Université de Montréal

Développement d'un algorithme de branch-and-price-and-cut pour le problème de conception de réseau avec coûts fixes et capacités

par
Mathieu Larose

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Décembre 2011

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Développement d'un algorithme de branch-and-price-and-cut pour le problème de conception de réseau avec coûts fixes et capacités

présenté par:

Mathieu Larose

a été évalué par un jury composé des personnes suivantes:

Fabian Bastin,	président-rapporteur
Bernard Gendron,	directeur de recherche
Teodore Gabriel Crainic,	membre du jury

Mémoire accepté le: 1er juin 2012

RÉSUMÉ

De nombreux problèmes en transport et en logistique peuvent être formulés comme des modèles de conception de réseau. Ils requièrent généralement de transporter des produits, des passagers ou encore des données dans un réseau afin de satisfaire une certaine demande tout en minimisant les coûts. Dans ce mémoire, nous nous intéressons au problème de conception de réseau avec coûts fixes et capacités. Ce problème consiste à ouvrir un sous-ensemble des liens dans un réseau afin de satisfaire la demande, tout en respectant les contraintes de capacités sur les liens. L'objectif est de minimiser les coûts fixes associés à l'ouverture des liens et les coûts de transport des produits.

Nous présentons une méthode exacte pour résoudre ce problème basée sur des techniques utilisées en programmation linéaire en nombres entiers. Notre méthode est une variante de l'algorithme de branch-and-bound, appelée branch-and-price-and-cut, dans laquelle nous exploitons à la fois la génération de colonnes et de coupes pour la résolution d'instances de grande taille, en particulier, celles ayant un grand nombre de produits.

En nous comparant à CPLEX, actuellement l'un des meilleurs logiciels d'optimisation mathématique, notre méthode est compétitive sur les instances de taille moyenne et supérieure sur les instances de grande taille ayant un grand nombre de produits, et ce, même si elle n'utilise qu'un seul type d'inégalités valides.

Mots clés: Problème de conception de réseau, branch-and-bound, génération de colonnes, génération de coupes

ABSTRACT

Many problems in transportation and logistics can be formulated as network design models. They usually require to transport commodities, passengers or data in a network to satisfy a certain demand while minimizing the costs. In this work, we focus on the multicommodity capacited fixed-charge network design problem which consists of opening a subset of the links in the network to satisfy the demand. Each link has a capacity and a fixed cost that is paid if it is opened. The objective is to minimize the fixed costs of the opened links and the transportation costs of the commodities.

We present an exact method to solve this problem based on mixed integer programming techniques. Our method is a specialization of the branch-and-bound algorithm, called branch-and-price-and-cut, in which we use column generation and cutting-plane method to solve large-scale instances.

We compare our method with CPLEX, currently one of the best solver. Numerical results show that our method is competitive on medium-scale instances and better on large-scale instances.

Keywords: Multicommodity network design problem, branch-and-bound, column generation, cutting-plane method.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1

INTRODUCTION

De nombreux problèmes en transport et en logistique peuvent être formulés comme des modèles de conception de réseau. Ils requièrent généralement de transporter des produits, des passagers ou encore des données dans un réseau afin de satisfaire une certaine demande tout en minimisant les coûts de conception et de transport. Bien que les problèmes de conception de réseaux soient simples à comprendre et à formuler, ils sont difficiles à résoudre puisque les décisions relatives à la conception du réseau (par exemple, la construction d'une route, l'ouverture d'un vol entre deux villes ou l'installation d'un câble réseau) correspondent à des problèmes combinatoires qui sont fondamentalement difficiles à résoudre. Il faut ajouter à cela l'interdépendance entre les décisions de conception et d'opération qui les rend davantage difficiles à résoudre. En effet, en général, plus les coûts de conception du réseau sont élevés, moins les coûts de transport le sont, et vice-versa. De plus, la présence de contraintes de capacités, de topologie ou de fiabilité sur le réseau rend ces problèmes encore plus complexes.

Dans ce mémoire, nous nous intéressons au problème de conception de réseau avec coûts fixes et capacités (MCND). Les décisions de conception du réseau consistent à sélectionner un sous-ensemble des liens dans le réseau afin de satisfaire la demande tout en respectant les contraintes de capacités sur les liens. L'objectif est de minimiser les coûts de conception associés à l'ouverture des liens et des installations et les coûts de transport des produits.

Plusieurs problèmes étudiés dans la littérature sont des cas particuliers du problème de conception de réseau avec coûts fixes et capacités. Magnanti et al. [31] en dressent une liste. On y retrouve des problèmes pour lesquels il existe des algorithmes de résolution ayant une complexité en temps polynomiale tels que le problème du plus court chemin et le problème de l'arbre partiel minimum. On y retrouve également des problèmes plus difficiles pour lesquels nous ne connaissons pas d'algorithme de résolution en temps polynomial pour les résoudre tel que le problème du voyageur de commerce.

Bien que le problème de conception de réseau avec coûts fixes et capacités puisse être résolu par des techniques utilisées en programmation linéaire en nombres entiers, des solutions optimales ne peuvent être identitiées que pour des instances de petite taille ayant peu de produits. Nous présentons dans ce mémoire une méthode exacte qui une variante de l'algorithme de branch-and-bound, appelée branch-and-price-and-cut, dans laquelle nous exploitons à la fois la génération de colonnes et de coupes. L'avantage de notre méthode, comparativement aux autres méthodes exactes, est qu'elle permet de considérer un nombre restreint de variables. Elle peut alors résoudre des instances de grande taille, en particulier celles ayant un très grand nombre de produits, beaucoup plus rapidement que les autres méthodes.

Nous débutons ce mémoire en présentant au chapitre 2 les notions élémentaires en programmation linéaire en nombres entiers. Ce chapitre permet de mieux situer la revue de la littérature et la description du problème qui font l'objet du chapitre 3. Notre méthode est décrite et analysée au chapitre 4. Enfin, nous présentons et analysons nos résultats au chapitre 5.

CHAPITRE 2

PROGRAMMATION LINÉAIRE EN NOMBRES ENTIERS

Nous définissons dans ce chapitre ce qu'est un problème de programmation linéaire en nombres entiers. Puis, nous présentons une façon de le relaxer et de quantifier la différence entre les valeurs optimales du problème original et de sa relaxation. Nous décrivons ensuite l'algorithme de branch-and-bound. Ce chapitre est inspiré des références suivantes : Achterberg [1], Achterberg et al. [2], Atamtürk et Savelsberg [4], Desrosiers et Lübbecke [17] et Gondzio et al. [23].

2.1 Définitions

Problème de programmation linéaire en nombres entiers. Soit $A \in \mathbb{R}^{m \times n}$ et $G \in \mathbb{R}^{m \times p}$ des matrices, et $c \in \mathbb{R}^n, d \in \mathbb{R}^p$ et $b \in \mathbb{R}^m$ des vecteurs. On peut alors formuler un problème de programmation linéaire en nombres entiers comme suit :

$$\min c^T x + d^T y \tag{2.1}$$

Sujet à

$$Ax + Gy \geq b \tag{2.2}$$

$$x \in \mathbb{R}^n_{\geq 0} \tag{2.3}$$

$$y \in \mathbb{N}^p \tag{2.4}$$

$x = (x_1, \ldots, x_n)$ sont les variables continues et $y = (y_1, \ldots, y_p)$ sont les variables entières. On définit alors une solution optimale du problème comme suit :

$$(x^*, y^*) = \underset{(x,y) \in R^n_{\geq 0} \times N^p}{\arg\min} \left\{ c^T x + d^T y \mid Ax + Gy \leq b \right\} \tag{2.5}$$

et sa valeur optimale

$$z^* = c^T x^* + d^T y^* \tag{2.6}$$

Relaxation linéaire. La relaxation linéaire pour un problème de programmation linéaire en nombres entiers correspond à relaxer les contraintes d'intégralité sur les variables entières. La relaxation linéaire s'écrit donc ainsi :

$$\begin{aligned} &\min c^T x + d^T y \\ &\text{Sujet à} \\ &Ax + Gy \geq b \\ &x \in \mathbb{R}^n_{\geq 0} \\ &y \in \mathbb{R}^p_{\geq 0} \end{aligned} \tag{2.7}$$

Nous dénotons par $(\tilde{x}, \tilde{y})$ la solution optimale de la relaxation linéaire et par $\tilde{z}$ sa valeur optimale.

Écart d'intégralité. Puisque la relaxation linéaire élimine les contraintes d'intégralité sur les variables entières et qu'elles deviennent donc des variables continues, la valeur optimale de la relaxation linéaire est inférieure ou égale à celle du problème original. On définit donc l'écart d'intégralité comme étant l'écart relatif entre la valeur optimale du problème original et la valeur optimale de la relaxation linéaire, soit

$$\frac{z^* - \tilde{z}}{z^*} \tag{2.8}$$

2.2 Branch-and-Bound

L'algorithme de branch-and-bound est une méthode qui permet d'énumérer implicitement l'ensemble des solutions réalisables d'un problème de programmation linéaire en nombres entiers. Il suit le principe de « diviser pour régner » en décomposant récursivement un problème en plusieurs sous-problèmes plus simples à résoudre. La solution

optimale du problème est retrouvée en prenant la solution du meilleur sous-problème, c'est-à-dire le sous-problème qui permet d'identifier la solution réalisable ayant la plus petite borne.

L'algorithme génère et parcourt un arbre de recherche où chaque noeud de l'arbre de recherche correspond à un sous-domaine du domaine réalisable du problème original. La racine de l'arbre de recherche correspond au domaine réalisable du problème. La méthode est résumée à la figure 2.1.

À chaque itération, un noeud Q est sélectionné (lignes 4-5) parmi les feuilles de l'arbre de recherche non évaluées, un noeud non évalué étant un noeud dont la relaxation linéaire n'a pas été résolue. Une fois le noeud choisi, sa relaxation linéaire $\tilde{Q}$ est évaluée (lignes 6-7). La résolution de la relaxation linéaire fournit une borne inférieure sur le problème défini dans le noeud Q et permet d'arriver à l'une des trois conclusions suivantes :

1. Le noeud Q est élagué (ligne 8) s'il vérifie l'une des conditions suivantes :

   (a) Le problème relaxé $\tilde{Q}$ est non réalisable.

   (b) La solution optimale du problème relaxé $\tilde{Q}$ est supérieure ou égale à la meilleure solution réalisable trouvée jusqu'à maintenant par la méthode (solution candidate).

   Si $\tilde{Q}$ n'est pas réalisable, alors il n'existe aucune solution réalisable au problème original dans cette région de l'arbre. Nous pouvons donc éliminer cette région.

   Si la valeur optimale de $\tilde{Q}$ est supérieure ou égale à la valeur de la solution candidate, alors il n'existe aucune solution réalisable pour le problème original ayant une valeur optimale inférieure à la solution candidate. Ainsi, nous pouvons éliminer cette région.

2. La solution optimale de $\tilde{Q}$ devient la nouvelle solution candidate (ligne 9), si elle satisfait les deux conditions suivantes :

   (a) Elle est réalisable pour le problème original.

Entrée: Problème de minimisation R
Sortie: x^* la solution optimale et z^* la valeur optimale (si $z^* = \infty$, alors R n'a pas de solution réalisable)

Initialisation
1: $L \leftarrow \{R\}$
2: $z^* = \infty$

Critère d'arrêt
3: **Si** $L = \emptyset$ **alors** x^* est la solution optimale et z^* la valeur optimale. **Arrêter**

Sélection du noeud
4: Choisir $Q \in L$
5: $L \leftarrow L \setminus \{Q\}$

Évaluation
6: Résoudre la relaxation $\tilde{Q}$ de Q
7: **Si** $\tilde{Q}$ est non réalisable **alors** $\tilde{z} \leftarrow \infty$ **sinon** poser $\tilde{x}$ la solution optimale de $\tilde{Q}$ et $\tilde{z}$ sa valeur optimale

Élagage
8: **Si** $\tilde{z} \geq z^*$ **alors** aller à la ligne 3

Vérification
9: **Si** $\tilde{x}$ est une solution réalisable pour R **alors** $x^* \leftarrow \tilde{x}, z^* \leftarrow \tilde{z}$ et aller à la ligne 3

Branchement
10: Séparer Q en des sous-problèmes tel que $F(Q) = F(Q_1) \cup \ldots \cup F(Q_k)$
11: $L \leftarrow L \cup \{Q_1, \ldots, Q_k\}$
12: Aller à la ligne 3

Figure 2.1 – Algorithme de branch-and-bound

(b) Sa valeur optimale est inférieure à celle de la solution candidate.

3. Le noeud Q est divisé en plusieurs sous-problèmes $Q_1, \ldots, Q_k$, si la solution optimale de $\tilde{Q}$ satisfait les deux conditions suivantes :

   (a) Elle n'est pas réalisable pour Q, c'est-à-dire qu'au moins une variable entière prend une valeur fractionnaire.

   (b) Sa valeur optimale est inférieure à celle de la solution candidate.

   Étant donné qu'une solution optimale peut se trouver dans cette région de l'arbre, nous poursuivons donc la recherche dans cette région de l'arbre en insérant les sous-problèmes $Q_1, \ldots, Q_k$ dans l'arbre de recherche.

L'algorithme se termine lorsque tous les noeuds de l'arbre de recherche ont été évalués (ligne 3).

Durant l'exécution de l'algorithme, on maintient une borne inférieure et une borne supérieure sur z^*, la valeur optimale du problème sont conservées. On associe à chaque noeud non évalué une borne inférieure, qui est initialisée à la valeur optimale du problème relaxé de son parent, sauf pour le noeud racine, pour lequel elle est initialisée à une valeur arbitrairement petite. La borne inférieure sur z^* est égale à celle du noeud ayant la plus petite borne inférieure parmi l'ensemble des noeuds non évalués. La borne supérieure sur z^* est égale à la valeur optimale de la solution candidate.

L'algorithme se termine lorsque l'écart entre la borne inférieure et la borne supérieure sur z^* est nul. La borne inférieure augmente ou reste égale lorsque le meilleur noeud non évalué est évalué et la borne supérieure diminue lorsqu'une nouvelle solution candidate est trouvée. L'élagage (ligne 8) permet d'éviter une énumération explicite de toutes les solutions du problème en éliminant les régions de l'arbre de recherche ne pouvant contenir de solution optimale.

L'implantation d'un algorithme de branch-and-bound pour un problème nécessite de définir trois composantes : l'évaluation d'un noeud, la règle de branchement et le parcours de l'arbre de recherche. Nous résumons dans les prochaines sous-sections quelques stratégies utilisées dans la littérature.

2.2.1 Évaluation d'un noeud

L'évaluation d'un noeud consiste à résoudre la relaxation du sous-domaine associé à celui-ci, ce qui fournira une borne inférieure sur la valeur optimale du problème correspondant à ce sous-domaine.

L'approche la plus courante pour les algorithmes de branch-and-bound est de résoudre la relaxation linéaire du problème, c'est-à-dire de relaxer les contraintes d'intégralité sur les variables entières. On peut alors utiliser la méthode du simplexe pour résoudre la relaxation linéaire. Il est préférable d'utiliser la méthode duale du simplexe pour résoudre le nouveau sous-problème, puisqu'elle permet une réoptimisation rapide en partant de la base optimale du parent, qui demeure toujours réalisable pour le nouveau sous-problème. Une autre approche utilisée est de résoudre la relaxation lagrangienne qui consiste à relaxer certaines contraintes et à pénaliser leur violation en insérant un terme dans la fonction objectif.

Nous décrivons dans les deux prochaines sous-sections deux techniques permettant d'améliorer les performances de l'évaluation d'un noeud.

2.2.1.1 Génération de colonnes

La génération de colonnes est une méthode utilisée pour résoudre un problème de programmation linéaire ayant un très grand nombre de variables. Le principe de la méthode est de résoudre le problème de programmation linéaire en considérant un sous-ensemble des variables du problème et donc de conclure à l'optimalité sans énumérer explicitement toutes les variables du problème. Nous présentons la méthode en l'appliquant aux variables $x \in \mathbb{R}^n$ de la formulation (2.7), qui est alors considéré comme le problème maître. C'est ce problème que l'on veut résoudre en faisant l'hypothèse qu'il contient un très grand nombre de variables x.

La méthode de génération de colonnes propose de résoudre le problème maître en résolvant à chaque itération un problème maître restreint contenant un sous-ensemble des variables x. Étant donné J, l'ensemble des indices des variables x, on peut définir le problème maître restreint ainsi :

$$
\begin{aligned}
&\min c^T x + d^T y \\
&\text{Sujet à} \\
&\quad Ax + Gy \geq b \\
&\quad x_j \geq 0,\ j \in J' \\
&\quad x_j = 0,\ j \in J \setminus J' \\
&\quad y \in \mathbb{R}^p_{\geq 0}
\end{aligned}
\tag{2.9}
$$

Seules les variables $x_j,\ j \in J' \subseteq J$ peuvent prendre une valeur positive dans le problème (2.9). Les autres variables $x_j,\quad j \in J \setminus J'$ peuvent donc être considérées comme des variables hors-base. Il serait avantageux de permettre à l'une de ces variables de prendre une valeur positive seulement si son coût réduit est négatif. Autrement, une solution optimale pour le problème maître restreint est nécessairement optimale pour le problème maître.

Ainsi, une fois le problème maître restreint résolu, la méthode de génération de colonnes doit déterminer si la solution optimale pour le problème maître restreint l'est aussi pour le problème maître. Le cas échéant, le problème maître est résolu. Sinon, la méthode doit déterminer une variable à ajouter au problème maître restreint qui pourrait améliorer sa solution. C'est la fonction du sous-problème de déterminer si toutes les variables x_j telles que $j \in J \setminus J'$ ont un coût réduit non négatif, ou sinon, de déterminer s'il existe une variable ayant un coût réduit négatif.

Si au moins une variable $x_j,\ j \in J \setminus J'$ a un coût réduit négatif, alors cette variable peut améliorer la solution optimale du problème maître restreint et peut être ajoutée à celui-ci. Si toutes les valeurs ont un coût réduit non négatif, alors la solution du problème maître restreint est aussi optimale pour le problème maître.

La méthode de génération de colonnes est résumée à la figure 2.2. Le problème maître restreint est résolu (ligne 1) et le coût réduit des variables $x_j, j \in J \setminus J'$ est calculé en résolvant le sous-problème (ligne 2). Puis, si le coût réduit de toutes les variables $x_j, j \in J \setminus J'$ est non négatif (ligne 3), l'algorithme termine (ligne 4) et la solution

1: Résoudre le problème maître restreint
2: Calculer le coût réduit des variables $x_j, j \in J \setminus J'$ en résolvant le sous-problème
3: **Si** toutes les variables $x_j, j \in J \setminus J'$ ont un coût réduit non négatif **Alors**
4: **Arrêter**
5: **Sinon**
6: Ajouter les variables $x_j, j \in J \setminus J'$ ayant un coût réduit négatif dans le problème maître restreint
7: Aller à l'étape
8: **Fin Si**

Figure 2.2 – Génération de colonnes

optimale du problème maître restreint est optimale pour le problème maître. Sinon, les variables ayant un coût réduit négatif sont ajoutées au problème maître restreint et celui-ci est résolu à nouveau (lignes 6 et 7).

La méthode de génération de colonnes est assurée de converger en un nombre fini d'itérations si le nombre de variables x est fini et si on ajoute au moins une variable ayant un coût réduit négatif à chaque itération. On pourrait par exemple ajouter la variable ayant le plus petit coût réduit. En pratique, il est toutefois préférable d'ajouter plusieurs variables (ayant un coût réduit négatif) à chaque itération, afin de réduire le temps d'exécution de la méthode.

2.2.1.2 Génération de coupes

Étant donné que la performance d'un algorithme de branch-and-bound dépend en grande partie de la qualité de la borne inférieure fournie par la relaxation linéaire, il est naturel d'ajouter des contraintes au modèle afin de réduire l'écart d'intégralité. Il en résulte alors un problème ayant un très grand nombre de contraintes, donc plus difficile à résoudre et souvent propice à la dégénérescence. On peut alors utiliser la génération de coupes pour résoudre un problème de programmation linéaire en nombres entiers ayant un très grand nombre de contraintes. L'idée de la méthode est de résoudre le problème en considérant un sous-ensemble des contraintes, puis d'ajouter à chaque itération des nouvelles contraintes violées par la solution courante et de résoudre le problème à nouveau.

Une coupe est une contrainte (ou inégalité valide) qui est satisfaite par toutes les solutions réalisables (entières), mais qui ne l'est pas forcément par une une solution (fractionnaire) de la relaxation linéaire. Ainsi, en ajoutant une coupe à la formulation, on réduit le domaine réalisable de la relaxation linéaire $\tilde{Q}$ sans toutefois modifier celui de Q.

Formellement, une inégalité

$$a^T x + g^T y \leq b \tag{2.10}$$

où $a \in \mathbb{R}^n, g \in \mathbb{R}^p$ et $b \in \mathbb{R}$ est une inégalité valide si

$$a^T \tilde{x} + g^T \tilde{y} \leq b, \ \forall (\tilde{x}, \tilde{y}) \in S \tag{2.11}$$

où S est l'ensemble des solutions réalisables du problème linéaire en nombres entiers.

Le problème de séparation consiste à trouver une inégalité valide qui est violée par la solution optimale actuelle de $\tilde{Q}$. En ajoutant cette inégalité valide, puis en résolvant à nouveau $\tilde{Q}$, on obtient une nouvelle solution optimale qui a une valeur supérieure à la précédente. La borne de Q est alors augmentée et permet possiblement d'élaguer le noeud Q si elle est supérieure ou égale à la borne supérieure (valeur optimale de la solution candidate).

La méthode de génération de coupes est décrite à la figure 2.3. Une itération de la méthode consiste à résoudre dans un premier temps le problème de séparation (ligne 2) pour identifier une ou plusieurs inégalités valides violées par la solution optimale $\tilde{x}$ de $\tilde{Q}$. Elle ajoute ensuite les inégalités valides violées à $\tilde{Q}$ (ligne 3), puis elle résout à nouveau $\tilde{Q}$ (ligne 4). Le critère d'arrêt diffère d'une implantation à l'autre, mais il consiste généralement à arrêter s'il n'y a plus de coupes valides violées ou encore si la valeur de la solution $\tilde{x}$ est supérieure ou égale à la borne supérieure. On peut également arrêter la génération de coupes si l'amélioration de la borne est minime.

L'algorithme de branch-and-cut est une généralisation de l'algorithme de branch-and-bound dans la mesure où la génération de coupes est appliquée à chaque noeud.

Entrée: $\tilde{Q}$ et sa solution optimale $\tilde{x}$

1: **Répéter**
2: Résoudre le problème de séparation et identifier les inégalités valides
3: Ajouter les inégalités valides violées à $\tilde{Q}$
4: Résoudre $\tilde{Q}$ et poser $\tilde{x}$ comme étant sa nouvelle solution optimale
5: **Jusqu'à** Critère d'arrêt

Figure 2.3 – Génération de coupes

2.2.1.3 Branch-and-Price-and-Cut

Il est également possible de combiner la génération de colonnes avec la génération de coupes à chaque noeud lors de l'algorithme de branch-and-bound. On appelle alors cette méthode l'algorithme de branch-and-price-and-cut. L'évaluation d'un noeud (figure 2.1, ligne 6) dans une telle méthode correspond alors à la figure 2.4.

2.2.2 Règles de branchement

Les règles de branchement servent à diviser le domaine réalisable du noeud parent Q en imposant de nouvelles contraintes aux sous-problèmes générés $Q_1, Q_2, \ldots, Q_k$. Elles servent à diviser l'espace de recherche en séparant le domaine réalisable du noeud Q de

Entrée: La relaxation linéaire $\tilde{Q}$
Sortie: $\tilde{x}$ la solution optimale de $\tilde{Q}$ et $\tilde{z}$ sa valeur optimale (si $\tilde{z} = \infty$, alors $\tilde{Q}$ n'a pas de solution réalisable)

1: **Répéter**
2: Résoudre $\tilde{Q}$
3: Génération de colonnes
4: Résoudre $\tilde{Q}$
5: Génération de coupes
6: **Jusqu'à** Critère d'arrêt

Figure 2.4 – Évaluation d'un noeud

telle sorte que la solution optimale fractionnaire soit éliminée de l'espace de recherche :

$$s(\tilde{Q}) \notin \bigcup_{i=1}^{k} F(\tilde{Q}_i) \tag{2.12}$$

où $s(\tilde{Q})$ est la solution optimale fractionnaire précédemment trouvée. De plus, les règles de branchement ne doivent pas éliminer de solutions entières du domaine réalisable :

$$F(Q) = \bigcup_{i=1}^{k} F(Q_i) \tag{2.13}$$

Il est nécessaire que les conditions (2.12) et (2.13) soient satisfaites afin d'assurer la convergence de l'algorithme de branch-and-bound. D'une part, si la condition (2.12) n'est pas respectée, alors l'algorithme pourrait découvrir la même solution optimale fractionnaire à l'infini dans cette région de l'arbre. D'autre part, si la condition (2.13) est violée, alors on pourrait éliminer à tort une solution optimale du domaine réalisable du noeud racine R.

Bien que cela ne soit pas nécessaire pour assurer la convergence de l'algorithme de branch-and-bound, il est préférable que le domaine réalisable des sous-problèmes $Q_1, Q_2, \ldots, Q_k$ soit une partition du domaine réalisable de Q :

$$F(Q_i) \cap F(Q_j) = \emptyset \quad \forall i, j = 1, \ldots, k \text{ tel que } i \neq j \tag{2.14}$$

Cette condition permet d'éviter de rechercher plus d'une fois dans une même partie du domaine réalisable et donc de ne pas faire de calculs inutiles.

Par exemple, si la variable entière y_i a une valeur fractionnaire, disons 2.4, dans la solution optimale de $\tilde{Q}$, on peut alors introduire les contraintes $y_i \leq \lfloor 2.4 \rfloor = 2$ et $y_i \geq \lceil 2.4 \rceil = 3$ dans les sous-problèmes Q_1 et Q_2, respectivement. La solution fractionnaire où $y_i = 2.4$ est ainsi éliminée du domaine réalisable des deux nouveaux sous-problèmes, sans pour autant éliminer de solutions entières.

La règle de branchement utilisée au cours de l'algorithme de branch-and-bound a une grande influence sur le nombre de noeuds générés dans l'arbre de recherche. Une

bonne règle de branchement génèrera moins de noeuds au total dans l'arbre de recherche qu'une règle de branchement plus naïve.

Il existe des dizaines de règles de branchement dans la littérature. Certaines d'entre elles sont génériques et s'appliquent à tout problème de programmation linéaire en nombres entiers, alors que d'autres exploitent une structure spéciale du problème. Puisque nous utilisons exclusivement des variables entières binaires dans notre modèle (voir chapitre 3), nous nous limitons aux règles de branchement qui impliquent uniquement des variables binaires. De plus, nous considérons les règles de branchement basées sur la résolution de la relaxation linéaire à chaque noeud de l'arbre de recherche, puisque ce sont celles que nous utilisons dans notre méthode (voir chapitre 4).

Une règle de branchement est un algorithme qui détermine quelles sont les contraintes à ajouter pour partitionner le domaine réalisable. Étant donné que nous nous intéressons seulement aux règles de branchement qui s'appliquent à des variables binaires et qui divisent le domaine réalisable en deux parties, les contraintes ajoutées seront du type $y_i = 0$ et $y_i = 1$, pour une variable y_i sélectionnée.

Bien que chaque règle de branchement soit unique, elles réalisent toutes le même objectif : évaluer les variables candidates, soit les variables binaires qui ont une valeur fractionnaire dans la solution optimale de la relaxation linéaire, et sélectionner la « meilleure » variable afin de brancher sur celle-ci. La « meilleure » variable est généralement celle qui permettra de réduire le nombre de noeuds générés dans l'arbre de recherche.

On peut alors interpréter une règle de branchement comme étant simplement une fonction qui assigne un score à chaque variable candidate et ainsi définir l'algorithme de sélection générique d'une variable (figure 2.5). Les règles de branchement doivent donc implanter la fonction score$(\cdot)$ appelée à la ligne 2 de l'algorithme de sélection générique d'une variable.

Nous décrivons dans les prochaines sous-sections comment les règles implantent la fonction score$(\cdot)$.

Entrée: Le problème $\tilde{Q}$ ainsi qu'une solution optimale fractionnaire $(\tilde{x}, \tilde{y})$
Sortie: L'indice d'une variable binaire ayant une valeur fractionnaire dans la solution $\tilde{y}$

1: Soit I l'ensemble des indices des variables candidates
2: $\forall i \in I$, calculer score$(\tilde{y}_i) \in \mathbb{R}$
3: $i^* \leftarrow \arg\max_{i \in I}\{\text{score}(\tilde{y}_i)\}$
4: **Retourner** i^*

Figure 2.5 – Sélection générique d'une variable

2.2.2.1 Variable aléatoire

Cette règle de branchement choisit aléatoirement une variable parmi l'ensemble des variables candidates. Cette règle n'est jamais utilisée en pratique ; elle sert comme base de comparaison pour évaluer les autres règles de branchement. Le score associé à une variable est défini comme suit :

$$\text{score}(\tilde{y}_i) = U(0, 1) \tag{2.15}$$

où $U(\cdot)$ est la loi de probabilité uniforme.

2.2.2.2 Variable la plus fractionnaire

Cette règle de branchement sélectionne la variable qui est la plus fractionnaire. Le principe de cette règle est qu'elle sélectionne une variable pour laquelle il n'est pas certain si elle doit être fixée à 0 ou à 1. Achterberg et al. [2] ont démontré que la performance de cette règle est généralement pire que la règle de branchement aléatoire.

$$\text{score}(\tilde{y}_i) = 1 - \min\{\tilde{y}_i, 1 - \tilde{y}_i\} \tag{2.16}$$

2.2.2.3 Pseudo coût

Le pseudo coût d'une variable est une estimation de l'accroissement de la borne par unité lorsqu'on fixe la variable à 0 ou à 1. Il est calculé en utilisant l'historique des branchements effectués sur cette variable dans l'arbre de recherche.

Pour calculer le pseudo coût, on définit Δ_i^0 et Δ_i^1 comme étant l'accroissement de la borne lorsqu'on branche sur cette variable à 0 et à 1. On définit ainsi ρ_i^0 et ρ_i^1 comme étant l'accroissement de la borne par unité :

$$\rho_i^0 = \frac{\Delta_i^0}{\tilde{y}_i} \qquad \rho_i^1 = \frac{\Delta_i^1}{1 - \tilde{y}_i} \tag{2.17}$$

On définit ensuite le pseudo coût $\bar{\rho}_i^0$ et $\bar{\rho}_i^1$ comme étant la moyenne des accroissements de la borne par unité :

$$\bar{\rho}_i^0 = \frac{\sigma_i^0}{\eta_i^0} \quad \text{et} \quad \bar{\rho}_i^1 = \frac{\sigma_i^1}{\eta_i^1} \tag{2.18}$$

où σ_i^0 et σ_i^1 représentent la somme des accroissements par unité lorsqu'on branche à 0 et à 1 sur la variable y_i et η_i^0 et η_i^1 représentent le nombre de branchements effectués à 0 et à 1 sur la variable y_i.

Le score associé à une variable est alors une combinaison linéaire de l'estimation de l'accroissement de la borne :

$$\text{score}(\tilde{y}_i) = \lambda \cdot \min\left\{\tilde{y}_i\bar{\rho}_i^0,\ (1-\tilde{y}_i)\bar{\rho}_i^1\right\} + (1-\lambda) \cdot \max\left\{\tilde{y}_i\bar{\rho}_i^0,\ (1-\tilde{y}_i)\bar{\rho}_i^1\right\} \tag{2.19}$$

où $\lambda \in [0, 1]$ est un paramètre. On choisit généralement une valeur de λ près de 1 afin de conserver l'arbre de recherche équilibré.

Nous pouvons observer que cette méthode est strictement empirique et utilise peu d'information locale pour prendre une décision. De plus, au début de l'algorithme $\sigma_i^0 = \sigma_i^1 = \eta_i^0 = \eta_i^1 = 0,\ \forall i \in I$. Le pseudo coût d'une variable $i \in I$ est non défini pour le branchement à 0, si $\eta_i^0 = 0$. Le pseudo coût des variables dont le branchement à 0 n'est pas défini est fixé à la moyenne des pseudo coûts de toutes les variables dont le pseudo coût pour le branchement à 0 est initialisé. Si aucun pseudo coût pour le branchement à 0 n'est défini, alors il est fixé à 0. Le même raisonnement s'applique pour le pseudo coût pour le branchement à 1. Malgré ces inconvénients, cette règle de branchement est rapide à calculer et prend en moyenne de meilleures décisions que la règle « variable aléatoire » [2].

2.2.2.4 Branchement fort

La règle de branchement fort (« *Strong branching* ») évalue le score de chaque variable en résolvant deux sous-problèmes de programmation linéaire. L'un de ces sous-problèmes correspond à la relaxation linéaire dans lequel la variable est fixée à 0, alors que l'autre correspond à la relaxation linéaire dans lequel la variable est fixée à 1. Les valeurs optimales des deux sous-problèmes de programmation linéaire donnent une estimation de l'accroissement de la borne lorsqu'on branche sur cette variable. Le score associé à une variable est alors une combinaison linéaire de l'accroissement de la borne inférieure.

$$\text{score}(\tilde{y}_i) = \lambda \cdot \min\left\{\Delta_i^0, \Delta_i^1\right\} + (1-\lambda) \cdot \max\left\{\Delta_i^0, \Delta_i^1\right\} \tag{2.20}$$

où $\lambda \in [0,1]$ est un paramètre et Δ_i^k correspond à la différence entre la valeur optimale du sous-problème associé à la variable y_i lorsqu'elle est fixée à k et la valeur optimale de la relaxation linéaire.

Il est à noter que le branchement fort est très long à calculer, mais prend une décision en analysant l'information locale. En pratique, on ne résout pas à l'optimum les relaxations linéaires, mais on effectue plutôt un nombre limite d'itérations du simplexe dual.

2.2.2.5 Pseudo coût avec une initialisation par branchement fort

Un des inconvénients d'utiliser les pseudo coûts est qu'ils ne sont pas initialisés tant qu'on n'a pas branché sur la variable. On peut donc utiliser le branchement fort pour initialiser le pseudo coût d'une variable et ensuite utiliser le pseudo coût pour accélérer les calculs.

2.2.2.6 Branchement fiable

Le branchement fiable est une généralisation du branchement pseudo coût avec une initialisation par le branchement fort. Au lieu de seulement utiliser le branchement fort

sur les variables ayant un pseudo coût non initialisé, le branchement fort est appliqué sur les variables dont le pseudo coût n'est pas jugé fiable. Le pseudo coût de la variable y_i n'est pas jugé fiable si $\min\{\eta_i^0, \eta_i^1\} < \eta_{\text{fiable}}$, où η_{fiable} est un paramètre.

2.2.3 Parcours de l'arbre de recherche

Après le traitement d'un noeud, l'algorithme doit choisir un noeud à évaluer parmi les feuilles de l'arbre de recherche. Il existe deux approches fondamentales pour le parcours de l'arbre de recherche : la recherche en profondeur et la recherche meilleur d'abord. La recherche en profondeur choisit le noeud le plus en profondeur alors que la recherche meilleur d'abord choisit le noeud ayant la plus petite valeur optimale de la relaxation.

2.2.3.1 Recherche en profondeur

La recherche en profondeur choisit le noeud le plus en profondeur dans l'arbre de recherche. Cette stratégie descend donc d'un niveau dans l'arbre de recherche à chaque itération, jusqu'à ce que le noeud puisse être élagué ou qu'une solution réalisable ait été trouvée. Le cas échéant, la recherche en profondeur choisira un noeud du même niveau ou d'un niveau supérieur ; on dit alors qu'il revient sur ses pas (*backtracking*).

La recherche en profondeur permet de trouver rapidement des solutions réalisables et donc d'améliorer rapidement la borne supérieure, puisque les solutions réalisables sont généralement trouvées dans les niveaux inférieurs de l'arbre de recherche.

Une propriété intéressante de la recherche en profondeur est que le noeud sélectionné est similaire au noeud précédent. En particulier, la relaxation linéaire du noeud courant sera alors similaire à celle du noeud précédent et nécessitera souvent seulement un changement de borne sur une variable, ce qui est géré efficacement par la plupart des logiciels d'optimisation mathématique.

Un autre avantage de cette stratégie de recherche est qu'elle consomme peu de mémoire et garde en pire cas $O(bd)$ noeuds non élagués en mémoire, où d est le niveau du noeud courant et b le facteur de branchement, qui est généralement 2.

En l'absence d'une bonne borne supérieure, la taille de l'arbre de recherche peut exploser puisque la recherche en profondeur peut s'enfoncer dans une région de l'arbre de recherche qui aurait pu être élaguée si une meilleure borne supérieure avait été fournie.

2.2.3.2 Recherche meilleur d'abord

La recherche meilleur d'abord permet d'améliorer la borne inférieure en sélectionnant le noeud ayant la plus petite borne inférieure. Cette stratégie permet d'explorer en un nombre minimum de noeuds l'arbre de recherche (si la règle de branchement est fixée), puisque sans même connaître la valeur optimale du problème, elle garantit qu'elle explorera seulement les régions de l'arbre de recherche où la borne inférieure est plus petite que la valeur optimale. Elle utilise cependant beaucoup plus de mémoire que la recherche en profondeur puisqu'elle garde en pire cas $O(b^d)$ noeuds non élagués en mémoire et qu'elle ne trouvera généralement pas de solutions réalisables rapidement.

2.2.3.3 Recherche en profondeur avec redémarrage

Il existe également des stratégies hybrides entre la recherche meilleur d'abord et en profondeur telle que la recherche profondeur avec redémarrage, qui consiste à choisir le noeud ayant la meilleure borne inférieure (meilleur d'abord), puis d'appliquer une recherche en profondeur sur l'un des enfants. Cette stratégie a pour avantage d'améliorer la borne inférieure à chaque redémarrage tout en trouvant rapidement des solutions réalisables.

CHAPITRE 3

DESCRIPTION DU PROBLÈME ET REVUE DE LA LITTÉRATURE

Nous présentons dans ce chapitre un modèle général pour le problème de conception de réseau, ainsi que deux modèles pour le problème de conception de réseau avec coûts fixes et capacités. Nous présentons par la suite une revue de la littérature sur les approches pour résoudre des problèmes de conception de réseau, en particulier le problème de conception avec coûts fixes et capacités.

3.1 Conception de réseau

Étant donné un graphe orienté $G = (V, A)$, constitué d'un ensemble de sommets V et d'un ensemble d'arcs A disponibles pour la conception du réseau, on associe à ce graphe un ensemble de produits K et un ensemble d'installations T à ouvrir sur chaque arc. Pour chaque produit $k \in K$, une quantité connue $d^k > 0$ de ce produit est introduite dans le réseau à partir du sommet origine $O(k)$ et doit aboutir au sommet destination $D(k)$. Le problème général de conception de réseau consiste alors à ouvrir un sous-ensemble des installations sur les arcs de telle sorte que la conception du réseau satisfasse les demandes, respecte certaines contraintes et minimise les coûts. Il revient à chaque cas particulier du problème général de définir avec précision les contraintes et la fonction objectif à minimiser.

On peut définir une formulation pour ce problème général de conception de réseau contenant deux types de variables pour modéliser le flot des produits et l'ouverture des installations. La variable continue x_{ij}^k représente la quantité du flot du produit k sur l'arc (i, j) et la variable entière y_{ij}^t représente le nombre d'installations de type t installés sur l'arc (i, j).

En utilisant les deux types de variables précédents, on peut formuler le modèle suivant, qui s'inspire de ceux énoncés par Magnanti et al. [31] et Gendron et al. [5] :

$$\min \phi(y, x) \tag{3.1}$$

Sujet à

$$\sum_{j \in V_i^+} x_{ij}^k - \sum_{j \in V_i^-} x_{ji}^k = \begin{cases} d^k & \text{si } i = O(k) \\ -d^k & \text{si } i = D(k) \\ 0 & \text{sinon} \end{cases} \quad \forall i \in V, k \in K \tag{3.2}$$

$$\sum_{k \in K} x_{ij}^k \leq \sum_{t \in T} u_{ij}^t y_{ij}^t \quad \forall (i,j) \in A \tag{3.3}$$

$$(y, x) \in S \tag{3.4}$$

$$x_{ij}^k \geq 0 \quad k \in K, (i,j) \in A \tag{3.5}$$

$$y_{ij}^t \geq 0 \text{ et entiers} \quad t \in T, (i,j) \in A \tag{3.6}$$

Les contraintes de conservation de flots (3.2) indiquent, pour chaque sommet et chaque produit, que la somme du flot des arcs incidents vers l'intérieur doit être égale à la somme du flot des arcs incidents vers l'extérieur, sauf pour l'origine et la destination. Les contraintes (3.3) imposent un flot nul si aucune installation n'est ouverte sur l'arc et servent également de contraintes de capacités lorsque $\sum_{t \in T} u_{ij}^t < \sum_{k \in K} d^k$. Les contraintes additionnelles (3.4) permettent d'exprimer toute contrainte ne pouvant être définie par les deux types de contraintes précédents. On peut, par exemple, imposer un budget maximum sur les coûts de conception du réseau. Les contraintes de non négativité (3.5) empêchent un flot négatif sur les variables de flots x_{ij}^k et les contraintes (3.6) indiquent que les variables y_{ij}^t sont des variables entières non négatives.

On retrouve généralement deux types de coûts sur chaque arc (i,j) : $c_{ij}^k \geq 0$ représente le coût unitaire de transport pour le produit k sur l'arc (i,j) et $f_{ij}^t \geq 0$ représente le coût fixe pour l'ouverture de l'installation de type t sur l'arc (i,j) dans le réseau. Ainsi,

la fonction objectif (3.1) prend une forme linéaire :

$$\phi(y,x) = \sum_{(i,j)\in A}\sum_{k\in K} c_{ij}^k x_{ij} + \sum_{(i,j)\in A}\sum_{t\in T} f_{ij}^t y_{ij}^t \tag{3.7}$$

et la formulation précédente est un problème de programmation linéaire en nombres entiers.

3.1.1 Cas particuliers

Plusieurs problèmes classiques peuvent être considérés comme des cas particuliers de la formulation présentée à la sous-section précédente en utilisant la fonction objectif (3.7) et aucune contrainte additionnelle (3.4). Les deux exemples suivants utilisent au plus un seul type d'installation ; l'indice t peut donc être supprimé de la notation.

Le problème du plus court chemin d'un sommet vers tous les autres sommets peut être exprimé comme un problème de conception de réseau sans capacité où il n'y a aucun coût pour la conception du réseau ($f_{ij} = 0, \forall (i,j) \in A$) et en utilisant $|V| - 1$ produits de demande unitaire ayant tous la même source ($O(k) = s \in V, \forall k \in K$) et une destination différente pour chaque produit. Une solution optimale correspond à une solution du problème du plus court chemin d'un sommet vers tous les autres sommets.

Le problème du voyageur de commerce peut également être interprété comme un cas particulier d'un problème de conception de réseau. On définit un réseau $G = (V, A)$ tel que 1) la demande est complète, c'est-à-dire qu'on associe à chaque paire de sommets un produit k ayant une demande unitaire ; 2) tous les arcs ont le même coût fixe $f = f_{ij}, (i,j) \in A$, une valeur arbitrairement grande ; 3) le coût de transport sur un arc est le même pour tous les produits $c_{ij} = c_{ij}^k, (i,j) \in A, k \in K$. Puisque la demande est complète, une solution réalisable à ce problème possède au moins $|V|$ arcs. De plus, étant donné que le coût fixe, f, est très grand, une solution solution optimale contiendra au plus $|V|$ arcs. Ces $|V|$ arcs formeront alors un circuit hamiltonien. Finalement, puisque la somme des coûts fixes est constante et que le flot est le même sur chacun des arcs sélectionnés, une solution optimale pour ce problème consiste à trouver un circuit hamiltonien de coût minimum par rapport aux coûts c_{ij}, ce qui correspond à résoudre le

problème du voyageur de commerce avec les distances c_{ij} entre les sommets.

Le lecteur est invité à consulter la référence [31] pour une présentation d'autres cas particuliers du problème de conception de réseau.

3.2 Revue de la littérature en conception de réseau

Lorsque la fonction objectif est linéaire, la formulation présentée à la section 3.1 est un problème de programmation linéaire en nombres entiers. Toutes les techniques disponibles pour résoudre ce type de problème peuvent également servir à résoudre des problèmes de conception de réseau. On y retrouve, entre autres, des méthodes basées sur la relaxation lagrangienne, des méthodes d'ascension duale, des méthodes de coupes, des méthodes d'énumération implicite ou encore des méthodes heuristiques. Nous présentons dans cette section quelques méthodes permettant de mieux situer nos travaux. Le lecteur intéressé à une revue de littérature plus approfondie est invité à consulter les références suivantes : Magnanti et al. [31], Minoux [32], Gendron et al. [5] et Crainic [11].

Tout comme pour les problèmes de programmation linéaire en nombres entiers, une méthode basée sur une approche polyédrale pour résoudre des problèmes de conception de réseau consiste à représenter une partie de l'enveloppe convexe du domaine réalisable à l'aide d'inégalités valides. Si on réussit à représenter l'enveloppe convexe à l'aide d'inégalités valides, on peut alors utiliser un algorithme de programmation linéaire pour résoudre un problème de programmation linéaire en nombres entiers. En pratique, on ne réussit généralement qu'à identifier un sous-ensemble des inégalités qui définissent l'enveloppe convexe. Cela permet tout de même de développer des algorithmes de branch-and-bound plus efficaces, puisqu'on peut y intégrer une méthode de coupes réduisant l'écart d'intégralité lors de de l'évaluation des noeuds (voir section 2.2.1.2).

Les inégalités valides dérivées pour le problème général de conception de réseau avec capacités sont basées principalement sur la notion de coupures (« cutsets »), provenant de la théorie des graphes. Une coupure d'un graphe $G = (V, A)$ est un sous-ensemble non vide de sommets $S \subset V$ auquel on associe les arcs de la coupure $(S, \bar{S}) =$

$\{(i,j) \in A | i \in S, j \in \bar{S}\}$, où $\bar{S} = V \setminus S$. La capacité d'une coupure est définie ainsi : $\sum_{(i,j)\in(S,\bar{S})} \sum_{t\in T} u_j^t y_{ij}^t$. Le principe des inégalités valides est que la capacité totale de toute coupure doit suffire à la demande de tous les produits tels que le sommet origine et le sommet destination sont séparés par la coupure. Magnanti et al. [29, 30] et Bienstock et Günlük [7] ont utilisé ce type d'inégalités valides, ainsi que quelques inégalités valides additionnelles, pour résoudre des problèmes de conception de réseau avec capacité à l'aide de méthodes de coupes intégrées à un algorithme de branch-and-cut. Günlük [24] a par la suite intégré d'autres inégalités valides dans un algorithme de branch-and-cut pour un problème de conception de réseau avec capacités.

3.3 Conception de réseau avec coûts fixes et capacités

Le problème de conception de réseau avec coûts fixes et capacités est un cas particulier du problème présenté à la section 3.1 pour lequel : 1) la fonction objectif prend la forme (3.7) ; 2) les variables y_{ij}^t sont binaires, i.e., (3.4) prend la forme $y_{ij}^t \leq 1, (i,j) \in A, t \in T$; 3) il n'y a qu'un seul type d'installation, i.e., $|T| = 1$ et on peut donc éliminer l'indice t. La fonction objectif à minimiser comporte deux termes : des coûts linéaires de transport, correspondant au transit des flots sur les arcs, et des coûts fixes de conception, qui sont encourus dès qu'un arc est utilisé pour le transit des flots. On retrouve ainsi deux types de coûts sur chaque arc (i,j) : $c_{ij}^k \geq 0$ représente le coût unitaire de transport pour le produit k et $f_{ij} \geq 0$ représente le coût fixe. On impose ainsi une capacité $u_{ij} > 0$ sur chaque arc $(i,j) \in A$.

Le problème de conception de réseau avec coûts fixes et capacités peut être modélisé comme un problème de programmation linéaire en nombres entiers. Nous pouvons le formuler dans l'espace des arcs ou des chemins. Dans les deux types de formulations, on modélise la sélection ou non de l'arc (i,j) dans le réseau en introduisant la variable binaire y_{ij} qui est égale à 1 si l'arc (i,j) est sélectionné, et 0 sinon.

Nous présentons une formulation dans l'espace des arcs provenant de Magnanti et al. [31] et une formulation dans l'espace des chemins inspirée de Ahuja et al. [3].

3.3.1 Formulation dans l'espace des arcs

En plus d'utiliser les variables binaires y_{ij}, la formulation dans l'espace des arcs utilise des variables continues pour représenter le flot dans le réseau. On décompose les flots par produit et par arc. Ainsi, la variable continue x_{ij}^k représente le flot du produit k sur l'arc (i, j).

Le modèle se formule comme suit :

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.8}$$

Sujet à

$$\sum_{j \in V_i^+} x_{ij}^k - \sum_{j \in V_i^-} x_{ji}^k = \begin{cases} d^k & \text{si } i = O(k) \\ -d^k & \text{si } i = D(k) \\ 0 & \text{sinon} \end{cases} \quad \forall i \in V,\ \forall k \in K \tag{3.9}$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij} y_{ij} \quad \forall (i,j) \in A \tag{3.10}$$

$$x_{ij}^k \geq 0 \quad \forall (i,j) \in A,\ \forall k \in K \tag{3.11}$$

$$y_{ij} \in \{0, 1\} \quad \forall (i,j) \in A \tag{3.12}$$

où $V_i^+ = \{j \in V \mid (i,j) \in A\}$ et $V_i^- = \{j \in V \mid (j,i) \in A\}$.

La fonction objectif (3.8) comporte deux types de termes : la somme des coûts de transport et la somme des coûts de conception. Les contraintes de conservation de flots (3.9) indiquent que, pour chaque sommet et chaque produit, la somme du flot des arcs incidents vers l'intérieur doit être égale à la somme du flot des arcs incidents vers l'extérieur, sauf pour l'origine et la destination qui doit être, respectivement, positive et négative. Les contraintes de capacités sur les arcs (3.10) permettent d'assurer que le flot ne dépasse pas la capacité d'un arc lorsqu'il est sélectionné et imposent un flot nul sinon. Les contraintes de non négativité (3.11) empêchent un flot négatif sur les variables

de flots x_{ij}^k et les contraintes (3.12) indiquent que les variables y_{ij} sont des variables binaires.

3.3.2 Formulation dans l'espace des chemins

On peut également formuler le problème de conception de réseau avec coûts fixes et capacités dans l'espace des chemins. On décompose le flot par produit et par chemin. La variable x_p^k représente alors le flot du produit k sur le chemin p, où p est un chemin de $O(k)$ vers $D(k)$. $P(k)$ représente l'ensemble des chemins de $O(k)$ vers $D(k)$. On définit également δ_{ijp}^k comme suit :

$$\delta_{ijp}^k = \begin{cases} 1 & \text{si l'arc } (i,j) \text{ fait partie du chemin } p \in P(k) \\ 0 & \text{sinon} \end{cases} \tag{3.13}$$

Le modèle peut alors s'écrire ainsi :

$$\min \sum_{k \in K} \sum_{p \in P(k)} \sum_{(i,j) \in A} \delta_{ijp}^k c_{ij}^k x_p^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.14}$$

Sujet à

$$\sum_{p \in P(k)} x_p^k = d^k \qquad \forall k \in K \tag{3.15}$$

$$\sum_{k \in K} \sum_{p \in P(k)} \delta_{ijp}^k x_p^k \leq u_{ij} y_{ij} \qquad \forall (i,j) \in A \tag{3.16}$$

$$x_p^k \geq 0 \qquad \forall k \in K,\ \forall p \in P(k) \tag{3.17}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.18}$$

La structure de la formulation dans l'espace des chemins est semblable à celle du modèle dans l'espace des arcs. La fonction objectif (3.14) minimise les coûts de transport et de conception. Les contraintes (3.15) et (3.16) correspondent, respectivement, aux contraintes de conservation de flot et de capacités.

En pratique, il n'est pas réaliste d'insérer tous les chemins dans la formulation puisqu'il en existe un nombre exponentiel. Une approche basée sur la génération de colonnes (voir section 2.2.1.1) est donc souvent utilisée pour résoudre cette formulation.

3.4 Revue de la littérature pour le problème de conception de réseau avec coûts fixes et capacités

Plusieurs approches ont été utilisées pour résoudre le problème de conception de réseau avec coûts fixes et capacités. Nous présentons dans un premier temps les principales méthodes heuristiques pour le résoudre. Nous présentons ensuite des méthodes exactes qui ont été développées pour résoudre le MCND.

3.4.1 Méthodes heuristiques

L'une des heuristiques les plus simples développées pour un problème semblable au MCND a été proposée par Powell [33]. Sa méthode est basée sur le principe de la descente locale, c'est-à-dire qu'elle s'arrête au premier minimum local rencontré. Le voisinage considéré est l'ajout et la suppression d'un arc au réseau.

Crainic et al. [14] ont développé une métaheuristique basée sur la recherche tabou, l'algorithme du simplexe et la génération de colonnes. Le voisinage exploré par leur algorithme correspond à un pivot de l'algorithme du simplexe dans l'espace des chemins. Il correspond donc à faire dévier un produit à la fois en fermant un chemin pour le faire passer par un nouveau chemin. La génération de colonnes est utilisée afin d'ajouter de nouveaux chemins. Leur voisinage permet de considérer plus de solutions que le voisinage de type ajout/suppression, mais il demeure toutefois limité puisqu'il ne considère qu'un seul produit à la fois. Une version parallèle basée sur cette approche a été développée par Crainic et al. [13].

Ghamlouche et al. [21] ont ensuite développé une autre métaheuristique basée sur la recherche tabou. Leur algorithme choisit deux sommets dans le réseau et deux chemins les reliant, formant ainsi un cycle. Le flot d'un chemin est entièrement redirigé vers l'autre chemin. Les arcs sur le premier chemin ayant désormais un flot nul sont fermés

et les arcs sur le deuxième chemin sont forcés à être ouverts. Ils ont ensuite combiné ce voisinage à base de cycle dans un algorithme de « path-relinking » [22]. Une version parallèle basée sur la recherche tabou a été développée par Crainic et al. [15].

Les méthodes développées ont ensuite combiné des techniques de programmation linéaire en nombres entiers à des méthodes heuristiques. La méthode de Rodríguez-Martín et al. [34] est basée sur le branchement local (« *local branching* »), une méthode proposée par Fishetti et al. [18]. Basée sur le principe de la recherche locale, le branchement local explore l'espace de recherche en ajoutant des inégalités valides au modèle dans l'espace des arcs.

Tout comme Rodríguez-Martín et al. [34], Hewitt et al. [25] ont proposé une méthode basée sur la recherche locale où le voisinage est exploré en résolvant des sous-problèmes à l'aide de logiciels d'optimisation mathématique. Chouman et al. [8] ont développé une méthode hybride combinant une méthode de génération de coupes, un algorithme de branch-and-bound et la recherche tabou.

Finalement, Katayama et al. [27] ont combiné le « capacity scaling » et le branchement local. Ils ont d'ailleurs répertorié tous les résultats des méthodes heuristiques décrites précédemment sur des jeux d'instances utilisés dans la littérature[1] [2].

La conclusion qui se dégage de cette revue de la littérature sur les méthodes heuristiques est qu'il existe de bonnes heuristiques ; en particulier, l'écart d'optimalité moyen de la méthode proposée par Katayama et al. [27] sur l'ensemble des instances est de moins de 1%. Étant donné que les solutions trouvées par les méthodes heuristiques sont proches des solutions optimales, le but des méthodes exactes est généralement de prouver que les solutions trouvées par les méthodes heuristiques sont optimales, ou de les améliorer légèrement afin de trouver une solution optimale.

3.4.2 Méthodes exactes

Du côté des méthodes exactes, on retrouve principalement des méthodes se basant sur des techniques en programmation linéaire en nombres entiers telles que des méthodes

[1] `http://www.rku.ac.jp/~katayama/CND_GAP_C.html`
[2] `http://www.rku.ac.jp/~katayama/CND_GAP_R.html`

de coupes, la relaxation lagrangienne et la décomposition de Benders.

Holmberg et al. [26] ont développé un algorithme de branch-and-bound, dans lequel ils appliquent une méthode de sous-gradient au dual lagrangien provenant de la relaxation des contraintes de conservation de flot. Cet algorithme a par la suite été modifié par Kliewer et al. [28], qui y ont ajouté une méthode de génération de coupes. Crainic et al. [12] ont comparé les méthodes de sous-gradient et de faisceau appliquées à deux types de relaxation lagrangienne.

Costa et al. [10] ont appliqué la décomposition de Benders [6] sur la formulation dans l'espace des arcs. Le problème maître correspond à une assignation des variables de conception et le sous-problème correspond à un problème de multiflots. L'algorithme résout itérativement les deux problèmes jusqu'à ce que la borne inférieure, fournie en résolvant le problème maître, soit suffisamment près de la borne supérieure, qui est obtenue lorsque la solution du problème maître et du sous-problème correspondent à une solution réalisable. La décomposition de Benders appliquée au MCND fonctionne pour des instances de petite taille, mais n'est pas utilisable pour des instances de grande taille.

Finalement, Chouman et al. [9] ont étudié plusieurs familles d'inégalités valides pour le MCND en développant une méthode de coupes qui est utilisée afin d'améliorer la borne inférieure sur la valeur optimale. Ils ont comparé plusieurs types d'inégalités valides, notamment des inégalités basées sur les coupures (« cutsets »). Parmi ces types d'inégalités, les inégalités fortes (SI) sont particulièrement intéressantes. En effet, les résultats expérimentaux démontrent que les autres inégalités valides sont soit dominées par les SI, en terme de la qualité de la borne, soit qu'elle fournissent sensiblement la même borne, mais en un temps beaucoup plus long que les SI. De plus, les SI fournissent la meilleure borne sur des instances de grande taille, en particulier celles ayant un grand nombre de produits. Les inégalités fortes s'énoncent ainsi :

$$x_{ij}^k \leq d^k y_{ij}, (i,j) \in A, k \in K \tag{3.19}$$

Étant donné que les SI réduisent considérablement l'écart d'intégralité, on désigne par formulation forte la formulation dans l'espace des arcs à laquelle on a ajouté les SI, alors que la formulation faible est tout simplement la formulation dans l'espace des arcs sans ajout d'inégalités valides. Il est intéressant d'observer que toutes les méthodes exactes décrites précédemment utilisent la formulation forte pour la relaxation linéaire.

CHAPITRE 4

ALGORITHME DE BRANCH-AND-PRICE-AND-CUT

Nous décrivons dans cette section l'algorithme de branch-and-price-and-cut que nous avons développé pour le problème de conception de réseaux avec coûts fixes et capacités. Notre méthode est basée sur l'algorithme de branch-and-bound où l'évaluation du noeud consiste à résoudre la relaxation linéaire du problème en utilisant une méthode de générations de colonnes combinée à une méthode de génération de coupes.

4.1 Limites et hypothèses de notre méthode

La méthode de branch-and-bound sert à trouver une solution optimale. Elle doit donc trouver une solution réalisable en explorant l'arbre de recherche, puis prouver que celle-ci est la meilleure solution en visitant tous les noeuds de l'arbre de recherche.

Une façon de réduire le temps d'exécution est d'obtenir une borne supérieure afin d'élaguer le plus possible de noeuds dans l'arbre de recherche. Étant donné qu'il existe d'excellentes méthodes heuristiques pour le MCND (décrites à la section 3.4.1), nous avons développé notre méthode en supposant que la solution optimale, ou une excellente solution, est connue *a priori*. La tâche de notre méthode est donc de prouver que la solution donnée est optimale ou, dans certains cas, de l'améliorer légèrement en identifiant une solution optimale.

4.2 Évaluation d'un noeud

L'évaluation d'un noeud consiste à résoudre la relaxation linéaire de la formulation forte dans l'espace des arcs. Il a été observé que l'écart d'intégralité est très grand pour la formulation faible, ce qui a pour effet d'augmenter le temps d'exécution de l'algorithme, puisque très peu de noeuds sont élagués. Nous avons donc ajouté les inégalités fortes (décrites à la section 3.4.2) à la relaxation linéaire en utilisant la méthode de génération de coupes. Nous appliquons également la génération de colonnes sur les variables de

flots x_{ij}^k afin de conserver un nombre raisonnable de variables pour la résolution de la relaxation linéaire. De plus, nous appliquons une technique de fixation sur les variables de conception y_{ij}.

4.2.1 Génération de coupes

Au début de la méthode de génération de coupes, seules les contraintes de la formulation faible sont présentes dans la formulation. Les inégalités fortes (SI) violée sont alors ajoutées à la formulation et celle-ci est résolue à nouveau. La méthode termine lorsqu'aucune SI n'est violée. La méthode de génération de coupe résout donc la formulation forte en passant par la formulation faible et en ajoutant itérativement les SI violées.

L'avantage de résoudre la formulation forte par rapport à la formulation faible est la réduction de l'écart d'intégralité (voir section 2.1). Rappelons que les SI sont définies comme suit :

$$x_{ij}^k \leq d^k y_{ij} \qquad \forall (i,j) \in A,\ \forall k \in K \tag{4.1}$$

Bien qu'il en existe un nombre polynomial $O(|K||A|)$, elles sont ajoutées dynamiquement afin de réduire la dégénérescence et d'accélérer la résolution de la relaxation linéaire.

4.2.1.1 Génération et ajout des coupes violées

Une SI est ajoutée si elle est violée par la solution optimale de la relaxation linéaire. Ainsi, pour trouver les SI violées, nous itérons sur tous les arcs et tous les produits et nous ajoutons celles qui sont violées, soit celles qui satisfont l'inégalité suivante :

$$x_{ij}^k > d^k y_{ij} \quad \forall (i,j) \in A, k \in K \tag{4.2}$$

4.2.1.2 Résolution de la relaxation linéaire

L'ajout de nouvelles contraintes au problème rend la solution primale non réalisable. Par contre, la solution duale demeure toujours réalisable, mais n'est plus optimale puisque l'ajout de contraintes au modèle primal est équivalent à ajouter de nouvelles colonnes au modèle dual. Nous utilisons donc l'algorithme dual du simplexe pour résoudre à nouveau le problème puisqu'il peut redémarrer en partant de la base optimale à l'itération précédente. La méthode de génération de coupes est ensuite appliquée sur le nouveau problème. La génération de coupes se termine lorsqu'aucune coupe n'est violée.

4.2.2 Génération de colonnes

La relaxation linéaire est résolue en appliquant la génération de colonnes sur les variables de flots x_{ij}^k de la formulation dans l'espace des arcs. Notre méthode débute avec un sous-ensemble des variables de flots, puis elle ajoute itérativement les variables de flots pouvant améliorer la solution courante de la relaxation linéaire. Comme nous l'avons décrit à la section 2.2.1, la génération de colonnes est une méthode itérative qui résout un problème maître, en résolvant un problème maître restreint, ainsi qu'un ou plusieurs sous-problèmes. Nous décrivons ces trois composantes, problème maître, problème maître restreint et sous-problème, dans les prochaines sous-sections.

4.2.2.1 Problème maître

La génération de colonnes n'est généralement pas appliquée directement sur la formulation originale du problème, mais sur une reformulation du problème qui a un très grand nombre de variables. Par exemple, en appliquant la méthode de décomposition de Dantzig-Wolfe [16] sur la formulation dans l'espace des arcs, on obtient la formulation dans l'espace des chemins comme problème maître sur lequel on applique la génération de colonnes. Dans notre cas, nous n'effectuons pas de reformulation du modèle original. Le problème maître correspond donc à la formulation forte dans l'espace des arcs.

4.2.2.2 Problème maître restreint

La formulation du problème maître restreint est composée d'un sous-ensemble des variables de flots du problème maître ; certaines variables x_{ij}^k sont donc restreintes à être nulles.

Des arcs artificiels ont été ajoutés au modèle afin de toujours avoir une solution réalisable même si la configuration du réseau n'est pas réalisable. Nous avons ajouté pour chaque produit k un arc artificiel reliant les sommets $O(k)$ et $D(k)$. Ces arcs n'ont aucune limite de capacité, un coût fixe nul et un coût variable arbitrairement grand (la somme des coûts variables et fixes de tous les arcs). Ils sont ajoutés au réseau afin d'obtenir une première solution réalisable pour démarrer la génération de colonnes.

Ces arcs artificiels permettent aussi d'obtenir une solution réalisable même si une configuration du réseau (une affectation partielle des variables y_{ij}) est non réalisable (i.e s'il n'y a pas assez de capacité dans le réseau). Le sous-problème a alors toujours accès à une solution duale du problème maître restreint et peut donc générer, si possible, de nouvelles colonnes. Cela facilite grandement l'implantation de la génération de colonnes.

Il est à noter que l'ajout de ces nouvelles variables fait en sorte que toutes les solutions non réalisables pour le problème original seront tout de même élaguées lors de la recherche. En effet, étant donné qu'une solution non réalisable pour le problème original manque de capacité dans le réseau, cette solution utilisera forcément au moins un arc artificiel et la valeur de cette solution sera alors plus grande que la valeur de la solution candidate, puisque le coût d'un arc artificiel est très grand. Cette solution sera donc élaguée par l'algorithme de branch-and-bound.

4.2.2.2.1 Résolution du problème maître restreint La solution primale du problème maître restreint n'est plus optimale après l'ajout de nouvelles colonnes, mais elle demeure réalisable. Les nouvelles colonnes sont alors considérées comme des variables hors base à leur borne inférieure, soit zéro. Nous utilisons donc l'algorithme primal du simplexe pour résoudre à nouveau le problème maître restreint.

4.2.2.3 Sous-problème

Le sous-problème calcule le coût réduit des variables de flots qui ne sont pas encore générées et identifie les variables pouvant améliorer la solution courante du problème maître restreint. Pour le générer, nous associons une variable duale à toutes les contraintes de la formulation forte. En reformulant les contraintes du problème, on obtient le problème suivant :

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{4.3}$$

$$\sum_{j \in V_i^+} x_{ij}^k - \sum_{j \in V_i^-} x_{ji}^k = \begin{cases} d^k & \text{si } i = O(k) \\ -d^k & \text{si } i = D(k) \\ 0 & \text{sinon} \end{cases} \qquad \left(\pi_i^k\right) \qquad \forall i \in V,\ \forall k \in K \tag{4.4}$$

$$u_{ij} y_{ij} - \sum_{k \in K} x_{ij}^k \geq 0 \qquad (\alpha_{ij}) \qquad \forall (i,j) \in A \tag{4.5}$$

$$d^k y_{ij} - x_{ij}^k \geq 0 \qquad \left(\beta_{ij}^k\right) \qquad \forall (i,j) \in A,\ \forall k \in K \tag{4.6}$$

$$-y_{ij} \geq -1 \qquad (\gamma_{ij}) \qquad \forall (i,j) \in A \tag{4.7}$$

et son dual :

$$\max \sum_{k \in K} d^k \left(\pi_{O(k)}^k - \pi_{D(k)}^k\right) - \sum_{(i,j) \in A} \gamma_{ij} \tag{4.8}$$

$$\pi_i^k - \pi_j^k - \alpha_{ij} - \beta_{ij}^k \leq c_{ij}^k \qquad \left(x_{ij}^k\right) \qquad \forall k \in K,\ \forall (i,j) \in A \tag{4.9}$$

$$u_{ij} \alpha_{ij} + \sum_{k \in K} d^k \beta_{ij}^k - \gamma_{ij} \leq f_{ij} \qquad (y_{ij}) \qquad \forall (i,j) \in A \tag{4.10}$$

$$\alpha_{ij} \geq 0 \qquad \forall (i,j) \in A \tag{4.11}$$

$$\beta_{ij}^k \geq 0 \qquad \forall k \in K,\ \forall (i,j) \in A \tag{4.12}$$

$$\gamma_{ij} \geq 0 \qquad \forall (i,j) \in A \tag{4.13}$$

Nous pouvons formuler le coût réduit des variables x_{ij}^k à partir des contraintes (4.9) :

$$c_{ij}^k - \pi_i^k + \pi_j^k + \alpha_{ij} + \beta_{ij}^k \qquad \forall k \in K,\ \forall (i,j) \in A \tag{4.14}$$

Ainsi, les variables x_{ij}^k pouvant améliorer la solution primale du problème maître restreint sont celles dont le coût réduit est négatif :

$$c_{ij}^k - \pi_i^k + \pi_j^k + \alpha_{ij} + \beta_{ij}^k \ < \ 0 \qquad \forall k \in K,\ \forall (i,j) \in A \tag{4.15}$$

4.2.2.3.1 Résolution du sous-problème La résolution du sous-problème consiste à ajouter au problème maître restreint les variables x_{ij}^k ayant un coût réduit négatif, c'est-à-dire celles qui satisfont l'inégalité (4.15).

Étant donné que les inégalités fortes sont générées de façon dynamique à l'aide de la génération de coupes (décrite à la section 4.2.1), nous n'avons pas toutes les valeurs des variables β_{ij}^k à notre disposition. Nous décrivons donc la procédure pour les calculer. Nous définissons les contraintes d'écarts complémentaires, puisqu'elles sont utilisées dans notre analyse.

$$x_{ij}^k \left(c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} + \beta_{ij}^k\right) = 0 \quad \forall (i,j) \in A, \forall k \in K \tag{4.16}$$

$$y_{ij} \left(f_{ij} + \gamma_{ij} - u_{ij}\alpha_{ij} - \sum_{k \in K} d^k \beta_{ij}^k \right) = 0 \quad \forall (i,j) \in A \tag{4.17}$$

$$\alpha_{ij} \left(u_{ij} y_{ij} - \sum_{k \in K} x_{ij}^k \right) = 0 \quad \forall (i,j) \in A \tag{4.18}$$

$$\beta_{ij}^k \left(d^k y_{ij} - x_{ij}^k\right) = 0 \quad \forall (i,j) \in A, \forall k \in K \tag{4.19}$$

$$\gamma_{ij} \left(1 - y_{ij}\right) = 0 \quad \forall (i,j) \in A \tag{4.20}$$

À chaque itération, un certain nombre des variables de flots dans le problème maître restreint sont contraintes à être nulles. On dénote cet ensemble K_{ij}^0. Ainsi,

$$x_{ij}^k = 0 \quad \forall (i,j) \in A, \forall k \in K_{ij}^0 \tag{4.21}$$

Nous supposons que toutes les contraintes sont présentes dans le problème maître restreint (i.e toutes les SI ont été générées). Nous poursuivons l'analyse en distinguant deux cas pour la variable y_{ij} : soit $y_{ij} > 0$ ou $y_{ij} = 0$.

Cas $y_{ij} > 0$ Supposons que $k \in K_{ij}^0$. Si la solution du problème maître restreint était optimale, on aurait :

$$\beta_{ij}^k (\underbrace{d^k y_{ij}}_{>0} - \underbrace{x_{ij}^k}_{=0}) = 0 \quad \Longrightarrow \quad \beta_{ij}^k = 0 \tag{4.22}$$

Dans ce cas, si

$$c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} < 0 \tag{4.23}$$

alors la contrainte du dual

$$c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} + \beta_{ij}^k \geq 0 \tag{4.24}$$

serait violée et la solution du problème maître restreint ne serait donc pas optimale pour le problème maître. On génère alors les variables de flots x_{ij}^k telles que

$$c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} < 0 \tag{4.25}$$

Cas $y_{ij} = 0$ Puisque $y_{ij} = 0$, alors $x_{ij}^k = 0 \ \ \forall k \in K$. Posons

$$\bar{K}_{ij} = \left\{ k \in K \mid c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} < 0 \right\} \tag{4.26}$$

Si la solution du problème maître restreint était optimale, nous aurions

$$\gamma_{ij}(\underbrace{1 - y_{ij}}_{\neq 0}) = 0 \quad \Longrightarrow \quad \gamma_{ij} = 0 \tag{4.27}$$

De plus, nous connaissons la valeur des variables duales en ayant résolu le problème maître restreint. Lorsqu'on fixe les valeurs de ces variables duales, le problème dual se décompose par arc. Regroupons les valeurs des variables duales $\pi_i^k, \pi_j^k \ \ \forall k \in K$ et α_{ij} dans les termes de droite du problème dual associé à l'arc (i, j) :

$$0 = \max \gamma_{ij} \tag{4.28}$$

Sujet à

$$-\beta_{ij}^k \ \leq \ c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} \quad \forall k \in K \tag{4.29}$$

$$\sum_{k \in K} d^k \beta_{ij}^k - \gamma_{ij} \ \leq \ f_{ij} - u_{ij}\alpha_{ij} \tag{4.30}$$

$$\beta_{ij}^k \geq 0 \quad \forall k \in K \tag{4.31}$$

Ce modèle, ayant une valeur optimale finie, est équivalent au modèle suivant :

$$0 = \min \sum_{k \in K} \left(c_{ij} + \pi_i^k - \pi_j^k + \alpha_{ij}\right) x_{ij}^k + \left(f_{ij} - u_{ij}\alpha_{ij}\right) y_{ij} \tag{4.32}$$

Sujet à

$$0 \leq x_{ij}^k \leq d^k y_{ij} \quad \forall k \in K \tag{4.33}$$

$$0 \leq y_{ij} \leq 1 \tag{4.34}$$

La solution optimale de ce problème est nulle si et seulement si

$$f_{ij} - u_{ij}\alpha_{ij} \geq - \sum_{k \in \bar{K}_{ij}} \left(c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij}\right) d^k \tag{4.35}$$

Donc, si l'inégalité (4.35) est satisfaite, alors il existe une solution duale réalisable pour (i, j) et qui satisfait les contraintes d'écarts complémentaires où les variables $\beta_{ij}^k \ \ \forall k \in K$ prennent les valeurs suivantes :

$$\beta_{ij}^k = \begin{cases} -(c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij}) & \text{si } k \in \bar{K}_{ij} \\ 0 & \text{si } k \notin \bar{K}_{ij} \end{cases} \tag{4.36}$$

Il n'est donc pas nécessaire de générer de variables de flots. Par contre, si l'inégalité (4.35) n'est pas satisfaite, alors la solution optimale du problème maître restreint n'est pas optimale pour le problème maître. On génère alors les variables de flots x_{ij}^k telles que

$$c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} < 0 \tag{4.37}$$

Résumé Pour générer de nouvelles colonnes, on effectue les tests suivants :

1. si $y_{ij} > 0$ et $c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} < 0$, alors on ajoute la variable x_{ij}^k.
2. si $y_{ij} = 0, f_{ij} - u_{ij}\alpha_{ij} < -\sum_{k \in \bar{K}_{ij}} \left(c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij}\right) d^k$ et $c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} < 0$, alors on ajoute la variable x_{ij}^k.

4.2.3 Fixation des variables de décision

Il est possible de réduire le domaine des variables en utilisant leur coût réduit, ainsi qu'une borne inférieure et supérieure sur la valeur optimale du problème. L'avantage de réduire le domaine des variables est de réduire l'espace de recherche afin de résoudre plus rapidement le problème.

Au cours de l'algorithme, certaines variables de conception y_{ij} sont fixées à l'une de leurs bornes, alors que d'autres sont traitées comme des variables continues ayant pour domaine $[0, 1]$. Il est donc possible qu'une variable continue prenne une valeur entière (i.e., soit à l'une de ses bornes) à une certaine itération sans toutefois y être contrainte. Elle pourrait donc prendre une valeur fractionnaire à une autre itération, puis prendre à nouveau une valeur entière à une autre itération.

Soit z_{inf} la valeur optimale de la relaxation linéaire et $\bar{f}_{ij}$ le coût réduit de la variable y_{ij} dans cette solution. Si y_{ij} est hors base, alors $z_{\text{inf}}+|\bar{f}_{ij}|$ fournit une borne inférieure sur la valeur optimale du nouveau problème dans lequel la variable y_{ij} est fixée à son autre borne. Nous pouvons utiliser cette propriété pour fixer la variable à sa valeur actuelle si $z_{\text{inf}} + |\bar{f}_{ij}| > z_{\text{sup}}$, où z_{sup} est la valeur de la solution candidate.

Étant donné que la fixation des variables est très rapide, nous appliquons cette technique avant chaque itération de la génération de coupes.

4.3 Règle de branchement

Notre méthode utilise la règle de branchement fiable qui, selon nous, est l'une des meilleures règles de branchement décrite à ce jour dans la littérature. En effet, cette règle de branchement est un bon compromis entre le temps nécessaire pour la sélection de la variable parmi les variables candidates pour le branchement et le nombre total de noeuds générés.

4.4 Parcours de l'arbre de recherche

Lors de nos tests préliminaires, nous avons comparé trois types de parcours d'arbre de recherche : la recherche meilleur d'abord, la recherche en profondeur et la recherche en profondeur avec redémarrage. Nos tests préliminaires ont démontré que la recherche en profondeur avec redémarrage est dominée par les deux autres types de parcours, c'est-à-dire que lorsqu'elle est combinée avec le branchement fiable, le temps de calcul est systématiquement plus long que les deux autres types de parcours. Nous comparons donc dans notre expérimentation seulement la recherche meilleur d'abord et la recherche en profondeur.

CHAPITRE 5

RÉSULTATS

Dans ce chapitre, nous présentons et analysons les résultats expérimentaux de notre méthode appliquée sur des instances utilisées dans la littérature.

5.1 Instances

Notre expérimentation s'est déroulée sur 196 instances largement utilisées dans la littérature. Elles ont été générées par Gendron et al. [19, 20]. Les instances sont divisées en quatre groupes représentés au tableau 5.I. La taille d'une instance est définie par un triplet représentant le nombre de sommets, d'arcs et de produits dans le réseau. Les nombres entre parenthèses indiquent le nombre d'instances ayant cette taille. Les instances des groupes **C** et **R-II** ont un grand nombre de produits, alors que les instances du groupe **C+** et **R-I** ont peu de produits.

En plus d'avoir des tailles variées, certaines instances possèdent des coûts de conception plus élevés que les coûts de transport, et vice versa. Enfin, la capacité du réseau est parfois plus contraignante sur certaines instances.

5.2 Code et environnement

Nous avons implanté notre méthode en C++ en utilisant la librairie logicielle *Solving Constraint Integer Programs* (SCIP)[1]. Développée dans le cadre de la thèse de doctorat de Tobias Achterberg [1], SCIP est une librairie logicielle écrite en C permettant de faciliter le développement d'algorithmes de branch-and-bound. La relaxation linéaire est résolue par CPLEX 12.3.

Les tests ont été exécutés dans un environnement Linux sur un ordinateur ayant un processeur Intel Xeon X5660 cadencé à 2.80 GHz.

[1] Disponible à `http://scip.zib.de`

C	(31)	**C+**	(12)	**R-I**	(72)	**R-II**	(81)
20,230,40	(3)	25,100,10	(3)	10,35,10	(6)	20,120,40	(9)
20,230,200	(4)	25,100,30	(3)	10,35,25	(6)	20,120,100	(9)
20,300,40	(4)	100,400,10	(3)	10,35,50	(6)	20,120,200	(9)
20,300,200	(4)	100,400,30	(3)	10,60,10	(9)	20,220,40	(9)
30,520,100	(4)			10,60,25	(9)	20,220,100	(9)
30,520,400	(4)			10,60,50	(9)	20,220,200	(9)
30,700,100	(4)			10,85,10	(9)	20,320,40	(9)
30,700,400	(4)			10,85,25	(9)	20,320,100	(9)
				10,85,50	(9)	20,320,200	(9)

Tableau 5.I – Les instances utilisées pour tester notre méthode. Les triplets représentent la taille de l'instance (sommets, arcs, produits) et les nombres entre parenthèses représentent le nombre d'instances ayant cette taille pour chaque groupe.

5.3 Analyse des résultats

Avant de nous lancer dans le développement d'un algorithme de branch-and-price-and-cut, nous avons appliqué la génération de coupes et de colonnes à la racine de l'arbre de recherche. Ces résultats préliminaires nous ont ensuite motivés à étendre notre méthode à l'ensemble de l'abre de recherche. Nous présentons dans cette section les résultats de notre méthode appliquée à la racine, puis les résultats pour l'ensemble de l'arbre de recherche.

5.3.1 Méthode de price-and-cut à la racine

Nous avons appliqué notre méthode à la relaxation linéaire du MCND dans le but d'obtenir rapidement des résultats préliminaires. Nous l'avons ensuite comparée au logiciel d'optimisation mathématique CPLEX et à une méthode de génération de coupes afin d'évaluer sa réalisabilité. Nous décrivons dans les deux premières sous-sections les deux méthodes auxquelles nous nous comparons. Puis, nous décrivons dans la troisième sous-section notre méthode appliquée à la racine.

5.3.1.1 Méthode de génération de coupes génériques

Nous comparons notre méthode avec CPLEX, puisqu'il est l'un des meilleurs logiciels d'optimisation mathématique. Nous fournissons à CPLEX la formulation faible dans l'espace des arcs du MCND et nous changeons quelques paramètres dans le but d'optimiser la résolution linéaire par CPLEX. Tout d'abord, nous indiquons à CPLEX de résoudre seulement la relaxation linéaire du problème sans appliquer aucune règle de branchement en affectant la valeur 0 au paramètre `NodeLim`; puis, nous désactivons toutes les heuristiques primales afin que CPLEX se concentre sur la résolution de la relaxation linéaire. Nous laissons à CPLEX la liberté d'utiliser tous ses types de coupes. La résolution termine lorsqu'aucune coupe ne permet d'augmenter la borne.

5.3.1.2 Méthode de génération d'inégalités fortes

Dans la méthode précédente, nous utilisons CPLEX comme un logiciel de résolution de type boîte noire, c'est-à-dire que nous modifions quelques paramètres afin d'aider CPLEX, mais nous n'avons pas de contrôle lors de la résolution. Nous avons donc développé une méthode qui utilise CPLEX pour résoudre la formulation forte du MCND où les inégalités fortes sont ajoutées itérativement. De manière similaire à la méthode précédente, la résolution se termine lorsqu'aucune inégalité forte ne permet d'augmenter la borne.

5.3.1.3 Méthode de génération de colonnes et d'inégalités fortes

Cette méthode est similaire à la précédente, puisqu'elle résout également la formulation forte en ajoutant itérativement les coupes au modèle et arrête lorsqu'aucune inégalité forte n'est violée. Elle résout cependant la formulation forte à l'aide de la génération de colonnes sur les variables de flots.

5.3.1.4 Analyse des résultats à la racine

Les résultats sont rapportés au tableau 5.II. La méthode de génération de coupes génériques est identifiée par A, la méthode de génération d'inégalités fortes par B et

notre méthode par C. Les résultats sont séparés par groupe d'instances et par méthode (colonnes « Groupe » et « Méthode »). Puisque la méthode de génération de coupes génériques applique plusieurs types de coupes, la borne aura tendance à être supérieure par rapport à celles des deux autres méthodes. On retrouve alors sous la colonne « Écart relatif » l'écart relatif moyen entre la borne obtenue par la méthode de génération de coupes génériques et la méthode correspondante sur la ligne. La colonne « Temps » représente le temps moyen en secondes pour chacune des méthodes. Finalement, on retrouve sous la colonne « Gain en temps » le gain relatif moyen en temps par rapport à la méthode de générations de coupes génériques.

L'analyse du tableau 5.II permet de conclure que la qualité des bornes inférieures générées par les inégalités fortes est presque équivalente à celle générée par les coupes de CPLEX. En effet, l'écart relatif moyen pour les méthodes B et C est presque nul pour les instances des groupes **C** (0.2 %) et **R-II** (0.2%) et est légèrement plus élevé pour les instances des groupes **C+** (2.5%) et **R-I** (0.9%).

En comparant les temps d'exécution moyens des méthodes, il en ressort que la méthode B est légèrement plus rapide que la méthode A, puisque la méthode B ne génère qu'un seul type de coupes. Puis, en comparant le temps d'exécution moyen de notre méthode à ceux des deux autres méthodes, on remarque qu'elle est beaucoup plus rapide sur les instances du groupe **C** et **R-II** où le nombre de produits est élevé.

En somme, les trois méthodes calculent des bornes sensiblement égales pour la majorité des instances. Cependant, l'utilisation de la génération de colonnes dans notre méthode permet de résoudre plus rapidement les instances des groupes **C** et **R-II** que les deux autres méthodes.

5.3.2 Branch-and-price-and-cut

Nous comparons dans cette sous-section notre méthode à l'algorithme de branch-and-bound de CPLEX, ainsi qu'à une version sans génération de colonnes développée avec SCIP où les seules coupes activées sont les inégalités fortes. Ces méthodes sont des extensions des méthodes à la racine de la section précédente qu'on applique à l'ensemble de l'arbre de recherche.

Groupe	Méthode	Écart relatif	Temps (s)	Gain en temps
C	A	0 %	26,2	0 %
	B	0,2 %	23,3	11 %
	C	0,2 %	6,3	76 %
C+	A	0 %	2,3	0 %
	B	2,5 %	0,6	74 %
	C	2,5 %	0,6	74 %
R-I	A	0 %	0,09	0 %
	B	0,9 %	0,02	77 %
	C	0,9 %	0,02	77 %
R-II	A	0 %	6,3	0 %
	B	0,2 %	5,4	14 %
	C	0,2 %	2,6	59 %
Tous	A	0 %	6,9	0 %
	B	0,3 %	6,0	13 %
	C	0,3 %	2,1	70 %

Tableau 5.II – Comparaison des méthodes à la racine.

Nous distinguons trois types d'instances. Les instances résolues sont les instances pour lesquelles les trois méthodes ont prouvé, dans un temps limite de trois heures, la solution optimale. Les instances non résolues sont les instances pour lesquelles les trois méthodes n'ont pu prouver, dans le temps limite, la solution optimale. Enfin, les instances mixtes consistent en des instances pour lesquelles certaines méthodes ont prouvé la solution optimale, alors que d'autres méthodes n'ont pas réussi.

Les métriques pour comparer les performances des trois méthodes ne sont pas les mêmes dépendamment du type d'instances (i.e., résolues, non résolues et mixtes). Étant donné que les trois méthodes ont prouvé la solution optimale pour les instances résolues, on compare alors le temps d'exécution. Plus le temps est petit, meilleure est la méthode. Puisque toutes les méthodes ont atteint le temps limite avant d'avoir réussi à résoudre les instances non résolues, on considère qu'une méthode est supérieure à une autre si sa borne inférieure est meilleure. Pour les instances mixtes, la mesure de performance principale est le nombre d'instances résolues ; si deux méthodes résolvent le même nombre d'instances, on regarde alors le temps et la qualité de la borne inférieure.

Pour les trois méthodes, nous analysons la performance des algorithmes en faisant varier le type de parcours utilisé pour l'arbre de recherche. Puisque la valeur optimale (ou la meilleure solution connue) est donnée à priori, il serait judicieux de parcourir en profondeur l'arbre de recherche, puisqu'en théorie c'est le parcours le mieux adapté pour exploiter la réoptimisation lors de l'évaluation du noeud. Cependant, l'interaction entre l'évaluation des noeuds, le type de branchement et le type de parcours crée des effets croisés qui complexifient l'analyse théorique. Par exemple, l'ordre dans lequel les noeuds sont visités influence la règle de branchement, qui, à son tour, influence le nombre de noeuds générés dans l'arbre de recherche. C'est pour cette raison que la recherche meilleur d'abord supplante la recherche en profondeur sur certaines instances résolues, alors qu'en théorie la recherche en profondeur devrait prendre un temps d'exécution plus petit.

Nous présentons dans un premier temps les résultats sur les quatre ensembles d'instances (i.e., **C**, **C+**, **R-I** et **R-II**). Puis, étant donné que le but de notre méthode est d'exploiter la génération de colonnes afin de réduire la taille des formulations ayant un grand nombre de produits, nous comparons les méthodes sur les instances ayant un très grand nombre de produits provenant des ensembles **C** et **R-II**.

Nous identifions l'algorithme de branch-and-bound de CPLEX par l'acronyme *cplex*, la méthode de branch-and-cut où seules les inégalités fortes sont générées par l'acronyme *B&C* et notre méthode par l'acronyme *B&P&C*.

Instances de l'ensemble C Les résultats sur les instances de l'ensemble **C** sont rapportés au tableau 5.III. On dénombre 16 instances dans la catégorie « résolues », 16 dans la catégorie « non résolues » et 3 dans la catégorie « mixtes ». Le temps d'exécution de la méthode *B&P&C* (419 s) pour les instances résolues est deux fois plus rapide que celui de *cplex* (845 s) et *B&C* (1 013 s). L'écart relatif moyen est également plus petit pour la méthode *B&P&C* (0,7 %) que celui pour *cplex* (0,9%) et *B&C* (0,9%) sur les instances non résolues. La méthode *B&P&C* résout deux instances mixtes sur trois, alors que la méthode *cplex* en résout une et *B&C* aucune. On peut observer que le nombre de noeuds générés sur les instances résolues est du même ordre pour les trois méthodes, alors que

le nombre de noeuds explorés par la méthode *B&P&C* sur les instances non résolues est plus du double de celui des deux autres méthodes. Enfin, la génération de colonnes est particulièrement efficace sur les instances de l'ensemble **C**, car elle génère seulement le quart des variables de la formulation dans l'espace des arcs.

Méthode	**Parcours**	**Temps (s)**	**Nb. noeuds**	**Nb. variables**
cplex	Profondeur	845	1 487	28 942
	Meilleur d'abord	1 204	1 487	28 942
B&C	Profondeur	1 017	1 657	29 031
	Meilleur d'abord	1 013	1 756	29 031
B&P&C	Profondeur	474	2 352	6 692
	Meilleur d'abord	419	2 062	6 662

(a) Instances résolues

Méthode	**Parcours**	**Écart relatif (%)**	**Nb. noeuds**	**Nb. variables**
cplex	Meilleur d'abord	0,9	6 792	130 563
B&C	Meilleur d'abord	0,9	3 545	130 805
B&P&C	Meilleur d'abord	0,7	14 640	30 364

(b) Instances non résolues

Méthode	**Nb. instances résolues**
cplex	1/3
B&C	0/3
B&P&C	2/3

(c) Instances mixtes

Tableau 5.III – Résultats sur les instances de l'ensemble **C**

Instances de l'ensemble C+ Nous nous attendions à ce que notre méthode ne soit pas efficace sur les instances du groupe **C+** où le nombre de produits est petit, ce qui fut effectivement le cas. D'autant plus que les inégalités fortes ne sont pas suffisantes pour résoudre ces instances, comme le démontre Chouman et al. [9], ainsi que nos résultats. L'ajout d'inégalités valides basées sur les coupures semble essentiel pour s'attaquer à ces instances. Les résultats sur les instances de l'ensemble **C+** sont rapportés au tableau 5.IV. Huit instances sont dans la catégorie « résolues », deux instances sont dans la catégories « non résolues » et trois instances sont dans la catégorie « mixtes ».

Méthode	Parcours	Temps (s)	Nb. noeuds	Nb. variables
cplex	Profondeur	26	630	2 676
	Meilleur d'abord	30	627	2 676
B&C	Profondeur	327	6 080	2 693
	Meilleur d'abord	322	5 767	2 693
B&P&C	Profondeur	687	7 966	1 573
	Meilleur d'abord	496	5 870	1 557

(a) Instances résolues

Méthode	Parcours	Écart relatif (%)	Nb. noeuds	Nb. variables
cplex	Meilleur d'abord	3,2	16 517	8 401
B&C	Meilleur d'abord	7,2	429 172	8 420
B&P&C	Meilleur d'abord	7,7	169 185	6 790

(b) Instances non résolues

Méthode	Nb. instances résolues
cplex	2/2
B&C	0/2
B&P&C	0/2

(c) Instances mixtes

Tableau 5.IV – Résultats sur les instances de l'ensemble **C+**

Instances de l'ensemble R-I Les instances de l'ensemble **R-I** sont considérées comme étant faciles. En effet, il faut en moyenne un peu moins de trois secondes pour résoudre chacune des 72 instances. Il est alors difficile de discriminer les méthodes à partir de cet ensemble d'instances. Les résultats sur les instances de l'ensemble **R-I** sont rapportés au tableau 5.V. Il est intéressant d'observer que la génération de colonnes obtient des temps de calcul similaires à ceux des deux autres méthodes.

Instances de l'ensemble R-II Les résultats sur les instances de l'ensemble **R-II** sont présentés au tableau 5.VI. On dénombre 60 instances dans la catégorie « résolues », 17 dans la catégorie « non résolues » et 4 dans la catégorie « mixtes ». Pour les instances résolues, le temps d'exécution moyen de *cplex* (766 s) est plus rapide que celui de *B&C* (1114 s) et *B&P&C* (855 s). Pour les instances non résolues, les écarts relatifs moyens sont semblables pour les trois méthodes : 1,3 % pour *cplex* et *B&P&C*, et 1,4 % pour

Méthode	Parcours	Temps (s)	Nb. noeuds	Nb. variables
cplex	Profondeur	2,0	213	1 822
	Meilleur d'abord	2,5	215	1 822
B&C	Profondeur	2,4	285	1 850
	Meilleur d'abord	2,6	303	1 850
B&P&C	Profondeur	2,0	421	745
	Meilleur d'abord	2,5	396	743

(a) Instances résolues

Tableau 5.V – Résultats sur les instances de l'ensemble **R-I**

B&C. La méthode *cplex* résout les quatres instances mixtes, alors que *B&C* en résout deux et *B&P&C* en résout une. Ainsi, sur l'ensemble des instances de **R-II**, la méthode *cplex* performe mieux. Il faut cependant noter que le tiers des instances ont seulement 40 produits. Lorsqu'on sépare les instances ayant peu de produits (moins de 100 produits) et celles en ayant beaucoup (100 produits ou plus), on observe alors que la méthode *B&P&C* performe mieux que *cplex*. Au prochain paragraphe, nous comparons les trois méthodes sur les instances ayant un grand nombre de produits.

Instances avec un grand nombre de produits Les méthodes ont été précédemment comparées en séparant les instances selon quatre ensembles utilisés dans la littérature. Pour véritablement apprécier notre méthode, nous comparons les méthodes sur les 78 instances ayant au moins 100 produits provenant des ensembles **C** et **R-II**. Les résultats sont rapportés au tableau 5.VII. On dénombre 41 instances dans la catégorie « résolues », 31 dans la catégorie « non résolues » et 6 dans la catégorie « mixtes ». Autant pour les instances résolues que les instances non résolues, la méthode *B&P&C* est supérieure aux deux autres méthodes. Cependant, la méthode *cplex* résout une instance de plus parmi les instances mixtes que la méthode *B&P&C*.

5.3.3 Conclusion

Les résultats expérimentaux nous permettent de conclure que notre méthode est meilleure que les deux autres méthodes sur les instances de grande taille et qu'elle est

Méthode	Parcours	Temps (s)	Nb. noeuds	Nb. variables
cplex	Profondeur	766	3 516	19 409
	Meilleur d'abord	1 051	3 507	19 409
B&C	Profondeur	1 114	15 874	19 510
	Meilleur d'abord	1 213	15 926	19 510
B&P&C	Profondeur	901	21 131	7 739
	Meilleur d'abord	855	16 829	7 733

(a) Instances résolues

Méthode	Parcours	Écart relatif (%)	Nb. noeuds	Nb. variables
cplex	Meilleur d'abord	1,3	9 900	44 966
B&C	Meilleur d'abord	1,4	10 280	45 123
B&P&C	Meilleur d'abord	1,3	18 550	22 353

(b) Instances non résolues

Méthode	Nb. instances résolues
cplex	4/4
B&C	2/4
B&P&C	1/4

(c) Instances mixtes

Tableau 5.VI – Résultats sur les instances de l'ensemble **R-II**

compétitive sur les instances de petite et moyenne taille. Cela est dû au fait que notre méthode résout une formulation plus compacte tout au long de l'exécution ; la taille des formulations est deux fois moins grandes que les formulations des deux autres méthodes.

Méthode	Parcours	Temps (s)	Nb. noeuds	Nb. variables
cplex	Profondeur	1 246	3 150	29 983
	Meilleur d'abord	1 733	3 020	29 983
B&C	Profondeur	1 580	2 847	30 128
	Meilleur d'abord	1 669	2 826	30 128
B&P&C	Profondeur	1 087	3 722	11 024
	Meilleur d'abord	1 015	3 317	11 013

(a) Instances résolues

Méthode	Parcours	Écart relatif (%)	Nb. noeuds	Nb. variables
cplex	Meilleur d'abord	1,1	6 065	91 215
B&C	Meilleur d'abord	1,2	4 004	91 424
B&P&C	Meilleur d'abord	1,0	12 235	27 389

(b) Instances non résolues

Tableau 5.VII – Résultats sur les instances ayant un très grand nombre de produits

CHAPITRE 6

CONCLUSION

Nous avons présenté dans ce mémoire une méthode exacte basée sur des techniques de programmation linéaire mixte pour résoudre le problème de conception de réseau avec coûts fixes et capacités. Nous avons décrit une méthode utilisant à la fois la génération de colonnes et la génération de coupes dans un algorithme de branch-and-bound.

En nous comparant à CPLEX, actuellement l'un des meilleurs logiciels d'optimisation mathématique, notre méthode s'est montrée compétitive sur les instances de taille moyenne et supérieure sur les instances de grande taille ayant un grand nombre de produits, et ce, même si elle n'utilise qu'un seul type d'inégalités valides.

Plusieurs possibilités sont envisagées pour le développement de travaux connexes basés sur notre méthode. Une extension directe de notre méthode serait d'incorporer de nouvelles inégalités valides. Nous estimons ainsi que le nombre de noeuds générés dans l'arbre de recherche serait grandement diminué, et par conséquent, le temps d'exécution aussi. Notre méthode pourrait également servir au développement de méthodes heuristiques. En effet, notre méthode est très rapide pour l'évaluation des noeuds. Ainsi, la méthode de branch-and-bound, considérée comme une méthode exacte puisqu'elle parcourt tous les noeuds générés dans l'arbre de recherche, peut être transformée en une méthode heuristique en limitant le nombre de noeuds visités. Notre méthode pourrait servir à parcourir certaines régions en profondeur de l'arbre de recherche afin de trouver des solutions réalisables.

Finalement, bien que les instances sur lesquelles nous avons effectué notre expérimentation contenait jusqu'à 400 produits, il serait intéressant d'appliquer notre méthode sur des problèmes similaires ayant un plus grand nombre de produits pour en mesurer l'extensibilité (*scalability*).

BIBLIOGRAPHIE

[1] T. Achterberg. *Constraint Integer Programming*. Thèse de doctorat, Technische Universität Berlin, 2007.

[2] T. Achterberg, T. Koch et A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42 – 54, 2005.

[3] R. K. Ahuja, T. L. Magnanti et J. B. Orlin. *Network flows : theory, algorithms, and applications*. Prentice-Hall, 1993.

[4] A. Atamtürk et M. Savelsbergh. Integer-programming software systems. *Annals of Operations Research*, 140:67–124, 2005.

[5] T.G. Crainic B. Gendron et A. Frangioni. Multicommodity capacitated network design. Dans B. Sanso et P. Soriano, éditeurs, *Telecommunications Network Planning*, pages 1–19. Kluwer Academic Publishers, 1998.

[6] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[7] D. Bienstock et O. Günlük. Capacitated network design – polyhedral structure and computation. *Journal on Computing*, 8:243–259, 1994.

[8] M. Chouman et T. G. Crainic. A mip-tabu search hybrid framework for multicommodity capacitated fixed-charge network design. Rapport technique CIRRELT-2010-31, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT), 2010.

[9] M. Chouman, T. G. Crainic et B. Gendron. Commodity representations and cutset-based inequalities for multicommodity capacitated fixed-charge network design. Rapport technique CIRRELT-2011-56, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport (CIRRELT), 2011.

[10] A. Costa, J. Cordeau et B. Gendron. Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42:371–392, 2009.

[11] T. G. Crainic. Service network design in freight transportation. *European Journal of Operational Research*, 122(2):272 – 288, 2000.

[12] T. G. Crainic, A. Frangioni et B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1-3):73 – 99, 2001.

[13] T. G. Crainic et M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8:601–627, 2002.

[14] T. G. Crainic, M. Gendreau et J. M. Farvolden. A simplex-based tabu search method for capacitated network design. *INFORMS Journal on Computing*, 12(3): 223–236, 2000.

[15] T. G. Crainic, Y. Li et M. Toulouse. A first multilevel cooperative algorithm for capacitated multicommodity network design. *Computers & Operations Research*, 33(9):2602 – 2622, 2006.

[16] G. B. Dantzig et P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[17] J. Desrosiers et M. E. Lübbecke. A primer in column generation. Dans G. Desaulniers, J. Desrosiers et M. M. Solomon, éditeurs, *Column Generation*, pages 1–32. Springer US, 2005. ISBN 978-0-387-25486-9.

[18] M. Fischetti et A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[19] B. Gendron et T. G. Crainic. Relaxations for multicommodity capacitated network design problems. Rapport technique CRT-965, Centre de recherche sur les transports, 1994.

[20] B. Gendron et T. G. Crainic. Bounding procedures for multicommodity capacitated fixed charge network design problems. Rapport technique CRT-96-06, Centre de recherche sur les transports, 1996.

[21] I. Ghamlouche, T. G. Crainic et M. Gendreau. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research*, 51(4):pp. 655–667, 2003.

[22] I. Ghamlouche, T. G. Crainic et M. Gendreau. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131:109–133, 2004.

[23] J. Gondzio, P. González-Brevis et P. Munari. New developments in the primal-dual column generation technique. Rapport technique ERGO 11-001, University of Edinburgh, 2011.

[24] O. Günlük. A branch-and-cut algorithm for capacitated network design problems. *Mathematical Programming*, 86:17–39, 1999.

[25] M. Hewitt, G. L. Nemhauser et M. W. P. Savelsbergh. Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, 22(2):314–325, 2010.

[26] K. Holmberg et D. Yuan. A lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48(3):pp. 461–481, 2000.

[27] N. Katayama, M. Chen et M. Kubo. A capacity scaling heuristic for the multicommodity capacitated network design problem. *Journal of Computational and Applied Mathematics*, 232(1):90 – 101, 2009.

[28] G. Kliewer et L. Timajev. Relax-and-cut for capacitated network design. Dans Gerth Brodal et Stefano Leonardi, éditeurs, *Algorithms - ESA 2005*, volume 3669 de *Lecture Notes in Computer Science*, pages 47–58. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-29118-3.

[29] T. L. Magnanti et P. Mirchandani. Shortest paths, single origin-destination network design, and associated polyhedra. *Networks*, 23(2):103–121, 1993.

[30] T. L. Magnanti, P. Mirchandani et R. Vachani. Modeling and solving the two-facility capacitated network loading problem. *Operations Research*, 43(1):pp. 142–157.

[31] T. L. Magnanti et R. T. Wong. Network design and transportation planning : Models and algorithms. *Transportation Science*, 18(1):1, 1984.

[32] M. Minoux. Networks synthesis and optimum network design problems : Models, solution methods and applications. *Networks*, 19(3):313–360, 1989.

[33] W. B. Powell. A local improvement heuristic for the design of less-than-truckload motor carrier networks. *Transportation Science*, 20(4):246–257, 1986.

[34] I. Rodríguez-Martín et J. J. Salazar-González. A local branching heuristic for the capacitated fixed-charge network design problem. *Computers & Operations Research*, 37(3):575 – 581, 2010.