terme de solution du PTV, la quantité $w = \sum_{j=1}^{s} \left\lceil x(\delta(T_j)) - 2 \right\rceil$ représente exactement deux fois le nombre de coupures dans I pour la solution x. Le nombre de véhicules nécessaires pour servir H dans x est supérieur au résultat du problème de bin packing après $w/2$ coupures et donc à $bp(H|T_1, \ldots, T_s) - w/2$ $\diamond$

Notons que la contrainte précédente a un intérêt seulement si

$bp(H|T_1, \ldots, T_s) > r(H)$. Dans le cas contraire, la contrainte est l'addition de contraintes de capacité. De plus, comme $x(\delta(T_j)) \geq 2$ dans toute solution de PTV, un ensemble T_j ne sera mis dans la contrainte que si son retrait fait réduire la valeur de la solution du sous-problème de bin packing. Autrement, la contrainte sans l'ensemble T_j dominerait celle avec T_j.

Dans le cas général, où la condition (i) de (2.23) n'est pas satisfaite, l'ensemble I n'est pas défini. On définit alors un nouveau problème de bin packing. Soit I' l'ensemble des objets de ce problème. Tout sommet $i \in H \setminus (\cup_{j=1}^{s} T_j)$ correspond à un objet de taille $d(i)$. Tout ensemble $T_j : j = 1, \ldots, s$ satisfaisant (i) correspond à un objet de taille $d(T_j)$ dans I'. Tout autre ensemble T_j correspond dans I' à q objets de taille C et un objet de taille $d(T_j) - qC$ avec $q = \lfloor d(T_j)/C \rfloor$. On note encore $bp(H|T_1, \ldots, T_s)$ la solution de ce problème de bin packing. On dira que I' est un découpage des

T_j.

Lemme 2.3.2 *Soit $(H, T_1, \ldots, T_s)$ une famille d'ensembles satisfaisant les conditions (ii), (iii) et (iv) de (2.23).*

Si $\lceil d(T_1 \cup T_2)/C \rceil = \lceil d(T_1)/C \rceil + \lceil d(T_2)/C \rceil$, on a

$$bp(H|T_1, \ldots, T_s) \geq bp(H|T_1 \cup T_2, T_3, \ldots, T_s) \tag{2.25}$$

et sinon

$$bp(H|T_1, \ldots, T_s) + 1 \geq bp(H|T_1 \cup T_2, T_3, \ldots, T_s) \tag{2.26}$$

Preuve :

Considérons le découpage I' pour les ensembles $T_1, \ldots, T_s$. Soient t_1 et t_2 des objets issus respectivement de T_1 et T_2 de taille minimum. Notons que tous les autres objets issus de T_1 et T_2 ont une taille égale à C. Si $d(t_1) + d(t_2) \leq C$, on peut transformer la solution optimale du problème de bin packing défini sur $(H, T_1, \ldots, T_s)$ en une solution avec une boîte de plus contenant uniquement t_1 et t_2. Cette dernière solution est aussi une solution réalisable du problème de bin packing défini sur $(H, T_1 \cup T_2, \ldots, T_s)$, car elle partitionne $T_1 \cup T_2$ en $(\lceil d(T_1)/C \rceil - 1) + (\lceil d(T_2)/C \rceil - 1) + 1 = \lceil d(T_1 \cup T_2)/C \rceil$ objets dont au plus un n'est pas de taille C. On a dans ce cas $bp(H|T_1, \ldots, T_s) + 1 \geq bp(H|T_1 \cup T_2, T_3, \ldots, T_s)$.

Si $d(t_1) + d(t_2) \geq C$, notons b_1 et b_2 les boîtes auxquelles ces objets sont affectés dans la solution optimale du problème de bin packing défini sur $(H, T_1, \ldots, T_s)$. On remplace alors ces 2 objets par le couple d'objets de taille respectvement C et $d(t_1) + d(t_2) - C$, On met dans b_1 l'objet de taille C et dans b_2 celui de taille $d(t_1) + d(t_2) - C$ et tous les autres objets se trouvant au départ dans b_1 ou b_2. Ceci forme une solution qui partitionne $T_1 \cup T_2$ en $\lceil d(T_1)/C \rceil + \lceil d(T_2)/C \rceil = \lceil d(T_1 \cup T_2)/C \rceil$ objets dont au plus un n'est pas de taille C. C'est donc une solution réalisable du problème de bin packing défini sur $(H, T_1 \cup T_2, T_3, \ldots, T_s)$. On a dans ce cas $bp(H|T_1, \ldots, T_s) \geq bp(H|T_1 \cup T_2, T_3, \ldots, T_s)$. Notons que $(H, T_1 \cup T_2, T_3, \ldots, T_s)$ satisfait encore les conditions (ii), (iii) et (iv) de (2.23) $\diamond$

Théorème 2.3.3

$$x(\delta(H)) \geq 2\ bp(H|T_1, \ldots, T_s) - \sum_{j=1}^{s} \left[x(\delta(T_j)) - 2\ \lceil d(T_j)/C \rceil \right] \tag{2.27}$$

est valide pour le PTV.

Preuve :

Soit x une solution du PTV. Pour chaque $j = 1, \ldots, s$, cette solution partitionne T_j en $p(j) = 1/2\ x(\delta(T_j))$ sous-ensembles tels que chaque sous-ensemble $T_j^k,\ k = 1, .., p(j)$ satisfait $x(T_j^k) = 2$. Il suit de la proposition (2.3.1) que

$x(\delta(H)) \geq 2bp(H|T_1^1, \ldots, T_1^{p(1)}, T_2^1, \ldots, T_s^{p(s)})$.

Pour chaque ensemble T_j, on applique itérativement $p(j)-1$ fois le lemme (2.3.2) en regroupant à chaque fois deux sous-ensembles de T_j. Notons α_p le nombre de fois où on diminue la borne de 1 pour les p premières itérations. Le lemme (2.3.2) exprime que $\alpha_1 = \lceil d(T_j^1)/C \rceil + \lceil d(T_j^2)/C \rceil - \lceil d(T_j^1 \cup T_j^2)/C \rceil = 2 - \lceil d(T_j^1 \cup T_j^2)/C \rceil$. Supposons que $\alpha_p = p + 1 - \lceil \sum_{i=1}^{p}\ d(T_j^i)/C \rceil$, on a alors $\alpha_{p+1} = \alpha_p + \lceil d(T_j^{p+1})/C \rceil + \lceil \sum_{i=1}^{p}\ d(T_j^i)/C \rceil - \lceil \sum_{i=1}^{p+1}\ d(T_j^i)/C \rceil = p + 2 - \lceil \sum_{i=1}^{p+1}\ d(T_j^i)/C \rceil$. On tire de cette récurence que

$x(\delta(H) \geq 2bp(H|T_1, \ldots, T_s) - \ 2\ \sum_{i=1}^{s}\ \Big[p(j) - \lceil d(T_j)/C \rceil\Big]$,

ce qui est le résultat annoncé $\diamond$

Un cas particulier de cette contrainte est obtenu avec $H = V_0$. On retrouve alors la contrainte de capacité généralisée de la section précédente. La contrainte (2.24) peut s'écrire

$\sum_{j=1}^{s} x(\delta(T_j)) \geq 2\ bp(H|T_1, \ldots, T_s)\ -\ 2\ k\ +\ 2\ s$.

On a donc $\Re(T_1, \ldots, T_s) \geq bp(V_0|T_1, \ldots, T_s) + s - k$.

2.3.2 Contraintes d'hypotour

Nous présentons maintenant une nouvelle classe de contraintes valides pour le PTV. Nous allons rechercher des sous-graphes (V, F) du graphe complet (V, E) dans lequel on ne peut pas construire de solution du PTV. Cette étude a déjà été réalisée pour le PVC. Grötschel [Grö80], Papadimitriou et Yannakakis [PY84] ont dérivé des contraintes à partir de certaines classes de graphes non hamiltoniens. L'intérêt de ces contraintes est limité dans le cas du PVC par la complexité de leur identification (cf Grötschel et Padberg [GP85]). Dans le cas du PTV, les demandes sur les clients introduisent de nouveaux critères pour savoir si un sous-graphe contient une solution réalisable. Dans leur expérimentation de l'algorithme de «Branchement et

Coupe», Cornuejols et Harche [CH93] identifient visuellement de tels sous-graphes et nomment les contraintes correspondantes des contraintes d'hypotour, suivant le nom donné par Grötschel et Padberg pour le PVC. Nous présentons maintenant deux types de sous-graphe pouvant mener à des contraintes valides puis nous donnerons quelques extensions et montrerons que ces contraintes généralisent des contraintes connues.

Soit S_n l'ensemble des sous-ensembles d'arêtes de solutions de PTV. Pour chaque sous-ensemble $F \subseteq E$, on peut définir un nombre $r(F)$ appelé rang de F, $r(F) = \max\{|S| \ : \ S \subset F \ et \ S \in S_n\}$. Une conséquence triviale de la définition est que la contrainte $x(F) \leq r(F)$ est valide pour le polyèdre PTV. Ces contraintes sont d'ailleurs valides pour tout problème combinatoire et sont en général appelées des **contraintes de rang**. Pour tout sous-graphe G' de G, on note par $V(G')$ et $E(G')$ l'ensemble des sommets et des arêtes de G'. On dit que $E(G')$ est un **hypotour**, si $E(G')$ ne contient pas de solution de PTV mais que $E(G') \cup \{e\}$ en contient pour toute arête $e \in E \setminus E(G')$. Comme le nombre d'arêtes de toute solution réalisable est $n+k$, la contrainte d'hypotour $x(E(G')) \leq n+k-1$ est une contrainte de rang valide pour PTV. Le but des paragraphes qui suivent est de décrire quelques classes de graphes qui sont des hypotours.

Contrainte d'arbre de rang 1

Une définition plus restrictive du rang permet d'obtenir un premier type de contrainte. Soit v un sommet de V et S_v l'ensemble des sous-ensembles d'arêtes de routes contenant v dans les solutions du PTV. Pour chaque sous-ensemble $F \subseteq E$, on définit le nombre $r_v(F) = \max\{|S| \ : \ S \subset F \ et \ S \in S_v\}$. Pour un sommet $v \in V$, on définit alors une **contrainte d'arbre** (ce nom sera justifié plus loin) à partir d'un ensemble d'arêtes F_v qui intersecte tous les tours contenant v. La contrainte $x(F_v) \geq n + k - r_v(E \setminus F_v)$ est alors valide et équivalente à la contrainte de rang $x(E \setminus F_v) \leq r_v(E \setminus F_v)$.

Si $r_v(E \setminus F_v) = n+k-1$, la contrainte $x(F_v) \geq 1$ est une contrainte d'arbre de rang 1. Si F_v est un ensemble minimal pour cette propriété, $x(F_v) \geq 1$ est une contrainte d'hypotour. Pour donner une idée de la structure et de la puissance des contraintes d'arbre, nous introduisons maintenant un support générique de ces contraintes.

Soit $v \in V$ la racine d'un arbre $A \subseteq G_0$. On partitionne $V(A)$ en deux ensembles, N_f l'ensemble des feuilles de A et N_{nf} l'ensemble des sommets qui ne sont pas des feuilles. On appellera A-chemin un chemin inclus dans A entre deux feuilles de A et contenant la racine v. On appellera v-chemin un chemin dans A entre v et une feuille de A. On note par ailleurs $F(A)$ l'ensemble des arêtes incidentes aux sommets de $N_{nf} \cup \{v\}$ qui ne sont pas dans l'arbre A : $F(A) = (\gamma(N_{nf} \cup \{v\}) \cup \delta(N_{nf} \cup \{v\})) \setminus E(A)$.

Lemme 2.3.4 *Si v n'est pas un sommet pendant de A, on a :*

toute tournée dans G contenant v contient au moins un A-chemin ou une arête de $F(A)$;

toute tournée dans G contenant v contient au moins un v-chemin ou deux arêtes de $F(A)$.

Preuve :

Soient x une solution de PTV et T_v la tournée contenant v induite par x. Si T_v ne contient pas d'arêtes de $F(A)$, toutes les arêtes de T_v incidentes aux sommets de N_{nf} sont dans $E(A)$. Il existe alors un A-chemin dans T_v. Si T_v contient exactement une arête de $F(A)$, toutes les arêtes de T_v incidentes aux sommets de N_{nf} sauf une sont dans $E(A)$. Il existe alors un v-chemin dans T_v $\diamond$

Si v est un sommet pendant de A, un v-chemin est aussi un A-chemin et on a le lemme suivant.

Lemme 2.3.5 *Si v est un sommet pendant de A, on a :*

toute tournée dans G contenant v contient au moins un v-chemin ou deux arêtes de $F(A)$.

Preuve :

Soient x une solution de PTV et T_v la tournée contenant v induite par x. Comme le degré de v est deux dans T_v et un dans $E(A)$, T_v contient au moins une arête de $F(A)$. Si elle en contient exactement une, toutes les autres arêtes de T_v incidentes aux sommets de $N_{nf} \cup \{v\}$ sont dans $E(A)$ et T_v contient un v-chemin $\diamond$

Ces deux lemmes appliqués à des conditions sur les A-chemins ou les v-chemins d'un arbre A vont nous permettre de définir des contraintes valides de PTV. Supposons que tous les A-chemins ont une demande strictement supérieure à la capacité. On peut alors définir un sous-graphe incluant A et ne contenant pas de tournées passant par v.

Lemme 2.3.6 *Soit (A, v, N_f, N_{nf}) un arbre tel que v n'est pas une feuille de A et tel que tout A-chemin P satisfait $d(V(P)) > C$ alors la contrainte*

$$x(E(A)) \; + \; x(\gamma(N_{nf})) \leq 2 \; |N_{nf}| \; - 1 \tag{2.28}$$

est équivalente à la contrainte d'arbre

$$x(F(A)) \geq 1 \tag{2.29}$$

et elle est valide pour PTV.

Preuve :

En ajoutant les contraintes de degré pour tous les sommets de N_{nf}, on obtient :

$2 \; |N_{nf}| \; = \; 2 \; x(\gamma(N_{nf})) + x(\delta(N_{nf}))=$
$x(\gamma(N_{nf})) \; + x((\gamma(N_{nf}) \cup \delta(N_{nf})) \cap E(A)) \; + \; x((\gamma(N_{nf}) \cup \delta(N_{nf})) \setminus E(A))$.

Comme $E(A) \subset \gamma(N_{nf}) \cup \delta(N_{nf})$, on a $2 \; |N_{nf}| \; = \; x(\gamma(N_{nf})) + x(E(A)) + x((\gamma(N_{nf}) \cup \delta(N_{nf})) \setminus E(A))$.

Comme $F(A) = (\gamma(N_{nf}) \cup \delta(N_{nf})) \setminus E(A)$. La contrainte (2.28) est équivalente à $x(F(A)) \geq 1$. La condition du lemme implique qu'aucune tournée contenant v ne contient de A-chemin ce qui grâce au lemme (2.3.4) permet de conclure $\diamond$

La figure (2.6) montre un arbre A satisfaisant les conditions du lemme précédent pour $C = 10$ et $N_{nf} = \{v, 4\}$. A cet arbre correspondent 7 A-chemins tous de demande 11.

En réalité, la contrainte $x(F(A)) \geq 1$ peut être renforcée facilement. Nous présenterons plus loin une procédure permettant d'obtenir une meilleure contrainte lorsque le nombre de véhicules est illimité. Dans le cas général, on peut quand même renforcer la contrainte précédente. Pour cela, on étend les notions de A-chemin et de v-chemin. Pour un sous-graphe $T \subseteq G$, on dira que P est un **A-chemin** de T si $P \subseteq T$ et P contient au moins la racine de A et deux feuilles de A. On dira aussi que P un **v-chemin** de T si $P \subseteq T$ et P contient au moins la racine v de A et une feuille de A différente du sommet v.

Théorème 2.3.7 *Soient (A, v, N_f, N_{nf}) un arbre et A_* un sous-graphe de G tel que :*

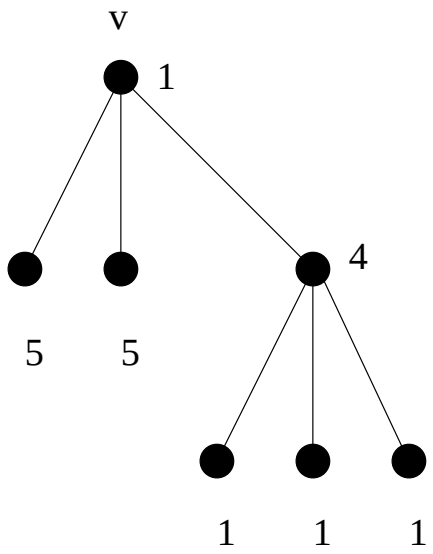


FIG. 2.6 - *Support d'une contrainte d'arbre*

(i) $V(A) = V(A_*)$
(ii) $E(A) \subseteq E(A_*) \subseteq E(A) \cup \gamma(N_{nf}) \cup \delta(N_{nf})$
(iii) tout A-chemin P *de* A_* *satisfait* $d(V(P)) > C$
alors

$$x((\gamma(N_{nf}) \cup \delta(N_{nf})) \setminus E(A^*)) \geq 1 \tag{2.30}$$

est une contrainte valide pour PTV.

Preuve :

La condition du théorème implique la condition du lemme précédent donc $x(F(A)) \geq 1$ est valide pour PTV. Soit x une solution de PTV. Supposons $x(F(A)) = x(F(A) \cap E(A^*))$, alors toutes les arêtes incidentes aux sommets de N_{nf} sont dans $E(A^*)$. Il existe donc un A-chemin de A^* dans la route contenant v. Ceci est impossible vu la condition (iii). On a donc $x(F(A)) > x(F(A) \cap E(A^*))$ ce qui implique $x((\gamma(N_{nf}) \cup \delta(N_{nf})) \setminus E(A^*)) \geq 1$ $\diamond$

Notons que le théorème reste valide si on remplace $d(V(P)) > C$ par $R(V(P)) > 1$ dans la condition (iii). Les démonstrations sont inchangées si on modifie la définition d'une route F, en remplaçant la condition $d(V_0(F)) \leq C$ par $R(V_0(F)) = 1$, ce qui est tout à fait valide.

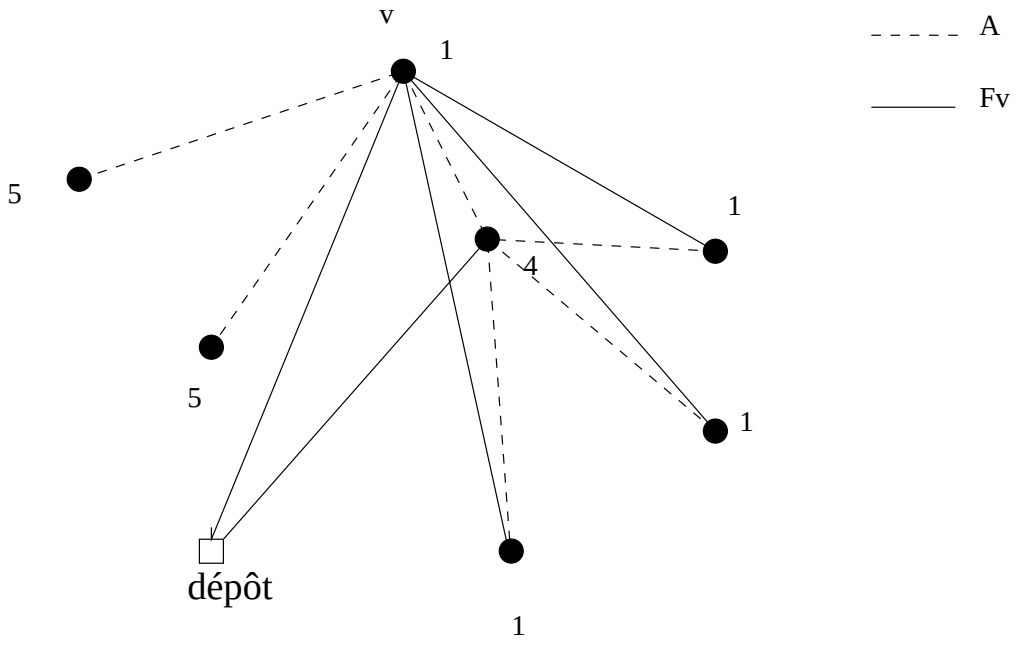


FIG. 2.7 - $C = 10$

La figure (2.7) montre un graphe avec 8 sommets. Les chiffres indiqués sont les demandes et les lignes pointillées correspondent à l'arbre A. L'ensemble d'arêtes $F(A)$ a été remplacé par l'ensemble F_v pour obtenir une contrainte d'arbre minimale. Pour cela, on a construit un graphe A^* contenant A et les arêtes entre les sommets de demande 5 et 4. Ce graphe satisfait les propriétés du théorème. L'ensemble $F_v = F(A) \setminus E(A^*)$ est dessiné en lignes pleines.

La figure (2.8) est une solution fractionnaire qui viole $x(F(A)) \geq 1$ et $x(F_v) \geq 1$. On peut vérifier par énumération que les différentes contraintes de capacité sont satisfaites par cette solution.

Contrainte de forêt

La non faisabilité d'un sous-graphe va maintenant être caractérisée comme une violation de bin packing. Soit $S = \{v_i \ : \ i = 1, \ldots, s\}$, un ensemble de sommets qui sont les racines des arbres d'une forêt $A(S) = \{A_i \ : \ i = 1, \ldots, s\}$. On définit pour chaque arbre A_i l'ensemble des feuilles $N_f(A_i)$ et celui des non feuilles $N_{nf}(A_i)$. Si v_i est une feuille de A_i, on définit un A_i-

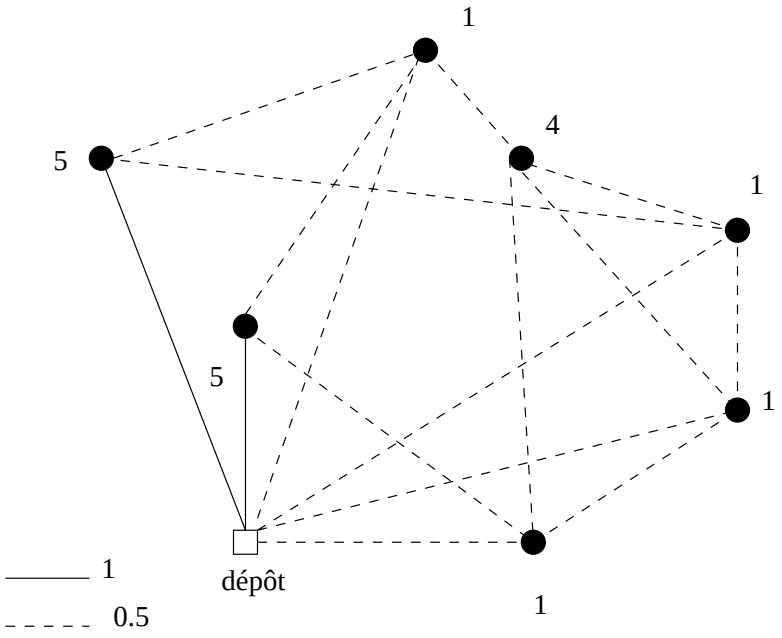


FIG. 2.8 - *Violation de contrainte d'arbre de rang 1*

chemin de A_i comme un chemin entre v_i et une autre feuille de A_i. Sinon, un A_i-chemin de A_i est défini comme à la section précédente. Pour donner une idée simple de la signification d'une contrainte de forêt, nous faisons le parallèle avec les contraintes de capacité généralisée. Dans ces dernières, on additionne des contraintes de capacité en remarquant que ces contraintes de capacité ne peuvent être serrées en même temps. Ici, on suppose que tout A_i-chemin de A_i peut être un sous-graphe d'une k-route, mais que ce n'est pas le cas pour une forêt de s chemins respectivement A_i-chemins de A_i, pour $i = 1, \ldots, s$. La figure (2.9) montre un exemple de forêt. Il y a dans cet exemple trois A_1-chemins possibles tous de demande 5, exactement un A_2-chemin et un A_3-chemin tous les deux de demande 6. Comme la capacité est 10 et le nombre de véhicules 2, cette forêt satisfait bien les conditions précédentes.

Pour $i = 1, \ldots, s$, on notera $F(A_i) = (\gamma(N_{nf}(A_i) \cup \{v_i\}) \cup \delta(N_{nf}(A_i) \cup \{v_i\})) \setminus E(A_i)$.

Théorème 2.3.8 *Si la condition suivante est satisfaite :*

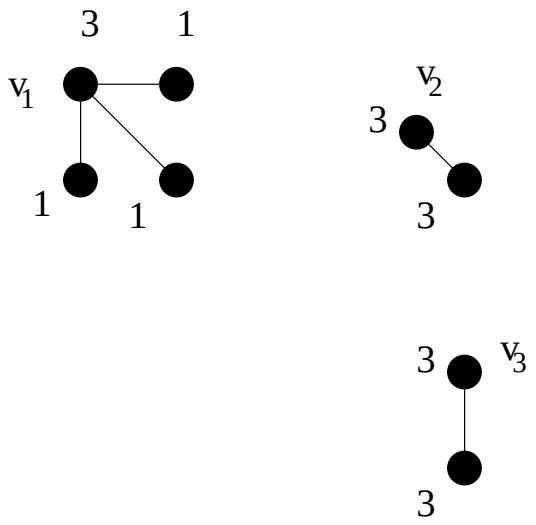


FIG. 2.9 - *Support d'une contrainte de forêt, C=10, k=2*

pour toute famille $(P_1, ..., P_s)$ *de chemins tels que* P_i, $i = 1, \ldots, s$ *est un* A_i*-chemin, on a* $bp(V_0|V(P_1), \ldots, V(P_s)) > k$ *, la contrainte*

$$\sum_{i=1}^{s} x(F(A_i)) \geq 1 + \sum_{i=1}^{s} |N_f(A_i) \cap \{v_i\}| \tag{2.31}$$

est valide pour PTV.

Preuve :

Soit x une solution réalisable de PTV. Notons d'abord, indépendamment de l'hypothèse du théorème, que la quantité $\sum_{i \in I} x(F(A_i))$ est toujours supérieure au nombre d'éléments v_i qui sont des feuilles. Ceci justifie le terme $\sum_{i \in I} |N_f(A_i) \cap \{v_i\}|$. L'hypothèse du théorème implique par ailleurs qu'il existe au moins un indice $i \in \{1, \ldots, s\}$ tel que le graphe induit par x ne contienne pas de A_i-chemin de A_i. Cette propriété et les lemmes (2.3.4) et (2.3.5) permettent de conclure $\diamond$

Contrainte d'arbre généralisée

Considérons maintenant une généralisation des contraintes d'arbre. Le support d'une contrainte d'arbre généralisée sera dérivé d'un support de contrainte d'arbre en remplaçant les feuilles de l'arbre par des cliques. La

figure (2.10) montre le support d'une contrainte d'arbre généralisée, obtenu en modifiant légèrement l'exemple illustrant les contraintes d'arbre de rang 1. On a dans cet exemple une clique formée de deux sommets, une clique formée de trois sommets et trois cliques formées d'un seul sommet.

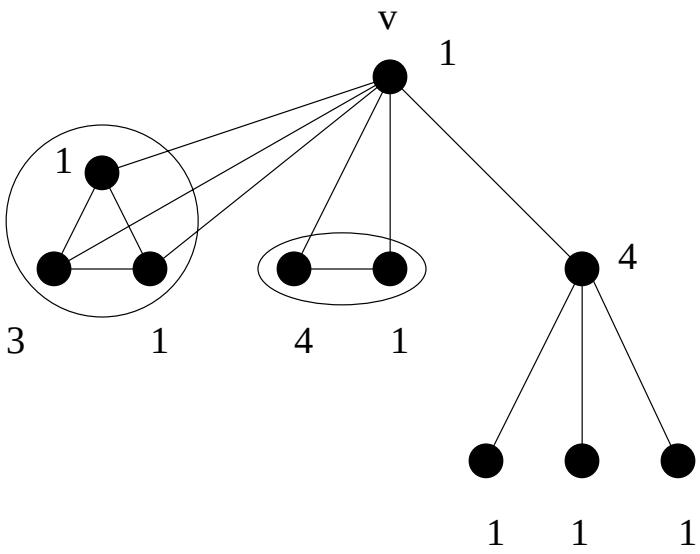


FIG. 2.10 - *Support d'une contrainte d'arbre généralisée, C=10*

Soit $v \in V$ un sommet non pendant d'un arbre A. Soient N_f et N_{nf} les ensembles des feuilles et non feuilles de A.

On définit $(W_i, \gamma(W_i))$, $i \in [1, \ldots, p]$, p cliques disjointes telles que tous les éléments d'une clique sont des feuilles et tous les éléments d'une même clique ont le même père dans A. On supposera par ailleurs que toute feuille appartient à exactement une clique et que $d(W_i) \leq C$, $i = 1, \ldots, p$.

Proposition 2.3.9 *Si la condition suivante est satisfaite :*

pour tout A-chemin P entre $i \in N_f$ et $j \in N_f$, $j \neq i$, on a $d(P \cup W_i \cup W_j) > C$, alors

$$x(E(A)) \; + \; x(\gamma(N_{nf})) \; + \; \sum_{i=1}^{p} x(\gamma(W_i)) \; \leq 2 \; |N_{nf}| \; + \; \sum_{i=1}^{p} (|W_i| - 1) \; - 1 \qquad (2.32)$$

est une contrainte valide de PTV.

Preuve :
La contrainte est équivalente à
$\sum_{i=1}^{p} x(\gamma(W_i)) \leq \sum_{i=1}^{p}(|W_i|-1)) + x(F(A)) - 1$
qui est la somme de p contraintes de sous-tour à un terme près. Ce terme $x(F(A)) - 1$ est toujours supérieur à -1, ce qui implique que la contrainte est satisfaite par toute solution de PTV pour laquelle au moins une des p contraintes de sous-tour est lâche. Dans le cas contraire, aucune tournée dans cette solution ne peut contenir de A-chemin, car celui ci contiendrait alors deux cliques et aurait par hypothèse une demande supérieure à C. On peut alors utiliser le lemme (2.3.4) pour conclure $\diamond$

Notons que si $N_{nf} = \{v\}$, la contrainte (2.32) est la contrainte d'étoile décrite dans la section (2.2.4).

Contrainte d'arbre de rang 2

Nous considérons maintenant un ensemble d'arêtes F tel que $r_v(E \setminus F) = n + k - 2$ (la quantité r_v est définie en section (2.3.2)). On peut alors dériver de nouvelles contraintes même si $E \setminus F$ n'est plus un hypotour, au sens où nous l'avons défini.

Soit $v \in V$ la racine d'un arbre A tel que v ne soit pas une feuille de A. Soient N_f et N_{nf} les ensembles des feuilles et non feuilles de A. Pour chaque sommet $i \in N_f$, on définit P_i le v-chemin entre v et i dans A.

Proposition 2.3.10 *Si pour tout $i \in N_f$, $d(P_i) > C$ alors*

$$x(F(A)) \geq 2 \tag{2.33}$$

est valide pour PTV.

Preuve :
La condition de la proposition implique qu'une tournée contenant v ne contient pas de v-chemin, ce qui permet de conclure en utilisant le lemme (2.3.5) $\diamond$

Comme exemple de support de contrainte d'arbre de rang 2, on peut prendre la figure (2.6) en supposant cette fois la capacité égale à 5.

2.3.3 Contraintes de chemin-boîte

Nous présentons dans cette section la contrainte de chemin-boîte pour le polyèdre du problème de tournées de véhicules. Cette contrainte est une extension des contraintes de peigne et de boîte présentées auparavant, i.e. elle prend en compte à la fois les problèmes de parité et de bin packing.

Les contraintes de peigne pour le PTV ne considèrent pas l'aspect «capacité» (ou «bin packing» de manière globale mais cet aspect est vu au mieux pour les dents prises séparément ou par deux. De même, les contraintes de boîte ne prennent pas du tout en compte les problèmes de parité. Le but des contraintes de chemin-boîte est d'intégrer ces aspects, en conservant un support de contrainte proche de celui d'un peigne.

Le support d'une contrainte de chemin-boîte est défini par un manche H et un certain nombre de sous-ensembles $T_1, T_2, \ldots, T_s$ appelés dents ou spots satisfaisant les conditions suivantes

$$\begin{array}{ll} (i) & H, T_1, T_2, \ldots, T_s \subset V_0 \\ (ii) & d(T_j) \leq C \quad \forall\ 1 \leq j \leq s \\ (iii) & T_j \cap H \neq \emptyset \quad \forall\ 1 \leq j \leq s \\ (iv) & T_i \cap T_j = \emptyset \quad \forall\ 1 \leq i < j \leq s \\ (v) & s \geq 1 \end{array} \tag{2.34}$$

La différence par rapport à un peigne classique est que la demande de chaque dent T_j est inférieure à C et qu'une dent T_j peut être totalement incluse dans le manche. Une telle dent sera appelée un spot. On réservera par la suite le nom de dent à un ensemble qui n'est pas un spot. La figure (2.11) représente le support d'une contrainte de chemin-boîte avec 3 dents T_1, T_2, T_3 et 1 spot T_4. Les chiffres indiqués à côté des sommets sont les demandes.

Nous associons au support d'une telle contrainte le sous-problème de bin packing avec contraintes défini comme suit :

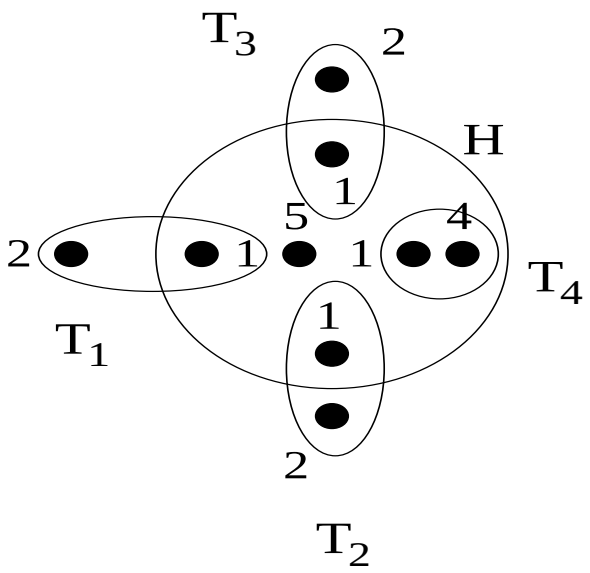


FIG. 2.11 - *Un support de chemin-boîte avec 3 dents et 1 spot.*

Soit I l'ensemble des objets de ce problème. Chaque dent ou spot T_j, $1 \leq j \leq s$ définit un objet de taille $d(T_j)$. Chaque sommet $i \in H \setminus \cup_{j=1}^{s} T_j$ définit un objet de taille d_i dans I. La taille des boîtes est fixée à C. Notons $r(H|T_1, \ldots, T_s)$ le nombre minimum de boîtes nécessaires pour ranger tous les éléments de I avec la contrainte additionnelle qu'une boîte peut contenir au plus deux dents (les spots ne sont pas concernés par cette restriction). Le sous-problème de chemin-boîte associé au support de la figure (2.11) est :

- 3 objets de taille 3 ne pouvant être rangés tous les trois dans la même boîte.

- 2 objets de taille 5

Si $C = 10$, le résultat du sous-problème est $r(H|T_1, \ldots, T_s) = 3$, ce qui correspond par exemple à la solution décrite dans la figure (2.12).

On remarque que la seule différence avec les contraintes de boîte réside dans la contrainte additionnelle, et donc que ces dernières contraintes sont un cas particulier des contraintes de chemin-boîte. Si on fixe toutes les demandes à zéro, le problème de chemin-boîte conserve un intérêt car il permet

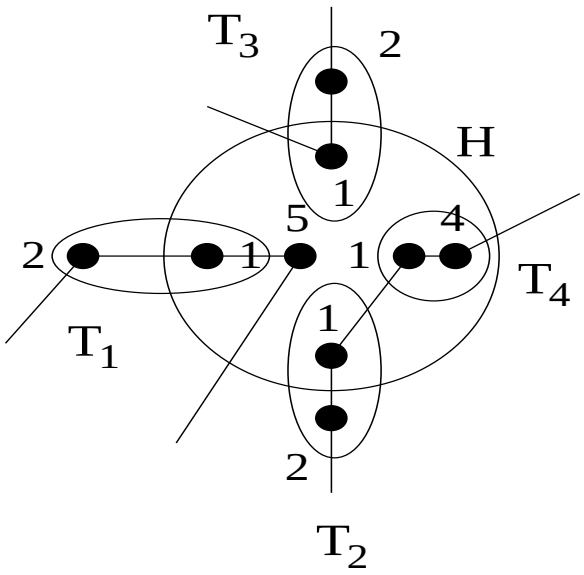


FIG. 2.12 - *Solution partielle correpondant à un sous-problème de chemin-boîte*

d'exprimer les problèmes de parité, et la contrainte obtenue est alors une contrainte de peigne.

Pour lier ce problème de bin packing avec contraintes aux solutions de PTV, nous constatons que la valeur du cocycle de H dans une solution de PTV n'est pas vraiment liée au nombre minimum de véhicules nécessaires pour servir H. Un même véhicule peut entrer et sortir plusieurs fois de H et chaque entrée et sortie doivent être comptabilisées dans le cocycle de H. Le nombre $r(H|T_1, \ldots, T_s)$ exprime alors le nombre minimum d'entrées de véhicules dans H avec la condition que les sommets à l'intérieur d'un ensemble donné T_j sont servis par un seul véhicule et en une fois, i.e. en entrant une seule fois dans la dent. En effet, le nombre d'entrées dans H est exactement le nombre de composantes connexes induites par la solution dans H. Une de ces composantes connexes ne peut intersecter plus de deux dents si la condition précédente sur les dents est repectée. Ceci explique la contrainte annexe du problème de bin packing.

Exprimé comme nous l'avons fait, le calcul de bin packing est local en

ce sens que l'on ne tient pas compte des demandes hors du support de la contrainte de chemin-boîte. Notre calcul donne donc une borne inférieure du nombre minimum parmi toutes les solutions x de PTV de véhicules entrant dans H avec les contraintes annexes déjà exprimées. Ce résultat est formalisé par la contrainte suivante

Proposition 2.3.11 *Si x est une solution de PTV telle que $x(\delta(T_j)) = 2$, $1 \leq j \leq s$, alors*

$$x(\delta(H)) \geq \ 2\ r(H|T_1, \ldots, T_s) \tag{2.35}$$

Preuve :

Soit x^* une solution de PTV minimisant la valeur $x(\delta(H))$. Considérons le graphe induit sur H par la solution x^*. La valeur $x^*(\delta(H))$ est exactement le double du nombre de composantes connexes de ce nouveau graphe. Supposons qu'une composante connexe dans le graphe intersecte strictement plus de deux dents dans un ordre T_{i0}, T_{i1}, T_{i2}. Alors la dent T_{i1} ne peut satisfaire $x(\delta(T_{i1})) = 2$ car elle contient au moins un élément hors de H. Vu l'hypothèse de la proposition, on en conclut que les routes définies par x^* satisfont les contraintes annexes du problème de bin packing $\diamond$

Théorème 2.3.12 *Soit $(H, T_1, \ldots, T_s)$ un support de chemin-boîte, la contrainte de chemin-boîte*

$$x(\delta(H)) \geq \ 2r(H|T_1, \ldots, T_s) \ - \ \sum_{j=1}^{s} \Big[x(\delta(T_j)) - 2\Big] \tag{2.36}$$

est valide pour PTV.

Preuve :

Le problème de bin packing avec contraites annexes a la même propriété que le problème de bin packing sans contraintes annexes. Soit p la solution du problème de bin packing avec contraintes après une coupure de I. En regroupant les deux parties de l'objet coupé dans une nouvelle boîte, on obtient une solution avec $p+1$ boîtes qui satisfait la proposition (2.3.11). On a donc $p+1 \geq r(H|T_1, \ldots, T_s)$. On peut utiliser la même fin de démonstration que pour les contraintes de boîte $\diamond$

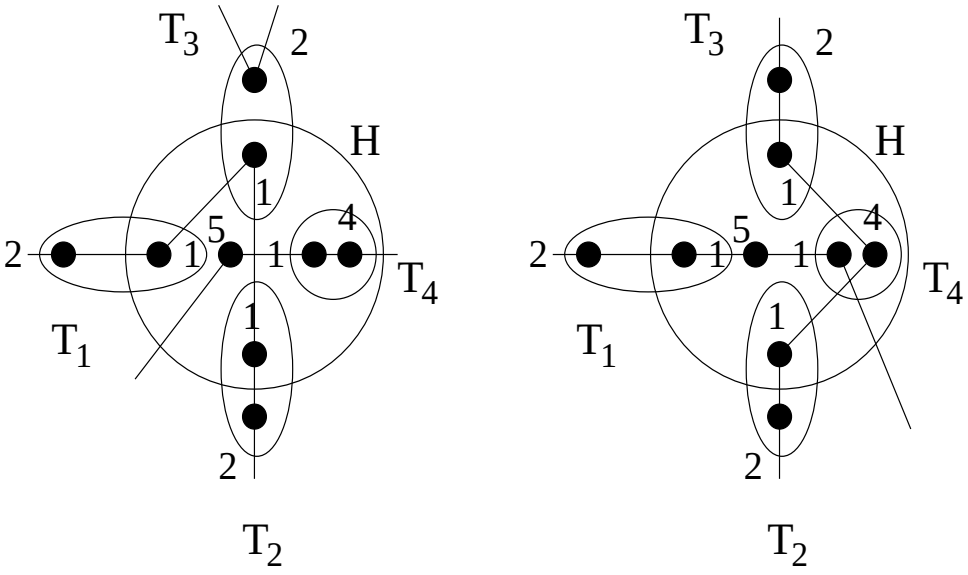


FIG. 2.13 - *Solutions serrées pour un problème de chemin-boîte*

La figure (2.13) représente 2 points serrés de la contrainte de chemin-boîte (points satisfaisant celle-ci avec égalité) utilisant le même support qu'en figure (2.11). L'un des deux a une dent découpée en deux, l'autre un spot.

Le calcul de $r(H|T_1,\ldots,T_s)$ est bien sûr plus difficile que celui de $bp(H|T_1,\ldots,T_s)$. Cependant les mêmes idées que pour le bin packing sans contraintes annexes peuvent être utilisées. Pour de petits ensembles H, l'algorithme exact pour le bin packing de Martello et Toth [MT90a] peut être adapté pour gérer les contraintes additionnelles. Pour des ensembles plus grand, on fera de même avec la procédure de Martello et Toth [MT90b] donnant une borne inférieure de $bp(H|T_1,\ldots,T_s)$. Enfin, deux bornes inférieures triviales de $r(H|T_1,\ldots,T_s)$ sont $\lceil t/2\rceil$ où t est le nombre de dents et $\lceil d(H\cup(\cup_{i=1}^{s}T_i))/C\rceil$. Comme $r(H|T_1,\ldots,T_s)$ n'apparaît que dans le membre de droite de (2.36), on peut remplacer sa valeur par une borne inférieure en obtenant une contrainte encore valide.

Comme $x(\delta(T_j))\geq 2$ dans toute solution de PTV, un ensemble T_j ne sera mis dans la contrainte que si son retrait fait réduire la valeur de la solution du sous-problème de chemin-boîte. Autrement, la contrainte sans l'ensemble T_j dominerait celle avec T_j.

En modifiant les contraintes annexes du sous-problème de bin packing, on peut construire une contrainte de chemin-boîte qui généralise les différents types de peignes présentés dans la littérature. Pochet ([Poc]) a prouvé la validité d'une formulation des contraintes de chemin-boîte intégrant des dents de taille quelconque (peignes de Laporte et al.), des dents petites et larges (peignes introduit par Araque), des dents contenant le dépôt (peignes de Cornuejols et Harche).

2.4 Combinaison de contraintes

Soit une famille de contraintes valides pour PTV, $C_\alpha=\{\alpha^i x\geq\alpha_0^i\ :\ i=1,\ldots,s\}$. Pour simplifier on supposera que les $\alpha^i x$ prennent uniquement des valeurs paires dans PTV. On peut définir un nombre $p(C_\alpha)$ qui représente le nombre minimum de contraintes de C_α non serrées pour une solution de

PTV. Il est clair que la contrainte

$$\sum_{i=1}^{s} \alpha^i x \geq \sum_{i=1}^{s} \alpha_0^i \; + \; 2\; p(C_\alpha) \tag{2.37}$$

est alors valide pour PTV. Le problème est de calculer un tel nombre $p(C_\alpha)$. Nous l'avons déjà fait (voir section «contraintes de capacité généralisées») en combinant des contraintes de sous-tour pour obtenir une contrainte de capacité généralisée. Le calcul de $p(C_\alpha)$ dans ce cas revient au calcul d'un bin packing $r(V_0|T_1,\ldots,T_s)$. Les s ensembles correspondent aux s contraintes de sous-tour.

Plus généralement, plaçons nous à un nœud et une itération quelconque de notre algorithme de «Branchement et Coupe». Si certaines variables, ou certains cocycles, ont été fixés par une contrainte de branchement, cela veut dire que l'on optimise non plus sur PTV mais sur une face de PTV. Nous considérons dans cette section des contraintes valides dans une certaine face du polyèdre PTV, mais pas forcement valides pour PTV. On transformera ces contraintes en contraintes valides pour PTV.

Nous allons présenter deux cas que nous avons rencontrés lors de notre expérimentation numérique. Le premier combine une contrainte d'arbre et une contrainte de sous-tour. Le second considère la transformation d'une contrainte de peigne lorsqu'une ou plusieurs routes sont fixées dans la solution.

2.4.1 Combinaison à base d'arbre

Les contraintes d'arbre peuvent être combinées avec les autres contraintes. Soit (A, v, N_f, N_{nf}) un arbre dont v n'est pas une feuille et soit $S \subset V_0$ un ensemble de sommets contenant v et tel que $d(S) \leq C$. On définit encore $F(A) = (\gamma(N_{nf}) \cup \delta(N_{nf})) \setminus E(A)$.

Proposition 2.4.1 *Si tout A-chemin P satisfait $d(V(P) \cup S) > C$, la contrainte*

$$x(\delta(S)) \; \geq 4 \; - \; 2x(F(A)) \tag{2.38}$$

est valide pour PTV.

Preuve :

Soit x une solution de PTV. Si la contrainte de sous-tour sous-jacente à (2.38) est lâche, (2.38) est satisfaite par la solution x. Sinon, la solution x ne peut contenir de A-chemin et on peut appliquer le lemme (2.3.4) $\diamond$

Ce genre de combinaison est possible pour les autres contraintes d'arbre. Nous présenterons dans le chapitre sur la séparation, une liste de contraintes utiles de ce type. Nous donnons seulement ici une seconde contrainte dans la mesure, où elle généralise la contrainte «one connector partial multistar» définie dans Araque et al. [AHM90]. Il s'agit de la combinaison d'une contrainte de sous-tour et d'une contrainte d'arbre de rang 2. Elle est définie à partir des mêmes données v, S, A que la combinaison précédente.

Proposition 2.4.2 *Si tout v-chemin P de A satisfait $d(V(P) \cup S) > C$, la contrainte*

$$x(\delta(S)) \geq 4 - x(F(A)) \tag{2.39}$$

est valide pour PTV.

Preuve :

Toute solution x de PTV pour laquelle la contrainte de sous-tour est lâche, satisfait (2.39). Les autres solutions ne peuvent contenir de v-chemin de A et on peut utiliser le lemme (2.3.5) pour conclure $\diamond$

2.4.2 Combinaison à base de peigne

Considérons une solution fractionnaire x, où un certain nombre de routes sont complètement fixées. Notons $W \subset V_0$ l'ensemble des sommets qui n'appartiennent pas à ces routes. On a alors $x(W : \bar{W}) = 0$. Dans $W \cup \{0\}$, on peut définir un support de contrainte de peigne $(H, T_1, \ldots, T_s)$ tel que la dent T_1 contienne le dépôt. Si ce support satisfait les conditions de la contrainte de peigne (2.16), on a

$$x(\delta(H)) \geq (s+1) - \Big[x(\delta(T_1)) - 2R(V \setminus T_1)\Big] - \sum_{j=2}^{s} \Big[x(\delta(T_j)) - 2\Big]$$

Cette contrainte est globale dans la mesure où le terme $R(V \setminus T_1)$ est calculé en tenant compte de toutes les demandes. Pour les solutions telles que $x(W :$

$\bar{W}) = 0$, on est tenté de écrire la contrainte en fonction de $R(W \setminus T_1)$. Nous présentons maintenant une condition permettant cette écriture.

Proposition 2.4.3 *Si* $R(W \setminus T_1) + R(\bar{W}) = R(V \setminus T_1) + 1$, *la contrainte suivante*

$$x(\delta(H)) \geq (s+1) - 2(x(W:\bar{W}) - 1) \qquad (2.40)$$
$$- \left[x(\delta(T_1)) - 2R(V \setminus T_1)\right] - \sum_{j=2}^{s} \left[x(\delta(T_j)) - 2\right]$$

est valide pour PTV.

L'hypothèse exprime que le nombre de véhicules nécessaires pour servir $V \setminus T_1$ augmente d'une unité quand T_1, $W \setminus T_1$ et $\bar{W}$ sont servis de manière indépendante. La contrainte se lit alors : si $x(W:\bar{W}) = 0$, le second membre de l'inégalité de peigne initiale est augmenté de deux.

Preuve :

Soit x une solution du PTV. Le cas $x(W:\bar{W}) \geq 1$ est trivial, car le second membre est alors plus petit que dans la première contrainte. Si $x(W:\bar{W}) = 0$, la restriction x^W de x à $W \cup \{0\}$ satisfait la contrainte de peigne

$$x^W(\delta(H)) \geq (s+1) - \left[x^W(\delta(T_1))\right.$$
$$\left. -2R(W \setminus T_1)\right] - \sum_{j=2}^{s} \left[x^W(\delta(T_j)) - 2\right] \qquad (2.41)$$

dans le polyèdre PTV^W, défini à partir des vecteurs de PTV, par leurs composantes dans $\gamma(W \cup \{0\})$. Comme $x(\delta(T_1)) = x^W(\delta(T_1)) + 2R(\bar{W})$ et $x^W(\delta(T_j)) = x(\delta(T_j))$, $i = 2, \ldots, s$, on en déduit que x satisfait (2.40) si la condition $R(W \setminus T_1) + R(\bar{W}) = R(V \setminus T_1) + 1$ est respectée $\diamond$

2.5 Polyèdres en relation avec PTV

Prouver qu'une contrainte valide de PTV est une facette de PTV est un problème difficile, car le polyèdre PTV n'est pas de pleine dimension. Par

ailleurs, dans le cas général, on ne peut calculer la dimension de PTV. Chaque solution de PTV est en effet en même temps la solution d'un problème de bin packing. Il est donc impossible d'utiliser les méthodes usuelles de preuve de facette, en particulier d'énumérer des solutions génériques serrées. Pour contourner ce problème, il est usuel de travailler dans un polyèdre P tel que $PTV \subset P$ ayant de meilleures propriétés. Cornuejols et Harche [CH93] introduisent ainsi la relaxation graphique du PTV (que nous appellerons GPTV). Araque et al. [AHM90] ne fixent pas le nombre de véhicules (on notera $P\tilde{T}V$ le polyèdre correspondant). Grâce à un changement de variable, ils éliminent aussi les arêtes incidentes au dépôt ce qui permet de travailler sur un polyèdre de pleine dimension. Nous rappelons maintenant quelques uns de ces résultats car nous allons aussi utiliser ces polyèdres plutôt que le PTV pour préciser les propriétés des contraintes valides pour PTV.

2.5.1 Les polyèdres $P\tilde{T}V$ et PPC

On considère maintenant le problème, où le nombre de véhicules n'est pas fixé. On peut donc éventuellement avoir autant de véhicules que de clients. En terme de programmation entière, on considérera le programme suivant identique à celui associé au PTV moins la contrainte (2.2).

$$
\begin{aligned}
\min Z(P\tilde{T}V) = \sum_{e \in E} l_e \; x_e & \\
x(\delta(\{i\})) = 2 & \qquad \forall \; i \in V_0 \\
x(\delta(S)) \geq \; 2 \left\lceil \frac{d(S)}{C} \right\rceil & \qquad \forall \; S \subseteq V_0, \; S \neq \emptyset \\
0 \leq x_e \leq 1 & \qquad \forall \; e \in \gamma(V_0) \\
0 \leq x_e \leq 2 & \qquad \forall \; e \in \delta(V_0) \\
x_e \text{ entier} & \qquad \forall \; e \in E
\end{aligned}
$$

On note $P\tilde{T}V$ le polyèdre défini par l'enveloppe convexe des solutions réalisables de ce programme. Soit F l'ensemble des arêtes qui n'appartiennent à aucune solution, i.e. les arêtes ij telles que $d(i) + d(j) > C$, on a alors

Proposition 2.5.1

$$
dim(P\tilde{T}V) = m \; - \; n \; - |F| \tag{2.42}
$$

Preuve :

$m - |F|$ est le nombre de variables pouvant apparaître dans une solution. De plus, les n contraintes de degré limitent la dimension de $P\tilde{T}V$ à $m - n - |F|$. Pour chaque arête $e = (u, v) \in E \setminus F \setminus \delta(\{0\})$, on peut construire la solution $T_e = \{(0, u, v, 0), (0, t, 0) \ : \ \forall t \notin \{u, v\}\}$. En ajoutant la solution $T^* = \{(0, t, 0) \ : \ \forall t \in V_0\}$, on obtient un ensemble de $n - |F| + 1$ solutions indépendantes de $P\tilde{T}V$ et donc la dimension du polyèdre $P\tilde{T}V$ $\diamond$

On supposera dans toute la suite du document et pour faciliter les démonstrations que $|F| = 0$.

En terme d'optimisation, travailler sur PTV et $P\tilde{T}V$ peut être très différent. Le problème de bin packing sous-jacent à PTV peut engendrer une solution optimale avec des arêtes très longues qui ne seront pas dans la solution optimale de $P\tilde{T}V$. Même si on choisit comme valeur de k le plus petit nombre de véhicules pour lequel il existe une solution, on pourra avoir une solution strictement plus petite pour $P\tilde{T}V$ que pour PTV. Dans notre expérimentation numérique, nous avons rencontré ce cas une seule fois en traitant une centaine de problèmes.

Il est souvent délicat de travailler sur un polyèdre qui n'est pas de pleine dimension. Queyranne et Wang [QW93] décrivent cependant une procédure de normalisation des contraintes qui simplifie les démonstrations dans ce cas. Cette procédure est applicable à PTV. Sans entrer dans les détails, cela revient à soustraire à une contrainte donnée les équations de degré pour les clients, pondérées par les coefficients adéquats, de manière à éliminer tous les coefficients des arêtes adjacentes au dépôt. Une autre façon équivalente de faire est donnée par Araque [Ara90a]. Il transforme (avec le même objectif que Queyranne et Wang) le problème du PTV en un problème de partition de chemins. Pour toute arête $ij \in E$, $i \in V$, $j \in V$, on définit $s_{ij} = l_{i0} + l_{j0} - l_{ij}$. Ces coûts sont égaux aux «savings» définis dans l'heuristique de Clarke et Wright [CW64]. Le PTV est alors équivalent au problème de partition de chemin PPC suivant

$$\max Z(PPC) = \sum_{e \in E \setminus \delta(\{0\})} s_e \ x_e$$

$$x(\delta(\{i\})) \leq 2 \qquad \text{pour tout } i \in V_0$$

$$
\begin{aligned}
x(\gamma(S)) \leq\ |S|\ - \left\lceil \frac{d(S)}{C} \right\rceil \qquad & \text{pour tout } S \subseteq V_0,\ S \neq \emptyset \\
0 \leq x_e \leq 1 \qquad & \text{pour tout } e \in \gamma(V_0) \\
x_e \text{ entier} \qquad & \text{pour tout } e \in E \setminus \delta(\{0\})
\end{aligned}
$$

Le polyèdre PPC, qui correspond à l'enveloppe convexe des solutions de ce programme est de pleine dimension. Les résultats de Queyranne et Wang permettent d'associer à toute contrainte valide de $P\tilde{T}V$ une contrainte normalisée telle que tous les coefficients des arêtes incidentes au dépôt soient nuls et telle que cette contrainte soit valide pour PPC.

2.5.2 Le polyèdre GPTV

Dans la relaxation graphique du PTV, un véhicule peut passer par un sommet sans prendre la commande du client correspondant (mais aussi peut passer plusieurs fois par le même sommet). En terme de programmation entière cela conduit à relaxer les contraintes de degré ainsi que les bornes supérieures sur la valeur des variables. Notons que si le graphe est complet et si les coefficients $l_e,\ e \in E$, dans (2.1) satisfont l'inégalité triangulaire, l'optimisation dans GPTV est équivalente à l'optimisation dans PTV.

La relaxation graphique a d'abord été introduite dans le cas du PVC par Cornuejols et al. [CFN85] et utilisée depuis comme un outil pour étudier la structure du polyèdre du PVC (cf Naddef et Rinaldi [NR91]). Cornuejols et Harche ont par la suite étudié le cas du PTV [CH93]. Nous présentons maintenant ce polyèdre en suivant leurs notations.

Un **tour** $T(F,U)$ est défini comme une famille non vide d'arêtes F de E (une arête donnée peut apparaître plusieurs fois dans un tour) et un sous-ensemble de sommets $U \subseteq V_0(F)$ tels que $G(F)$ est un cycle contenant le dépôt (i.e. $0 \in V(F)$), $G(F)$ est connexe, le degré de chaque sommet dans $G(F)$ est pair) et $d(U) \leq C$. Un tour représente le voyage associé à un véhicule et son ensemble U représente les clients affectés au véhicule. Rappelons que ce tour peut passer par des clients qui ne lui sont pas affectés.

Un **k-tour** est un ensemble de routes $\{T_1(F_1,U_1),\ldots,T_k(F_k,U_k)\}$ (où k n'est pas fixé mais $k \geq r(V_0)$ pour que la solution soit réalisable) telle

que chaque sommet $i \in V_0$ appartienne à exactement un des ensembles $U_j, 1 \leq j \leq k$. Chaque k-tour définit une solution réalisable de GPTV. La longueur d'un k-tour est la somme pondérée (i.e. en tenant compte des arêtes multiples) des longueurs des arêtes le définissant. GPTV est le problème de trouver un k-tour de longueur minimum. Notons qu'un k-tour est une solution du PTV si $V(F_i) = U_i \cup \{0\}$ et $|F_i| = |U_i| - 1$, pour tout $i = 1, \ldots, k$.

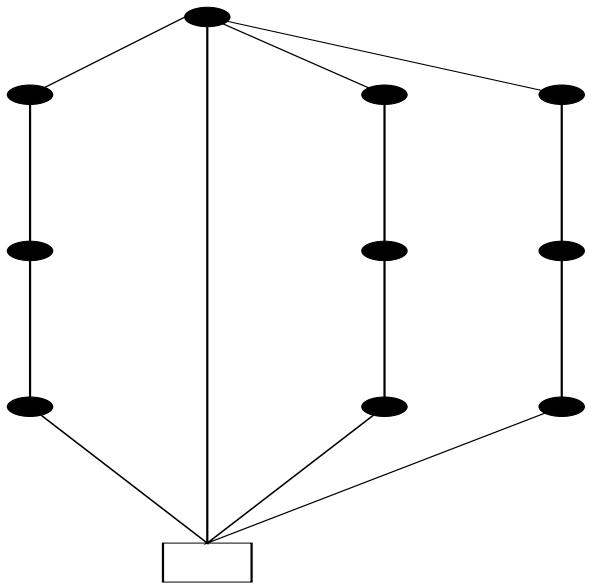


FIG. 2.14 - *Un exemple de k-tour qui n'est pas une solution de PTV, C=10, demandes unitaires*

La figure (2.14) donne un exemple de 2-tour qui n'est pas une solution de PTV (i.e. pas une 2-route). La capacité est 10 et les commandes sont unitaires. On peut extraire 3 solutions differentes du PTV de ce 2-tour.

Si nous définissons la variable x_e comme le nombre de fois où $e \in E$ est utilisée dans une solution de GPTV, on peut associer un vecteur $x \in R^{n(n+1)/2}$ à toute solution de GPTV et définir le polyèdre GPTV comme l'enveloppe convexe de toutes les solutions de GPTV, $X^{GPTV} \subset R^{n(n+1)/2}$.

Proposition 2.5.2 *Si le problème PTV a au moins une solution alors*

$$dim(GPTV) = m \tag{2.43}$$

Preuve :

PTV est une face de GPTV. Soit x une solution de PTV et pour $e \in E$ on définit $y^e = (y^e_i)$ le m-vecteur tel que $y^e_e = 1$ et toutes les autres composantes sont nulles. Alors les m solutions $z^e = x + 2y^e$ forment un ensemble de vecteurs affinement indépendants $\diamond$

2.6 Résultats de facettes

Soit $\alpha x \geq \alpha_0$ une contrainte valide d'un polyhèdre P de dimension $dim(P)$. On dit que $\alpha x \geq \alpha_0$ est une facette de P si le sous-espace $\{x \in P : \alpha x = \alpha_0\}$ est de dimension $dim(P) - 1$. On trouve dans la littérature des conditions pour que les contraintes présentées jusqu'ici soient des facettes de PTV, $GPTV$, ou $P\tilde{T}V$. Ces conditions sont en général longues à énumérer et difficiles à vérifier lorsqu'on veut utiliser une contrainte dans le cadre d'un algorithme de coupe. Nous ne présentons donc aucun détail. Le lecteur se reportera aux articles cités.

Cornuejols et Harche [CH93] donnent des conditions suffisantes (resp. nécessaires et suffisantes) pour que la contrainte de capacité (2.11) soit une facette de PTV (resp. GPTV). Araque et al. [AHM90] donnent aussi des conditions suffisantes pour que les contraintes de capacité soient facettes de PPC. Ils étudient le cas où toutes les demandes sont unitaires et les contraintes (2.4), (2.8) et (2.11) sont alors identiques.

Cornuejols et Harche donnent des conditions suffisantes pour que la contrainte de peigne soit une facette du PTV et GPTV. Ces conditions sont en fait données pour les contraintes de chemin qui sont une généralisation des contraintes de peigne.

Harche and Rinaldi [HR91] donne des conditions nécessaires et suffisantes pour que les contraintes (2.20) soient des facettes de GPTV. Araque et al. donnent des conditions suffisantes pour que les contraintes d'étoile soient des facettes de PPC.

Nous présentons maintenant le même genre de conditions, pour les nouvelles contraintes décrites dans ce travail.

Comme nous l'avons déjà mentionné, certaines contraintes décrites sont locales dans le sens où elles n'utilisent que les informations sur les sommets de leur support. Parce que le nombre de véhicules k est fixé, une solution partielle x restreinte au support d'une contrainte peut être serrée pour ce dernier, sans pour autant donner une solution de PTV réalisable. Plutôt que de donner une liste de conditions pour qu'une contrainte locale induise une facette, nous étudierons seulement les cas où la contrainte est une facette locale. Nous formalisons ce concept dans la définition suivante.

Définition Une contrainte induit une facette locale de PTV si et seulement si elle permet de définir une facette du problème relaxé $P\tilde{T}V$.

Soit $F \subset E$ un sous-ensemble d'arêtes, on définit PTV(F) l'enveloppe convexe des solutions de PTV qui sont dans F. On définit de la même manière $P\tilde{T}V(F)$ et on continuera d'écrire PTV=PTV(E), $P\tilde{T}V = P\tilde{T}V(E)$.

Soit

$$\sum_{f \in F} \alpha_f x_f \leq \alpha_0 \tag{2.44}$$

une contrainte valide de PTV dont le support est un ensemble d'arêtes $F \subset E$.

Proposition 2.6.1 *Si (2.44) est une facette de $P\tilde{T}V(F \cup \delta(\{0\}))$ alors il existe des coefficients $\beta_e, e \in E \setminus F$ tels que*

$$\sum_{f \in F} \alpha_f x_f \; + \sum_{f \in E \setminus F} \beta_f x_f \leq \alpha_0 \tag{2.45}$$

soit une facette de $P\tilde{T}V$.

Preuve :

Nous allons appliquer à notre problème le principe décrit par Padberg [Pad75] pour obtenir des facettes d'un polyèdre à partir de facettes d'un polyèdre de dimension inférieure. Ce principe porte le nom de «sequential lifting ». Supposons que (2.44) soit une facette de $P\tilde{T}V(F \cup \delta(\{0\}))$ et soit $e \in E \setminus (F \cup \delta(\{0\}))$. Considérons le problème suivant :

$$\min \; Z(e) = \alpha_0 \; - \sum_{f \in F} \alpha_f x_f \tag{2.46}$$

t.q.

$$x \in P\tilde{T}V(F \cup \delta(\{0\}) \cup \{e\}) \tag{2.47}$$

$$x_e = 1 \tag{2.48}$$

Notons qu'il existe une solution réalisable de ce problème qui ne contient que des routes avec un client, plus une route avec deux clients contenant l'arête e. La validité de (2.44) assure par ailleurs que la solution optimale de ce problème $Z(e)$ est positive. On pose $\beta_e = Z(e)$, il est clair que la contrainte

$$\sum_{f \in F} \alpha_f x_f \; + \beta_e x_e \leq \alpha_0 \tag{2.49}$$

est une facette de $P\tilde{T}V(F \cup \delta(\{0\}) \cup \{e\})$. Comme on peut réitérer le calcul sur $F \cup \{e\}$ pour une arête de E dont le coefficient n'a pas encore été calculé. On peut obtenir une contrainte

$$\sum_{f \in F} \alpha_f x_f \; + \sum_{f \in E \setminus F} \beta_f x_f \leq \alpha_0 \tag{2.50}$$

qui est une facette de $P\tilde{T}V$. Notons que cette contrainte dépend de l'ordre dans lequel on considère les variables de $E \setminus (F \cup \delta(\{0\}))$ $\diamond$

2.6.1 Contraintes de chemin-boîte

Nous avons trouvé des instances de PTV (i.e. des distributions de demande) et des supports de contrainte de chemin-boîte qui constituent des facettes du polytope PTV correspondant. Dans ces instances de petite taille, nous avons énuméré toutes les solutions serrées. Comme toute solution serrée pour une telle contrainte est solution d'un problème de bin packing, il semble plus difficile de donner des conditions générales de facettes pour les contraintes de chemin-boîte. Ce travail est en cours.

2.6.2 Contraintes d'arbre

Nous donnons maintenant des conditions suffisantes pour qu'une contrainte d'arbre de rang 1 soit une facette locale de PTV. Soit un arbre (A, v, N_f, N_{nf}) tel que v n'est pas une feuille et tout A-chemin a une demande supérieure à la capacité C. Nous avons déjà montré que la contrainte suivante est valide.

$$x(E(A)) \; + \; x(\gamma(N_{nf})) \; \leq \; 2 \; |N_{nf}| \; - \; 1 \tag{2.51}$$

Théorème 2.6.2 *Si de plus les deux conditions suivantes sont satisfaites,*

(i) pour tout A-chemin $P = (f, \ldots, v, \ldots, g)$*, on a* $d(V(P) \setminus \{f\}) \leq C$ *et* $d(V(P) \setminus \{g\}) \leq C$

(ii) pour tout $s \in N_{nf} \setminus \{v\}$*, il existe un chemin* P *dans* A *contenant* s *et dont les extrémités sont des feuilles de* A *et satisfaisant* $d(V(P)) \leq C$

alors la contrainte (2.51) est une facette locale de PTV.

La figure (2.15) montre un arbre satisfaisant pour $C = 10$ la condition de validité de (2.51) et les deux conditions du thérème (2.6.2).

Preuve :

Considérons d'abord le polytope $P\tilde{T}V(E(A) \cup \delta(\{0\}))$, c'est-à-dire l'enveloppe convexe des solutions du problème $P\tilde{T}V$ dont le support est inclus dans l'ensemble d'arêtes $E(A) \cup \delta(\{0\})$. Nous allons d'abord montrer que (2.51) est une facette de ce polytope, puis nous utiliserons la proposition précédente pour conclure.

Soit Q l'ensemble des solutions de $P\tilde{T}V(E(A) \cup \delta(\{0\}))$ qui satisfont (2.51) avec égalité. On définit plus facilement Q par l'autre forme (cf section 2.3.2) de la contrainte (2.51), c'est-à-dire

$$x(F(A)) \geq 1 \tag{2.52}$$

où $F(A) = (\gamma(N_{nf}) \cup \delta(N_{nf})) \setminus E(A)$. En effet, l'intersection de $F(A)$ et de $E(A) \cup \delta(\{0\})$ se réduit à l'ensemble d'arêtes $(N_{nf} : \{0\})$. On a donc $Q = \{x \in P\tilde{T}V(E(A) \cup \delta(\{0\})) \ : \ x(N_{nf} : \{0\}) = 1\}$ ce qui permet de vérifier facilement qu'une solution est dans Q.

On fixera par ailleurs comme écriture normalisée des contraintes de $P\tilde{T}V$ (cf Queyranne et Wang [QW93]), celle où tous les coefficients des arêtes incidentes au dépôt sont nuls.

Soit

$$\alpha \ x \leq \alpha_0 \tag{2.53}$$

une contrainte normalisée telle que $x \in Q$ implique $\alpha \ x = \alpha_0$. On a en particulier $\alpha_{0:i} = 0$, pour tout $i \in V_0$. Nous allons montrer que (2.53) est un

multiple de la contrainte (2.51). Pour cela, on va calculer itérativement tous les coefficients du vecteur α en comparant des solutions génériques de Q.

Soient $f \in N_f$ et $h \in N_{nf}$ deux sommets ayant v pour seul ancêtre commun dans A. Nous appellerons solution générique z_{fh} associée à (f,h) la solution construite de la manière suivante. La première route $(0, f, \ldots, v, \ldots, h, 0)$ existe grâce à la condition (i) appliquée à f et une feuille descendante de h. Les autres routes sont construites de manière itérative. A un moment donné de la construction, on choisit le sommet t non utilisé le plus proche de v (en terme de profondeur dans A). Aucun de ses descendants n'est encore inclus dans la solution partielle. La condition (ii) permet de construire une route contenant t sans arêtes dans $(N_{nf} : \{0\})$. Enfin, pour les sommets $t \in V \setminus V(A)$, on ajoute une route $(0, t, 0)$. Il est clair que cette solution est dans Q car seule la première des routes construites contient une arête de $N_{nf} : \{0\}$ et exactement une. La figure (2.15) montre un arbre de racine v et deux solutions génériques z_{fh} et $z_{fh'}$, où h' est le père de h.

Soient f et g deux feuilles de A ayant v comme seul ancêtre commun dans A. Soient f' et g' leur père respectif. Les deux solutions génériques $z_{fg'}$ et $z_{gf'}$ satisfont $\alpha z_{fg'} = \alpha z_{gf'}$. Ceci est équivalent à $\alpha_{f:f'} = \alpha_{g:g'}$ car ces deux solutions ne diffèrent que par ces deux arêtes et des arêtes incidentes au dépôt. Comme ceci est vrai pour toute paire de feuilles f, g ayant v comme seul ancêtre commun, c'est aussi vrai par transitivité pour toute paire de feuilles. On notera $\tilde{\alpha}$ cette valeur commune.

Nous allons maintenant calculer par construction la valeur des autres coefficients. Soit h un sommet de A différent de v. Soit h' son père dans A. On définit A^h le sous-graphe de G induit par les descendants de h dans A. Pour $x \in Q$, on définit enfin $M^h(x) = \sum_{e \in A^h} \alpha_e \, x_e$.

Supposons que h n'est pas une feuille et que toute arête $ij \in A^h$ satisfait :

$$\begin{aligned} \alpha_{ij} &= 0 \qquad si\ i = 0\ ou\ j = 0 \\ \alpha_{ij} &= 2\tilde{\alpha} \quad si\ i \in N_{nf}\ et\ j \in N_{nf} \\ \alpha_{ij} &= \tilde{\alpha} \qquad\qquad sinon. \end{aligned}$$

Nous allons montrer que $\alpha_{hh'} = 2\tilde{\alpha}$.

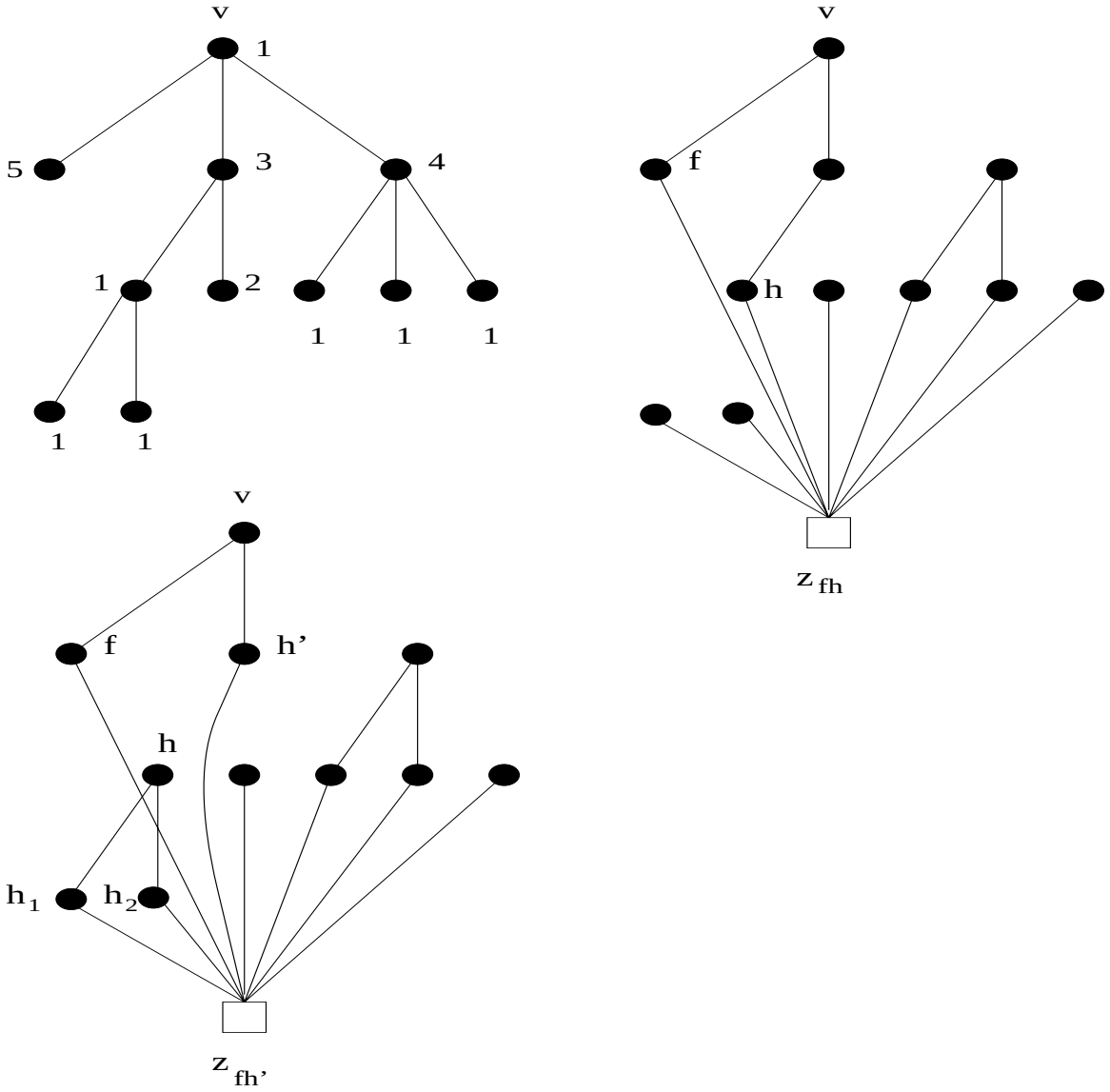


FIG. 2.15 - *Arbre et solutions génériques*

Soit $f \in N_f$ un sommet tel que le seul ancêtre en commun de f et h soit v. Soient z_{fh} et $z_{fh'}$ les solutions génériques correspondantes. Notons $h_i,\ i = 1, \ldots, s$ les fils de h. L'équation $\alpha z_{fh} = \alpha z_{fh'}$ est équivalente à $\alpha_{hh'} + \sum_{i=1}^{s} M^{h_i}(z_{fh}) = M^h(z_{fh'})$. On peut facilement calculer $M^h(z_{fh'})$ car on connait grâce à notre hypothèse tous les coefficients des arêtes de A^h. Soit p le nombre de routes dans $z_{fh'}$ qui intersectent $N_{nf} \cap V(A^h)$. Ces routes définissent $2p$ arêtes incidentes à des feuilles car toutes ces routes contiennent deux feuilles. Notons q le nombre d'arêtes de $\gamma(N_{nf} \cap V(A^h))$ dans ces routes. En sommant les degrés des sommets de $N_{nf} \cap V(A^h)$ on obtient $2|N_{nf} \cap V(A^h)| = 2p + 2q$ donc $q = |N_{nf} \cap V(A^h)| - p$. On conclut que $M^h(z_{fh'}) = 2p\ \tilde{\alpha} + (|N_{nf} \cap V(A^h)| - p)\ 2\tilde{\alpha} = 2\tilde{\alpha}|N_{nf} \cap V(A^h)|$. Le même résultat peut être obtenu pour le calcul de $M^{h_i}(z_{fh}),\ i = 1, \ldots, s$ sauf si h_i est une feuille où on a $M^{h_i}(z_{fh}) = 0$. Ceci nous permet de calculer $\alpha_{hh'} = M^h(z_{fh'}) - \sum_{i=1}^{s} M^{h_i}(z_{fh}) = 2\tilde{\alpha}(|N_{nf} \cap V(A^h)| - \sum_{i=1}^{s} |N_{nf} \cap V(A^{h_i})|) = 2\tilde{\alpha}$.

On peut donc effectuer ce calcul pour les sommets dont tous les fils sont des feuilles puis le réiterer autant de fois que nécessaire sur les sommets h tel que tous les coefficients des arêtes de A^h sont connus. Ainsi, on calcule tous les coefficients du membre de gauche de $\alpha x \leq \alpha_0$.

La façon dont a été construite z_{fg} permet aussi de calculer α_0. Soit nc le nombre de routes intersectant N_{nf} dans z_{fg}. Le nombre d'arêtes dans $E(A) \cap \delta(N_{nf})$ est $2\ nc - 1$. On en déduit en sommant les degrés des sommets de N_{nf} le nombre d'arêtes de z_{fg} dans $E(A) \cap \gamma(N_{nf})$. Ce nombre est exactement $|N_{nf}| - nc$. On a donc $\alpha_0 = \alpha z_{fg} = 2\tilde{\alpha}\ (|N_{nf}| - nc) + \tilde{\alpha}\ (2\ nc\ -\ 1) = \tilde{\alpha}(2\ |N_{nf}|\ -\ 1)$. On peut donc conclure que (2.53) et (2.51) sont les mêmes à un scalaire près. (2.51) est donc une facette de $P\tilde{T}V(E(A) \cup \delta(\{0\}))$. La procédure générale de lifting décrite plus haut nous assure donc d'obtenir une facette de $P\tilde{T}V$ à partir de (2.51) et montre donc que (2.51) est une facette locale de PTV $\diamond$

En conclusion de ces descriptions de contraintes, nous présentons un schéma (figure 2.16) de leurs interactions.

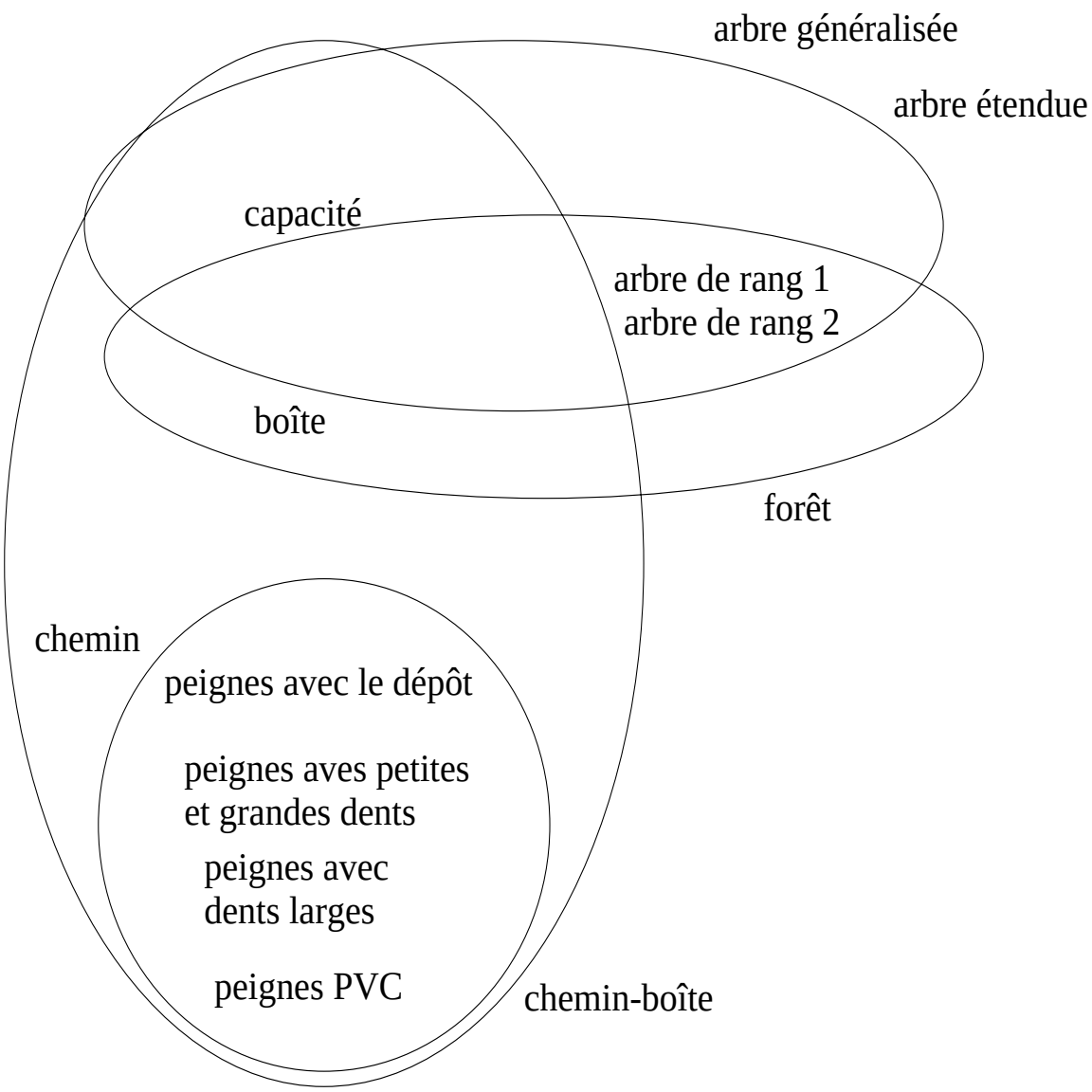


FIG. 2.16 - *Résumé : Les contraintes valides du PTV*

2.7 Autres formulations d'intérêt pour l'approche polyédrale

Nous présentons maintenant une autre formulation du PTV, la formulation à trois index. Cette formulation plus complexe a été introduite par Fisher et Jaikumar [FJ81]. Elle utilise le numéro du véhicule comme troisième index. Soit x_e^h le nombre de fois où l'arête $e \in E$ est utilisée par le véhicule h dans la solution (k-route). Soit y_i^h une variable prenant la valeur 1, si le véhicule h visite le client i, et la valeur 0 dans le cas contraire. Le PTV peut alors se formuler ainsi en utilisant ces variables

$$\min Z(PTV2) = \sum_{h=1}^{k} \sum_{i=0}^{n} \sum_{j=0}^{n} l_{ij} x_{ijh} \tag{2.54}$$

t.q.

$$\sum_{h=1}^{k} y_i^h = \begin{cases} k & \text{si } i = 0 \\ 1 & \forall i = 1, \ldots, n \end{cases} \tag{2.55}$$

$$\sum_{i=1}^{n} d_i y_i^h \leq C \qquad \forall h = 1, \ldots, k \tag{2.56}$$

$$\sum_{i=0}^{n} x_{ijh} = y_{jh} \qquad \forall j = 0, \ldots, n \qquad \forall h = 1, \ldots, k \tag{2.57}$$

$$\sum_{j=0}^{n} x_{ijh} = y_{ih} \qquad \forall i = 0, \ldots, n \qquad \forall h = 1, \ldots, k \tag{2.58}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijh} \leq |S| - 1 \qquad \forall S \subseteq N \setminus \{0\},\ S \neq \emptyset \qquad \forall h = 1, \ldots, k \tag{2.59}$$

$$y_i^h \in \{0,1\} \text{ et } x_{ijh} \in \{0,1\} \qquad \forall i = 0, \ldots, n, \quad j = 0, \ldots, n \qquad \forall h = 0, \ldots, k \tag{2.60}$$

Cette formulation est utilisée par Kubo [Kub92] avec la contrainte (2.59) remplacée par

$$\sum_{i \in S} \sum_{j \in S} x_{ijh} \leq \sum_{i \in S} y_i^h - 1 \qquad \forall S \subseteq N \setminus \{0\},\ S \neq \emptyset \text{ et } h = 1, \ldots, k \tag{2.61}$$

L'avantage est que cette formulation fait apparaître le problème de sac à dos sous-jacent au PTV. En revanche, le nombre de variables dans cette formulation est très grand (à peu près k fois celui de la formulation précédente). D'autre part, on peut à partir d'une solution de PTV obtenir d'autres solutions par permutation des indices des véhicules. Ceci risque d'engendrer pour notre méthode de nombreux problèmes.

Une formulation très similaire est utilisée par Balas [Bal89], [Bal93] pour le problème du voyageur de commerce avec récompense. Ce problème consiste à chercher dans un graphe un cycle (pas forcement hamiltonien) de longueur minimale. Des commandes sont aussi définies sur les sommets du graphe ainsi qu'une contrainte imposant que la somme des commandes correspondant à un cycle réalisable soit supérieure à une valeur fixée C_{min}. Ce problème se formalise ainsi.

$$\min Z(PVCR) = \sum_{i=0}^{n}\sum_{j=0}^{n} l_{ij}x_{ij} \tag{2.62}$$

t.q.

$$\sum_{i=1}^{n} d_i y_i \geq C_{min} \tag{2.63}$$

$$\sum_{i=0}^{n} x_{ij} = y_j \qquad \forall j = 0,\ldots,n \tag{2.64}$$

$$\sum_{j=0}^{n} x_{ij} = y_i \qquad \forall i = 0,\ldots,n \tag{2.65}$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1 \qquad \forall S \subseteq N \setminus \{0\},\ S \neq \emptyset \tag{2.66}$$

$$y_i \in \{0,1\} \qquad x_{ij} \in \{0,1\} \qquad \forall i = 1,\ldots,n \qquad \forall j = 1,\ldots,n \tag{2.67}$$

$$x_{0j} \in \{0,1,2\} \qquad \forall j = 1,\ldots,n \tag{2.68}$$

$$x_{0j} \in \{0,1,2\} \qquad \forall j = 1,\ldots,n \tag{2.69}$$

Remarquons, que pour une valeur donnée des variables k, C, $\{d_i : i = 1,\ldots,n\}$, on peut construire un PVCR tel que toute route faisant partie d'une k-route réalisable de PTV soit une solution réalisable de PVCR. Il suffit de fixer $Cmin = C - (k * C - \sum_{i=1}^{n} d_i)$. Réciproquement, une solution

du PVCR sera aussi une route d'une k-route si elle respecte des conditions (de type bin-packing) simple. Pour les deux problèmes, toutes les contraintes de type sac à dos sont valides en particulier les théorèmes de lifting de facettes (voir encore Kubo et Balas pour chacun des deux problèmes).

Nous présentons maintenant une formulation prenant en compte à la fois le problème de sac à dos et de bin packing mais contenant moins de variables. Associons deux variables x_{ij} et y_{ij} à chaque arête de $ij \in E$ qui représentent, pour x_{ij}, le nombre de fois, où l'arête ij est utilisée dans la solution (k-route), et pour y_{ij}, le fait que les i et j sont ou non servis par le même véhicule (on fixera par ailleurs $y_{ii} = 1$ et $y_{0i} = 0$). Une k-route minimale est alors solution du problème suivant.

$$\min Z(PTV3) = \sum_{e \in E} l_e \; x_e \tag{2.70}$$

t.q.

$$\begin{aligned}
x(\delta(\{0\})) &= 2k & & \\
x(\delta(\{i\})) &= 2 & \forall\, i \in V_0 & \\
x(\delta(S)) &\geq 2 & \forall\, S \subseteq V_0,\ S \neq \emptyset & (2.71)\\
x_e &\leq y_e & \forall\, e \in E \setminus \delta(\{0\}) & (2.72)\\
y_{ij} \;+\; y_{jk} &\leq \;1\; + \; y_{ij} & \forall\, i,j,k \in V_0 & (2.73)\\
\sum_{i=1}^{n} y_{ij}\; d_i &\leq C & \forall\, j \in V_0 & (2.74)\\
y_e &\in \{0,1\} & \forall\, e \in \gamma(V_0) & \\
x_e &\in \{0,1\} & \forall\, e \in \gamma(V_0) & \\
x_e &\in \{0,1,2\} & \forall\, e \in \delta(\{0\}) &
\end{aligned}$$

Les contraintes (2.71) assurent la connexité et les contraintes (2.74) imposent les contraintes de capacité. Le lien entre les contraintes x_e et y_e est fait par les deux autres contraintes (2.72) et (2.73).

Nous avons expérimenté les formulations PTV2 et PTV3 dans un algorithme de coupe, c'est-à-dire introduit des variables supplémentaires à la fin de l'algorithme appliqué à la formulation à deux index. Nous avons alors essayé, sans succès, de trouver visuellement des contraintes violées de type sac à dos. Dans le cas du problème PTV3, nous avons en revanche trouvé

des violations du type suivant. Posons $C_{min} = C - (k * C - \sum_{i=1}^{n} d_i)$ et pour $i = 1, \ldots, n$, $z_i = \sum_{j=1}^{n} d_i y_{ij}$. Soient j et h deux sommets de V_0, la contrainte suivante

$$z_j + z_h \geq C + C_{min} - y_{jh}(C - C_{min}) \tag{2.75}$$

est valide pour le polyèdre associé aux solutions de PTV3. Cette contrainte se lit ainsi. Si j et h ne sont pas servis par le même véhicule, la somme des charges de deux véhicules concernés est supérieure à $C+C_{min}$. Une contrainte identique peut être obtenue en considérant $p > 2$ clients.

La découverte de telles violations incite à étudier dans le futur l'aspect bin packing du PTV avec une formulation comme celle que nous venons de présenter PTV3 donnant plus d'informations sur ce sous-problème que la formulation classique à deux index.

2.8 Difficulté de la formulation de GPTV

Nous présentons dans cette section, deux points qui sont solutions du système :

$$\begin{array}{lcl} (i) & x_e \geq 0 & \forall e \in E \\ (ii) & x(\delta(\{0\})) \geq 2k & et\ pair \\ (iii) & x(\delta(v)) \geq 2 & \forall v \in V_0 \\ (iv) & x(\delta(S)) \geq 2R(S) & \forall\ S \subset V_0,\ S \neq \emptyset \\ (v) & x_e\ entier & \forall e \in E \end{array}$$

mais qui ne sont pas des solutions de $GPTV$. Ces points violent respectivement une contrainte de boîte (figure 2.17) et une contrainte d'arbre de rang 2 (figure 2.18). Dans la première instance, considérons la partition de

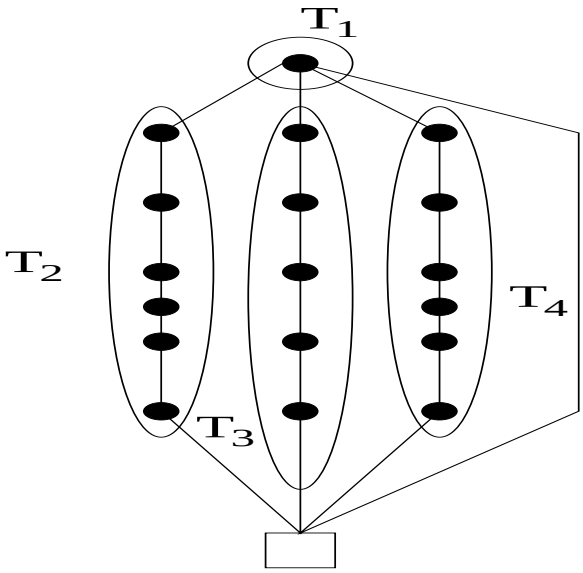


FIG. 2.17 - *Violation d'une contrainte de boîte, $C = 10$, commandes unitaires*

V_0 en quatre ensembles $T_1, \ldots, T_4$ définis sur le dessin. La demande des trois derniers ensembles est respectivement $6, 5, 6$ et on a $r(V_0|T_2, \ldots, T_4) = 3$. On conclut que la contrainte de boîte associée à $(V_0, T_2 \ldots, T_4)$ est violée, c'est-à-dire :

$$x(\delta(V_0)) + \sum_{i=2,\ldots,4} x(\delta(T_i) \geq 12 \qquad (2.76)$$

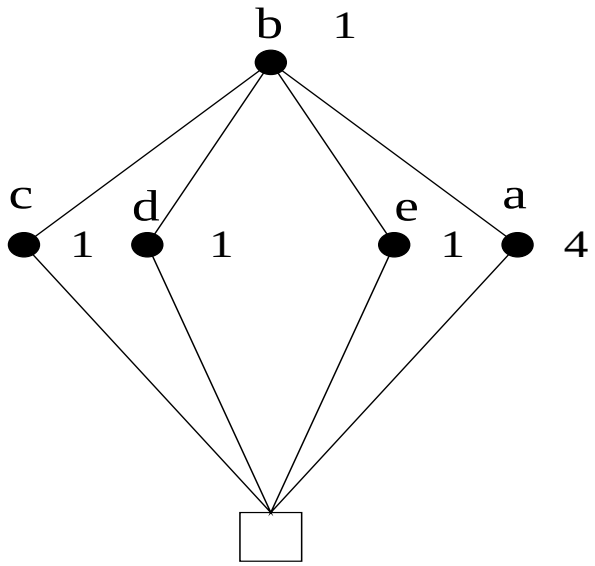


FIG. 2.18 - *Violation d'une contrainte d'arbre, $C = 4$*

Dans la seconde instance, nous présentons un exemple classique introduit par Fleischmann [Fle82]. On considère l'arbre $A \subset \gamma(V_0)$ de racine a obtenu en supprimant les arêtes incidentes au dépôt dans la solution dessinée figure (2.18). Comme tous les chemins entre ce sommet et les feuilles de l'arbre ont une demande strictement supérieure à la capacité $C = 4$, on peut générer une contrainte d'arbre de rang 2. L'ensemble des feuilles de l'arbre est $N_f = \{c, d, e\}$ alors que l'ensemble des non feuilles est $N_{nf} = \{a, b\}$. Notons $F(A) = (\gamma(N_{nf}) \cup \delta(N_{nf})) \setminus E(A)$. La contrainte suivante

$$x(F(A)) \; + \; \sum_{i \in N_f} x(\delta(i)) \; \geq 8 \tag{2.77}$$

est valide pour GPTV et violée par le point de la figure (2.18). En fait cette contrainte correspond à la contrainte d'arbre de rang 2 (2.33) pour le PTV.

Nous n'avons pas d'exemple pour l'instant de points satisfaisant à la fois le système précédent, et les familles de contraintes de boîte et d'arbre, mais qui ne soit pas une solution de GPTV.

Chapitre 3

Les problèmes de séparation

3.1 Généralités

Les résultats exposés dans ce chapitre sont le fruit de collaborations avec J.M. Belenguer, E. Benavent, J.M. Clochard, A. Corberan, D. Naddef et G. Rinaldi.

Comme nous l'avons mentionné dans le premier chapitre, la connaissance partielle du polyèdre du PTV doit être associée à de bons algorithmes pour l'analyse des solutions fractionnaires. Le lien entre le sous-problème de séparation et le problème principal d'optimisation est précisé par les résultats de Grötschel et al. [GLS81] et Padberg et Rao [PR81]. Considérons un polytope P dans $\mathbb{R}^m$, c un m-vecteur et les deux problèmes suivants :

Problème principal d'optimisation

$$\min cx$$
$$x \in P$$

Problème de séparation

Soit un vecteur $\tilde{x} \in \mathbb{R}^m$, trouver une contrainte valide pour P violée par $\tilde{x}$

ou prouver qu'il n'existe pas de telle contrainte.

Le premier problème peut être résolu par un algorithme polynomial si et seulement si le second peut être résolu de manière polynomiale pour tout vecteur rationnel $\tilde{x}$.

Dans le cas du PTV, nous connaissons deux types de contraintes pour lesquelles existe un algorithme polynômial d'identification. Il s'agit des contraintes de sous-tour et de 2-couplage (cf [GP85] pour la description de ces algorithmes). Pour les autres contraintes intégrant la capacité, il est prouvé que le problème d'identification est NP-difficile. Nous présentons dans les sections suivantes des heuristiques pour résoudre les problèmes d'identification correspondant à chaque contrainte, c'est-à-dire

Soit L une famille de contraintes valides du PTV, soit un point $\tilde{x} \in \mathbb{R}^m$, trouver une contrainte appartenant à L violée par $\tilde{x}$.

On notera x la solution fractionnaire (optimale) du programme linéaire à une itération donnée et $G(x)$ le sous-graphe induit par les arêtes telles que $x_e > 0$.

3.2 Séparation des contraintes de capacité

Nous rappelons d'abord la contrainte de capacité sur laquelle nous allons travailler (contrainte 2.4).

$$x(\delta(S)) \geq 2 \left\lceil \frac{d(S)}{C} \right\rceil \qquad \forall S \subseteq V_0,\ S \neq \emptyset$$

Les idées de base pour la séparation des contraintes de capacité ont été données par Harche et Rinaldi [HR91]. Elles sont développées dans le paragraphe suivant. Araque et al. [AKMP94] ont écrit en même temps que nous des algorithmes de contraction gloutons. Un algorithme glouton est décrit dans la procédure (3.2.1). Enfin, nous présenterons un algorithme de recherche tabou (procédure (3.2.2)). Les résultats numériques obtenus montreront l'importance des contraintes de capacités.

Harche et Rinaldi ont montré que le problème de séparation pour la contrainte (2.4) était NP-complet et ont écrit plusieurs heuristiques d'identification qui vont être décrites brièvement maintenant.

heuristique 1) Vérifier si (2.4) est satisfaite pour l'ensemble des sommets correspondant à chaque composante connexe de $G(x)$ et de $G(x) \setminus \{0\}$.

heuristique 2) Contracter récursivement les arêtes e non incidentes au dépôt telles que $x_e \geq 1$ (voir Padberg and Rinaldi [PR91] pour une description détaillée de la procédure de contraction dans le cas du PVC). Contracter une arête e revient à remplacer les deux extrémités de e par un super sommet dont la demande est égale à la somme des demandes des extrémités de e. Si le super sommet a une demande supérieure à C, l'ensemble des sommets du graphe d'origine «inclus» dans le super sommet induit une violation de (2.4).

Soit $GS(x)$ le graphe contracté obtenu. Il peut être facilement montré que la recherche d'une contrainte de capacité violée dans $G(x)$ est équivalente à la recherche d'une contrainte violée dans $GS(x)$. A partir de maintenant, toutes les heuristiques de capacité seront appliquées à $GS(x)$, mais pour limiter les notations, nous dirons qu'elles sont appliquées à $G(x)$.

heuristique 3) La contrainte de capacité fractionnaire est obtenue en remplaçant $\lceil \frac{d(S)}{C} \rceil$ dans (2.4) par $\frac{d(S)}{C}$. Soit $f(S) = x(\delta(S)) - 2\frac{d(S)}{C}$. Il est possible de trouver un ensemble $S \subseteq V_0$ pour lequel $f(S)$ est minimum en utilisant un algorithme de flot maximum. Si $f(S) < 0$, la contrainte de capacité fractionnaire et la contrainte (2.4) correspondante sont violées. Sinon, l'ensemble S est un candidat uniquement pour une violation de (2.4).

Nous avons implémenté toutes ces procédures. En les utilisant conjointement, on obtient les résultats présentés dans les tables (3.5.1) (colonne «Harche et Rinaldi») et (3.5.2) (colonne «H. et R.»).

Algorithme glouton de contraction

Procédure 3.2.1 *(Procédure gloutonne)*

G.1. *Choisir un ensemble initial S*

G.2. *Si le dépôt appartient à S, vérifier que (2.4) n'est pas violée pour l'ensemble $V_0 \setminus S$, autrement vérifier (2.4) à la fois sur S et $V_0 \setminus S$.*

G.3. *Choisir un sommet dans $v \in V_0 \setminus S$ qui maximise $x(S : \{v\})$, c'est à dire la somme des valeurs des arêtes entre S et v. En cas d'égalité, on choisit de préférence un sommet non adjacent au dépôt, puis s'il reste des ex æquo, on choisit aléatoirement un de ces sommets. Ajouter v à S et aller à G.2.*

Voici une liste de candidats pour figurer comme ensemble initial. Pour chaque catégorie, on choisira plusieurs instances d'ensemble et on exécutera autant de fois la procédure gloutonne.

1 - Un ensemble composé d'un sommet de V_0

2 - Les extrémités d'une arête e

3 - Un ensemble correspondant à une contrainte de capacité serrée actuellement dans le programme linéaire

4 - Le complémentaire d'un tel ensemble

5 - Un ensemble tiré aléatoirement dans V_0

6 - Un ensemble tiré aléatoirement contenant le dépôt

Dans les stratégies (1 - 4), le nombre d'instances possibles est limité, alors que dans les deux dernières, on peut générer aléatoirement autant d'instances que de sous-ensembles de V. Nous avons fixé dans ce dernier cas un nombre maximum d'appel égal à 10 fois le nombre de sommets. Notons, aussi, que les stratégies (1, 2, 3, 5) reviennent à élargir de manière gloutonne un ensemble de sommets. Les stratégies (4, 6) reviennent au contraire à réduire de manière gloutonne la taille d'un ensemble de sommets. Les meilleurs résultats ont été obtenus avec la stratégie (5), puis renforcés en utilisant en même temps les stratégies (1) (la moins coûteuse) et (5). Ces résultats sont présentés en table (3.5.1) et (3.5.2). La suite logique de cette étude était d'envisager une heuristique capable de revenir quelquefois en arrière ce qui a été fait par Belenguer et Benavent sous la forme d'une heuristique de recherche tabou.

Algorithme de recherche tabou

Soit S un sous-ensemble de V_0, on note $N^+(S)$ l'ensemble des sommets de $V_0 \setminus S$ adjacents à au moins un sommet de S et par $N^-(S)$ l'ensemble des sommets de S adjacents à au moins un sommet de $V_0 \setminus S$. Étant donné un ensemble de sommets S, tel que $d(S)$ et $x(\delta(S))$ ont été calculés pour vérifier (2.4), il est facile de faire de même pour les ensembles $S \cup \{v\}$, pour $v \in N^+(S)$, et $S \setminus \{v\}$, pour $v \in N^-(S)$.

A chaque itération de la procédure, un ensemble S est modifié en lui ajoutant, ou retirant, un sommet puis (2.4) est vérifiée pour le nouvel ensemble. La procédure peut se décomposer en deux phases, la phase d'expansion et la phase d'échange qui sont appliquées itérativement. A chaque itération de la phase d'expansion (resp. d'échange) un sommet est ajouté (resp. ajouté ou retiré) à S. La phase d'expansion se termine si aucun sommet ne peut être ajouté à S sans violer $d(S) \geq (p + ulimit)C$, où $ulimit$ est un paramètre donné, et p prend les valeurs 1 à $k-1$ pendant l'algorithme. Dans la phase d'échange, les mouvements sont sélectionnés de telle manière que l'ensemble S obtenu vérifie $(p-llimit)C \leq d(S) \leq (p+ulimit)C$, où $llimit$ est un autre paramètre.

Définissons, $smax = C(p+ulimit) - d(S)$ et $smin = d(S) - C(p-llimit)$. Alors l'ensemble des candidats à l'ajout dans S est $C^+(S) = \{v \in N^+(S) : d_v \leq smax\}$ alors que l'ensemble des candidats au retrait de S est $C^-(S) = \{v \in N^-(S) : d_v \leq smin\}$. Si un sommet v est ajouté, ou retiré, de S, le mouvement inverse est déclaré tabou pendant tll itérations.

A chaque itération des deux phases, l'objectif est de sélectionner un mouvement qui conduit à un ensemble S tel que $x(\delta(S))$ soit le plus petit possible. Cependant, dans la phase d'expansion, le sommet ajouté est choisi aléatoirement parmi un ensemble de bons candidats (voir le pas E.2 de l'algorithme). Ainsi, en exécutant plusieurs fois l'algorithme, on peut obtenir des résultats différents. Un paramètre $ntimes$ détermine le nombre de fois où l'algorithme est appelé et le paramètre per contrôle la taille de l'ensemble des bons candidats. L'algorithme suivant est lancé avec $S = \{i\}$, pour $i = 1, \ldots n$.

Procédure 3.2.2 *(Algorithme Tabou)*

Fixer $p = 1$.

Phase d'expansion

E.1. Calculer $smax$ et $C^+(S)$. Si $C^+(S)$ est vide, aller à I.0 (Phase d'échange).

E.2. Calculer $M = max\{x((S:v)) \ : \ v \in C^+(S)\}$ et sélectionner aléatoirement un sommet $v \in C^+(S)$ parmi ceux qui vérifient $M - per \leq x((S:v)) \leq M$.

E.3. Ajouter v à S, vérifier (2.4) et aller à E.1.

Phase d'échange

I.0. Fixer $iter = 1$.

I.1. Calculer $smax, smin, C^+(S)$ et $C^-(S)$. Retirer de $C^-(S)$ $(C^+(S))$ les sommets qui ont été ajoutés (retirés) à S lors des dernières tll itérations. Si $C^+(S) \cup C^-(S)$ est vide, aller à I.4.

I.2. Soit v le sommet pour lequel le maximum suivant est atteint

$$max\{\{x((S:j)), j \in C^+(S)\}, \{x((V_0 \setminus S:j)), j \in C^-(S)\}\}$$

I.3. Selon que $v \in C^+(S)$ ou $v \in C^-(S)$, ajouter ou retirer v à S et vérifier (2.4). Fixer $iter = iter + 1$, Si $iter > tope$ aller à I.4, autrement, aller à I.1.

I.4. Fixer $p = p + 1$, si $p \leq k - 1$ aller à E.1. Autrement, stopper.

La procédure précédente est donc appelée $ntimes$ fois avec les paramètres suivants $ulimit = 0.3, llimit = 0.1, tll = 5, per = 0.2$.

La table suivante compare les résultats obtenus par les trois familles de procédures décrites. L'algorithme tabou est noté deux fois avec des paramètres différents. Dans le premier cas, $ntimes = 1$ et le nombre maximum d'itérations dans la phase d'échange est fixé à $tope = 10$. Dans le second cas, $ntimes = 3$ et $tope = 20$.

Le premier tableau donne les résultats pour un certain nombre d'instances de la littérature. Nous notons les bornes supérieures BS et les bornes inférieures BI. Les bornes supérieures sont les valeurs des solutions réalisables obtenues avec une heuristique tabou. On peut à partir du nom de l'instance retrouver le jeu dont elle est issue, le nombre de sommets et de véhicules. Par exemple, l'instance E-n101-k8 est issue du jeu E et contient 101 sommets et 8 véhicules. Les jeux d'instances sont décrits dans le dernier chapitre.

Instance	BS	BI Harche et Rinaldi	BI glouton	BI tabou1	BI tabou2
E-n22-k4	375	375	375	375	375
E-n23-k3	569	569	569	569	569
E-n30-k3	534	508.5	508.5	508.5	508.5
E-n33-k4	835	831.1	833.5	833.2	833.5
E-n51-k5	521	510.9	514.524	514.524	514.524
E-n76-k10	832	773.592	787.449	787.472	789.31
E-n76-k7	683	659.467	660.316	660.834	661.251
E-n76-k8	735	703.484	710.188	709.682	711.053
E-n101-k8	817	788.758	795.875	795.081	796.149
F-n45-k4	724	723.8	723.667	724	724
F-n72-k4	238	232.5	232.5	232.5	232.5
F-n135-k7	1166	1154.2	1155.89	1157.88	1157.55
M-n101-k10	820	818.165	819.412	819.333	819.333

Table 3.5.1 : Comparaison des heuristiques d'identification des contraintes de capacité.

Nous avons effectué les mêmes tests sur un plus grand jeu d'instances (89 problèmes). Le tableau suivant détaille les résultats moyens pour l'écart (pourcentage restant à combler entre borne inférieure et borne supérieure), le nombre de coupes, le nombre d'appel de Cplex, et le temps cpu. Tous les calculs ont été effectués sur une station SUN sparc 10-51 et les temps de

calcul sont exprimés en secondes de cpu pour cette machine.

procédure	H. et R.	glouton	tabou 1	tabou 2
écart	3.29	2.48	2.43	2.51
nombre de coupes	339	946	1908	2299
nombre d'appel de Cplex	45.2	29.9	42.4	41
temps cpu	65.3	62.4	374	263

Table 3.5.2 : Comparaison des heuristiques d'identification des contraintes de capacité.

Notons que les résutats obtenus avec la première méthode sont nettement moins bons que les autres. L'utilisation de la méthode tabou est justifiée par un meilleur écart moyen alors que l'algorithme glouton obtient des résultats presque aussi bons en beaucoup moins de temps.

Nous présentons maintenant les résultats globaux pour les procédures de capacité. Lors de l'algorithme de coupe, celles ci sont exécutées l'une après l'autre et seulement si la précédente n'a pas trouvé de contrainte violée. L'ordre choisi est le suivant : 1) procédure gloutonne, 2) procédure tabou, 3) procédures «H.R.». Cet ordre tient compte de la rapidité des différentes procédures. Nous avons testé un ordre différent avec la procédure tabou en tête. Le nombre d'itérations a été réduit de manière importante, mais le temps de calcul a largement augmenté. En fait, ceci est autant dû aux procédures de séparation qu'à la résolution du programme linéaire. Comme la procédure tabou trouve plus de contraintes violées la résolution du programme linéaire est plus lente. Le temps cpu total est la somme du temps pris par la séparation, la résolution du programme linéaire et l'analyse de sensibilité du programme linéaire. On détaillera cette analyse dans le chapitre suivant.

Instance	BS	BI	écart	cpu séparation	cpu Cplex	itérations
E-n22-k4	375	375	0	1.06	1.07	22
E-n23-k3	569	569	0	0.42	0.33	9
E-n30-k3	534	508.5	4.8	2.17	1.2	21
E-n33-k4	835	833.5	0.18	3.73	2.91	23
E-n51-k5	521	514.524	1.2	8.01	8.58	25
E-n76-k10	832	789.416	5.1	152.36	194.42	70
E-n76-k7	683	661.256	3.2	51.14	95.79	42
E-n76-k8	735	711.17	3.2	81.91	176.14	73
E-n101-k8	817	796.314	2.5	100.09	194.31	56
F-n45-k4	724	724	0	5	2.78	27
F-n72-k4	238	232.5	2.3	8.56	2.87	19
F-n135-k7	1166	1158.25	0.66	1198.07	314.79	123
M-n101-k10	820	819.5	0.061	138.1	42.26	46

Table 3.5.3 : Résultats avec l'algorithme glouton plus l'algorithme tabou.

Sur un jeu de test plus large, nous avons obtenu un écart de 2.37 pour un temps de calcul moyen de 100 secondes. Ceci constitue une amélioration par rapport à l'algorithme tabou1 en terme d'écart (2.37 au lieu de 2.43) mais surtout en temps (100 au lieu de 374). Pour améliorer encore ces performances, il faudrait envisager une procédure de classement et de sélection des contraintes violées afin de limiter la taille du programme linéaire.

3.3 Séparation des contraintes de capacité généralisées

Nous incluerons aussi dans cette famille les contraintes de boîte dont le support est proche. Le problème d'identification pour les contraintes de capacités généralisées est NP-complet, car le calcul du second membre de ces contraintes inclut un calcul de bin packing. Nous allons donc présenter des heuristiques pour identifier des violations de (2.20) puis de (2.24).

Soit $\Omega = \{S_1, \ldots, S_s\}$ une famille d'ensembles disjoints de V_0. Dans la section (2.3.1), nous avons calculé une borne inférieure de $\Re(\Omega)$. C'est cette

borne que nous utilisons et on cherchera donc des violations de la contrainte

$$\sum_{j=1}^{s} x(\delta(S_j)) \geq 2\Big[bp(V_0|S_1,\ldots,S_s) - k + \sum_{j=1}^{s} \lceil d(S_j)/C \rceil\Big] \qquad (3.1)$$

Dans la suite, on notera $bp(\Omega) = bp(V_0|S_1,\ldots,S_s)$.

Dans la procédure suivante, la valeur de $INCRE$ est égale à la violation de (2.20) pour une partition Ω dans le cas où $bp(\Omega) = k+1$. Chaque fois que $INCRE \geq 0$, il est utile de calculer la valeur de $bp(\Omega)$. Initialement, $INCRE$ est fixé à la valeur 2 et on travaille sur le graphe contracté. On notera $d(u)$ la demande d'un super sommet u et $GS(x)$ le graphe obtenu de $G(x) \setminus \{0\}$ en contractant récursivement les arêtes dont la valeur est supérieure ou égale à 1. Les deux procédures qui suivent sont gloutonnes. La première procédure contracte itérativement l'arête la plus forte du graphe alors que la seconde fait de même au voisinage d'un super sommet.

Procédure 3.3.1 *(procédure gloutonne globale)*

GG.0) *Soit $HT(x) = GS(x)$. Fixer $INCRE = 2$.*

GG.1) *Soit Ω une partition définie par l'ensemble des sommets de $HT(x)$. Si $INCRE > 0$, calculer $bp(\Omega)$; dans le cas $bp(\Omega) > k$, la contrainte (2.20) est violée.*

GG.2) *Sélectionner l'arête la plus forte dans $HT(x)$. En cas d'égalité, on sélectionne l'arête dont la plus petite demande est la plus grande. Soit (u,v) l'arête choisie et y sa valeur.*

GG.3) *Fixer $INCRE = INCRE - 2(1-y)$. Si $\lceil \frac{d(u)+d(v)}{C} \rceil = \lceil \frac{d(u)}{C} \rceil + \lceil \frac{d(v)}{C} \rceil$, alors fixer $INCRE = INCRE + 2$.*

GG.4) *Contracter l'arête (u,v) et appeler $HT(x)$ le graphe résultant. Si $HT(x)$ ne contient pas d'arête, stopper. Autrement vérifier la contrainte de capacité (2.4) pour le super sommet que l'on vient de créer et aller à GG.1.*

La figure 3.1 montre une violation de la contrainte (2.20). Dans cet exemple, le graphe $GS(x)$ contient 3 super-sommets correspondant aux ensembles $S_1,\ldots,S_3$ et 3 sommets $S_4,\ldots,S_6$ de demandes respectives $(3175, 3925, 3700, 1500, 300,$

La capacité est $C = 4500$. Dans cette figure et les suivantes, le dépôt est le sommet numéro 1 (et non pas 0). On a donc $bp(S_1, \ldots, S_6) = 4$ alors que seulement 3 véhicules sont disponibles.

Nous présentons maintenant une seconde procédure dite «locale» pour la contrainte (2.20). La procédure locale est appliquée à chaque sommet v de $GS(x)$ et effectue des contractions à chaque itération dans le cocycle de v.

Procédure 3.3.2 *(Procédure gloutonne locale)*

GL.0) *Soit $HT(x) = GS(x)$. Fixer $INCRE = 2$.*

GL.1) *Soit Ω une partition définie par l'ensemble des sommets de $HT(x)$. Si $INCRE \geq 0$ et $bp(\Omega) > k$, la contrainte (2.20) est violée.*

GL.2) *Sélectionner l'arête la plus forte parmi les arêtes incidentes à v dans $HT(x)$. Soit (u, v) l'arête choisie et y sa valeur.*

GL.3) *Si $INCRE - 2(1 - y) \leq 0$, appliquer alors la procédure de contraction globale en commençant au pas GG.1.*

GL.4) *Contracter l'arête (u, v) et appeler $HT(x)$ le graphe résultant. Si $HT(x)$ ne contient pas d'arête, stopper. Autrement vérifier la contrainte de capacité (2.4) pour le super sommet que l'on vient de créer et aller à GL.1.*

Maintenant, nous présentons une heuristique de séparation pour la contrainte de boîte (2.24). Elle utilise le même schéma, mais le calcul de bin packing et les contractions ne sont pas faits sur V_0 mais sur un sous-ensemble H. Nous générons aléatoirement n ensembles (un pour chaque sommet v de $GS(x)$) et nous exécutons alors la procédure gloutonne.

Procédure 3.3.3 *(Procédure d'identification pour la contrainte de boîte)*

0) *Sélectionner aléatoirement un ensemble de sommets de V_0 de cardinalité $2k$. Ajouter v à cet ensemble.*

1) *Calculer de manière heuristique un flot maximum entre cet ensemble et le dépôt. Soit $(H : V \setminus H)$ la «pseudo» coupe minimum obtenue par ce calcul.*

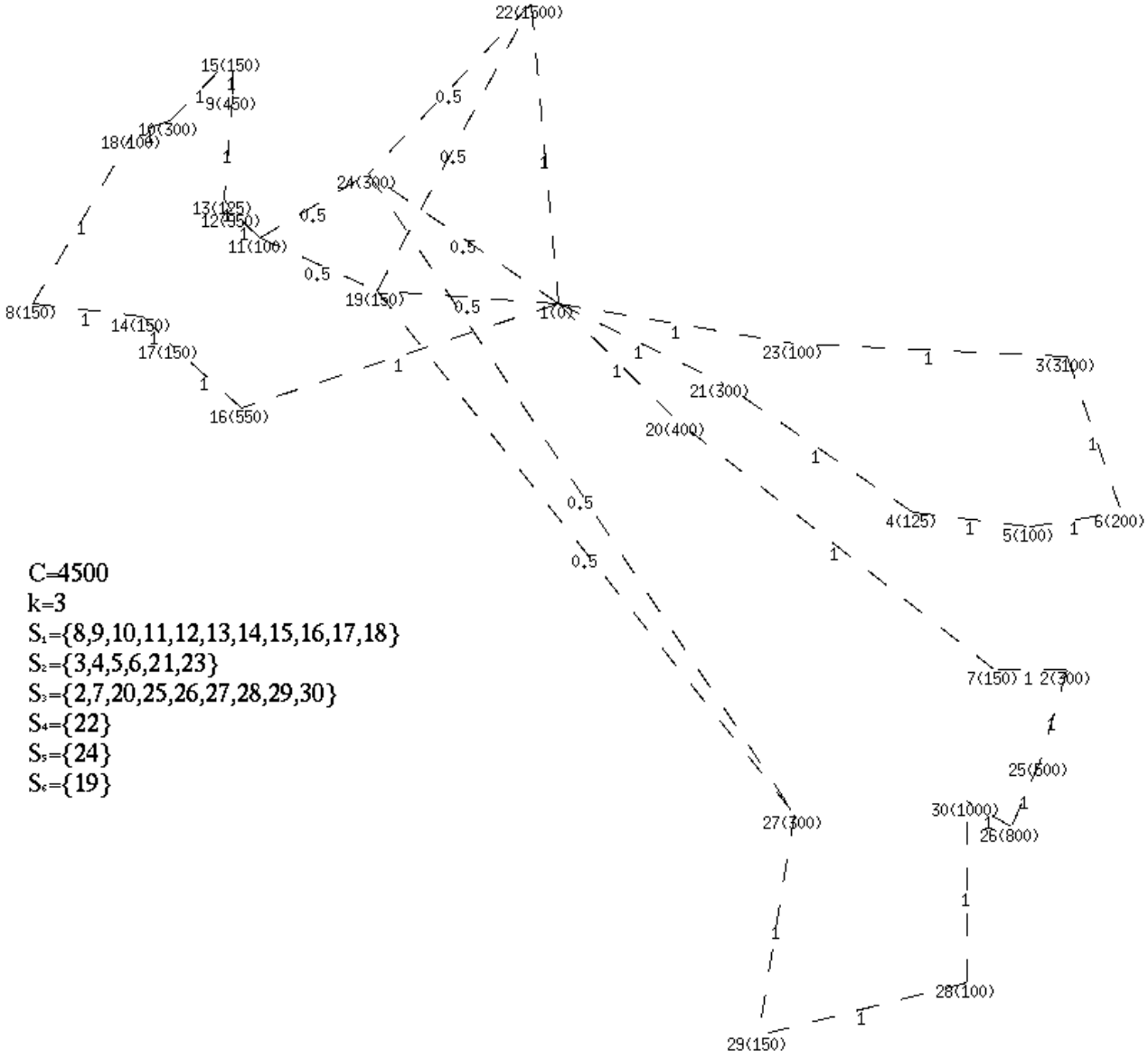


FIG. 3.1 - *Contrainte de capacité généralisée violée dans le problème E-n30-k3*

2) *Si* $x(\delta(H)) < 2(\lceil \frac{d(H)}{C} \rceil + 1)$, *appliquer la procédure globale de contraction en commençant au pas GG.1 avec* $HT(x)$ *égal au graphe induit par* H *dans* $GS(x)$ *et* $INCRE = 2(\lceil \frac{d(H)}{C} \rceil + 1) - x(\delta(H))$.

Les résultats obtenus en utilisant conjointement ces 3 procédures sont présentés dans la table (3.5.4) (colonne «capacité»).

3.4 Séparation des contraintes de peignes

Nous allons présenter une heuristique pour identifier des peignes violés de trois types correspondant respectivement à la contrainte de peigne classique valide pour le PVC (2.15), à l'extension introduite par Laporte et al. (2.17) et enfin à la contrainte dont le support contient le dépôt, introduite par Cornuejols et Harche (2.16). La procédure suivante est très proche de celle utilisée par Padberg et Rinaldi [PR91] pour le PVC. Elle travaille en deux étapes. Dans la première, on détermine un manche H candidat (ou plusieurs). Dans la seconde, on recherche des dents appropriées et on teste la violation des contraintes correspondantes. La principale modification par rapport au PVC est que l'on doit traiter de nouvelles catégories de dents.

Procédure 3.4.1 *étape 1 (Manches candidats)*

Pour chaque valeur de $EPS = 0,\ 0.1,\ 0.2,\ 0.3$, *et pour chacun des deux graphes* $G(x)$ *and* $G(x)\backslash\{0\}$, *faire :*

1.1) *Retirer du graphe les arêtes dont la valeur est inférieure ou égale à EPS ou supérieure ou égale à 1-EPS. Soit* $F(x')$ *le graphe résultant.*

1.2) *Trouver les composantes biconnexes de* $F(x')$.

1.3) *Les ensembles suivants sont alors considérés comme des manches candidats*

- *chaque composante biconnexe (ou bloc),*
- *pour chaque point d'articulation,* v, *de* $F(x')$, *l'union des blocs incidents à* v.

Remarque 1

En fonction de certains paramètres, on retire plus d'arêtes lors de l'étape 1.1 :

1.a) *les arêtes de $G(x)$ (ou $G(x)\backslash\{0\}$) à l'intérieur des super sommets du graphe $HS(x)$,*

1.b) *les arêtes de $G(x)$ (ou $G(x)\backslash\{0\}$) à l'intérieur de certains ensembles S contenant le dépôt tels que $x(\delta(\bar{S})) = 2 * r(\bar{S})$, où $\bar{S} = V_0 \setminus S$.*

1.c) *les arêtes de $G(x)$ (ou $G(x)\backslash\{0\}$) à l'intérieur de certains ensembles correspondant à des contraintes de capacité serrées du programme linéaire courant, i.e. des ensembles S n'incluant pas le dépôt tels que $x(\delta(S)) = 2 * r(S)$ et satisfaisant de plus pour au moins un sommet $a \in S$: $r(S \setminus \{a\}) = r(S)$.*

Les deux derniers types d'ensembles sont générés en prenant une contrainte de capacité du programme linéaire et selon la taille de l'ensemble S en considérant $\bar{S}$ (premier type d'ensemble pour la contrainte (2.16)) ou S (second type pour la contrainte (2.17)).

L'étape 2 est appliquée pour chaque manche découvert à l'étape 1. Notons H un tel manche.

Procédure 3.4.2 *étape 2 (identification des dents)*

2.1) *pour chaque sommet $v \in H$, considérons les dents candidates suivantes :*

- *les super sommets de $HS(x)$ contenant v et non entièrement dans H si la remarque 1 a) a été appliquée.*
- *les ensembles spéciaux contenant v et non entièrement dans H si les remarques 1 b) ou 1 c) ont été appliquées.*
- *les arêtes de $G(x)$ incidentes à v et dont l'autre extrémité est hors de H.*
- *Si v est un point d'articulation de $F(x')$, on considère aussi les ensembles suivants*
  - *les blocs incidents à v non inclus dans H*
  - *les semi-blocs (i.e. un graphe induit dans $G(x)$ par un sommet v point d'articulation et ses voisins dans le bloc)*

2.1.1) *Sélectionner la dent T qui maximise l'expression suivante : $x(E(T)) + R(T) + 1 - |T|$. On appellera cette expression «valeur de la dent».*

2.1.2) *Si cette valeur est supérieure à 0.5, la dent est ajoutée au peigne, autrement elle est rejetée. La meilleure des dents rejetée est cependant sauvée pour être utilisée au pas suivant.*

2.2) *Si le nombre de dents est impair, aller à 3). Autrement, soit T la dent du peigne dont la valeur est la plus faible, et soit T' la dent hors peigne sauvée.*

- *si $x_{T'} \geq 1 - x_T$, ajouter T' au peigne*
- *si $x_{T'} < 1 - x_T$, ou si en fait, aucune dent n'a été sauvée, on retire T du peigne*

2.3) *Tester si le peigne satisfait les conditions requises (voir la section (2.2.2), puis tester la violation de l'une des 3 contraintes.*

Lorsqu'un peigne est presque violé, nous utilisons une procédure pour élargir le peigne, ou les dents, décrite dans Clochard et Naddef [CN94].

Nous présentons les résultats obtenus avec cette heuristique dans la table (3.5.4) (colonne «peigne»). Les paramètres utilisés pour obtenir ces résultats sont les suivants. L'algorithme est exécuté d'abord sans les options de la remarque. On identifie ainsi les contraintes de types PVC. Pour chacun des ensembles décrits dans la remarque, on lance l'algorithme ce qui permet d'identifier d'autres composantes biconnexes et donc d'autres manches. Dans ce cas, tous les sommets appartenant à un des ensembles sont marqués une fois pour toutes, de telle manière que la recherche des dents de valeur maximum, pour un manche donné, soit plus facile.

La figure (3.2) montre une violation de contrainte de peigne avec le dépôt dans une dent. La première procédure permet d'identifier le manche $(4, 11, 13)$ comme une composante biconnexe du graphe obtenu à partir des arêtes e, $0 < x_e < 1$. Le sommet 11 a été marqué au préalable comme appartenant à la dent T_1, car la contrainte $x(\delta(\bar{T}_1)) \geq 4$ est serrée dans le programme linéaire courant.

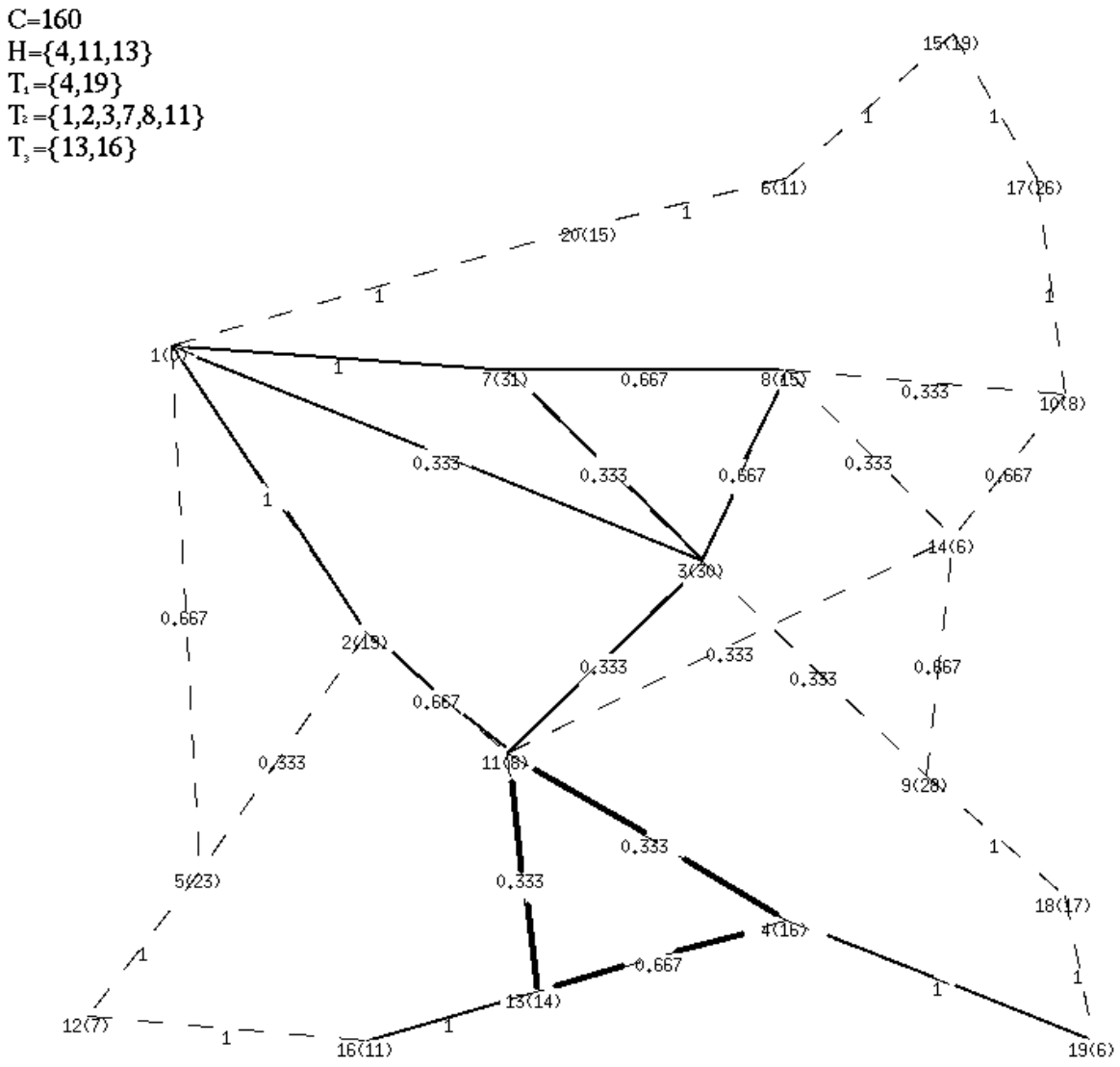


FIG. 3.2 - *Contrainte de peigne violée dans P-n20-k2*

3.5 Séparation des contraintes d'hypotour

Nous étudions maintenant le problème d'identification pour les contraintes d'hypotour. Nous présentons d'abord le cas de la contrainte d'arbre de rang 1 (2.30).

3.5.1 Contrainte d'arbre de rang 1

En théorie, on doit trouver un sommet $v \in V_0$ et un arbre A de racine v tels que

(i) tout A-chemin a une demande strictement plus grande que C

(ii) $x((\delta(N_{nf}) \cup \gamma(N_{nf})) \setminus E(A)) < 1$

où N_{nf} est l'ensemble des non feuilles de A. Ce n'est pas tout à fait ce que l'on cherche en pratique. On s'attache seulement à trouver un ensemble d'arêtes qui intersecte toutes les routes contenant un sommet v. La structure d'arbre ne sera jamais explicite. En fait nous allons procéder en deux phases. Soit $G(x) = (V, E(x))$ le graphe associé à une solution fractionnaire x. On va d'abord tester si $x(E \setminus E(x)) \geq 1$ est une contrainte valide (elle serait alors violée). Une condition suffisante est qu'il existe un sommet v tel que $E \setminus E(x)$ intersecte toute les routes contenant v. Ceci est équivalent à dire que $G(x)$ ne contient aucune route contenant v ou encore que tout chemin P dans $G(x)$ contenant v satisfait au moins une des deux propriétés suivantes :

(i) la demande P est strictement supérieure à C

(ii) au moins une des extrémités de P n'est pas connectée au dépôt dans $G(x)$

En énumérant tous les chemins de $G(x)$ contenant v de demande inférieure à C, on peut donc facilement décider si $x(E \setminus E(x)) \geq 1$ est une contrainte valide.

Dans la seconde phase, on renforce la contrainte $x(E \setminus E(x)) \geq 1$. Soit $F \subset (E \setminus E(x))$ l'ensemble des arêtes e telles qu'il existe un chemin contenant v de demande inférieure à C dans $(V, E(x) \cup \{e\})$. Il est clair que F intersecte toutes les routes contenant v, et par conséquent que $x(F) \geq 1$ est valide. On peut construire F en utilisant la même énumération que dans la première phase. Nous décrivons maintenant plus en détail les deux phases de l'algorithme.

Procédure 3.5.1 *(Recherche d'une violation)*

Énumérer tous les chemins contenant v dans $G(x)$ et de demande inférieure ou égale à C.

Si on trouve un chemin dont les extrémités sont connectées au dépôt (i.e. une route), on stoppe car il n'y a pas violation.

Ceci peut être fait en ajoutant à v, 2 voisins et en construisant de proche en proche un chemin jusqu'à ce que la demande du chemin dépasse la capacité C. Si on considère à chaque pas toutes les paires de sommets permettant de prolonger le chemin courant, on obtient effectivement un programme récursif qui énumère tous les chemins contenant v dans $G(x)$. Si aucune route n'est trouvée, on obtient une contrainte violée

$$x(E \setminus E(x)) \geq 1 \tag{3.2}$$

Procédure 3.5.2 *(Transformation en contrainte d'arbre)*

Énumérer tous les chemins contenant v dans $G(x)$ et de demande inférieure ou égale à C.

Pour chaque chemin trouvé P, calculer le sous-ensemble $Z_P \subset (E \setminus E(x))$ des arêtes qui permettent de prolonger P en un chemin de demande inférieure ou égale à C.

Si a et b sont les extrémités de P, on a $Z_P = \{e \in \delta(\{a\}) \cup \delta(\{b\}) : d(V(P \cup \{e\})) \leq C\} \setminus E(x)$.

Définissons Z l'union des Z_P pour tous les chemins énumérés. La contrainte

$$x(Z) \geq 1 \tag{3.3}$$

est la contrainte valide que nous générons dans notre algorithme.

Quelques remarques importantes améliorent les performances de l'algorithme présenté. Dans la première procédure, une énumération complète peut être coûteuse en temps de calcul. Notons cependant que le nombre d'arêtes dans $G(x)$ est en pratique beaucoup plus petit que $|E|$. Ainsi, le nombre de paires de voisins à considérer lors d'une itération est petit. Ce nombre peut être réduit de la manière suivante. Dans $G(x)$, on peut calculer pour chaque sommet $i \in V_0$, le chemin de demande minimum entre le sommet i et le dépôt. Notons $md(i)$ la demande d'un tel chemin. A une itération de l'algorithme récursif, notons P le chemin courant et a, b ses extrémités. On peut calculer une

borne inférieure de la demande du cycle dans $G(x)$ contenant P et le dépôt et de demande minimum. Si cette borne $d(V(P)) + md(a) + md(b) - d(a) - d(b)$ est plus grande que la capacité, il est inutile de continuer de traiter le chemin P.

La seconde difficulté est alors la profondeur de l'arbre des appels récursifs. Celui ci dépend clairement du ratio n/k. Dans le cas, où ce ratio est grand, on fixera une limite à la profondeur de l'arbre. Notons enfin qu'il serait plus coûteux de rechercher une violation en une seule étape. La première procédure est une énumération implicite dans $G(x)$ alors que la seconde est une énumération explicite dans G. Il est donc important de n'appeler la seconde procédure que quand on a identifié une violation.

Pour finir, nous précisons le lien entre l'ensemble Z et la structure d'arbre (v, A, N_f, N_{nf}, A^*) définissant les contraintes d'arbre. Notons N_{nf} l'ensemble des sommets des chemins énumérés dans la procédure (3.5.2). Notons $N_f = V(Z) \setminus N_{nf}$ et $A^* = x((\delta(N_{nf}) \cup \gamma(N_{nf})) \setminus Z$. Soit A un arbre couvrant les sommets de $V(Z)$. La structure (v, A, N_f, N_{nf}, A^*) définit exactement la contrainte (3.3).

3.5.2 Contrainte d'arbre étendue

Nous présentons maintenant la méthode utilisée pour identifier les contraintes d'arbre étendues dont nous avons déjà présenté un exemple (contrainte (2.38)). Dans cet exemple, la contrainte d'arbre est combinée avec une contrainte de sous-tour. La procédure (3.5.1) va alors être appelée avec un chemin initial déjà construit au lieu du singleton $\{v\}$. Si W est l'ensemble de sommets correspondant à la contrainte de sous-tour, ce chemin sera un chemin hamiltonien dans W.

Procédure 3.5.3 *(Identification d'arbre étendu)*

A.1. *Soit un client v, construire de manière gloutonne (par contraction d'arête) un ensemble W contenant v tel que $x(\delta(W)) < 4$ et $d(W) \leq C$.*

A.2. *Énumérer tous les chemins contenant v dans $G(x) \cap \gamma(W)$.*

A.3. *Utiliser successivement chacun de ces chemins comme entrée de la procédure récursive (3.5.1).*

Le reste de l'algorithme ne change pas. Si aucune route n'est trouvée, on a une violation et on génère alors l'ensemble Z avec la procédure (3.5.2).

Nous présentons maintenant d'autres extensions. De manière générale. On recherche une contrainte valide $\alpha x \geq \alpha_0$ et un ensemble d'arêtes Z qui satisfont : si x est une solution du PTV, alors soit $\alpha x > \alpha_0$, soit $G(x) \cap Z \neq \emptyset$. On a alors une contrainte valide

$$\alpha x \ + \ 2x(Z) \ \geq \ \alpha_0 \ + \ 2 \tag{3.4}$$

La première combinaison a déjà été décrite par la contrainte (2.38) comme la combinaison d'une contrainte de sous-tour et d'une contrainte d'arbre de rang 1. Cela revient à trouver un ensemble d'arêtes Z qui intersecte toutes les routes qui induisent un chemin hamiltonien sur W. On a alors

$$x(\delta(W)) + \ 2x(Z) \ \geq \ 4 \tag{3.5}$$

Soit e une arête dans $\delta(W)$. Si Z intersecte toutes les routes qui contiennent e et induisent un chemin hamiltonien sur W, alors

$$x(\delta(W)) + \ 2x(Z) \ \geq \ 2x_e \ + \ 2 \tag{3.6}$$

est une contrainte valide.

Soient e, f deux arêtes dans $\delta(W)$. Si Z intersecte toutes les routes qui contiennent e et f et induisent un chemin hamiltonien sur W, alors

$$x(\delta(W)) + \ 2x(Z) \ \geq \ 2x_e \ + \ 2x_f \tag{3.7}$$

est une contrainte valide.

La figure 3.3 montre une violation du type (3.7) dans laquelle $W = \{6, 15, 17, 10, 14, 9, 18, 19\}$, $e = (6, 20)$, $f = (19, 4)$. La capacité vaut 160 et la demande de $W \cup V(\{e, f\})$ est 152. Une route contenant W, e et f ne peut alors contenir qu'un sommet supplémentaire et même exactement un sommet parmi les sommets 4, 11 ou 12. L'ensemble d'arêtes $Z = \{(1, 4), (12, 4), (1, 11)\}$ intersecte toutes les routes qui contienent e, f et induisent un chemin hamiltonien sur W.

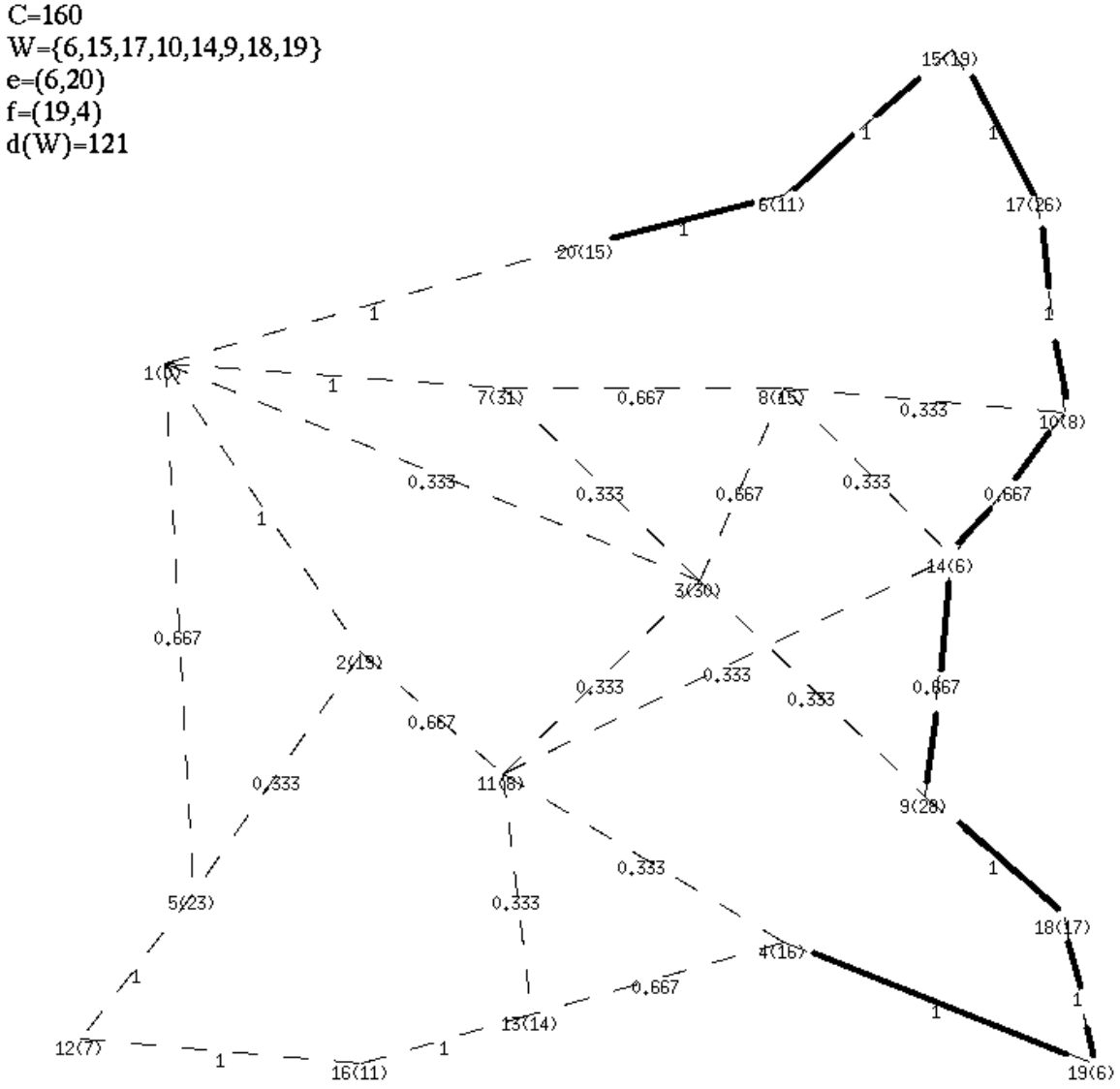


FIG. 3.3 - *Contrainte d'arbre étendue violée dans P-n20-k2*

Soit $\Omega = \{\Omega_i \ : \ i = 1, \cdots, p\}$, une famille d'ensemble de sommets. Si Z intersecte toutes les k-routes qui induisent un chemin hamiltonien sur chaque Ω_i,

$$\sum_{i=1,\cdots,p} x(\delta(\Omega_i)) + \ 2x(Z) \ \geq \ 2p \ + \ 2 \tag{3.8}$$

est une contrainte valide.

Si il existe un sommet v tel que $v \in \Omega_i, \ i = 1, \cdots, p$ et si Z intersecte toutes les routes qui contiennent v et induisent un chemin hamiltonien sur 2 des ensembles Ω_i.

$$\sum_{i=1,\cdots,p} x(\delta(\Omega_i)) + \ 2x(Z) \ \geq \ 4p \ - \ 2 \tag{3.9}$$

est une contrainte valide.

L'identification de ces 4 conditions (et contraintes) se fait en utilisant les mêmes procédures que pour la première combinaison. Pour (3.6) (resp. (3.7)), on ne considérera que les chemins contenant l'arête e (resp. les arêtes e et f). Pour (3.9) (resp. (3.8)), on travaillera non pas sur $G(x)$ mais sur $GS(x)$ le graphe contracté défini pour les contraintes de capacité simples, et l'ensemble Ω sera un sous-ensemble (resp. l'ensemble) des super sommets de $GS(x)$. En fait dans le cas de la contrainte (3.8), on recherchera un ensemble d'arêtes Z qui intersecte toute route R_l induisant un chemin hamiltonien sur Ω_1 et telle que le résultat du problème de bin packing appliqué à $(V(R_l), \Omega_2, \ldots, \Omega_p)$ soit plus grand que k.

Les résultats de l'utilisation de ces procédures sont présentés dans la table suivante. Pour chaque type de contrainte, l'algorithme est appliqué pour chaque sommet de $GS(x)$. Pour les deux dernières contraintes, nous nous sommes aperçu qu'il était utile de lancer l'algorithme sur les différents graphes contractés obtenus à chaque étape de l'algorithme pour les contraintes de capacité généralisées. Par contre, pour que le temps de calcul reste raisonnable, on fixe la profondeur d'exploration dans la procédure (3.5.1) à 1, c'est-à-dire qu'on se restreint à énumérer les chemins de longueur 2. Une autre amélioration a été apportée à l'algorithme. Dans la première étape , on n'arrête pas forcement l'énumération quand une route est découverte. On s'autorise à stocker un petit nombre de routes réalisables, puis ensuite, on recherche dans $\delta(\{0\})$ un ensemble d'arêtes F qui intersecte ces

routes et tel que $x(F) < 1$. Ces arêtes sont alors ajoutées à l'ensemble Z trouvé dans l'étape 2 pour obtenir une contrainte effectivement violée.

Les colonnes du tableau suivant représentent les résultats de l'algorithme de coupe, pour différentes associations des procédures de séparation. En colonne 2, apparaissent les résultats lorsque seules les contraintes de capacité sont utilisées. En colonne 3, on trouve les résultats pour les contraintes de capacité + les contraintes de peigne. En colonne 4, on trouve les résultats pour les contraintes de capacité + les contraintes de capacité généralisée. En colonne 5, on trouve les résultats pour les contraintes de capacité + les contraintes d'arbre. En colonne 6, on trouve les résultats quand on utilise toutes les contraintes.

Les lignes du tableau représentent le type de procédure, le nombre d'instances résolues, le nombre d'instances améliorées par rapport à la première colonne, l'amélioration moyenne pour les instances améliorées, le nombre moyen de coupes ajoutées au programme linéaire et le temps cpu total moyen en secondes. Notons que ces résultats sont obtenus pour l'algorithme de coupe (donc sans la phase d'énumération implicite).

procédures	capacité	peigne	génér.	arbre	toutes
écart	2.37	2.32	2.28	2.22	2.04
instances résolues	11/89	12/89	11/89	13/89	20/89
instances améliorées	x	41/78	6/78	67/78	76/78
amélioration moyenne	x	0.11	1.3	0.19	0.38
# moyen de coupes	60	61	60	71	73
temps cpu moyen	100	130	100	367	486

Table 3.5.4 : Comparaison des classes de contraintes.

On voit sur ce tableau que les contraintes de capacité généralisées ne concernent qu'un petit nombre d'instances, mais que leur utilité est alors très grande. Les contraintes de peignes sont utiles dans la moitié des cas et celles d'hypotour dans plus des deux tiers. Leur efficacité est relativement identique. Globalement, toutes les instances sauf deux ont été améliorées en utilisant les heuristiques autres que celles de capacité simple. On doit quand même constater que ces dernières sont de loin les plus importantes à la fois

par le nombre de contraintes violées et par leur efficacité.

Nous comparons maintenant nos résultats à ceux de la litterature. La table suivante présente nos résultats, ceux de Fisher ([Fis94a]), Harche et Rinaldi ([HR91]) et Mingozzi et al. ([MCH94]). Nous avons comparé les écarts obtenus avec celui que nous obtenons (colonne 1). Cette table permet aussi de comparer les temps de calcul.

Nom	écart	écart [Fis94a]	écart [HR91]	écart [MCH94]	cpu [a]	cpu [b] Fi.	cpu [c] H.R.	cpu [d] Min.
E-n101-k8	2.1	4.87	2.6		1708	18480	21285	
E-n30-k3	0		4.8		30		?	
E-n51-k5	0.67	3.34	1.8	0.71	129	11760	?	425
E-n76-k10	4.6	9.55		2.2	1919	11040		938
E-n76-k8	2.9		3.5		1282		7695	
F-n135-k7	0.6	2.57			2024	15240		
F-n45-k4	0	0.38			12	3000		
F-n72-k4	1.26	1.74			59	6300		
M-n101-k10	0	0.22			167	15600		

(a) secondes sur une Sun Sparc 10
(b) secondes sur un Apollo Domain 3000
(c) secondes sur une IBM risc 600
(d) secondes sur un intel 486 (33Mhz)

3.6 Améliorations possibles

Nous précisons dans cette section certaines limites de nos procédures de séparation et les améliorations possibles. En ce qui concerne la séparation des contraintes de capacité, nous avons de bonnes raisons de penser que nos algorithmes sont efficaces. En effet, nous avons éxecuté nos procédures gloutonnes et tabou avec des paramètres tels que le nombre d'ensembles exploré et le temps de calcul soient très grand (et déraisonnable dans le cadre de notre algorithme). Les résultats présentés dans ce chapitre ne sont pratiquement pas améliorés. En revanche, nous n'avons pas analysé l'influence du choix du second membre. Rappelons que plusieurs contraintes de capacité sont valides

pour un ensemble S de clients. Les seconds membres sont alors liés aux valeurs $\lceil d(S)/C \rceil$ ou $r(S)$ ou $R(S)$ selon la manière (globale ou locale, avec ou sans problème de bin packing) de calculer le nombre minimum de véhicules nécessaires pour servir S. Nous avons utilisé le second membre le plus faible alors que d'autres stratégies sont possibles : par exemple, rechercher d'abord des ensembles S tels que $R(S) > \lceil d(S)/C \rceil$ puis calculer la valeur du cocycle $x(\delta(S))$. Ce travail est donc à faire.

Les contraintes de capacité généralisée apparaîssent très efficaces mais pour un petit nombre d'instance seulement. La première explication tient à la nature des certaines instances pour lesquelles le problème de bin packing peut être mineur. La seconde est liée à la taille des instances. Lorsque le nombre de véhicules est grand, le problème de bin packing peut être aisé à résoudre globalement mais difficile à résoudre localement. Les contraintes de capacité généralisée sont alors inutiles et ce sont les contraintes de boîte et de chemin-boîte qui seraient alors utiles. L'algorithme que nous avons présenté pour les contraintes de boîte est simplement hérité de celui pour les contraintes de capacité généralisée. La suite de notre travail sera donc de réfléchir à un algorithme plus spécifique et aussi à un algorithme pour les contraintes de chemin-boîte.

Pour les contraintes de peigne, l'amélioration des heuristiques pour le problème de voyageur de commerce nous sera directement profitable puisque ces heuristiques sont en quelque sorte des sous-programmes de notre algorithme.

Les deux points appréciables concernant les contraintes d'arbre sont leur efficacité en terme d'amélioration de la «borne inférieure» et la rapidité de détection d'une violation. Le point négatif vient de la difficulté de construire la contrainte lorsqu'une violation est découverte. Nous avons remédié à ce problème en construisant une contrainte qui n'est pas forcement une facette. En pratique, il arrive souvent que plusieurs contraintes presque identiques soient introduites dans le programme linéaire à des itérations différentes. Il est probable dans ce cas que ces contraintes correspondent à une même facette. Un travail est donc à faire pour améliorer la construction des contraintes d'arbre introduites dans le programme linéaire.

Chapitre 4

Algorithme de «Branchement et Coupe»

4.1 Structure de l'algorithme

Nous présentons maintenant un algorithme exact pour résoudre le problème du PTV, basé sur la méthode de «Branchement et coupe» Notre algorithme s'inspire largement de celui de Clochard et Naddef [CN93] écrit pour le PVC, qui suit lui même le modèle utilisé par Padberg et Rinaldi [PR91]. Pour le PTV, Laporte et al. [LND85] ont aussi présenté un algorithme assez similaire. Nous préciserons dans les sections ultérieures les différences entre ces algorithmes.

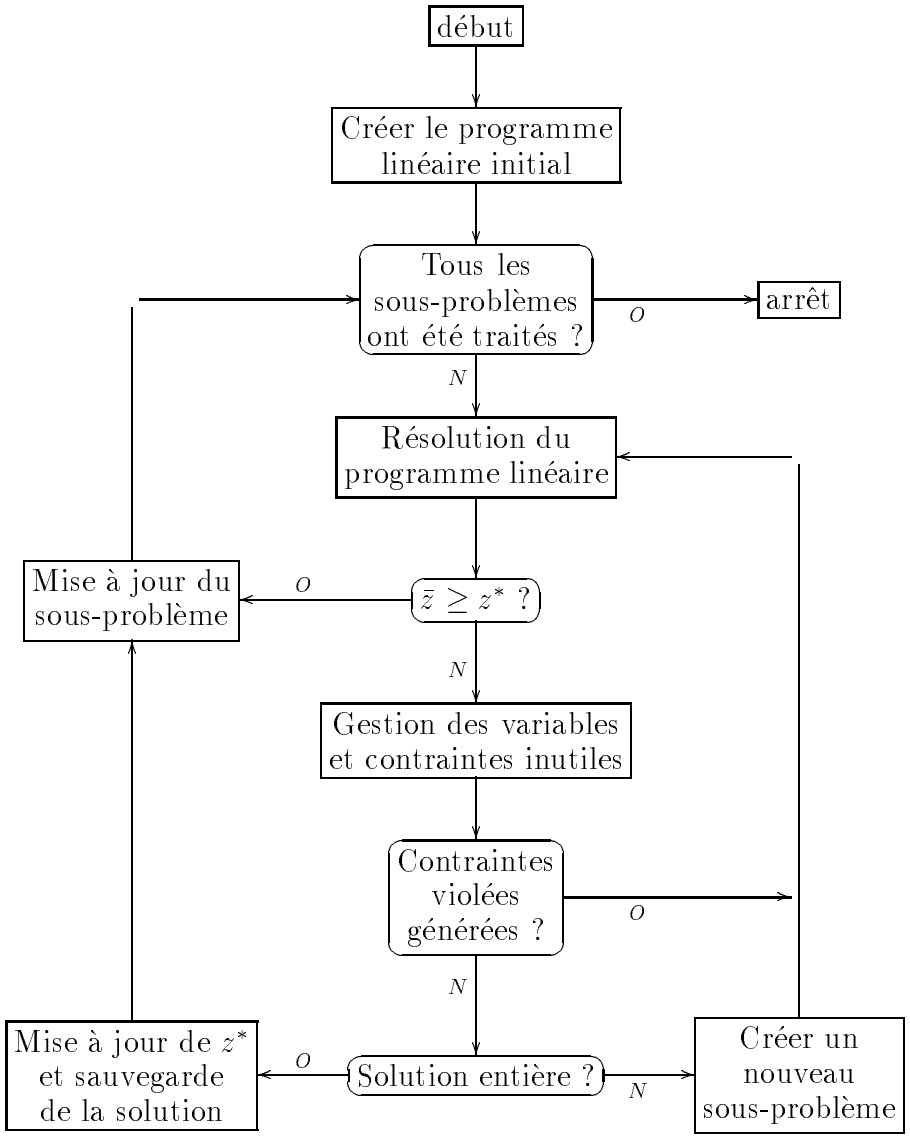
Les bases théoriques sur les algorithmes de coupe, de «Branchement et Evaluation» et de «Branchement et Coupe» peuvent être trouvées dans [PR91]. Le principe de l'algorithme de «Branchement et Coupe» a déjà été developpé à la section (1.6.4). Quatre éléments clefs cités dans [PR91] caractérisent un bon algorithme de «Branchement et Coupe». Il s'agit d'une bonne heuristique pour trouver une solution réalisable «proche» de la meilleure solution, d'un ensemble de procédures pour la séparation des contraintes faisant partie de la description partielle du polyèdre du PTV, d'une interface soignée avec le résolveur de programmes linéaires et enfin d'un programme d'énumération qui combine séparation en sous-problèmes et algorithme de coupe. Ce sont ces deux derniers éléments que nous traitons dans ce qui suit. Notons que des résultats numériques partiels sont donnés dans la plupart des sections.

Les jeux de test sont présentés à la fin du chapitre.

Nous présentons d'abord (page suivante) le diagramme de fonctionnement de notre algorithme. Dans ce schéma, z^* représente la valeur d'une solution réalisable (et donc une borne supérieure) et $\bar{z}$ la solution du programme linéaire courant (et donc une borne inférieure).

Le premier bloc qui concerne le choix du programme linéaire initial est décrit en section (4.2). Les modules «Résolution du programme linéaire» et «Gestion des variables et contraintes inutiles» seront traités dans la section (4.3). Les divers éléments (procédures de séparation) du module «Contraintes violées générées ?» ont été décrits dans le chapitre précédent. Nous précisons dans la section (4.4) comment nous utilisons ces éléments.

Il nous restera alors à présenter la gestion de l'arbre d'énumération. Comme nous réalisons une exploration en profondeur d'abord, la séparation d'un problème en deux sous-problèmes se fait en deux temps. D'abord, on ajoute une contrainte (dite de branchement, par exemple $x_e = 0$ pour une arête e) au problème courant (module «Créer un nouveau sous-problème»), on résoud le nœud correspondant puis on modifie la contrainte de branchement précédente (qui devient dans notre exemple $x_e \geq 1$) pour obtenir le second sous-problème. Le module «Mise à jour du sous-problème» consiste donc simplement à une modification de la dernière contrainte de branchement (non modifiée) introduite et à la suppression des contraintes de branchement déjà modifiées. Ces différents modules sont détaillés dans la section (4.6).



Si la solution d'un sous-problème est supérieure à la borne supérieure z^*, le sous-problème est considéré comme résolu. De même, un sous-problème est considéré comme résolu si les deux sous-problèmes qu'il a engendré sont résolus. Enfin, si la solution d'un sous-problème est entière, on doit mettre à jour la borne supérieure z^*.

4.2 La phase d'initialisation

La première étape de l'algorithme est l'obtention d'une borne supérieure et d'un ensemble initial de contraintes. Dans un premier temps, nous avons utilisé une heuristique améliorant celle de Clarke et Wright [CW64]. Cependant, les méthodes de recherche itérative apparaissent depuis quelques années comme les plus efficaces et nous utilisons maintenant une heuristique tabou que nous avons décrite au premier chapitre.

Le choix du programme linéaire initial est largement inspiré de la formulation du PTV comme un programme en nombres entiers (section 2.1). Cependant, la relaxation linéaire correspondante ne peut être utilisée telle quelle, car le nombre des contraintes de capacité est trop important. On doit considérer plutôt la relaxation où ces dernières contraintes sont enlevées. Cornuejols et Harche [CH93] proposent d'anticiper les contraintes de capacité qui vont être violées dans la solution de cette relaxation, et de les mettre dans le programme linéaire initial. Ces contraintes sont obtenues en calculant un k-arbre minimum sur le graphe et en générant une contrainte pour chaque chemin entre deux feuilles du k-arbre ainsi que pour le complémentaire de ce chemin. Nous présentons dans le tableau suivant une comparaison des écarts et des temps de calcul pour les deux relaxations linéaires (avec et sans un ensemble initial de contraintes). Ces tests ont été réalisés sur 90 instances. La première colonne indique l'écart moyen (en pourcentage) entre bornes inférieure et supérieure. La seconde donne le temps cpu moyen en secondes. La troisième colonne et le terme «victoires» indique le nombre de fois où la stratégie est meilleure que l'autre en terme d'écart.

Ensemble initial de contraintes	écart	cpu	# victoires
contraintes de degré	2.04	490	30
contraintes de degré et de capacité	2.04	481	32

On voit que la différence est globalement faible entre les deux méthodes. Par ailleurs quand une méthode est meilleure pour une instance donnée, l'écart est très faible.

4.3 L'aspect programmation linéaire

C'est le logiciel Cplex 2.0 ([Inc93]) qui a été utilisé comme résolveur de programme linéaire. A chaque itération de l'algorithme, on rajoute des contraintes violées et on utilise la méthode duale du simplex pour résoudre le nouveau programme linéaire. On doit veiller à ce que la résolution du programme linéaire soit rapide. Pour cela, on doit fixer soigneusement certains paramètres du logiciel Cplex. D'autre part, on doit éviter que la taille du programme linéaire ne devienne trop importante. Nous décrivons dans cette section les choix les plus important concernant cette gestion.

La plupart des paramètres par défaut de Cplex ont été utilisés. Seules les tailles des tables représentant le programme linéaire sont fixées par notre programme. Ce choix est important, car il influe sur le temps de calcul. Par exemple, si le nombre de coefficients maximum (MATSZ) est fixé trop petit. Il faudra en cours de résolution ajuster la taille de l'espace des données. Si au contraire MATSZ est choisi trop grand, le programme va prendre beaucoup de place en mémoire avec le risque de faire du swapping (transferts entre la mémoire et le disque de la station). Nous avons choisi de fixer $MATSZ = MATRZ * m/2$, où MATRZ est le nombre maximal de lignes, fixé lui même en fonction de la «capacité mémoire» de notre machine (m est le nombre d'arêtes). Une seule mise à jour de l'espace des données est réalisée avant la première séparation en sous-problèmes au nœud racine. La valeur de MATSZ est alors mise à jour. On calcule d'abord le nombre moyen de coefficients par lignes du programme linéaire courant et on fixe le nouveau rapport $MATSZ/MATRZ$ égal à ce nombre.

Dès que le programme linéaire est résolu, il est possible de le mettre à jour par analyse des coûts réduits. Pour une variable non basique e dont la valeur vaut zéro, on sait que la solution optimale ne contient pas e, si la somme des valeurs du coût réduit correspondant et de la fonction objective est strictement supérieure à la valeur z^* d'une solution réalisable connue. Pour une variable non basique e dont la valeur vaut un, on comparera z^* avec la valeur de la fonction objective moins le coût réduit (qui est négatif). Dans le premier cas, on peut alors retirer définitivement la variable du programme linéaire. Dans le second, on fixe la variable à la valeur 1 dans le programme linéaire. La fixation de variables intervient toutes les IFIX itérations. Notons

que dans ces calculs, «a strictement plus grand que b» veut dire $a \geq b + ZL$, où ZL est un paramètre que nous avons fixé à 0.001. De plus, une contrainte sera dite violée si la violation est plus grande que $3 * ZL$.

On peut aussi enlever des contraintes du programme linéaire. Nous enlevons toutes les IREM itérations, les contraintes dont la variable d'écart associée est basique. Le pool des contraintes supprimées est relu si on ne peut pas trouver de contrainte violée avec les heuristiques de séparation. Dans tous les cas si le nombres de contraintes atteint MARSZ, on essaiera d'éliminer des contraintes avec cette méthode.

La table qui suit donne les statistiques sur les bornes et le temps de calcul obtenus en fonction de IFIX et IREM. On peut noter le gain de temps obtenu grâce à la gestion des variables et des contraintes. Pour la suite, on utilisera donc $IFIX = 1$ et $IREM = 1$.

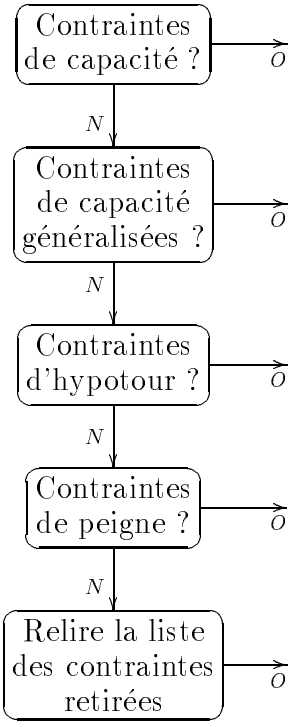
paramètres	Borne inférieure	cpu
IREM=0 IFIX=0	2.05	668
IREM=0 IFIX=1	2.04	513
IREM=10 IFIX=1	2.04	495
IREM=10 IFIX=10	2.04	493
IREM=1 IFIX=1	2.04	481

4.4 Le générateur de contraintes

Les différentes procédures de séparation sont décrites dans le chapitre précédent. Nous traitons dans cette section des conditions d'appel de ces heuristiques. Il apparaît en effet que l'intérêt d'un type de contrainte dépend du type de problème traité. Il est en particulier important de bien choisir l'ordre des procédures de séparation mais aussi de ne pas appeler à toutes les itérations une procédure qui ne trouve pas de contrainte violée.

Nous avons défini une stratégie basée sur cinq paramètres qui traduisent les dernières améliorations de la fonction objectif et l'influence de chaque type de contrainte. Ces paramètres sont *noimprove*, *nocap*, *nogen*, *nohypo*,

nocomb. Lorsqu'ils sont tous égaux à 0, on a le comportement par défaut du générateur de contraintes. Dans le diagramme général présenté en début de chapitre, on peut alors remplacer le bloc «Contraintes violées générées ?» par l'enchaînement suivant



Si *noimprove* vaut 1, toutes les procédures de séparations sont exécutées à la suite pour l'itération courante. La valeur de *noimprove* change à chaque itération et vaut 1 si l'amélioration de la fonction objective durant les 10 dernières itérations a été inférieure à $2 * ZL * z^*/100$. Si *noimprove* reste égal à 1 durant *maxnoimprove* itérations, aucune procédure de séparation n'est appelée et on recherche une contrainte pour brancher. Si *noimprove* vaut 0 mais un autre paramètre vaut 1, la procédure correspondante à ce paramètre n'est pas appelée du tout. Enfin si *noimprove* vaut 1 mais un autre paramètre vaut 1, la procédure correspondante est appelée mais avec une limite de temps de calcul. Le calcul des paramètres associés aux procédures de séparation est basé sur le nombre de contraintes violées trouvées pendant les 3 dernières itérations. Si aucune contrainte de type capacité (resp. capacité généralisée, peigne, hypotour) n'a été trouvée lors des 3 dernières itérations,

nocap (resp. *nogen*, *nocomb*, *nohypo*) vaut 1. L'intérêt de ces paramètres est donc de réduire le temps de calcul global. *noimprove* a un effet sur le nombre d'itérations total et conjointement avec les autres paramètres permet d'adapter l'algorithme au type de problème.

L'influence du paramètre *maxnoimprove* est traduite par la table suivante. La valeur 1000 de *maxnoimprove* correspond en pratique au choix d'arrêter l'algorithme de coupe uniquement si on ne trouve pas de contrainte valide violée. Si un tiers des bornes inférieures sont améliorées, ce n'est pas de manière significative. Le temps de calcul est en revanche presque doublé. Il est donc important d'utiliser le paramètre *maxnoimprove*.

maxnoimprove	écart moyen	cpu moyen	# victoires
10	2.04	481	0
1000	2.04	756	34

L'ordre dans lequel nous exécutons les différentes procédure est important. L'ordre présenté correspond au meilleur choix en moyenne pour les instances que nous avons testé, mais on obtient des résultats différents suivant les problèmes. Il est donc difficile de conclure sur un ordre optimal.

Finalement, nous présentons dans la table suivante les résultats obtenus quand les contraintes éliminées sont sauvegardées et relues lorsque l'on ne trouve plus de contrainte violée. Au vu de ses résultats, l'amélioration en terme d'écart ne justifie pas l'augmentation de temps de calcul.

Relecture des contraintes éliminées	écart	cpu	#victoires
oui	2.03	821	47
non	2.04	481	0

4.5 Particularités de la phase d'énumération

On doit faire la différence entre la phase d'algorithme de coupe initiale et la phase d'énumération qui suit (on dira aussi phase de branchement). Si on considère l'arbre d'exploration des solutions, la première phase correspond au nœud racine et la suivante aux autres nœuds.

Une procédure heuristique est exécutée à intervalles réguliers. Elle utilise la solution fractionnaire courante pour essayer d'améliorer la borne supérieure. Toutes les arêtes dont la valeur est supérieure à un seuil donné sont fixées à la valeur 1 à priori. L'heuristique des «savings» est alors utilisée pour générer une solution réalisable et une procédure tabou permet d'améliorer cette solution. La borne supérieure utilisée par notre algorithme est mise à jour si l'heuristique trouve une meilleure solution.

Durant la phase de branchement, les variables du programme linéaire peuvent être fixées, mais pas retirées du programme linéaire, car les contraintes de branchement incluses à un instant donné dans le programme linéaire ne sont pas valides pour les autres nœuds de l'arbre d'exploration. Il y a une exception cependant, qui correspond au cas où tous les nœuds pour lesquels ces contraintes ne sont pas satisfaites ont été éliminés.

On n'a pas intérêt à retirer les contraintes suivant le même critère qu'au nœud racine. On risquerait en effet de supprimer une contrainte «inutile» pour le sous-problème courant et «utile» pour un autre sous-problème. Pendant la phase de branchement, on ne retirera une contrainte que si elle a été introduite après la création du dernier nœud de branchement.

4.6 Les stratégies d'énumération

Nous avons restreint notre étude au cas où l'exploration prend la forme d'un arbre binaire. Soit $\tilde{x}$ une solution fractionnaire du programme linéaire courant $LP_{\tilde{x}}$, un vecteur $\alpha \in \mathbb{R}^m$ et un scalaire $\alpha_0 \in \mathbb{R}$. On appelle contrainte de branchement une contrainte non valide de PTV, $\alpha\tilde{x} \geq \alpha_0$. Si $\alpha_0 < \alpha x < \alpha_0 + 1$, on peut créer deux sous-problèmes de $LP_{\tilde{x}}$ en ajoutant à $LP_{\tilde{x}}$ respectivement les contraintes $\alpha x \geq \alpha_0 + 1$ et $\alpha x \leq \alpha_0$. Par exemple, si la variable $\tilde{x}_e$ vaut 0.5, on va créer deux sous-problèmes à partir des inéquations $x_e \geq 1$ et $x_e \leq 0$. Nous avons comparé plusieurs stratégies dans le choix de la contrainte de branchement, nous présentons d'abord celles étudiées dans la littérature.

Padberg et Rinaldi [PR91] décrivent une sélection de variable de branchement, où les candidats sont comparés à l'aide des deux critères suivants : la valeur de la variable (doit être la plus proche de 0.5 possible) et la valeur du

coefficient correspondant dans la fonction objective (doit être la plus grande possible). Alors la variable x_e choisie est fixée d'un côté à 0 et de l'autre à 1. Cette expérimentation faite à l'origine sur le PVC a été faite ensuite sur le PTV donnant des résultats nettement moins bons que pour le PVC. Une stratégie différente est de choisir une variable proche de 0 ou de 1. Araque et al. [AKMP94] rapporte de meilleurs résultats en choisissant une arête proche de 0.75, plutôt que proche de 0.5. Ces tests ont été effectués dans le cas du PTV avec commandes unitaires. Au contraire, Thienel [Thi94], dans le cas du PVC a expérimenté que ce genre de sélection était toujours moins bon que la sélection de Padberg et Rinaldi.

Pour choisir la variable de branchement, on peut aussi estimer à priori l'augmentation de la fonction objective, lorsque l'on fixe une variable à 0 ou 1. On peut faire cela en considérant le programme linéaire écrit sous forme basique. Pour une variable en base, et la ligne correspondante dans le tableau, on peut calculer à l'aide des coûts réduits l'augmentation minimale de la fonction objective, pour un pivotage, lorsque la variable est fixée à 0 ou 1. On effectue en fait une itération de la méthode duale du simplex. Notons qu'il s'agit de la règle de branchement du code Land-Powell utilisée par Laporte et al. [LND85] et Achuttan et al. [ACH] pour le PTV. Si on veut analyser l'effet au delà d'un pivotage, on doit alors construire le programme linéaire avec la contrainte de branchement, puis exécuter le nombre voulu d'itérations de la méthode duale du simplex. Applegate et al. [ABCC94] utilisent avec succès cette technique pour le PVC après avoir présélectionné un certain nombre d'arêtes proches de 0.5.

Des alternatives au branchement sur variables ont été envisagées récemment par Clochard et Naddef [CN93] pour le PVC et Fisher [Fis94a] pour le PTV. Clochard et Naddef comparent le branchement sur variables avec celui sur des contraintes plus sophistiquées. Ils recherchent un ensemble de cocycle impair et utilisent le fait que la valeur du cocycle d'un ensemble doit être paire. En particulier, en utilisant les contraintes de sous-tours (i.e. des ensembles de cocycle proche de 3), ils obtiennent souvent de meilleurs résultats qu'avec le branchement sur variables. Finalement, ils concluent en adoptant une stratégie hybride, c'est-à-dire en branchant tantôt sur arête, tantôt sur contrainte de sous-tour. Fisher propose aussi une stratégie hybride. Il propose de brancher soit sur une arête, soit sur un client, i.e. il

construit successivement des affectations partielles en ajoutant un client à chaque fois. Notons que l'arbre d'exploration n'est plus binaire dans ce cas. Cette technique apparaît utile dans le cas, où les clients sont regroupés dans des régions précises.

Nous allons proposer maintenant d'autres critères de branchement afin de traduire la spécificité du problème. Auparavant et à titre de comparaison, nous présentons les résultats obtenus avec les stratégies suivantes.

- Branchement sur l'arête la plus proche de 0.5 (et en cas d'égalité sur l'arête la plus longue) ;
- Branchement sur l'arête la plus proche de 0.75 (et si égalité sur l'arête la plus longue) ;
- Branchement avec test d'une présélection d'arêtes

Dans le dernier cas, on considère, un ensemble de 10 arêtes entre 0.45 et 0.65. On calcule pour chaque arête la solution des deux sous-problèmes correspondants. On choisit alors l'arête donnant la meilleure amélioration minimale. On a fixé la profondeur maximale de l'arbre d'énumération à 30 nœuds. Ceci explique que certaines instances ne soient pas résolues. Les résultats pour ces 3 tests sont présentés dans le tableau suivant. Ils ont été réalisés sur 15 instances. La première colonne indique le nombre d'instances résolues. La seconde colonne indique le temps total d'exécution. Notons qu'il est impossible de comparer les temps de calcul des stratégies qui ne résolvent pas toutes les instances, car dans ce cas la stratégie la plus rapide est en fait celle qui a échoué le plus vite. Enfin, nous avons classé pour chaque instances les stratégies en fonction du nombre de nœuds nécessaires pour résoudre l'instance. La troisième colonne et le terme «victoires» indiquent le nombre d'instances où la stratégie a été première dans le classement précédent.

stratégie	instances résolues	temps	# victoires
0.5	13	24h30	0
0.75	12	18h40	0
test	15	14h	15

Ces résultats montrent l'intérêt de tester plusieurs variables plutôt que d'utiliser un critère fixe.

Nous allons maintenant comparer ces méthodes avec le branchement sur contrainte. Nous nous sommes restreints au branchement sur les contraintes de sous-tour, c'est-à-dire que nous recherchons dans une solution fractionnaire x des ensembles de sommets S tels que $d(S) < C$ et $x(\delta(S)) < 4$. Nous obtenons ce genre d'ensemble avec le même algorithme glouton que pour la séparation des contraintes de capacité. Pour chaque sommet de $G(x)$, on applique l'algorithme en stoppant dès que l'ensemble courant a une demande supérieure à C. Nous obtenons donc au moins n candidats S qu'il va falloir départager.

Malgré nos conclusions sur le branchement sur arête, nous avons essayé un critère fixe pour choisir l'ensemble S qui sera utilisé pour brancher. Le choix de l'ensemble S a d'abord été fait avec l'un des critères suivants.

- $x(\delta(S))$ le plus proche de 3 ;
- $x(\delta(S))$ le plus proche de 2.85 ;
- $x(\delta(S))$ le plus proche de 3.15 ;
- $x(\delta(S))$ entre 2.75 et 3 tel que $d(S)$ soit maximale ;
- $x(\delta(S))$ entre 2.75 et 3 tel que la distance entre S et le dépôt soit maximale ;
- $x(\delta(S))$ entre 2.75 et 3 tel que le nombre de super sommets dans S soit maximal.

En testant les trois premières conditions, on essaye de mettre en évidence l'intérêt d'équilibrer l'arbre d'énumération. Il apparaît en effet que le côté de l'arbre correspondant à la contrainte $x(\delta(S)) \geq 4$ est plus difficile à résoudre que celui où on introduit $x(\delta(S)) \leq 2$. Les trois dernières conditions traduisent le fait que l'on voudrait des sous-problèmes plus faciles au sens du problème d'identification. On essaye donc soit de créer des super sommets de forte demande, soit de fixer la solution loin du dépôt. Dans tout les cas, les candidats ex æquo sont départagés par leur demande. Le candidat choisi est celui qui a la demande la plus forte. Le tableau suivant présente les résultats.

stratégie	instances résolues	temps	# victoires
2.85	15	14h55	7
3	13	26h10	1
3.15	13	26h25	1
d(S)	14	5h40	4
distance	13	5h30	3
nombre de super sommets	13	5h25	2

La première stratégie est la meilleure mais elle n'est pas première dans la majorité des instances. Les 3 dernières stratégies qui sont spécifiques au PTV montrent aussi leur intérêt en prenant la première place dans 8 instances sur 15.

Pour améliorer ces résultats, nous allons utiliser, comme pour les arêtes, un test à priori d'une présélection d'ensembles candidats. Dans notre première série de tests, nous essaierons tous les ensembles candidats dont le cocycle se situe respectivement

- entre 2.5 et 2.85 ;
- entre 2.75 et 3 ;
- entre 2.85 et 3.1.

Les résultats sont assez ambigus. On ne peut résoudre toutes les instances. En revanche, ces stratégies se comportent souvent mieux que les précédentes en terme de nombre de nœuds. La troisième colonne du tableau suivant donne le nombre d'instances où c'est le cas.

stratégie	instances résolues	temps	améliorations
2.5-2.85	14	17h	4
2.75-3	14	5h50	4
2.85-3.1	14	18h	2

Notre seconde série de tests consiste à présélectionner tous les ensembles correspondants aux critères cités plus haut, à savoir les ensembles dont le cocycle est respectivement le plus proche de 3, de 2.85, de 3.15, et parmi ceux dont le cocycle est entre 2.75 et 3, celui dont la demande est la plus

forte, celui dont le nombre de sommets est le plus grand et enfin celui le plus loin du dépôt. On teste ensuite tous ces ensembles afin de choisir le meilleur. Finalement, nous avons fait la même expérience en ajoutant aux ensembles présélectionnés l'arête la plus proche de 0.75 et les dix arêtes les plus proches de 0.5. Dans la table suivante sont reportés les résultats pour ces stratégies ainsi qu'une comparaison avec les meilleures stratégies testées auparavant.

stratégie	temps cpu	# victoires	# nœuds moyen
2.85	14h55	0	76
présélection arêtes	14h	4	50
présélection ensembles	4h57	10	45
présélection ensembles + arêtes	5h51	6	35

C'est logiquement une des deux dernières stratégies que nous utiliserons dans notre algorithme. Elles permettent d'avoir un nombre réduit de nœuds dans l'arbre d'énumération. Le temps dépensé pour la sélection d'une contrainte de branchement est justifié, car le temps total de calcul est inférieur à celui des autres stratégies. Nous n'avons pas réussi à mettre en évidence que l'une des deux stratégies était meilleure que l'autre. Nos tests sur un plus grand nombre d'instances confirment les résultats précédents en terme de temps moyen de calcul et de nombre de nœuds moyen. Il semblerait quand même que pour les instances résolues les plus difficiles, la dernière stratégie soit meilleure.

Ces résultats nous encouragent à étudier d'autres stratégies, par exemple l'idée de brancher sur des contraintes de capacité qui ne sont pas de simples contraintes de sous-tour. Ce sera l'objet de prochains développements.

4.7 Améliorations possibles

Les remarques qui suivent décrivent des méthodes utilisées dans d'autres algorithmes de «Branchement et Coupe» mais que nous n'avons pas encore mis en œuvre. Elles concernent la fixation de variables, la méthode d'exploration de l'arbre d'énumération et la gestion des colonnes du programme linéaire.

L'intérêt de la fixation de variables à la valeur zéro a été montré dans la section (4.3). Nous avons montré comment fixer des variables non basiques mais il est aussi possible de fixer des variables basiques. On a aussi vu que la règle de branchement de Land-Powell permettait de trouver une borne inférieure de l'augmentation de la fonction objective, quand une variable en base est fixée à une de ses bornes. S'il apparaît que cette augmentation fait dépasser la borne supérieure, on peut fixer la valeur de la variable x_e. Ceci peut être fait lorsque l'on ne trouve plus de contrainte violée. Si la fixation d'arêtes basiques est possible, on peut résoudre de nouveau le programme linéaire sans avoir à ajouter de contrainte de branchement.

Le choix de la méthode d'exploration par profondeur a été d'abord conduit par un souci de facilité d'implémentation théorique mais aussi pratique (code pour le PVC déjà écrit). Il apparaît que cette méthode n'est pas la meilleure. Celle qui consiste à explorer le nœud dont la fonction objective est la plus faible domine la méthode précédente sauf si la solution optimale est connue à priori. Un développement ultérieur de l'algorithme sera donc d'utiliser cette nouvelle méthode d'exploration. On pourra aussi envisager de séparer un nœud en plus de deux fils.

La résolution de PVC de grande taille implique une gestion des variables très complexe (cf [PR91]). On sépare en général les variables en plusieurs ensembles, le premier étant utilisé pour construire le programme linéaire initial. A intervalles réguliers, on regarde dans les autres ensembles si des variables doivent entrer dans le programme linéaire (en calculant le coût réduit de la variable). Cette gestion apparaît comme un des points les plus difficiles à implémenter. Dans notre cas, nous avons introduit régulièrement de nouveaux types de contraintes aux supports assez différents. Ceci rendait encore plus complexe la gestion des colonnes hors programme linéaire et donc le calcul du coût réduit correspondant. Cette gestion est donc maintenant à implémenter pour améliorer les performances du système.

4.8 Problèmes de la littérature

Les premiers jeux de données (jeux E et M) diffusés largement pour le PTV l'ont été par Christofides et Eilon [CE69] puis Eilon [Eil71]. Ces données sont des problèmes générés aléatoirement ou bien des combinaisons de tels

problèmes. Un des problèmes seulement est construit de manière à ressembler à un problème réel. Les clients ont été répartis en régions. Ce problème est un des plus facile à résoudre malgré sa taille.

Des données tirées de problèmes réels sont décrites dans Fisher [Fis94a] (jeu F). Deux problèmes (45 et 135 sommets) modélisent un jour de distribution des terminaux Peterboro and Bramalea et Ontario de la société National Grocers Limited. Le troisième (72 sommets) concerne la distribution de pneus, batteries et accessoires aux stations services (données obtenues de la société Exxon).

4.9 Nouveaux problèmes

Nous avons crée quatre jeux d'instances particuliers. Pour le premier, les données sont aléatoires. Pour le second, on a essayé de se rapprocher des problèmes réels en particulier en répartissant les clients en régions. Le troisième jeux nous permet de tester le cas, où la capacité n'est pas le problème majeur. Nous avons choisi de fixer la capacité à une valeur plus grande que la somme des commandes. Enfin, un dernier cas permet de voir ce qui se passe quand le nombre de clients par véhicules est faible. Nous avons donc écrit un algorithme de génération de problèmes que nous présentons maintenant.

Algorithme CréerInstance
(TYPE,NN,CAPACITÉ,NV,L,l,NC,MD,DTYPE,OX,OY)

/* TYPE = ALÉATOIRE pour un problème généré aléatoirement */
/* TYPE = PVC pour un problème généré aléatoirement avec une capacité plus grande que la demande totale */
/* TYPE = AP pour un problème généré aléatoirement avec NC régions et NV $\leq$ NC-1 véhicules */
/* TYPE = KP pour un problème généré aléatoirement avec seulement quelques clients par véhicule */
/* NN nombre de clients */
/* CAPACITÉ = capacité des véhicules */
/* NV nombre de véhicules */
/* L taille du carré (L^2 est en fait la taille réelle) */
/* l taille d'une région (l^2 est en fait la taille réelle) */

/* NC nombre de régions */
/* MD demande moyenne */
/* DTYPE= 0, les demandes sont générées aléatoirement autour de MD */
/* DTYPE= p, les demandes sont d'abord générées aléatoirement puis p d'entre elles sont multipliées par 3 */
/* OX,OY coordonnées du dépôt */

{

/* Générer d'abord les coordonnées */
$si(TYPE == AP)$
{
GénérerSommet($currentNN, L, OX, OY$); /* si non fixé, généré de manière aléatoire */
$pour(i = 1; i <= NC \&\& currentNN < NN; i++)$
GénérerCoordonnées($NombreDeClients(NN, currentNN), l, L$); /* aléatoire dans le carré */
}
alors GénérerCoordonnées(NN, L, OX, OY);
/* aléatoire dans le carré */

/* Puis générer les demandes */
GénérerDemande($NN, CAPACITY, NV, MD$); /* aléatoire dans $[1, 2 * MD]$ */
$si(DTYPE > 0) ModifierDemande(NN, CAPACITY)$;
/* DTYPE clients ont leur demande multipliée par 3 */
/* Mettre à jour le nombre de véhicules */
$si(TYPE == ALATOIRE || TYPE == AP)$
MinimiserNV($NN, NV, CAPACITY$); /* Modifier le nombre de véhicules pour que le problème soit serré */
sinon $si(TYPE == PVC)$AugmenterCapacité($NN, CAPACITY$); /* Fixer la capacité égale à la demande totale*/
sinon $si(TYPE == KP)$ReduireCapacité($NN, NV, CAPACITY$); /* Fixer la capacité égale à la demande maximale et augmenter le nombre de véhicules */
}

Le jeu A d'instances a été généré avec les paramètres suivants
TYPE = ALÉATOIRE
NN = 32 to 69
CAPACITÉ = 100
MD = 15
DTYPE = NN/10
L = 100
DTYPE = 0
Tous les autres paramètres sont ignorés.

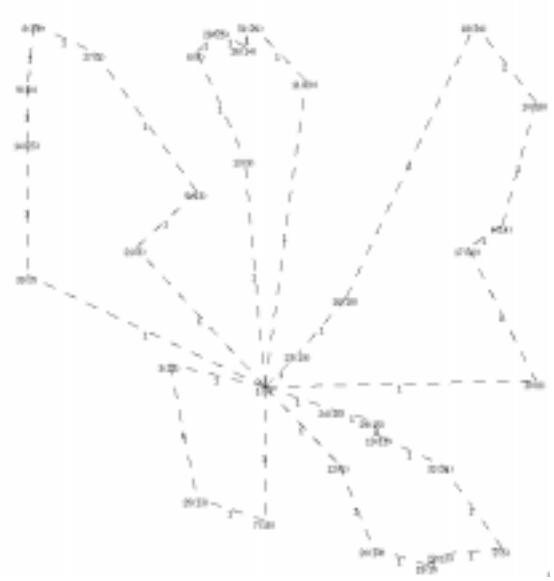


FIG. 4.1 - *Instance A-n33-k5 générée par le programme*

Le jeu B d'instances a été généré avec les mêmes paramètres excepté TYPE = AP.

Plutôt que de générer des problèmes de type PVC et KP avec le programme, nous avons préféré modifier les problèmes de la littérature de manière à avoir une base de comparaison. Nous avons donc créé deux jeux de données à partir des problèmes de grande taille pour le premier et de tous les problèmes de la littérature pour le second. Dans le premier (jeu C), la capacité des véhicules a été fixée à une valeur plus grande que la demande

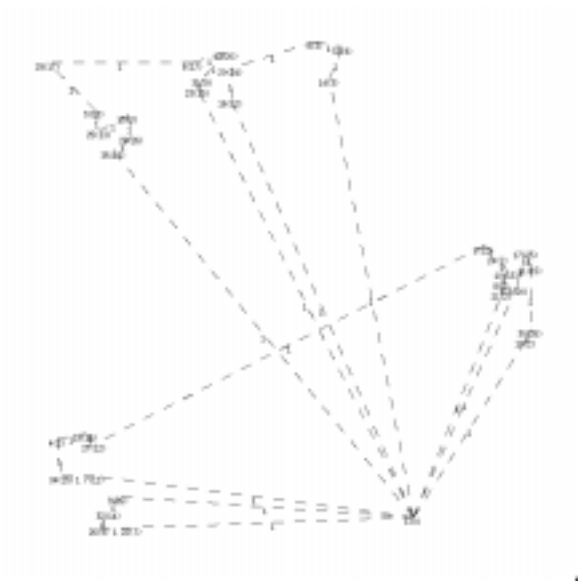


FIG. 4.2 - *Instance B-n35-k5 générée par le programme*

totale des clients. Dans le second (jeu P), nous avons multiplié le nombre de véhicules par deux et divisé la capacité des véhicules d'autant.

4.10 Analyse des résultats numériques

Cette section présente plusieurs types de remarques concernant les résultats obtenus avec notre algorithme. Nous avons d'abord essayé de dégager quel était, parmi les paramètres du PTV, celui qui influençait le plus la difficulté de ce problème. Dans un deuxième temps, nous analysons de manière plus précise l'effet des différentes stratégies de branchement sur nos résultats. Nous revenons aussi sur l'intérêt des différentes contraintes valides pour étayer les résultats du chapitre (3.5.2) par des résultats prenant en compte la phase d'énumération.

En utilisant les jeux de tests que nous avons créés, nous avons pu comparer la difficulté des différents types de problèmes. Le jeu C correspond à des problèmes de k-TSP (les données sont celles du jeu A excepté la capacité d'un véhicule qui est supérieure à la demande totale des clients). Toutes les instances ont été résolues facilement. Notons que le nombre de villes dans ce

jeu (entre 30 et 80) correspond à un PVC en général facile pour la méthode polyédrale. Les instances en dehors du jeu C sont serrées, c'est-à-dire que le rapport demande totale des clients sur capacité totale des véhicules est grand (proche de 1). Nous n'avons pas généré, ni trouvé dans la littérature d'instances intermédiaires entre ces instances serrées et les instances de type k-TSP.

Le tableau suivant permet de comparer les instances aléatoires (jeux A et E par exemple) à des instances tirées de cas réels ou construites pour simuler des problèmes réels (jeux B et F). Ces derniers apparaissent à taille égale plus facile que les premiers. Dans toute la suite, le terme écart désigne l'écart entre borne supérieure et borne inférieure à la fin de l'algorithme de coupe (ou nœud racine). L'écart moyen est la moyenne des écarts pour les instances d'une colonne. Une colonne correspond à un certain nombre d'instances satisfaisant un certain critère. Par exemple, la première colonne du tableau suivant précise l'écart moyen sur les 14 instances du jeu A ayant moins de 51 sommets.

jeu	A (≤ 50)	A (> 51)	B (≤ 50)	B (> 51)	E (≤ 50)	E (> 51)	F	M
nombre	14	12	12	11	4	7	3	4
écart moyen	1.65	3.4	0.52	3.4	0	3.7	0.6	5.8

Il peut être intéressant d'analyser quels autres paramètres font la difficulté d'un problème de tournées. Parmi les arguments traditionnels, citons : la position du dépôt, le nombre de sommets, le nombre de véhicules, le rapport «demande totale des clients» sur «capacité totale des véhicules». Nous n'avons pas fait d'analyse pour le premier argument. L'influence du dernier paramètre est traduite par la facilité des problèmes de type k-TSP, mais pour le reste des instances, nous n'avons pas pu dégager clairement son influence.

Nous présentons en revanche des tableaux montrant l'influence du nombre de sommets et du nombre de véhicules.

Étant donné la taille de notre jeu de test, ces résultats constituent moins des «théorèmes» que des indications pour comparer la difficulté de deux instances.

La table suivante montre l'influence du nombre de sommets. Le nombre

d'instances résolues correspond à la résolution par l'algorithme complet de «Branchement et Coupe» alors que l'écart est calculé au nœud racine..

# sommets	19-29	30-39	40-49	50-59	60-69	70-99	100-200
# instances	9	18	12	24	16	10	8
Instances résolues	9	18	11	13	5	2	2
écart moyen	1.17	0.91	1.01	2.49	3.1	3.42	4.04

La table suivante montre l'influence du nombre de véhicules.

# véhicules	2	3	4	5	6	7	8	9	10	11+
# instances	4	2	7	20	9	16	9	11	14	6
Instances résolues	4	2	7	19	8	9	4	4	5	2
écart moyen	0.35	0	0.25	0.98	1.6	2.5	3.2	3.98	3.8	5.2

Certaines instances du jeu P sont construites à partir d'autres instances en diminuant la capacité et en augmentant le nombre de véhicules. A chaque fois, on constate que le nouveau problème est plus difficile. On peut aussi faire la manœuvre inverse et augmenter la capacité en diminuant le nombre de véhicules. Le nouveau problème est alors plus facile. Les deux tableaux suivants montrent 7 couples d'instances et les écarts obtenus.

P-n22-k8	P-n34-k10	P-n51-k10	E-n76 -k8	E-n76 -k10	E-n101 -k8	M-n151 -k12
1.7	1.85	7.6	2.9	4.6	2.1	6.2

E-n22-k4	A-n34-k5	E-n51-k5	P-n76 -k4	P-n76 -k5	P-n101 -k4	P-n151 -k6
0	1.7	0.67	0	2.3	0.44	3.1

Au vu de ces résultats, le nombre de véhicules semble être le critère de difficulté majeur pour le PTV. Nous avons pensé réaliser d'autres comparaisons de ce type comme l'influence du nombre de client moyen par véhicules. Il faudrait pour cela un plus grand nombre d'instances de test.

Pour en terminer avec l'analyse de la difficulté du problème, nous précisons maintenant quels problèmes sont difficiles pour l'heuristique tabou.

Notons LB1 la valeur de la solution heuristique obtenue pour 1000 itérations de l'algorithme tabou. Cette valeur sert de borne inférieure dans notre algorithme de «Branchement et Coupe». Notons OPT la valeur de la solution obtenue par cet algorithme et LB2 la valeur de la solution heuristique obtenue en arrêtant l'heuristique au moment ou l'algorithme de «Branchement et Coupe» se termine. On peut alors classer les instances de notre jeu de test en quatre catégories.

	LB1=OPT	LB2=OPT	$LB2 > OPT$	OPT non trouvé
# Instances	50	2	8	39

On voit donc que 80 % des instances que nous pouvons résoudre sont «faciles» pour l'heuristique tabou. En revanche, nous avons pu résoudre par «Branchement et Coupe» 8 instances difficiles pour l'heuristique.

Nous présentons maintenant, en terme de gain de temps et de diminution de l'arbre d'énumération, un résumé des différentes améliorations que nous apportons par rapport aux algorithmes de «Branchement et Coupe» existants. Il faut d'abord rappeler que la gestion des contraintes et variables du programme linéaire permet de diminuer de 30 % le temps total de calcul. Cette gestion, tout comme celle de l'appel des heuristiques de séparation, est indispensable dans tout algorithme de «Branchement et Coupe».

Le choix d'une stratégie de branchement sur les inégalités de sous-tour plutôt que sur les valeur des variables est le choix le plus important puisqu'il permet de diviser par plus de deux le temps total de calcul (cf chapitre précédent). Indépendamment de ce choix, l'utilisation de la programmation linéaire pour la sélection de l'inégalité de branchement permet un gain de temps du même ordre. En utilisant notre meilleure stratégie de branchement, on divise par plus de 6, le temps de calcul par rapport à la stratégie habituellement utilisée dans la littérature.

La table suivante confirme cette estimation pour des instances plus difficiles. Le nombre de nœuds indiqué correspond au nombre de nœuds nécessaires pour combler 0.5% de l'écart à la fin de l'algorithme de coupe. Cela signifie que l'on ne développe pas complètement l'arbre d'énumération, très exactement que l'on élimine un nœud de l'arbre si la solution du programme linéaire est plus grande que la valeur au nœud racine divisée par 0.995. Cette

astuce permet de tester notre stratégie sur les instances de grande taille que nous ne pouvons pas résoudre à l'optimum. La première ligne de résultats indique le nombre de nœuds nécessaires quand on branche sur une variable en utilisant un critère fixe de choix (dans notre cas, la variable dont la valeur est la plus proche de 0.5). La seconde colonne correspond au branchement sur un ensemble sélectionné suivant un critère fixe (dans notre cas, l'ensemble dont la valeur du cocycle est la plus proche de 2.85). Enfin, la dernière colonne donne le nombre de nœuds obtenu en choisissant un ensemble par programme linéaire et parmi les ensembles correspondants aux six stratégies décrites au paragraphe (4.6). Les résultats présentés correspondent à des valeurs moyennes sur 20 instances difficiles testées.

stratégie d'énumération	une arête	un ensemble	6 ensembles
# nœuds	571	163	29
temps cpu	26086	4711	1520

Les gains obtenus sont donc encore plus appréciables pour cette série d'instances que pour les instances plus faciles que nous avons résolues.

Un second point apparaît comme fondamental dans la résolution du PTV par «Branchement et Coupe»: la nécessité d'une très bonne heuristique pour les contraintes de capacité simple. Les algorithmes que nous utilisons pour la séparation de ces contraintes font diminuer l'écart moyen de près d'un point par rapport aux algorithmes utilisés auparavant. Il est intéressant d'analyser la portée de cette amélioration dans l'arbre d'énumération. En ce qui concerne les instances que nous arrivons à résoudre, l'utilisation des nouvelles procédures pour les contraintes de capacité permet de diviser le temps de calcul par 4 et le nombre de nœuds par 8. En comparaison, les améliorations dues aux autres contraintes pourraient paraître marginales. Notons que nous avons fait moins d'efforts de développement pour ces contraintes. Malgré tout, elles permettent dans quelques cas de diminuer l'écart de manière notable, en particulier de résoudre des instances sans brancher. Elle permettent aussi de diminuer de 15 % (en moyenne sur les 39 instances que nous avons résolues) le nombre de nœuds de l'arbre d'énumération.

En reprenant, notre cadre de travail précédent, c'est à dire en étudiant

partiellement l'arbre d'énumération, on peut avoir une autre idée de la différence d'intérêt entre les procédures de séparation. Dans cette expérience, la borne supérieure est fixée à une valeur inférieure à la valeur réelle. Cette nouvelle valeur est calculée comme le minimum de la valeur d'une solution heuristique et de la valeur de la meilleure borne inférieure obtenue divisée par 0.995. Nous avons testé successivement les procédures de Harche et Rinaldi seules, nos procédures pour la séparation des contraintes de capacité, ces dernières plus respectivement les contraintes de peigne, les contraintes d'arbre et les contraintes de capacité généralisées, enfin toutes ces contraintes réunies. Ces procédures sont utilisées uniquement au nœud racine. Pendant la phase de branchement proprement dite, on se contente de séparer sur les contraintes de capacité. Les résultats sont décrits dans le tableau suivant. Les chiffres représentent des moyennes sur 20 instances difficiles.

stratégie	# de noeuds	temps cpu
H. & R.	310	3100
capacité	38	663
capacité + peigne	45	860
capacité + arbre	36	1425
capacité + généralisée	38	677
toutes contraintes	29	1520

Ces résultats confirment les résultats sur les instances plus faciles en terme de nombre de nœuds. En terme de temps cpu, les résultats sont faussés par le fait que le noeud racine est beaucoup plus coûteux en temps que la phase d'énumération partielle. Pour estimer les gains en temps, il faudrait allonger cette phase d'énumération partielle en essayant, par exemple, de combler 1% ou plus de l'écart au noeud racine.

Conclusion

Nous avons présenté dans cette thèse un algorithme de «Branchement et Coupe» pour le Problème de Tournées de Véhicules, en particulier des contributions importantes dans les quatre fondements d'un tel algorithme, l'étude du polytope PTV, les problèmes de séparation pour les contraintes valides, l'interface avec le résolveur de programme linéaire et l'arbre d'énumération implicite.

Pour le premier point, nous avons introduit de nouvelles contraintes valides pour le PTV : les contraintes de boîte, de chemin-boîte et d'arbre. Elles sont, sous certaines conditions, facettes de polytopes très proches de PTV. Ces contraintes généralisent la majorité des contraintes connues et sont originales par leur lien spécifique à l'aspect «capacité» du problème de tournées.

Des algorithmes ont été élaborés pour résoudre le problème de séparation pour quatre classes de contraintes : capacité, capacité généralisée, peigne et arbre. Ils permettent de mesurer l'efficacité respective de ces contraintes, mais aussi la variété des types de problèmes de tournées. Une gestion soigneuse du programme linéaire et de l'appel des algorithmes de séparation nous permet d'obtenir en un temps raisonnable des bornes inférieures qui améliorent en général celles trouvées dans la littérature.

Enfin, nous avons mis en évidence l'importance de la phase d'énumération en comparant différentes stratégies pour choisir une contrainte de branchement. L'augmentation du nombre d'instances que nous pouvons résoudre et le gain de temps sont très importants.

Nous n'arrivons pas cependant à résoudre certains problèmes difficiles (et jamais résolus) de la littérature. Ceci pousse bien sûr à réfléchir aux limites de notre méthode. Au vu de notre expérience, la principale limite vient de la formulation entière choisie. Il nous semble important maintenant d'étudier une formulation plus puissante du PTV, intégrant de manière plus explicite les problèmes de sac à dos et de bin packing. Les formulations décrites rapidement dans la section (2.7) satisfont cette condition. Elles permettent de plus d'utiliser sans modification tous les résultats trouvés pour notre formulation.

Par ailleurs, de nombreuses améliorations concernant les problèmes de séparation (section 3.6) et le programme de «Branchement et Coupe» (section 4.7) ont été envisagées et seront l'objet de développements futurs.

Bibliographie

[ABCC94] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Special session on tsp. In *15th International Symposium on Mathematical Programming*. University of Michigan, USA, aug 1994.

[AC91] N.R. Achuthan and L. Caccetta. Integer linear programming formulation for a vehicle routing problem. *European Journal of Operational Researchs*, 52:86–89, 1991.

[ACH] N.R. Achuthan, L. Caccetta, and S. P. Hill. A new subtour elimination constraint for the vehicle routing problem. to appear in EJOR.

[AHM90] Jesus R. Araque, Leslie Hall, and Thomas Magnanti. Capacitated trees, capacitated routing and associated polyhedra. Discussion paper 9061, CORE, Louvain La Neuve, 1990.

[AKMP94] J.R. Araque, G. Kudva, T.L. Morin, and J.F. Pekny. A branch-and-cut for vehicle routing problems. *Annals of Operations Research*, 50:37–59, 1994.

[AMS89] Agarwal, Mathur, and Salkin. Set partitioning approach to vehicle routing. *Networks*, 7:731–749, 1989.

[Ara89] Jesus R. Araque. *Contributions to the polyhedral approach to vehicle routing*. PhD thesis, State University of New York at Stony Brook, 1989.

[Ara90a] Jesus R. Araque. Lots of combs of different sizes for vehicle routing. Discussion paper 9074, CORE, Louvain La Neuve, 1990.

[Ara90b] Jesus R. Araque. Solution of a 48-city vehicle routing problem by branch and cut. Research Memorandum 90-19, Purdue University, 1990.

[Bal89] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[Bal93] E. Balas. The prize collecting traveling salesman problem: Polyhedral results. Technical Report MSRR-591, Graduate School of Industrial Administration, Carnegie Mellon University, July 1993.

[Bal94] R. Baldacci. *Algorithmi esatti per il Vehicle Routing Problem.* PhD thesis, University of Bologna, Italy, 1994.

[BG81] Lawrence Bodin and Bruce Golden. Classification in vehicle routing and scheduling. *Networks*, 11:97–108, 1981.

[BQ64] M. Balinski and R. Quadt. On an integer program for a delivery problem. *Operations Research*, 12:300–304, 1964.

[CCM91] Vincente Campos, Angel Corberan, and Enrique Mota. Polyhedral results for a vehicle routing problem. *European Journal of Operations Research*, 52:75–85, 1991.

[CE69] N. Christofides and S. Eilon. An algorithm for the vehicle dispatching problem. *Oprerations research*, 20, 1969.

[CFN85] G. Cornuejols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph amd some related integer polyhedra. *Math. Programming*, 33:1–27, 1985.

[CH93] Gerard Cornuejols and Farid Harche. Polyhedral study of the capacitated vehicle routing. *Mathematical Programming*, 60:21–52, jun 1993.

[Chr85] N. Christofides. *The Traveling Salesman Problem*, chapter 12 - Vehicle Routing, pages 431–448. John Wiley & Sons Ltd., 1985.

[Chv73] V. Chvatal. Edmonds polytopes and weakly hamiltonian graphs. *Mathematical Programming*, 5:29–40, 1973.

[CM90] N. Christofides and A. Mingozzi. *Vehicle routing : practical and algorithm aspects*. Logistics. Pergamon Press, 1990.

[CM94] V. Campos and E. Motta. Obtaining feasible solutions for the vehicle routing problem in a branch and cut scheme. Glasgow, 1994. Euro XIII.

[CMT81] N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11:145–164, 1981.

[CN93] Jean-Maurice Clochard and Denis Naddef. Use of path inequalities for tsp. In Giovanni Rinaldi anf Laurence Wolsey, editor, *Integer Programming and Combinatorial Optimization*, pages 291–312, 1993.

[CN94] Jean-Maurice Clochard and Denis Naddef. Some fast and efficient heuristics for comb separation in the symetric traveling salesman problem. Technical Report RR941M, submitted to Mathematical Programming, Institut IMAG, Grenoble, 1994.

[CW64] G. Clarke and J. Wright. Sheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[DKL86] Martin Desrochers, Antoon Kolen, and Abilio Lucena. A two-commodity flow approach for the vehicle routing problem. 1986.

[DL86] M. Dror and L.Levy. A vehicle routing improvement algorithm comparison of a greedy and a matching implementation for inventory routing. *Computer & Operations Research*, 13:33–45, 1986.

[DL91] M. Desrochers and G. Laporte. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Lettres*, 10(1):27–36, feb 1991.

[DLS90] Martin Desrochers, J.K. Lenstra, and M.W.P. Savelsbergh. A classification scheme for vehicule routing and scheduling problems. *European Journal of Operational Research*, 46:322–332, 1990.

[DSD84] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[DV89] M. Desrochers and T. W. Verhoog. A matching based saving algorithm for the vehicle routing problem. Technical Report G-89-04, Ecole des Hautes Etudes Commerciales de Montreal, 1989.

[Eil71] S. Eilon. *Distribution management: mathematical modelling and practical analysis"*, chapter 9. Griffin, London, 1971.

[FCG84] G. Finke, A. Claus, and E. Gunn. A two-commodity network flow approach to the traveling salesman problem. *Congressus numerantum*, 41:167–178, 1984.

[Fis81] Marshall L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, jan 1981.

[Fis94a] M. Fisher. Optimal solution of vehicle routing problem using minimum k-trees. *Operations Research*, 42(4):626–642, 1994.

[Fis94b] M. Fisher. A polynomial algorithm for the degree constrained k-tree problem. *Operations Research*, 42(4):776–780, 1994.

[FJ78] Marshall L. Fisher and Ramchandran Jaikumar. A decomposition algorithm for large-scale vehicle routing. Technical Report 78-11-05, The Wharton School, University of Pennsylvania, 1978.

[FJ81] Marshall L. Fisher and Ramchandran Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124, 1981.

[Fle82] B. Fleischmann. Linear programming approaches to traveling salesman and vehicle sheduling problems. XIth International Synposiym on Mathematical Programming, Bonn, 1982.

[GAG92] M. Gendreau, A.Hetz, and G.Laporte. A tabu search heuristic for the vehicle routing problem. Technical Report CRT-777, Centre de recherche sur les transports, Universite de Montreal, 1992.

[Gas67] T. Gaskel. Bases for vehicle fleet scheduling. *Operations Research Quaterly*, 18:281–295, 1967.

[GH91] M. Grötschel and O. Holland. Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming*, 51:141–202, 1991.

[GJ79] Michel R. Garey and David S. Johnson. *Computers and Intractability - A Guide to The Theory of NP-Completeness.* W.H. Freeman and Company - New York, 1979.

[GLS81] M. Grötschel, L. Lovasz, and A. Shrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[GM74] B. Gillet and L. Miller. A heuristic algorithm for the vehicle dispatch problem. *Operations Research*, 22:340–349, 1974.

[GP79] M. Grötschel and M.W. Padberg. On the symmetric traveling salesman problem: I and ii. *Mathematical Programming*, 16:265–280 and 281–302, 1979.

[GP85] M. Grötschel and M.W. Padberg. *The Traveling Salesman Problem*, chapter 8 - Polyhedral Theory, pages 281–290. 1985.

[Grö80] M. Grötschel. On the monotone symetric travelling salesman problem: hypohamiltonien/hypotraceable graphs and facets. *Math. Oper.Res.*, 5:285–292, 1980.

[HASG94] D.T. Hiquebran, A.S. Alfa, J.A. Shapiro, and D.T. Gittoes. A revised simulated annealing and cluster-first route second algorithm applied to the vehicle routing problem. *Engineering Optimization*, 22:77–107, 1994.

[HK71] M. Held and R.M. Karp. The traveling salesman problem and minimum spanning trees: Part ii. *Math. Prob.*, 1:6–25, 1971.

[HK85] M. Haimovitch and A.H Rinnoy Kan. Bounds and heuristics for capacitated routing problems. *Mathematic of Operations Research*, 10:527–542, 1985.

[HR91] Farid Harche and Giovanni Rinaldi. Vehicle routing. private communication, 1991.

[Inc93] CPLEX Optimization Inc. Using the cplex callable library and cplex mixed integer library. Technical report, 1993.

[Kar72] R.M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Plenum, new York, 1972.

[Kub92] Kubo. On the polyhedral structure of the vehicle routing. Technical report, Waseda University, 1992.

[Lap92] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[LK73] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the travelling salesman problem. *Operation Research*, (21):2245–2269, 1973.

[LK81] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

[LMN87] Gilbert Laporte, H. Mercure, and Yves Nobert. Cutting planes based on bin packing solutions for the vehicle routing problem. *Congressus Numerantium*, 59:165–178, 1987.

[LN84] Gilbert Laporte and Yves Nobert. Comb inequalities for the vehicle routing problem. *Methods of Operations research*, 51:271–276, 1984.

[LN87] Gilbert Laporte and Yves Nobert. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31:147–184, 1987.

[LND85] Gilbert Laporte, Yves Nobert, and Martin Desrochers. Optimal routing under capacity and distance restrictions. *Operations research*, 33:1058–1073, 1985.

[MCH94] A. Mingozzi, N. Christofides, and E. Hadjiconstantinou. An exact algorithm for the vehicle routing problem based on the set partitionning formulation. jun 1994.

[MJ76] R.H. Mole and S.R. Jamerson. A sequential route-building algorithm employing a generalised savings criterion. *Operationnal research Quaterly*, 27:503–511, 1976.

[MT90a] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementation*. John Wiley & Sons, 1990.

[MT90b] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[Nad90] D. Naddef. Handles and teeth in the symmetric traveling salesman polytope. In W. Cook and P.D. Seymour, editors, *Polyhedral Combinatorics*, volume 1 of *Discrete Mathematics and Theoritical Computer Science*, pages 61–74. DIMACS, 1990.

[NJR94] C.E. Noon, J.Mittenthal, and R.Pillai. A tssp+1 decomposition strategy for the vehicle routing problem. *European Journal of Operational Research*, 79, 1994.

[NNGW85] M.D. Nelson, K.E. Nygard, J.H. Griffin, and W.E.Shreve. Implementation techniques for the vehicle routing prolem. *Computers & Operations Research*, 12:273–283, 1985.

[NR91] Denis Naddef and Giovanni Rinaldi. The symetric traveling salesman polytope and its graphical relaxation: composition of valid inequalities. *Mathematical Programming*, 51:359–400, 1991.

[Orl76] C. Orloff. Route constrained fleet scheduling. *Transportation Science*, (10):149–168, 1976.

[Osm93] I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41, 1993.

[Pad75] M. Padberg. A note on zero-one programming. *Operation Research*, 23:833–837, 1975.

[Pas88] H. Passens. The savings algorithm for the vehicle routing problem. *European Journal of Operations Research*, 34:336–344, 1988.

[PF91] V.M. Pureza and P.M. Franca. Vehicle routing problems via tabu search metaheuristic. Technical Report CRT-747, Centre de recherche sur les transports, Montreal, 1991.

[Poc] Y. Pochet. Path-bin inequalities for the cvrp. (private communication).

[PR81] M.W. Padberg and M.R. Rao. The russian method and integer programming. Technical report, New York University, 1981.

[PR90] M. Padberg and G. Rinaldi. Facet identification for the symetric traveling salesman polytope. *Mathematical programming*, 47:219–257, 1990.

[PR91] Manfred Padberg and Giovanni Rinaldi. A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[PY84] C.H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *J. Comput. System Sci.*, 28:244–259, 1984.

[QW93] Maurice Queyranne and Yaoguang Wang. Hamiltonian path and symetric travelling salesman polytopes. *Mathematical Programming*, 58:89–110, 1993.

[Rin] G. Rinaldi. Facets and small polytopes of the k-tsp problem. (private communication).

[Tai93] É. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–674, 1993.

[Thi94] S. Thienel. Branch and cut session. Giens, France, 1994. Science Workshop on Large Combinatorial Problems.

[WH72] Anthony Wren and Alan Holliday. Computer scheduling of vehicles fron one or more depots to a number of delivery points. *Operational Research Quarterly*, 23(3):333–344, 1972.

[WMGS57] J.B. Johnson W. M. Garvin, H.W. Crandall and R.A. Spellman. Application of linera programming to oil industry. *Management Science*, 3:407–430, 1957.

[Yel70] P. Yellow. A computational modification of the savings method of vehicle scheduling. *Operational Research Quaterly*, 21:281–283, 1970.