Université de Montréal

# Méthode de génération de colonnes pour les problèmes de conception de réseaux avec coûts d'ajout de capacité

par
Souhaïla El Filali

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

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

Janvier 2014

## RÉSUMÉ

Les problèmes de conception de réseaux ont reçu un intérêt particulier et ont été largement étudiés de par leurs nombreuses applications dans différents domaines, tels que les transports et les télécommunications.

Nous nous intéressons dans ce mémoire au problème de conception de réseaux avec coûts d'ajout de capacité. Il s'agit d'installer un ensemble d'équipements sur un réseau en vue de satisfaire la demande, tout en respectant les contraintes de capacité, chaque arc pouvant admettre plusieurs équipements. L'objectif est de minimiser les coûts variables de transport des produits et les coûts fixes d'installation ou d'augmentation de capacité des équipements.

La méthode que nous envisageons pour résoudre ce problème est basée sur les techniques utilisées en programmation linéaire en nombres entiers, notamment celles de génération de colonnes et de coupes. Ces méthodes sont introduites dans un algorithme général de *branch-and-bound* basé sur la relaxation linéaire.

Nous avons testé notre méthode sur quatre groupes d'instances de tailles différentes, et nous l'avons comparée à CPLEX, qui constitue un des meilleurs solveurs permettant de résoudre des problèmes d'optimisation, ainsi qu'à une méthode existante dans la littérature combinant des méthodes exactes et heuristiques. Notre méthode a été plus performante que ces deux méthodes, notamment pour les instances de très grandes tailles.



**Mots clés : Problème de conception de réseaux, coûts d'ajout de capacité, reformulation 0-1, génération de colonnes, méthodes de coupes, *branch-and-bound*.**

## ABSTRACT

Network design problems received a particular interest and have been widely studied because of their many applications in different areas, such as logistics and telecommunications.

We focus in this work on the multicommodity capacitated network design problem with capacity expansion costs. It consists in opening a set of facilities on a network in order to meet the demand of some commodities, while respecting the capacity constraints. Each arc can admit several facilities. The objective is to minimize the commodities transportation costs, and the fixed costs of opening or increasing the capacity of the facilities.

The method we are using to solve this problem is based on techniques used in integer programming, including column generation and cutting-plane methods. These methods are introduced into a general branch-and-bound algorithm, based on linear relaxation.

We test our method on four groups of instances of different sizes, and we compare it with CPLEX, which is one of the best solvers available for optimization problems. We compare it also with an existing method in the literature, combining exact and heuristic methods.

Numerical results show that our method was able to outperform both methods, especially when tested on large scale instances.



**Keywords : Multicommodity capacitated network design problem, capacity expansion costs, 0-1 reformulation, column generation, cutting-plane methods, branch-and-bound.**

## TABLE DES MATIÈRES

## LISTE DES TABLEAUX

## LISTE DES FIGURES

# CHAPITRE 1

# INTRODUCTION

De nombreuses applications peuvent être modélisées comme des problèmes de conception de réseaux. Plusieurs de ces applications relèvent en particulier des domaines du transport et des télécommunications, et comprennent, à titre d'exemples, des décisions relatives à la construction de routes, à l'ouverture de nouvelles lignes ferroviaires ou aériennes, ou à l'installation de câbles de réseaux optiques.

Les problèmes de conception de réseaux, plus particulièrement, ceux avec coûts d'ajout de capacité, consistent à transporter un flot (biens, personnes, informations) sur un réseau qu'il faut construire en y installant un ou plusieurs types d'équipements, tout en minimisant les coûts variables de transport du flot, et les coûts fixes encourus suite à l'installation ou à l'augmentation de la capacité de ces équipements.

Ce problème, à l'image de tous les problèmes de conception de réseaux avec contraintes de capacité, est facile à modéliser en programme linéaire en nombres entiers. Il appartient toutefois à la classe de problèmes *NP*-difficiles, qui requièrent un effort de calcul considérable et un temps énorme de résolution pour des instances de grande taille.

Pour cette raison, plusieurs méthodes proposées dans la littérature pour résoudre ce problème se sont basées sur des méthodes heuristiques, ou sur une combinaison des méthodes exactes et heuristiques, notamment pour les instances de très grande taille. Les méthodes exactes n'ont été par contre capables de fournir une solution optimale au problème que pour des instances de tailles moyennes.

Dans ce mémoire, nous nous intéressons à une formulation du problème de conception de réseaux avec coûts d'ajout de capacité en variables binaires, renforcée d'un ensemble d'inégalités valides dites inégalités "étendues" (*extended linking inequalities*).

Nous proposons de résoudre cette formulation avec une méthode exacte exploitant la méthode de génération de colonnes, qui est introduite dans un algorithme général de *branch-and-bound* basé sur la relaxation continue. Nous suggérons de plus de générer les inégalités valides de manière dynamique en se basant sur les méthodes de coupes.

Ce choix de méthodes se justifie par le très grand nombre de variables et de contraintes constituant le problème.

Nous présentons dans le chapitre 2 de ce mémoire un rappel sur la programmation linéaire et la programmation linéaire en nombres entiers. Nous détaillons aussi les trois principales techniques qui constituent notre méthode, à savoir la génération de colonnes, les méthodes de coupes, et l'algorithme de *branch-and-bound*. Dans le chapitre 3, nous donnons un aperçu général sur les problèmes d'optimisation et de conception de réseaux. Nous passons en revue la littérature existante sur les différentes variantes du problème et leurs méthodes de résolution.

Le problème sur lequel porte le mémoire fait l'objet du chapitre 4. Nous y définissons le modèle adopté, et y détaillons la méthode proposée après un bref aperçu sur les méthodes existantes dans la littérature. Enfin, nous présentons dans le chapitre 5 les expérimentations effectuées pour tester notre méthode, ainsi que les résultats auxquels nous avons abouti.

# CHAPITRE 2

# ALGORITHMES FONDAMENTAUX

Nous présentons dans la première partie de ce chapitre quelques généralités sur la programmation linéaire et la programmation linéaire en nombres entiers. Nous détaillons par la suite trois méthodes principales permettant de résoudre les problèmes qui relèvent de ces deux domaines, à savoir, la méthode de génération de colonnes, les méthodes de coupes et l'algorithme de *branch-and-bound*.

Pour réaliser ce chapitre, nous nous sommes inspirés des références suivantes : Bazaraa et al. [11], Hillier er Lieberman [42], Nemhauser et Wolsey [56] et Wolsey [60] pour la première partie, et Achterberg [1], Achterberg et al. [2], Atamtürk et Savelsbergh [9], Dantzig et Wolfe [25], Desrosiers et Lübbecke [26] et Gondzio et al. [37] pour la seconde partie.

## 2.1 Quelques notions de base

### 2.1.1 Problème de programmation linéaire

Soit $A$ une matrice réelle de dimension $m \times n$ et de rang $m$, et $b$ et $c$ deux vecteurs de dimensions $m$ et $n$ respectivement. Nous entendons par problème de programmation linéaire, présenté ci-après sous sa forme standard, le problème de trouver le vecteur de réels positifs $x$, de dimension $n$, qui minimise la fonction linéaire $cx$ tout en satisfaisant les équations linéaires $Ax = b$.

$$(LP): \qquad z_{LP} = min\{cx : Ax = b, x \geqslant 0\} \tag{2.1}$$

Selon la théorie de dualité, en résolvant le problème $(LP)$, un autre problème linéaire est implicitement résolu. Il s'agit de son dual $(DL)$. $(PL)$ est alors dit problème primal.

$$(DL): \qquad z_{DL} = max\{ub : uA \leqslant c, \;\; u \in \mathbb{R}^m\} \tag{2.2}$$

Le problème (2.2) satisfait certaines propriétés particulières relatives à la théorie de la dualité, et ses variables $u$ fournissent des informations importantes sur l'ensemble des solutions optimales du problème primal, ainsi que des conditions nécessaires et suffisantes pour atteindre l'optimalité. Nous présentons dans ce qui suit trois résultats importants de cette théorie.

**Théorème de dualité faible** Le théorème de dualité faible, énoncé ci-après, stipule que toute solution réalisable du problème dual donne une borne inférieure sur la valeur optimale du problème primal, et inversement, toute solution réalisable du problème primal fournit une borne supérieure sur la valeur optimale du problème dual.

**Théorème 1.** *Si $\bar{x}$ est une solution réalisable du primal et $\bar{u}$ une solution réalisable du dual, alors : $c\bar{x} \geqslant z_{LP} \geqslant z_{DL} \geqslant \bar{u}b$.*

**Théorème de dualité forte** Le théorème de dualité forte, présenté ci-dessous, affirme que si deux solutions réalisables pour les problèmes primal et dual sont égales, alors ces deux solutions sont optimales.

**Théorème 2.** *Soient $\bar{x}$ et $\bar{u}$ deux solutions réalisables de $(LP)$ et $(DL)$ respectivement. Si $c\bar{x} = \bar{u}b$, alors $\bar{x}$ et $\bar{u}$ sont des solutions optimales respectives de $(LP)$ et $(DL)$.*

Un résultat important découlant de ce théorème est présenté dans le corollaire suivant :

**Corollaire 1.** *Uniquement une des quatre alternatives suivantes est vérifiée :*

1. *$(LP)$ et $(DL)$ admettent une solution optimale. Dans ce cas, on a : $z_{LP} = z_{DL}$,*
2. *$(LP)$ est non borné et $(DL)$ est non réalisable,*
3. *$(DL)$ est non borné et $(LP)$ est non réalisable,*
4. *$(LP)$ et $(DL)$ ne sont pas réalisables.*

Ce corollaire indique que si l'un des deux problèmes admet une solution optimale, alors l'autre problème en admet une également. Si, par contre, l'un des problèmes est non borné, l'autre est par conséquent non réalisable. Enfin, si l'un des deux problèmes est non réalisable, l'autre est ou bien non réalisable ou bien non borné.

**Théorème des écarts complémentaires** Le théorème des écarts complémentaires est l'un des résultats les plus importants reliant les problèmes primal et dual. Il indique qu'à l'optimum, si une variable dans le problème primal est strictement positive, alors la contrainte correspondante dans le problème dual doit être saturée (c'est-à-dire vérifiée à l'égalité). Inversement, si une contrainte dans le problème dual n'est pas saturée, alors la variable correspondante dans le problème primal doit être nulle. Le théorème s'énonce comme suit :

**Théorème 3.** *Soient $\bar{x}$ et $\bar{u}$ deux solutions réalisables de $(LP)$ et $(DL)$ respectivement. $\bar{x}$ et $\bar{u}$ sont des solutions optimales respectives de $(LP)$ et $(DL)$ si et seulement si :* $\bar{x}(c-\bar{u}A)=0$.

### 2.1.2 Problème de programmation linéaire en nombres entiers

Les problèmes de programmation linéaire en nombres entiers sont des problèmes linéaires dont certaines ou toutes les variables sont contraintes à prendre des valeurs entières. Ils prennent alors la forme générale suivante :

$$(IP): \quad min \ z_{IP} = cx + dy \tag{2.3}$$

*sujet à :*

$$Ax + Gy = b, \tag{2.4}$$

$$x \in \mathbb{R}^n_+, \ y \in \mathbb{Z}^n_+ \tag{2.5}$$

Lorsque seulement certaines variables sont contraintes à être entières, on parle de "problème linéaire en nombres entiers mixte". Par ailleurs, on dit qu'un problème est "linéaire en nombres entiers pur" lorsque toutes les variables du problème sont entières.

Dans certains cas, les variables entières sont contraintes à prendre des valeurs dans l'ensemble $\{0,1\}$. On parle alors de "problème linéaire en nombres entiers binaires".

Enfin, dans le cas particulier où aucune variable n'est entière, on retrouve la forme générale du problème linéaire tel que présenté dans la section 2.1.1.

Les contraintes d'intégralité sont des contraintes "complicantes". La difficulté du problème peut en effet augmenter de manière exponentielle avec l'augmentation du nombre de variables entières, notamment dans le cas de variables binaires. Dans ce sens,

les problèmes linéaires s'avèrent être plus faciles à résoudre que ceux en nombres entiers. C'est la raison pour laquelle les méthodes de résolution (voir section 2.2) empruntent des techniques de programmation linéaire pour trouver ou approcher les solutions des problèmes en nombres entiers. Un principe important derrière est celui de la relaxation.

### 2.1.3 Relaxation et relaxation linéaire

Afin de résoudre le problème linéaire en nombres entiers, nous avons besoin d'une borne supérieure et d'une borne inférieure sur sa valeur optimale. Lorsque ces deux bornes sont égales, nous pouvons déduire qu'une solution optimale du problème a été déterminée.

Pour trouver une borne supérieure au problème $(IP)$, on note que la valeur associée à toute solution réalisable en constitue une. Pour obtenir une borne inférieure, il est intuitivement intéressant de construire un problème plus facile à résoudre, qui soit défini sur un domaine plus large contenant celui de $(IP)$, ou qui possède une fonction objectif à valeurs inférieures à celles de la fonction objectif du problème en nombres entiers sur tout le domaine réalisable.

Le principe de relaxation combine ces deux propriétés et on parle, dans un contexte plus général, de "problème relaxé" ou "relaxation" $(R)$

$$(R): \quad z_R = min\{f(x) : x \in D_R \subseteq \mathbb{R}^n\}$$

par rapport à un problème initial $(P)$

$$(P): \quad z_P = min\{c(x) : x \in D_P \subseteq \mathbb{R}^n\}$$

lorsque :

- $D_P \subseteq D_R$,
- $f(x) \leqslant c(x), \ \forall x \in D_P$.

Déterminer un problème relaxé peut s'avérer difficile en pratique. Cependant, dans le cas du problème en nombres entiers, le problème linéaire associé où les contraintes d'intégralité ne sont pas incluses remplit cet objectif. Un tel problème est alors dit "relaxation linéaire" ou "relaxation continue", et on montre qu'il s'agit bien d'une relaxation, puisque le domaine réalisable du problème linéaire contient celui du problème en

nombres entiers, la fonction objectif demeurant inchangée.

En plus de fournir une borne inférieure sur la valeur optimale du problème en nombres entiers, la relaxation linéaire peut aussi donner des informations sur l'optimalité. En effet, dans le cas où la solution optimale du problème relaxé est entière, elle constitue une solution optimale du problème en nombres entiers. De plus, lorsque le domaine réalisable de la relaxation linéaire est vide, celui du problème en nombres entiers l'est aussi.

## 2.2 Méthodes de programmation linéaire en nombres entiers

En utilisant les informations fournies par la relaxation linéaire, des algorithmes tels que le *branch-and-bound* permettent de résoudre les problèmes en nombres entiers. De plus, pour améliorer le temps d'exécution et les solutions obtenues, les méthodes de génération de colonnes et de génération de coupes sont utilisées pour résoudre les problèmes relaxés. Nous détaillons dans cette section ces trois méthodes.

### 2.2.1 Méthode de génération de colonnes

La génération de colonnes est l'une des principales méthodes utilisées pour résoudre des problèmes linéaires de grande taille. La programmation linéaire et la programmation en nombres entiers étant des domaines d'optimisation fortement liés, la méthode constitue aussi un outil puissant pour résoudre des problèmes en nombres entiers.

Le principe de la méthode consiste à résoudre, de manière itérative, une série de problèmes de plus petites tailles, en vue d'obtenir une solution optimale au problème linéaire initial $(MP)$, dit "problème maître". Notons $N = \{1, ..., n\}$ l'ensemble des indices des variables du problème $\lambda_i$. Le problème maître est formulé comme suit :

$$(MP): \qquad min \;\; z_{MP} = \sum_{i \in N} \tilde{c}_i \lambda_i \tag{2.6}$$

*sujet à :*

$$\sum_{i \in N} \tilde{a}_i \lambda_i = b \tag{2.7}$$

$$\lambda_i \geqslant 0, \;\; i \in N \tag{2.8}$$

Nous faisons ici l'hypothèse que les coefficients $\tilde{a}_i$ ne sont pas donnés explicitement et qu'ils sont implicitement représentés comme des éléments d'un ensemble $\mathscr{A} \neq \emptyset$, pouvant être généré selon des règles bien déterminées. Nous supposons de plus que les coefficients $\tilde{c}_i$ sont liés aux coefficients $\tilde{a}_i$ par une fonction connue $\mathscr{C}$.

Pour résoudre $(MP)$, on considère un sous ensemble de variables $\bar{N} \subseteq N$ sur lequel on résout un "problème maître restreint" $(RMP)$.

$$(RMP): \qquad min \;\; z_{RMP} = \sum_{i \in \bar{N}} \tilde{c}_i \lambda_i \tag{2.9}$$

*sujet à :*

$$\sum_{i \in \bar{N}} \tilde{a}_i \lambda_i = b \tag{2.10}$$

$$\lambda_i \geqslant 0, \;\; i \in \bar{N} \tag{2.11}$$

Pour initialiser la méthode et déterminer le sous-ensemble d'indices des variables $\lambda_i$ qui vont constituer le problème maître restreint initial, il est possible d'introduire des variables artificielles ou d'utiliser des méthodes heuristiques.

Dans l'objectif de déterminer et ajouter de nouvelles variables au problème maître restreint $(RMP)$, on résout un "sous-problème" par rapport aux variables duales. Soit $\bar{\lambda}$ la solution optimale du problème maître restreint, et $\bar{\pi}$ le vecteur des variables duales optimales associées aux contraintes (2.7) du problème maître. Il est possible de vérifier si $\bar{\lambda}$ est une solution optimale pour le problème maître $(MP)$ en calculant les coûts réduits associés à chaque variable $\bar{\lambda_i}$ :

$$\bar{\tilde{c}}_i = \tilde{c}_i - \bar{\pi}\tilde{a}_i, \;\; i \in N \tag{2.12}$$

Si $\bar{\tilde{c}}_i \geqslant 0 \, \forall i \in N$, alors on peut conclure que la solution optimale du problème maître restreint actuel l'est aussi pour le problème maître $(MP)$. Cependant, du moment que les coefficients $\tilde{a}_i$ ne sont connus que partiellement, calculer $\bar{\tilde{c}}_i$ n'est pas possible pour tout $i \in N$. Comme alternative, on calcule le minimum des $\bar{\tilde{c}}_i$ pour tout $i \in N$, ce qui revient à

résoudre le sous-problème de *pricing* suivant :

$$(SP): \qquad z_{SP} = min\{\bar{\bar{c}}_i \,:\, \tilde{a}_i \in \mathscr{A}\} \tag{2.13}$$

Lorsque $z_{SP}$ est positif, ce qui signifie que tous les coûts réduits $\bar{\bar{c}}_i$ sont positifs $\forall i \in N$, on conclut que la solution actuelle $\bar{\lambda}$ est optimale pour le problème maître. Dans le cas contraire, une ou plusieurs variables ayant de coûts réduits négatifs sont introduites dans le problème maître restreint $(RMP)$ qui est réoptimisé par la suite.

#### 2.2.1.1 Bornes supérieures et inférieures

À chaque itération de la génération de colonnes, le problème maître restreint et le sous-problème permettent de calculer des bornes supérieures et inférieures sur la valeur optimale du problème maître.

En effet, puisque le domaine réalisable du problème maître restreint est inclus dans celui du problème maître, toute solution optimale de $(RMP)$ est réalisable pour $(MP)$, et la valeur correspondante $z_{RMP}$ constitue donc une borne supérieure $\bar{z}$ sur la valeur optimale du problème maître ($\bar{z} = z_{RMP} \geqslant z_{MP}$).

Contrairement à la borne supérieure qui est facile à calculer, la borne inférieure $\underline{z}$ est plutôt difficile à déterminer. Si l'on est capable toutefois de trouver une constante $\kappa$ telle que $\kappa \geqslant \sum_{i \in N} \bar{\lambda}_i^*$, où $\bar{\lambda}^*$ est la solution optimale du problème maître, alors $\underline{z} = z_{RMP} + \kappa z_{SP} \leqslant z_{MP}$ est une borne inférieure sur la valeur optimale de $(MP)$.

Le calcul de cette borne inférieure présente des difficultés pour deux raisons principales. D'une part, il n'existe pas de manière exacte pour déterminer la constante $\kappa$, sauf dans le cas de la décomposition de Dantzig-Wolfe comme nous allons le présenter dans la section 2.2.1.4. D'autre part, la borne inférieure ne croît pas nécessairement d'une itération à l'autre de la méthode de génération de colonnes et ce, contrairement à la borne supérieure qui constitue une fonction monotone décroissante.

La connaissance des bornes inférieure et supérieure permet, entre autres, de calculer "l'écart d'optimalité" $\bar{z} - \underline{z}$ ou $\bar{z} - LB$, où $LB$ est la plus grande borne inférieure connue, et de déterminer une solution $\varepsilon$-optimale si cet écart est inférieur à une constante $\varepsilon$ fixée

à l'avance.

#### 2.2.1.2 Convergence de la méthode

Etant donné qu'aucune variable dans le problème maître restreint ne possède un coût réduit négatif, toute colonne $\tilde{a}_i \in \mathscr{A}$ est donc générée au plus une seule fois. La procédure de génération de colonnes est alors exacte et se termine en un nombre fini d'itérations, pourvu que l'ensemble $\mathscr{A}$ soit fini.

#### 2.2.1.3 Reformulation de Dantzig-Wolfe

La reformulation de Dantzig-Wolfe permet de reformuler tout problème linéaire en un problème qui vérifie les hypothèses citées précédemment quand aux structures des coefficients $\tilde{a}_i$ et $\tilde{c}_i$ et ce, afin de pouvoir le résoudre par la procédure de génération de colonnes, dite dans ce cas "décomposition de Dantzig-Wolfe".

Considérons le problème linéaire $(LP)$ présenté dans la section 2.1.1, et supposons qu'il est possible de séparer les contraintes $Ax = b$ en deux sous-ensembles de contraintes $A^1x = b^1$ et $A^2x = b^2$. Nous pouvons donc réécrire $(LP)$ sous la forme :

$$z = \min\{cx : \ A^1x = b^1, \ x \in X\} \tag{2.14}$$

où X est un polyèdre de la forme $X = \{x \in \mathbb{R}^n_+ : A^2x = b^2\} \neq \emptyset$.

Tout élément $x$ de $X$ peut être écrit comme combinaison convexe de ses points extrêmes $\{x_p\}_{p\in P}$ et combinaison conique de ses rayons extrêmes $\{x_r\}_{r\in R}$ :

$$x = \sum_{p\in P} \lambda_p x_p + \sum_{r\in R} \lambda_r x_r \tag{2.15}$$

avec,

$$\sum_{p\in P} \lambda_p = 1, \ \lambda_p \geqslant 0, \ \forall p \in P \quad et \quad \lambda_r \geqslant 0, \ \forall r \in R.$$

En posant $\tilde{c}_i = c_i x_i$ *et* $\tilde{a}^1_i = a^1_i x_i$, $i \in P \cup R$, le problème (2.14) peut être reformulé comme suit :

$$min \;\; z = \sum_{p \in P} \tilde{c}_p \lambda_p + \sum_{r \in R} \tilde{c}_r \lambda_r \tag{2.16}$$

*sujet à :*

$$\sum_{p \in P} \tilde{a}^1_p \lambda_p + \sum_{r \in R} \tilde{a}^1_r \lambda_r = b^1 \tag{2.17}$$

$$\sum_{p \in P} \lambda_p = 1 \tag{2.18}$$

$$\lambda \geqslant \mathbf{0} \tag{2.19}$$

Le principe de la décomposition de Dantzig-Wolfe consiste à résoudre la formulation (2.16)-(2.19) en utilisant la méthode de génération de colonnes, initialisée par un sous-ensemble de points et de rayons extrêmes de $X$.

#### 2.2.1.4 Cas de problèmes linéaires en nombres entiers

La décomposition de Dantzig-Wolfe peut être adaptée au cas où $X$ est un ensemble discret, c'est-à-dire :

$$X = \{x \in \mathbb{Z}^n_+ : \; A^2 x = b^2\}.$$

La reformulation de Dantzig-Wolfe reste dans ce cas valable. La relation liant $\lambda$ à $x$ doit cependant figurer dans le modèle afin de conserver la contrainte d'intégralité des variables $x$.

$$min \;\; z = \sum_{p \in P} \tilde{c}_p \lambda_p + \sum_{r \in R} \tilde{c}_r \lambda_r \tag{2.20}$$

*sujet à :*

$$\sum_{p \in P} \tilde{a}^1_p \lambda_p + \sum_{r \in R} \tilde{a}^1_r \lambda_r = b^1 \tag{2.21}$$

$$\sum_{p \in P} \lambda_p = 1 \tag{2.22}$$

$$\lambda \geqslant \mathbf{0} \tag{2.23}$$

$$\sum_{p \in P} x_p \lambda_p + \sum_{r \in R} x_r \lambda_r = x \tag{2.24}$$

$$x \in \mathbb{Z}_+^n \tag{2.25}$$

Pour résoudre ce modèle, les contraintes d'intégralité (2.25) sont relaxées et par conséquent, les contraintes (2.24) peuvent être éliminées du problème. Le problème maître résultant correspond au problème linéaire (2.16)-(2.19), qui possède dans ce cas des colonnes définies par des points et rayons extrêmes de l'enveloppe convexe de $X$ ($conv(X)$). Il est résolu par la procédure de génération de colonnes décrite précédemment.

En effet, le problème maître restreint (2.26)-(2.29) est obtenu de (2.16)-(2.19) en ne considérant qu'un sous-ensemble de points extrêmes $\bar{P} \subseteq P$ et de rayons extrêmes $\bar{R} \subseteq R$ de $conv(X)$ :

$$min \;\; z = \sum_{p \in \bar{P}} \tilde{c}_p \lambda_p + \sum_{r \in \bar{R}} \tilde{c}_r \lambda_r \tag{2.26}$$

*sujet à :*

$$\sum_{p \in \bar{P}} \tilde{a}_p^1 \lambda_p + \sum_{r \in \bar{R}} \tilde{a}_r^1 \lambda_r = b^1 \tag{2.27}$$

$$\sum_{p \in \bar{P}} \lambda_p = 1 \tag{2.28}$$

$$\lambda \geqslant \mathbf{0} \tag{2.29}$$

Étant donné les variables duales $\bar{\pi}$ et $\bar{\pi}_0$ associées au contraintes (2.27) et (2.28) du problème maître restreint, le sous-problème de *pricing* s'énonce comme suit :

$$(SP_{DW}) : \begin{cases} \min\limits_{i \in P} \{\tilde{c}_i - \pi \tilde{a}_i^1 - \pi_0\} \\ \min\limits_{i \in R} \{\tilde{c}_i - \pi \tilde{a}_i^1\} \end{cases} \tag{2.30}$$

En reformulant (2.30) en termes des variables initiales $x$, nous obtenons le problème linéaire en nombres entiers :

$$(SP_{DW}) : \;\; z^*_{SP_{DW}} = min\{(c - \bar{\pi} A^1)x - \bar{\pi}_0, \;\; x \in X\} \tag{2.31}$$

Si $z^*_{SP_{DW}} \geqslant 0$, la méthode se termine avec une solution optimale de la relaxation linéaire du problème (2.20)-(2.25).

Si par contre $z^*_{SP_{DW}} < 0$, deux cas peuvent se présenter. Ou bien la valeur de $z^*_{SP_{DW}}$ est non bornée inférieurement et la solution correspond à un rayon extrême $\mathbf{x_r}$ de $conv(X)$ ; la colonne $[c^t\mathbf{x_r}, (A^1\mathbf{x_r})^t, 0]^t$ est alors introduite dans le problème maître restreint. Dans le cas où la valeur de $z^*_{SP_{DW}}$ est finie, le point extrême correspondant $\mathbf{x_p}$ donne lieu à la colonne $[c^t\mathbf{x_p}, (A^1\mathbf{x_p})^t, 1]^t$ qui est incluse dans le problème maître restreint avant de réoptimiser.

La borne inférieure sur la valeur optimale du problème maître peut dans ce cas être calculée par la relation :

$$\underline{z} = \bar{z} + z^*_{SP_{DW}}$$

Pour les problèmes en nombres entiers, la méthode de génération de colonnes est appliquée à la relaxation linéaire du problème maître, et est combinée avec la méthode de *branch-and-bound* pour obtenir une solution entière. On parle dans ce cas de méthode de *branch-and-price*.

Pour résumer, nous présentons dans la figure 2.1 la méthode de génération de colonnes en pseudo-code.

1. Initialisation : $q \leftarrow 0$. Déterminer un sous-ensemble d'indices $N_q \subseteq N$.
2. Résoudre le problème maître restreint $(RMP_q)$ composé des variables $x_i$ telles que $i \in N_q$.
3. Résoudre le sous-problème de *pricing*.
4. Si pour toute variable $x_i$ telle que $i \notin N_q$, les coûts réduits correspondants sont non négatifs : arrêter. Le problème maître a été résolu.
5. Sinon, introduire les variables $x_i$, $i \notin N_q$, ayant des coûts réduits négatifs dans $(RMP_q)$. Introduire leurs indices $i$ dans $N_q$.
6. $q \leftarrow q + 1$. Aller à l'étape 2.

Figure 2.1 – Méthode de génération de colonnes

### 2.2.2 Méthode de coupes

Dans plusieurs problèmes linéaires en nombres entiers, nous avons recours à l'introduction d'inégalités valides afin d'améliorer la formulation et d'obtenir de meilleures bornes inférieures sur la valeur optimale.

Considérons le problème linéaire en nombre entiers $(IP)$ : $\min\{cx, x \in X\}$, où $X$ est l'ensemble discret $X = \{x : Ax = b, x \in \mathbb{Z}^n_+\}$. Nous souhaitons avoir une description exacte de l'enveloppe convexe de $X$ afin de se ramener à la résolution d'un problème linéaire de la forme $(LP)$ : $\min\{cx, x \in conv(X)\}$.

Comme il n'est pas toujours facile de déterminer exactement cette enveloppe convexe, notamment pour des problèmes *NP*-difficiles, l'approche alternative consiste à approximer $conv(X)$ pour le problème en question en générant des inégalités valides. Cette méthode présente l'avantage qu'il n'est pas nécessaire de produire toutes les inégalités valides pour aboutir à la solution, puisqu'il suffit d'approximer $conv(X)$ dans un voisinage de la solution optimale pour résoudre le problème.

Il est possible de déterminer toutes les inégalités valides et de les introduire à priori dans le problème relaxé. Cependant, lorsque le nombre d'inégalités valides à introduire est énorme, il devient difficile de résoudre le problème dans sa globalité dans un temps raisonnable. Pour cette raison, l'approche qui consiste à générer les inégalités valides de manière itérative s'avère très utile et permet de réduire la taille des problèmes résolus.

Rappelons qu'une "inégalité valide" est une contrainte qui est satisfaite par toutes les solutions réalisables (entières) du problème en nombres entiers. La méthode de génération de coupes consiste d'abord à déterminer les inégalités valides non satisfaites par certains points réalisables du problème relaxé, dites dans ce cas "coupes". Ces coupes sont identifiées en résolvant le "problème de séparation" défini comme suit :

**Définition 1.** *Etant donné un ensemble d'inégalités valides pour un problème en nombres entiers quelconque et $x^*$ une solution de sa relaxation linéaire, le problème de séparation consiste à montrer que $x^*$ satisfait toutes ces inégalités valides, et si ce n'est pas le cas, permet de déterminer au moins une inégalité valide qui soit violée par $x^*$.*

Une fois qu'une coupe est trouvée, elle peut être introduite dans le problème $(LP)$

pour améliorer la relaxation sans pour autant modifier le domaine réalisable du problème $(IP)$. Suite à cette étape, la nouvelle formulation de $(LP)$ est résolue à nouveau, et le processus est réitéré jusqu'à ce qu'aucune inégalité valide ne soit violée ou qu'une solution optimale soit déterminée pour $(IP)$. La figure 2.2 présente l'algorithme général des méthodes de coupes.

1. Initialisation : $q \leftarrow 0$. Définir un problème relaxé $(LP_q)$ du problème initial $(IP)$.
2. Résoudre $(LP_q)$.
   Si $(LP_q)$ est non réalisable : arrêter. Le problème $(IP)$ est non réalisable.
   Sinon,
   Si $(LP_q)$ est non borné, identifier une solution réalisable $x^q$ tel que $cx^q < M$ où $M$ est un nombre négatif très grand.
   Sinon, soit $x^q$ la solution optimale de $(LP_q)$.
3. Résoudre le problème de séparation pour $x^q$.
   Si $x^q$ satisfait toutes les inégalités valides de $(IP)$. Arrêter.
   Sinon, ajouter les inégalités valides de $(IP)$ violées par $x^q$ à $(LP_q)$.
4. $q \leftarrow q + 1$. Aller à l'étape 2.

Figure 2.2 – Schéma général des méthodes de coupes

Notons enfin qu'il est possible d'introduire plusieurs coupes violées à la fois à chaque itération de l'algorithme. Si ce dernier se termine sans avoir trouvé une solution optimale entière pour $(IP)$, la dernière formulation obtenue constitue une version améliorée de la formulation initiale, et peut être utilisée dans un algorithme de *branch-and-bound* dans le but d'améliorer le temps de calcul. Lorsque les deux algorithmes sont combinés, ils donnent lieu à la méthode de *branch-and-cut*.

### 2.2.3 Méthode de *Branch-and-Bound*

La méthode de *branch-and-bound* est une méthode d'énumération implicite qui est souvent utilisée pour résoudre des problèmes de programmation linéaire en nombres entiers. Cette méthode repose sur le principe "diviser pour régner", permettant de réduire récursivement le problème initial en plusieurs sous-problèmes de plus petite taille dans

le cadre d'un arbre de recherche, en se basant sur le résultat suivant :

**Proposition 1.** *Soit le problème linéaire en nombres entiers suivant :*

$$(IP): \quad z_{IP} = \min\{cx : x \in X\}, \quad X = \{x : \ Ax = b,\ x \in \mathbb{Z}^n_+\},$$

*Considérons les sous-problèmes*

$$(IP^q): \quad z^q_{IP} = \min\{cx : x \in X^q\}$$

*où* $\{X^q\}^k_{q=1}$ *est une partition de* $X$*. Alors,*

$$z_{IP} = \min_{q=1,\ldots,k} z^q_{IP}$$

Nous présentons dans la figure 2.3 le schéma général de la méthode de *branch-and-bound*. Les règles d'élagage, de sélection des sous-problèmes et de parcours de l'arbre de recherche sont détaillées dans les sous-sections qui suivent.

1. Initialisation : $\mathscr{L} = \{IP\}$. $\quad z^*_{IP} = \infty$. $\quad X^0 = X$.
2. Test d'arrêt : si $\mathscr{L} = \emptyset$, alors la solution $x^*$ telle que $z^*_{IP} = cx^*$ est optimale.
3. Sélection du nœud : Choisir un problème $(IP^q)$ dans $\mathscr{L}$.
4. Evaluation du nœud : Résoudre la relaxation $(LP^q)$ de $(IP^q)$.
   Si $(LP^q)$ est non réalisable, supprimer le de $\mathscr{L}$. Aller à l'étape 2.
   Sinon, soit $x^q$ sa solution optimale et $z^q_{LP}$ la valeur optimale correspondante.
5. Elagage :
   Si $z^q_{LP} \geqslant z^*_{IP}$, supprimer $(LP^q)$ de $\mathscr{L}$. Aller à l'étape 2.
   Sinon, Si $x^q$ n'est pas entière, aller à l'étape 6.
   Sinon, poser $z^*_{IP} = z^q_{LP}$. Aller à l'étape 2.
6. Branchement : Soit $\{X^{q_j}\}^k_{j=1}$ une partition de $X^q$. Ajouter les problèmes $\{IP^{q_j}\}^k_{j=1}$ à $\mathscr{L}$. Aller à l'étape 2.

Figure 2.3 – Algorithme général de *branch-and-bound*

#### 2.2.3.1 Règles d'élagage

Pour éviter qu'une énumération explicite de tous les choix possibles ne soit effectuée, ce qui risque de prendre un temps exponentiel, des règles d'élagage ont été mises en

place pour éviter l'exploration non nécessaire de certains sous-ensembles $X^i$ de $X$. Elles permettent de déterminer si, à n'importe quel niveau de l'arbre de recherche, un nœud ne doit plus être exploré suite à sa non réalisabilité, son optimalité, ou sa dominance par une autre solution déjà trouvée.

En effet, on arrête l'exploration d'un nœud suite à trois situations possibles. Ou bien la résolution de la relaxation linéaire montre que la partie en question du domaine réalisable est vide. Dans ce cas, il n'est plus intéressant de continuer la recherche suivant cette branche. Ou bien la relaxation donne une solution entière, la partie correspondante du domaine réalisable a donc fourni sa meilleure solution. Ou encore, la solution de la relaxation linéaire obtenue est dominée par la meilleure solution connue, c'est-à-dire que sa valeur optimale est supérieure à la borne supérieure, elle n'est donc pas intéressante. Les règles d'élagage peuvent alors être énoncées comme suit :

**Définition 2.** *Considérons la relaxation linéaire $(LP^q)$ du problème en nombres entiers à un nœud quelconque de l'arbre de recherche :*

$$(LP^q): \qquad z^q_{LP} = min\{cx : x \in X^q_L\},$$

*Si $(LP^q)$ possède une solution optimale finie, on la dénote $x^q$. Les règles d'élagages sont alors les suivantes :*
*1- Non réalisabilité : $X^q_L = \emptyset$,*
*2- Optimalité : $x^q \in \mathbb{Z}^n_+$,*
*3- Dominance : $z^q_{LP} \geqslant z^*_{IP}$, où $z^*_{IP}$ est la valeur de la meilleure solution connue du problème initial $(IP)$.*

Notons que pour appliquer la troisième règle, il suffit de connaître une borne supérieure $\bar{z}$ sur $z^*_{IP}$, et une borne inférieure $\underline{z}$ sur $z^q_{LP}$. Si $\bar{z} \leqslant \underline{z}$, alors la règle est valide.

#### 2.2.3.2 Stratégies de recherche

Les stratégies de recherche permettent de déterminer de quelle manière l'arbre de recherche doit être exploré, et comment sélectionner le prochain nœud à évaluer.

Deux méthodes sont principalement connues, la recherche meilleur d'abord et la recherche en profondeur d'abord. Chacune d'entre elles présente des points forts et des

points faibles, raison pour laquelle d'autres stratégies ont été développées pour bénéficier des avantages qu'elles présentent, ou éviter leurs inconvénients.

**2.2.3.2.1 La recherche meilleur d'abord** La méthode de recherche meilleur d'abord fait partie des stratégies de recherche dites adaptives, puisqu'elle utilise l'information collectée sur les nœuds actifs au cours de la résolution afin de déterminer le prochain nœud à résoudre.

Selon cette stratégie, le nœud sélectionné est celui qui fournit la meilleure borne inférieure, et donc a la plus petite valeur optimale de la relaxation linéaire. L'objectif étant d'améliorer la borne inférieure globale afin de converger plus rapidement vers une solution optimale.

Du moment qu'il est plus probable d'obtenir de meilleures valeurs optimales aux relaxations linéaires dans le haut de l'arbre de recherche, la stratégie meilleur d'abord favorise la recherche près de la racine. Ceci risque toutefois de donner lieu à de très grandes listes de sous-problèmes actifs qui nécessitent un espace mémoire plus grand. D'autant plus que les problèmes résolus ne sont pas nécessairement proches les uns des autres dans la hiérarchie de l'arbre. Chaque problème doit donc être résolu à nouveau, ce qui conduit à un temps plus long de résolution.

Il reste que la stratégie présente l'avantage d'éviter l'exploration des nœuds superflus, ce qui permet cette fois d'économiser le temps d'exécution.

**2.2.3.2.2 La recherche en profondeur d'abord** Contrairement à la recherche meilleur d'abord, la recherche en profondeur d'abord est une stratégie qui détermine à priori l'ordre dans lequel l'arbre de recherche sera exploré. Cette stratégie présente deux avantages principaux.

D'une part, la relaxation linéaire d'un nœud fils est très proche de celle du nœud père, il est donc possible de réduire le temps de résolution en utilisant la méthode duale du simplexe pour réoptimiser. Ceci permet aussi d'économiser de l'espace mémoire puisque seule l'information sur les nœuds parents est à stocker.

D'autre part, étant donné que plus on avance dans l'arbre de recherche, plus les variables

sont contraintes à être entières, il existe alors plus de possibilité de trouver des solutions entières au problème dans les nœuds profonds que dans ceux proches de la racine. La méthode favorise ainsi l'amélioration de la borne supérieure.

La stratégie de recherche en profondeur d'abord présente tout de même l'inconvénient de permettre l'exploration des nœuds superflus, qui auraient pu être éliminés si une meilleure borne était connue.

**2.2.3.2.3 Stratégie hybride** Pour bénéficier des avantages que présentent les deux méthodes détaillées, l'idée de les combiner semble être prometteuse. Au début de la méthode de *branch-and-bound*, il est préférable d'aller plus en profondeur dans l'arbre de recherche afin de trouver une meilleure solution entière rapidement. La recherche meilleur d'abord est par la suite utilisée en vue d'améliorer la borne inférieure.

**2.2.3.2.4 La recherche avec meilleur estimé** Cette méthode consiste à calculer une estimation de la valeur de la meilleure solution réalisable d'un sous-problème à un nœud quelconque de l'arbre de recherche, puis choisir le nœud ayant le plus petit estimé. Il existe deux principaux critères pour calculer cette estimation :

**Critère de la meilleure projection** Selon ce critère, l'estimation est calculée par la relation :

$$E^q = z^q_{LP} + (\frac{z^*_{IP} - z^0_{LP}}{s^0})s^q \tag{2.32}$$

$z^q_{LP}$ étant la valeur de la solution optimale du nœud courant, $z^0_{LP}$ celle du nœud racine, $z^*_{IP}$ la valeur de la meilleure solution connue et $s^q$ est calculé par la formule (2.33), où $f_i^- = x_i - \lfloor x_i \rfloor$, et $f_i^+ = \lceil x_i \rceil - x_i$.

$$s^q = \sum_{i \in N} min\{f_i^-, f_i^+\} \tag{2.33}$$

Ce critère ne considère pas le coût pour atteindre l'intégralité pour chaque variable individuellement, et nécessite la connaissance d'une borne supérieure globale à priori.

**Le critère de la meilleure estimation** Ce critère permet de calculer l'estimation par la formule :

$$E^q = z^q_{LP} + \sum_{i \in N} min\{|P_i^- f_i^-|, |P_i^+ f_i^+|\} \tag{2.34}$$

$P_i^-$ et $P_i^+$ sont les pseudo-coûts résultants de l'arrondissement de la variable $x_i$ vers les valeurs entières inférieure et supérieure respectivement. Ils calculent l'amélioration unitaire de la fonction objectif due à la fixation de $x_i$ à son plancher ou à son plafond. Ce critère permet ainsi de mesurer la valeur minimale estimée d'une solution arrondie.

#### 2.2.3.3 Règles de branchement

La question qui se pose à présent est de déterminer la méthode de diviser le domaine réalisable du problème $(IP)$ pour construire l'arbre de recherche. Toute méthode doit tenir compte de l'objectif d'écarter les solutions optimales fractionnaires sans éliminer les solutions entières du domaine réalisable. Il est de plus préférable que les domaines des sous-ensembles générés soient mutuellement disjoints afin d'éviter l'exploration redondante de certaines régions du domaine réalisable.

La méthode la plus simple et intuitive existante est la "dichotomie", où le domaine réalisable d'une seule variable fractionnaire $x_i$ est modifié à la fois. Cette variable est choisie dans le problème linéaire du nœud père, et les contraintes $x_i \geqslant \lceil x_i \rceil$ et $x_i \leqslant \lfloor x_i \rfloor$ sont introduites dans les sous-problèmes des nœuds fils, dits fils de droite (*right child*) et fils de gauche (*left child*) respectivement.

Il existe d'autres méthodes plus compliquées pour générer les nœuds fils qui permettent dans certains cas de diviser le problème du nœud père en plus de deux sous-problèmes. Elles sont cependant peu appliquées quoique efficaces dans certains cas particuliers.

L'efficacité de la règle de branchement est cruciale pour assurer une meilleure performance de la méthode de *branch-and-bound* dans la résolution des problèmes linéaires en nombre entiers. Nous entendons par efficacité la capacité de la méthode à améliorer les bornes inférieures et supérieures le plus rapidement possible. Comme il est plus dif-

ficile de dériver des règles qui puissent agir sur la borne supérieure, les techniques de branchement existantes permettent principalement d'améliorer la borne inférieure.

Nous détaillons par la suite certaines méthodes qui permettent de choisir la variable sur laquelle le branchement doit être effectué. Comme il en existe un très grand nombre, nous portons notre attention sur les méthodes les plus utilisées, notamment par les solveurs disponibles de nos jours.

**2.2.3.3.1 Variable la plus fractionnaire** Cette règle de branchement est très intuitive. Elle suggère de brancher sur la variable ayant la partie fractionnaire la plus proche de 0.5, ce qui revient à choisir $x_{\hat{\imath}}$ telle que $\hat{\imath} = \text{argmin}_{i \in N}\{|0.5 - f_i^-|\}$. La performance de cette stratégie n'est pas meilleure que si $x_{\hat{\imath}}$ est choisie aléatoirement [1].

Il est aussi possible de choisir la variable la moins fractionnaire. La performance est toutefois aussi mauvaise que pour la règle de la variable la plus fractionnaire.

**2.2.3.3.2 Pseudo-coûts** La règle des pseudo-coûts prend en considération la variation de la valeur de la fonction objectif induite par la fixation d'une variable quelconque $x_i$ à une valeur entière. Cette variation est mesurée par la relation :

$$\begin{cases} D_i^- = P_i^- f_i^- & si\ x_i\ prend\ la\ valeur\ \lfloor x_i \rfloor \\ D_i^+ = P_i^+ f_i^+ & si\ x_i\ prend\ la\ valeur\ \lceil x_i \rceil \end{cases} \tag{2.35}$$

Les valeurs de $P_i^-$ et $P_i^+$ peuvent être prédéterminées ou estimées en observant, par exemple, l'augmentation de la borne inférieure suite au branchement sur $x_i$. Notons $z_{LP}^{q-}$ et $z_{LP}^{q+}$ les valeurs optimales des relaxations linéaires aux nœuds fils résultant du branchement sur la variable $x_i$. Il est possible de calculer $P_i^-$ et $P_i^+$ par les formules suivantes, où $z_{LP}^q$ est la valeur optimale du problème courant.

$$P_i^- = \frac{z_{LP}^{q-} - z_{LP}^q}{f_i^-} \qquad et \qquad P_i^+ = \frac{z_{LP}^{q+} - z_{LP}^q}{f_i^+} \tag{2.36}$$

Pour initialiser cette stratégie, on utilise le coefficient de $x_i$ dans la fonction objectif comme estimation de l'augmentation.

Il existe d'autres formules pour calculer les pseudo-coûts, comme par exemple :

$$P_i^- = \frac{\rho_i^-}{\nu_i^-} \qquad et \qquad P_i^+ = \frac{\rho_i^+}{\nu_i^+} \tag{2.37}$$

où $\rho_i^-$ et $\rho_i^+$ représentent la somme des variations unitaires de l'objectif lorsque $x_i$ est fixée à son plancher ou à son plafond respectivement, et $\nu_i^-$ et $\nu_i^+$ représentent le nombre de branchements effectués sur la variable $x_i$ en la fixant à l'une ou l'autre de ces valeurs.

Dans ce cas également, l'étape d'initialisation s'avère nécessaire, puisque $\nu_i^-$ et $\nu_i^+$ sont nuls au début de l'algorithme. $P_i^-$ et $P_i^+$ sont alors estimés par la moyenne des pseudo-coûts des autres variables initialisées. Si aucun pseudo-coût n'est encore calculé, cette moyenne est fixée à 1.

Une fois l'estimation de la valeur des accroissements $D_i^-$ et $D_i^+$ déterminée, il existe différents critères pour choisir la variable sur laquelle brancher, de manière à maximiser la différence entre la valeur de la fonction objectif de la relaxation linéaire du nœud père et celle des nœuds fils.

Il est, par exemple, possible de maximiser la somme des variations sur les deux branches, c'est-à-dire choisir $x_{\hat{\imath}}$ telle que $\hat{\imath} = \underset{i \in N}{\operatorname{argmax}}\{D_i^- + D_i^+\}$. Un autre critère consiste à maximiser la variation minimale sur les deux branches, ce qui revient à choisir $x_{\hat{\imath}}$ telle que $\hat{\imath} = \underset{i \in N}{\operatorname{argmax}}\{\min\{D_i^-, D_i^+\}\}$. Enfin, un troisième critère consiste à sélectionner $x_{\hat{\imath}}$ telle que $\hat{\imath} = \underset{i \in N}{\operatorname{argmax}}\{\lambda \min\{D_i^-, D_i^+\} + (1-\lambda)\max\{D_i^-, D_i^+\}\}$, où $\lambda$ est un paramètre dans l'intervalle [0,1] souvent choisi proche de 1.

Cette règle de branchement, quoique rapide, utilise peu d'information locale pour prendre une décision. Sa performance reste toutefois meilleure que celle de la règle de la variable la plus fractionnaire.

**2.2.3.3.3 Branchement fort** L'objectif du branchement fort est de déterminer la variable fractionnaire qui donne la meilleure amélioration de la fonction objectif lorsque fixée à une valeur entière et ce, avant d'effectuer aucun branchement.

Pour ce faire, la méthode suggère de résoudre les relaxations linéaires de tous les problèmes résultant du branchement sur chacune des variables, et de déterminer par la

suite la meilleure variable sur laquelle il faut brancher.

Lorsque toutes les relaxations linéaires sont résolues à l'optimum on parle de branchement fort complet (*full strong branching*). Ce type de branchement est cependant très consommateur de temps et de mémoire, vu le grand nombre de calculs qu'il requiert.

Pour accélérer la méthode, les relaxations linéaires ne sont résolues que partiellement. La décision est alors basée sur l'observation de l'amélioration de la fonction objectif le long des itérations de la méthode du simplexe. Une autre alternative consiste aussi à ne considérer qu'un sous-ensemble de variables les plus prometteuses.

**2.2.3.3.4 Pseudo-coûts avec initialisation par branchement fort** Malgré les amendements apportés à la stratégie de branchement fort, le fait de résoudre plusieurs problèmes linéaires dans l'unique objectif de déterminer la variable sur laquelle brancher reste très coûteux en terme de temps. Pour cette raison, cette technique n'est généralement pratiquée que pendant les premières itérations de la méthode de *branch-and-bound* et uniquement sur des variables non initialisées pour la méthode des pseudo-coûts.

**2.2.3.3.5 Branchement fiable [1]** Cette méthode de branchement est une généralisation de la méthode précédente, dans le sens où le branchement fort n'est pas effectué uniquement sur des variables ayant des pseudo-coûts non initialisés, mais aussi sur les variables ayant de pseudo-coûts $P_i^-$ et $P_i^+$ "non fiables", c'est-à-dire tels que $\min\{\nu_i^-, \nu_i^+\} < \nu_F$, où $\nu_F$ est un paramètre de fiabilité fixé à l'avance.

# CHAPITRE 3

# REVUE DE LITTÉRATURE SUR LES PROBLÈMES DE CONCEPTION DE RÉSEAUX

Les problèmes de conception de réseaux constituent une généralisation des problèmes d'optimisation de réseaux, dont le très célèbre problème de flot à coût minimum, un des classiques de la recherche opérationnelle.

Dans ce chapitre, nous effectuons un bref retour sur les problèmes d'optimisation de réseaux avant de décrire les différents types et formulations de problèmes de conception de réseaux, en particulier le problème de conception de réseaux à coûts fixes avec et sans capacités. Nous présentons aussi certaines méthodes de résolution proposées dans la littérature.

## 3.1 Classe des problèmes d'optimisation de réseaux

Les problèmes d'optimisation de réseaux constituent une classe importante de problèmes de la recherche opérationnelle ayant des applications en transport et en télécommunications, et se positionnant à la frontière entre la programmation linéaire et la programmation en nombres entiers.

Dans cette section, nous nous intéressons particulièrement au problème de flot à coût minimum et ses variantes. Nous rappelons par la suite quelques algorithmes connus pour résoudre ce type de problèmes.

### 3.1.1 Formulation et variantes

Étant donné un graphe orienté $G = (N, A)$ de $|N|$ sommets et $|A|$ arcs $(i, j)$, auxquels sont associés un coût $c_{ij}$ par unité de flot transportée, une capacité $u_{ij}$ et une borne inferieure $l_{ij}$ sur la quantité de flot.

À chaque nœud $i \in N$ du graphe $G$ est associée une demande de flot $b_i$, positive s'il s'agit d'un nœud origine, négative si $i$ est une destination, et nulle si c'est un nœud de

transfert.

Le problème de flot à coût minimum consiste à déterminer un coût minimum de transport du flot sur le réseau tout en satisfaisant la demande aux nœuds destinations à partir des nœuds origines. Les variables de décision $x_{ij}$ représentent alors le flot sur chaque arc $(i, j)$. De plus, les données du problème doivent satisfaire la relation $\sum_{i \in N} b_i = 0$. Le modèle est formulé comme suit :

$$min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{3.1}$$

sujet à

$$\sum_{i/(i,j) \in A} x_{ij} - \sum_{i/(j,i) \in A} x_{ji} = b_i, \;\; i \in N \tag{3.2}$$

$$l_{ij} \leqslant x_{ij} \leqslant u_{ij}, \;\; (i,j) \in A \tag{3.3}$$

Les contraintes (3.2) sont dites contraintes de conservation de flot. Elles signifient que le flot net à un nœud (sortant moins entrant) doit être égal à la demande à ce nœud. Les contraintes (3.3) sont des contraintes de capacité et de borne inférieure sur chaque arc.

Une des variantes les plus connues de ce modèle est le problème du plus court chemin entre deux sommets $s$ et $t$. Le flot est dirigé d'un nœud source $s$ vers un nœud puits $t$ en choisissant un chemin qui minimise le coût de transport. Dans ce cas nous avons $b_s = 1$, $b_t = -1$, et $b_i = 0 \; \forall i \in N \setminus \{s,t\}$. Si, de plus, nous fixons $l_{ij}$ à 0 et exigeons que $u_{ij} \geq 1$, la solution optimale permettra alors d'envoyer une unité de flot de $s$ à $t$ à coût minimum.

Une autre variante est le problème de flot multi-produit qui survient lorsque plusieurs produits sont à transporter sur le réseau. Chaque produit $k$ a un ensemble de nœuds origines $O(k)$ et un ensemble de nœuds destinations $D(k)$. L'objectif est d'assurer une meilleure allocation de la capacité sur chaque arc aux différents produits qui y circulent, de manière à minimiser le coût global de transport du flot, tous produits confondus. Le modèle correspondant est alors formulé comme suit :

$$min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k \tag{3.4}$$

sujet à :

$$\sum_{i/(i,j)\in A} x_{ij}^k - \sum_{i/(j,i)\in A} x_{ji}^k = \begin{cases} b_i^k & i \in O(k) \\ -b_i^k & i \in D(k) \\ 0 & autrement \end{cases} \quad i \in N,\ k \in K \tag{3.5}$$

$$\sum_{k\in K} x_{ij}^k \leqslant u_{ij},\ \ (i,j) \in A \tag{3.6}$$

$$l_{ij}^k \leqslant x_{ij}^k \leqslant u_{ij}^k,\ \ k \in K,\ \ (i,j) \in A \tag{3.7}$$

$u_{ij}$ est la capacité totale de l'arc $(i,j)$, tandis que $u_{ij}^k$ est le maximum de capacité de l'arc $(i,j)$ alloué au produit $k$.

### 3.1.2 Revue de littérature sur les algorithmes de résolution

Sous certaines conditions et hypothèses, il existe des méthodes classiques exactes pour résoudre le problème de flot à coût minimum et certaines de ses variantes dans un temps polynomial.

L'algorithme de Dijkstra [27] permet de résoudre le problème du plus court chemin sur un réseau connexe, ayant des poids non négatifs sur les arcs. Un autre algorithme, celui de Bellman-Ford, autorisant la présence d'arcs de poids négatifs, permet de détecter l'existence de circuits absorbants, c'est-à-dire de poids total négatif, accessibles depuis le sommet source. Lorsqu'aucun circuit absorbant n'est trouvé, l'algorithme se termine par une solution optimale au problème.

Le problème de flot à coût minimum peut être résolu par la méthode du simplexe standard. Il existe cependant une méthode du simplexe adaptée aux réseaux qui permet de résoudre le problème 200 à 300 fois plus rapidement [11]. Il se peut toutefois qu'elle requiert un temps exponentiel de résolution pour certaines classes de problèmes.

D'autres approches basées sur la méthode duale du simplexe ont été considérées. Bertsekas [14] a, par exemple, développé un algorithme primal dual qui domine celui du simplexe pour certaines instances du problème.

Plusieurs autres algorithmes polynomiaux et pseudo-polynomiaux ont été proposés dans la littérature. Nous référons le lecteur aux travaux de Ahuja et al. [5] pour une revue détaillée de ces méthodes, ainsi qu'à ceux de Ahuja et al. [4] pour différentes variantes

et applications du problème.

## 3.2 Problème général de conception de réseaux

Tandis que dans les modèles d'optimisation de réseaux, le graphe $G = (N,A)$ est prédéfini, dans les problèmes de conception de réseaux, nous ne disposons que d'un ensemble d'arcs potentiels. L'objectif est alors de déterminer à la fois un sous-ensemble d'arcs qui vont constituer le réseau $G$, et le flot optimal à y faire circuler.

### 3.2.1 Modèle général

Nous présentons une formulation générale du problème de conception de réseaux dans l'espace des arcs. Nous reprenons, avec quelques modifications, le modèle présenté par Crainic et al. [23]. Nous nous inspirons aussi des formulations décrites par Ahuja et al. [4] et Magnanti et Wong [50].

Nous considérons un graphe orienté $G = (N,A)$, où $N$ est l'ensemble des nœuds et $A$ l'ensemble des arcs. $K$ est l'ensemble de produits $k$ à faire circuler sur le réseau, ayant chacun un ensemble de nœuds origines $O(k)$ et un ensemble de nœuds destinations $D(k)$. À chaque nœud $i \in O(k)$ est associée l'offre $o_i^k$ du produit $k$, et pour tout $i \in D(k)$ est définie la demande $d_i^k$ du produit $k$. Nous supposons de plus que :

$$\sum_{i \in O(k)} o_i^k = \sum_{i \in D(k)} d_i^k, \ \ \forall k \in K \tag{3.8}$$

Nous définissons aussi l'ensemble $L$ des équipements à installer sur chaque arc. Le coût de transfert du flot du produit $k$ sur l'arc $(i,j)$ est noté $c_{ij}^k$, tandis que le coût d'installation de l'équipement $l$ sur l'arc $(i,j)$ est noté $f_{ij}^l$.

Nous considérons la possibilité que le flot du produit $k$ sur l'arc $(i,j)$ soit limité par une constante $b_{ij}^k$, et que le nombre d'équipements de type $l$ à installer sur l'arc $(i,j)$ ne puisse dépasser une borne $h_{ij}^l$. Nous supposons de plus que chaque arc $(i,j)$ possède une capacité initiale $v_{ij}$, et affectons à chaque équipement $l$ installé sur un arc $(i,j)$ une capacité $u_{ij}^l$. Le flot total (de tous les produits) à envoyer sur l'arc $(i,j)$ ne doit pas

excéder la capacité totale des arcs. Étant donné que chaque produit peut avoir sa propre pondération dans le calcul de ce flot total, nous définissons pour chaque arc $(i,j)$ une constante de pondération $e_{ij}^k$ par produit $k$.

Enfin, nous définissons les variables du modèle. Nous représentons la quantité de flot du produit $k$ à transporter sur l'arc $(i,j)$ par $x_{ij}^k$ (variables de flot), et le nombre d'équipements $l$ à installer sur l'arc $(i,j)$ par $y_{ij}^l$ (variables de conception).

L'objectif du problème est de minimiser une fonction de coût $\phi(x,y)$ en terme des variables $x$ et $y$ et dépendant des coûts fixes $f_{ij}^l$ et variables $c_{ij}^k$.

En notant $N^+(i) = \{j \in N \,/\, (i,j) \in A\}$ et $N^-(i) = \{j \in N \,/\, (j,i) \in A\}$, le problème est modélisé comme suit :

$$min \;\; \phi(x,y) \tag{3.9}$$

*sujet à :*

$$\sum_{j \in N^+(i)} x_{ij}^k - \sum_{j \in N^-(i)} x_{ji}^k = \begin{cases} o_i^k & i \in O(k), \\ -d_i^k & i \in D(k), \\ 0 & autrement \end{cases} \qquad i \in N, \;\; k \in K \tag{3.10}$$

$$\sum_{k \in K} e_{ij}^k x_{ij}^k \leqslant v_{ij} + \sum_{l \in L} u_{ij}^l y_{ij}^l, \quad (i,j) \in A \tag{3.11}$$

$$(x,y) \in \mathscr{S} \tag{3.12}$$

$$0 \leqslant x_{ij}^k \leqslant b_{ij}^k, \qquad (i,j) \in A, \;\; k \in K \tag{3.13}$$

$$0 \leqslant y_{ij}^l \leqslant h_{ij}^l, \qquad (i,j) \in A, \;\; l \in L \tag{3.14}$$

$$y_{ij}^l \;\; entier, \quad (i,j) \in A, \;\; l \in L. \tag{3.15}$$

Les contraintes (3.10) sont les contraintes de conservation de flot, et (3.11) celles de capacité. Les contraintes (3.12) représentent toute contrainte additionnelle sur les variables $x$ ou $y$. Il peut être question, par exemple, de contraintes de budget, c'est-à-dire que le coût global d'installation des équipements ne doit pas dépasser un budget

fixé. Il est possible aussi d'exiger certaines restrictions topologiques sur le réseau, de manière à avoir un arbre sous-tendant ou toute autre structure particulière, ou de forcer l'introduction de certains arcs dans le réseau.

### 3.2.2 Variantes et spécifications

Plusieurs types de problèmes peuvent être dérivés de la formulation générique présentée dans la section précédente.

Notons d'abord que, puisque toutes les contraintes sont linéaires, lorsque $\phi$ est linéaire et tient compte des coûts fixes d'installation, nous obtenons une formulation du modèle en problème linéaire en nombres entiers mixte, dit "problème de conception de réseaux à coûts fixes" :

$$\phi(x,y) = \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}^k x_{ij}^k + \sum_{l\in L}\sum_{(i,j)\in A} f_{ij}^l y_{ij}^l \tag{3.16}$$

Par ailleurs, dans plusieurs problèmes traités, certaines hypothèses sont considérées, donnant lieu à différents modèles.

Ainsi, dans certaines applications, chaque produit ne possède qu'une seule origine et une seule destination ($|O(k)| = |D(k)| = 1$).

D'autres hypothèses simplificatrices impliquent l'élimination des bornes supérieures sur les variables $x$ ou $y$, ou la fixation de la borne supérieure des variables $y$ à 1, ce qui donne lieu à un problème en variables binaires.

$$h_{ij}^l = 1, \ \ \forall l \in L, \ \ (i,j) \in A \tag{3.17}$$

La question qui se pose n'est donc plus de déterminer le nombre d'équipements de différents types à installer sur chaque arc, mais plutôt de savoir s'il serait profitable ou non d'installer tel équipement sur tel arc.

Les pondérations des produits dans le calcul du flot total passant par un arc sont en

général négligées. Nous avons alors :

$$e_{ij}^{k} = 1, \ \ \forall k \in K, \ \ (i,j) \in A \tag{3.18}$$

En outre, dans le modèle que nous avons présenté, même si aucun équipement n'est installé sur l'arc $(i,j)$, il possède tout de même une capacité initiale $v_{ij}$ qui permet d'y faire circuler une certaine quantité de flot. Cette considération n'est pas permise dans certains cas où l'on ne veut faire passer du flot que s'il existe un équipement sur l'arc, autrement dit, uniquement si l'arc est ouvert. Ceci revient à faire l'hypothèse suivante :

$$v_{ij} = 0, \ \ \forall (i,j) \in A \tag{3.19}$$

Cette réflexion en terme d'arcs s'étend aussi aux types d'équipements. Introduire plusieurs types d'installations sur un arc $(i,j)$ revient en fait à créer plusieurs arcs parallèles. Si cette situation n'est pas envisageable dans le cadre d'une application, il est possible de ne considérer qu'un seul type d'équipements ($|L| = 1$) et d'éliminer l'indice $l$ du modèle.

Il est aussi possible de considérer une version du problème sans capacité en fixant les variables $v_{ij}$ et $u_{ij}^{l}$ à des valeurs assez grandes de manière à ce que la capacité totale de chaque arc dépasse tout flot possible sur cet arc.

Enfin, les contraintes (3.12) ne sont généralement considérées que dans des cas particuliers. Parmi ces cas, le problème d'optimisation de flot multi-produits qu'on retrouve après fixation de toutes les variables $y$ à la valeur 1. La contrainte (3.12) correspond alors à :

$$y_{ij}^{l} = 1, \ \ \forall l \in L, \ \ (i,j) \in A \tag{3.20}$$

## 3.3 Problème de conception de réseaux à coûts fixes

Dans cette section, nous présentons deux versions du problème sans et avec capacité. Après avoir détaillé leurs formulations, nous donnons un aperçu sur leurs méthodes de résolutions heuristiques et exactes présentes dans la littérature.

### 3.3.1 Problème de conception de réseau à coûts fixes et sans capacité

Les problèmes de conception de réseaux sans contraintes de capacité permettent de modéliser les systèmes qui fonctionnent largement au dessous de leurs capacités de telle manière que l'effet de congestion est négligeable. Ce type de problèmes a plusieurs applications. Magnanti et Wong [50] en dressent une liste représentative.

#### 3.3.1.1 Formulations

Nous reprenons les notations de la section précédente, et en plus des hypothèses que $|O(k)| = |D(k)| = 1$, $o^k = d^k$, $b_{ij}^k$ est infinie, $e_{ij}^k = 1$, $v_{ij} = 0$ et $|L| = 1$, nous supposons que les variables $y$ sont binaires, et nous éliminons les contraintes additionnelles $(x, y) \in \mathscr{S}$.

Nous supposons de plus que $u_{ij} = \sum_{k \in K} d^k$, hypothèse nécessaire pour que le modèle soit sans capacité. Nous le présentons ainsi :

$$min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.21}$$

*sujet à :*

$$\sum_{j \in N^+(i)} x_{ij}^k - \sum_{j \in N^-(i)} x_{ji}^k = \begin{cases} d^k & i = O(k), \\ -d^k & i = D(k), \\ 0 & autrement \end{cases} \qquad i \in N,\ k \in K \tag{3.22}$$

$$\sum_{k \in K} x_{ij}^k \leqslant u_{ij} y_{ij}, \quad (i,j) \in A \tag{3.23}$$

$$x_{ij}^k \geqslant 0, \qquad (i,j) \in A,\ k \in K \tag{3.24}$$

$$0 \leqslant y_{ij} \leqslant 1, \qquad (i,j) \in A \tag{3.25}$$

$$y_{ij}\ \ entier, \quad (i,j) \in A. \tag{3.26}$$

Dans ce cas sans capacité, puisque $u_{ij} \geqslant d^k$ et $x_{ij}^k \leqslant d^k$, $\forall k \in K$, $(i,j) \in A$, les

contraintes (3.23) peuvent être désagrégées en $|K|$ contraintes de la forme :

$$x_{ij}^k \leqslant d^k y_{ij}, \quad k \in K, \ \ (i,j) \in A \tag{3.27}$$

Les deux formulations, sans et avec désagrégation des contraintes de capacité sont équivalentes. La seconde est cependant plus profitable pour le modèle en terme de possibilités de résolution puisque sa relaxation linéaire fournit de meilleures bornes inférieures.

#### 3.3.1.2 Revue de littérature sur les méthodes de résolution

Nous présentons une revue de littérature des principales méthodes heuristiques et exactes. Nous rappelons que cette partie est principalement inspirée du travail de Magnanti et Wong [50] et des références citées dans cet article.

**3.3.1.2.1 Méthodes exactes** Il existe plusieurs méthodes conçues pour résoudre exactement le problème de conception de réseaux à coûts fixes et sans contraintes de capacité. Parmi ces méthodes, on distingue celles basées sur la décomposition de Benders [12], le *branch-and-bound* et la relaxation lagrangienne.

Magnanti et al. [53] ont montré que plusieurs inégalités présentées dans la littérature pour améliorer la borne inférieure peuvent être vues comme des coupes de Benders. Ils ont proposé une méthode combinant la décomposition de Benders à la méthode d'ascension duale et ont dérivé de nouvelles coupes de Benders pour l'accélérer.

La méthode de Benders s'est aussi montrée prometteuse dans certaines applications du modèle tel que le problème de conception de réseaux de distribution [36], la modélisation de la circulation des avions [58], et la programmation des horaires des trains [29].

Balakrishnan et al. [10] ont présenté par la suite une famille d'algorithmes d'ascension duale pour résoudre le même problème, et les ont appliqué à des instances de grandes tailles. Ils ont montré de plus que ces algorithmes, lorsque combinés à des heuristiques du type suppression-ajout (*drop-add*), permettent d'atteindre des niveaux d'op-

timalité de 1 à 4 % dans un temps assez court.

La méthode de *branch-and-bound* a été plus utilisée pour des problèmes sans capacité avec contrainte de budget. Boyce et al. [16] ont dérivé un type d'inégalités valides basées sur la fonction de coût, qui sont générées et introduites dans le modèle au fur et à mesure de l'avancement de l'algorithme de *branch-and-bound*. Ils ont obtenu de bons résultats pour des instances de taille relativement moyenne.

Hoang [43] a par la suite essayé d'améliorer ces inégalités valides par un terme additionnel dépendant de $y$. Il a proposé de calculer la borne inférieure en relaxant les contraintes d'intégralité et en minimisant sur l'ensemble défini par les contraintes de budget, ce qui correspond à la résolution d'un problème de sac-à-dos. La borne inférieure obtenue a permis d'élaguer plus de nœuds, et d'accélérer l'algorithme de *branch-and-bound*.

Dionne et Florian [28] ont tenté d'améliorer l'algorithme de *branch-and-bound* et la méthode de calcul du terme additionnel proposé par Hoang. Pour ce faire, ils ont proposé de brancher sur les variables $y$ qui présentent la meilleure amélioration de l'objectif par unité de budget. Cette modification a permis d'améliorer considérablement le temps de résolution.

Pour les trois algorithmes, cependant, les auteurs ont constaté qu'ils performent mieux pour des instances de tailles moyennes, et que pour des instances plus grandes, des méthodes heuristiques seraient plus adaptées.

Finalement, Hellstrand et al. [40] se sont penché sur la dérivation de nouvelles inégalités valides et l'approximation de l'enveloppe convexe. Il ont montré en particulier que le polytope constituant la relaxation linéaire de la formulation forte du problème (c'est-à-dire incluant les inégalités fortes, voir section 3.3.2.1.2) est quasi-entier, dans le sens où tout chemin reliant les arêtes de l'enveloppe convexe est aussi un chemin à travers les arêtes de ce polytope.

**3.3.1.2.2 Méthodes heuristiques** Magnanti et Wong [50] ont montré, à travers plusieurs applications, qu'il est facile d'obtenir des solutions réalisables pour le problème de conception de réseaux à coûts fixes et sans capacité en utilisant des heuristiques clas-

siques.

En effet, étant donné que certaines applications du problème sont *NP*-difficiles, plusieurs auteurs se sont intéressés aux méthodes heuristiques pour les résoudre. Ces dernières peuvent fournir des bornes supérieures permettant d'accélérer le temps requis par les méthodes exactes.

Trois types d'heuristiques sont principalement connus. Les heuristiques d'ajout (*add heuristic*), de suppression (*drop heuristic*), et de suppression-ajout (*drop-add heuristic*). Le premier type commence par une structure réalisable du réseau et ajoute, à chaque itération, l'arc dont l'introduction permettra la meilleure amélioration de la fonction de coût. À l'inverse, le second type commence par un réseau contenant tous les arcs potentiels et procède, à chaque itération, à la suppression de l'arc qui permet la moindre dégradation de la fonction de coût. Le troisième type est une combinaison des deux autres. Il permet de supprimer et ajouter des arcs itérativement jusqu'à ce qu'aucune amélioration de l'objectif ne soit possible.

La majorité des heuristiques présentes dans la littérature sont des combinaisons des trois types. Nous citons particulièrement Billheimer et Gray [15] et Los et Lardinois [49].

### 3.3.2 Problème de conception de réseaux à coûts fixes et capacités

#### 3.3.2.1 Formulations

La formulation que nous présentons dans cette section est similaire à celle présentée pour le modèle sans contraintes de capacité. Il n'est cependant pas possible de désagréger les contraintes de capacité, puisque la relation $u_{ij} = \sum_{k \in K} d^k$ n'est plus vérifiée.

$$min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{3.28}$$

*sujet à :*

$$\sum_{j\in N^+(i)} x_{ij}^k - \sum_{j\in N^-(i)} x_{ji}^k = \begin{cases} d^k & i = O(k), \\ -d^k & i = D(k), \\ 0 & autrement \end{cases} \qquad i \in N,\ k \in K \tag{3.29}$$

$$\sum_{k\in K} x_{ij}^k \leqslant u_{ij} y_{ij}, \quad (i,j) \in A \tag{3.30}$$

$$x_{ij}^k \geqslant 0, \qquad (i,j) \in A,\ k \in K \tag{3.31}$$

$$0 \leqslant y_{ij} \leqslant 1, \qquad (i,j) \in A \tag{3.32}$$

$$y_{ij}\ \ entier, \quad (i,j) \in A. \tag{3.33}$$

Le modèle ainsi présenté est formulé dans l'espace des arcs. Une autre formulation équivalente, dans l'espace des chemins, existe aussi. Nous la présentons en se référant à celle décrite par Crainic [20].

**3.3.2.1.1 Formulation dans l'espace des chemins** Soit $P^k$ l'ensemble des chemins possibles $p$ du produit $k$, et définissons $\delta_{ij}^{kp}$ comme suit :

$$\delta_{ij}^{kp} = \begin{cases} 1 & si\ \ (i,j) \in p,\ \ p \in P^k, \\ 0 & sinon. \end{cases} \tag{3.34}$$

Considérons de plus les transformations suivantes :

$$\zeta_p^k = \sum_{(i,j)\in A} c_{ij}^k \delta_{ij}^{kp} \tag{3.35}$$

$$\chi_p^k = \sum_{(i,j)\in A} x_{ij}^k \delta_{ij}^{kp} \tag{3.36}$$

$\chi_p^k$ représente le flot du produit $k$ sur le chemin $p$, tandis que $\zeta_p^k$ représente le coût de transport du produit $k$ sur le chemin $p$. Le modèle résultant est le suivant :

$$min \sum_{k\in K}\sum_{p\in P}\zeta_p^k\chi_p^k + \sum_{(i,j)\in A} f_{ij}y_{ij} \tag{3.37}$$

*sujet à :*

$$\sum_{p\in P^k}\chi_p^k = d^k, \quad k\in K \tag{3.38}$$

$$\sum_{k\in K}\sum_{p\in P^k}\delta_{ij}^{kp}\chi_p^k \leqslant u_{ij}y_{ij}, \quad (i,j)\in A \tag{3.39}$$

$$\chi_p^k \geqslant 0, \qquad k\in K,\ \ p\in P^k \tag{3.40}$$

$$0\leqslant y_{ij}\leqslant 1, \qquad (i,j)\in A \tag{3.41}$$

$$y_{ij}\ \ entier, \quad (i,j)\in A. \tag{3.42}$$

Il s'agit en fait d'une reformulation de Dantzig-Wolfe. Par les relations (3.35) et (3.36), nous retrouvons les mêmes contraintes et fonction objectif que celles présentes dans la formulation dans l'espace des arcs. Cette formulation a cependant reçu moins d'attention dans le cadre des problèmes de conception de réseaux et ce, contrairement aux problèmes d'optimisation de flots où elle a été plus étudiée.

Finalement, la comparaison des deux formulations montre que leurs relaxations linéaires fournissent les mêmes bornes inférieures [57]. De plus, vu le grand nombre de chemins possibles, la méthode de génération de colonnes reste la plus adaptée à cette formulation. Crainic et al. [22] ont proposé de la combiner à une méthode heuristique pour résoudre le problème.

**3.3.2.1.2 Formulation forte** Afin d'améliorer la borne inférieure, il est possible d'introduire des inégalités valides dans la formulation dans l'espace des arcs. En particulier, nous obtenons une meilleure formulation suite à l'introduction d'un type d'inégalités dites "inégalités fortes" dans le modèle.

$$x_{ij}^k \leqslant d^k y_{ij}, \quad k\in K, \quad (i,j)\in A \tag{3.43}$$

Notons que ces inégalités sont redondantes pour le problème en nombres entiers, mais ne le sont pas pour sa relaxation linéaire. Elles permettent d'améliorer significativement la borne inférieure. Pour cette raison, on réfère souvent dans la littérature à la formulation dans l'espace des arcs sans inégalités fortes par la "formulation faible" et à celle avec les inégalités fortes par la "formulation forte".

#### 3.3.2.2 Revue de littérature sur les méthodes de résolution

Nous donnons dans cette partie un aperçu sur quelques méthodes exactes et heuristiques présentées dans la littérature pour résoudre le problème de conception de réseaux à coûts fixes et capacités. Nous soulignons que le problème est *NP*-difficile, et que la majorités des méthodes sont appliquées à la formulation forte du problème.

**3.3.2.2.1 Méthodes exactes** Parmi les principales méthodes exactes utilisées pour résoudre ce type de problèmes figurent les méthodes de coupes basées sur l'algorithme du simplexe, la décomposition de Benders, ainsi que la méthode de relaxation lagrangienne.

La première méthode bénéficie en fait de l'existence de logiciels informatiques permettant de résoudre la relaxation linéaire, et d'améliorer la qualité de la borne inférieure grâce à leur capacité à générer plusieurs types d'inégalités valides. Elle est cependant indifférente aux particularités que peut présenter chaque problème.

Il est possible de catégoriser les inégalités valides dérivées pour le problème en cinq groupes. On cite les inégalités fortes, qui sont d'un nombre assez grand mais polynomial, les inégalités de couverture, de cardinalité minimale, de couverture de flot (*flow cover*), et d'empaquetage de flot (*flow pack*), dont il existe un nombre exponentiel.

Dans la littérature, plusieurs travaux se sont intéressé à la dérivation d'inégalités valides et à l'amélioration de la formulation plutôt qu'à la résolution elle-même. Chouman et al. [17] ont proposé une méthode où les coupes sont d'abord générées avant que le modèle ne soit résolu par un algorithme de *branch-and-bound*. Après avoir testé la méthode avec les différents types d'inégalités valides, ils ont constaté que les inégalités de couverture de flot et d'empaquetage de flot permettent de réduire l'écart à la solution

optimale de manière efficace. Les inégalités fortes restent tout de même plus efficientes en terme de temps de calcul.

Concernant la décomposition de Benders, la méthode classique présente certaines faiblesses, notamment vu que le problème maître de Benders est un problème en nombres entiers. Il existe alors des tentatives d'améliorer la performance de l'algorithme en l'appliquant à la relaxation forte du modèle [18, 19].

La troisième méthode, quant à elle, permet d'exploiter la structure particulière que présente chaque problème pour résoudre des problèmes linéaires de plus petite taille. Elle contribue aussi, dans plusieurs cas, à la conception des méthodes heuristiques, et à la dérivation d'inégalités valides. Elle connait dans ce sens un succès particulier. Son inconvénient réside par contre dans la perte de la structure du sous-problème lagrangien suite à l'introduction des inégalités valides.

Gendron et Crainic [33] ont testé deux types de relaxations lagrangiennes. La première consiste à relaxer toutes les contraintes liantes (fortes et celles de capacité). Pour la seconde, ce sont les contraintes de flot qui sont relaxées. Ils montrent que dans les deux cas, la borne inférieure théorique obtenue est la même.

Crainic et al. [21] ont par la suite comparé plusieurs méthodes de sous-gradient et de faisceaux appliquées à ces différentes relaxations du problème, et ont montré que les méthodes de faisceaux sont plus robustes pour les deux types de relaxation, et convergent plus rapidement vers une solution optimale du problème dual lagrangien.

D'autres travaux ont été réalisés dans le même sens. Nous citons Sellmann et al. [59] et Kliewer et Timajev [47].

Plus récemment, Gendron et Larose [34] ont proposé une méthode de *branch-and-cut-and-price* pour résoudre le problème qu'ils initialisent avec une heuristique. Ils ont obtenu des résultats intéressants, notamment dans le cas des instances de grande taille.

**3.3.2.2.2 Méthodes heuristiques** Dans le cadre des méthodes heuristiques, Holmberg et Yuan [44] ont proposé une heuristique qui utilise une relaxation lagrangienne des contraintes de flot pour obtenir des sous-problèmes faciles à résoudre. La méthode du sous-gradient a été utilisée pour résoudre le dual lagrangien, et le problème est par la

suite résolu par *branch-and-bound*. La méthode permet de trouver de bonnes solutions réalisables pour des problèmes de grande taille dans un temps raisonnable.

Comme nous l'avons déjà cité, Crainic et al. [22] ont proposé une heuristique Tabou basée sur une combinaison de la méthode du simplexe et la génération de colonnes pour résoudre la formulation du problème dans l'espace des chemins. Ils ont obtenu de très bons résultats pour des instances de taille réelle.

Plusieurs autres méthodes proposées combinent des méthodes exactes et heuristiques, telles que celles présentées par Hewitt et al. [41], et par Kim et Barnhart [45] et Kim et al. [46], où une méthode heuristique permet de réduire la taille du problème avant de lui appliquer un algorithme de *branch-and-cut-and-price*.

Nous référons le lecteur intéressé par une littérature plus détaillée aux travaux de Crainic [20], Crainic et al. [23], Gendron [32] et Larose [48], références sur lesquelles nous nous sommes basés pour dresser cet aperçu sur les méthodes de résolution de problèmes de conception de réseaux à coûts fixes et capacités.

# CHAPITRE 4

# GÉNÉRATION DE COLONNES POUR LE PROBLÈME DE CONCEPTION DE RÉSEAUX AVEC COÛTS D'AJOUT DE CAPACITÉ

Contrairement aux problèmes de conception de réseaux à coûts fixes et capacités qui ont été largement étudiés, les problèmes avec coût d'ajout de capacité ont reçu peu d'attention. Ils n'ont été abordés avec intérêt que durant les deux dernières décennies, avec l'avènement du problème dans le domaine des réseaux de télécommunication.

Dans les sections suivantes, nous présentons d'abord deux formulations du problème et détaillons celle à laquelle notre méthode est appliquée. Nous passons par la suite en revue la littérature portant sur le problème et les méthodes proposées pour le résoudre. Enfin, nous décrivons de manière détaillée la méthode que nous avons développé et le cadre de son application.

## 4.1 Formulation et littérature du problème de conception de réseaux avec coûts d'ajout de capacité

Nous présentons deux formulations du problème, l'une avec des variables discrètes générales et l'autre avec des variables binaires. Nous nous inspirons des modèles présentés par Frangioni et Gendron [30] dans le cadre d'un travail récent portant sur le même type de problèmes.

### 4.1.1 Description du modèle

À la différence du problème de conception de réseaux à coûts fixes et capacités, pour lesquels un seul type d'arcs est permis, le problème avec coûts d'ajout de capacité permet l'introduction d'arcs parallèles ayant leurs propres capacités et coûts variables. L'objectif est donc de déterminer pour chaque paire de nœuds $(i, j)$ le nombre d'arcs qui vont les lier, ou en d'autres termes, le nombre d'équipements qui y seront installés. Ceci peut être modélisé par des variables générales $y \in \mathbb{Z}^+$ au lieu des variables binaires

considérées dans le cas du problème à coûts fixes.

En reprenant les notations du chapitre précédent, sauf pour les variables de flot $x_{ij}^k$, que nous considérons dans ce chapitre représenter la fraction (plutôt que la quantité) de flot du produit $k$ à faire passer par l'arc $(i, j)$, nous obtenons la formulation suivante du problème :

$$min \sum_{k \in K} \sum_{(i,j) \in A} d^k c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} y_{ij} \tag{4.1}$$

*sujet à :*

$$\sum_{j \in N^+(i)} x_{ij}^k - \sum_{j \in N^-(i)} x_{ji}^k = \begin{cases} 1 & i = O(k), \\ -1 & i = D(k), \\ 0 & autrement \end{cases} \qquad i \in N, \ k \in K \tag{4.2}$$

$$\sum_{k \in K} d^k x_{ij}^k \leqslant u_{ij} y_{ij}, \quad (i, j) \in A \tag{4.3}$$

$$0 \leqslant x_{ij}^k \leqslant 1, \qquad (i, j) \in A, \ k \in K \tag{4.4}$$

$$y_{ij} \geqslant 0, \qquad (i, j) \in A \tag{4.5}$$

$$y_{ij} \ \ entier, \quad (i, j) \in A. \tag{4.6}$$

Il est intéressant de souligner que les inégalités fortes, qui performent bien dans le cas de problèmes à coûts fixes et capacités, ne le sont plus pour le problème avec coûts d'ajout de capacité. Ceci se justifie par le fait que les variables de conception $y$ ne sont plus binaires, et peuvent donc prendre des valeurs supérieures à 1, tandis que les variables de flot $x$ sont bornées par la valeur 1. Ces inégalités perdent alors leur utilité et deviennent redondantes.

### 4.1.2 Reformulation du problème en variables binaires

Comme alternative à la formulation présentée, il est possible de modéliser le problème avec une fonction de coût linéaire par morceaux. Le problème résultant est non linéaire, mais peut être linéarisé afin d'obtenir un problème linéaire en nombres entiers

mixte.

Dans la suite, nous présentons une des reformulations utilisant des variables binaires, le "modèle de choix multiple", initialement dérivé pour des problèmes d'optimisation de flots. L'objectif de cette reformulation est d'améliorer la performance de certaines approches exactes et heuristiques lorsqu'elles sont appliquées au problème.

Nous introduisons les changements de variables suivants, étant donné l'ensemble $S_{ij}$ des valeurs possibles (non nulles) de $y_{ij}$ :

$$y_{ij}^{s} = \begin{cases} 1 & si \;\; y_{ij} = s, \\ 0 & sinon. \end{cases} \tag{4.7}$$

$$x_{ij}^{ks} = \begin{cases} x_{ij}^{k} & si \;\; y_{ij} = s, \\ 0 & sinon. \end{cases} \tag{4.8}$$

Le cardinal de $S_{ij}$, qui correspond à la plus grande valeur que peut prendre la variable $y_{ij}$, est donné par la formule suivante :

$$T_{ij} = \left\lceil \frac{\sum_{k \in K} d^{k}}{u_{ij}} \right\rceil \tag{4.9}$$

Le modèle reformulé peut alors être présenté ainsi :

$$min \sum_{k \in K} \sum_{(i,j) \in A} \sum_{s \in S_{ij}} d^{k} c_{ij}^{k} x_{ij}^{ks} + \sum_{(i,j) \in A} \sum_{s \in S_{ij}} s f_{ij} y_{ij}^{s} \tag{4.10}$$

*sujet à :*

$$\sum_{j \in N^{+}(i)} \sum_{s \in S_{ij}} x_{ij}^{ks} - \sum_{j \in N^{-}(i)} \sum_{s \in S_{ji}} x_{ji}^{ks} = \begin{cases} 1 & i = O(k), \\ -1 & i = D(k), \\ 0 & autrement \end{cases} \quad i \in N, \;\; k \in K \tag{4.11}$$

$$\sum_{k \in K} d^{k} x_{ij}^{ks} \leqslant s u_{ij} y_{ij}^{s}, \quad (i,j) \in A, \;\; s \in S_{ij} \tag{4.12}$$

$$\sum_{s \in S_{ij}} y_{ij}^{s} \leqslant 1, \quad (i,j) \in A \tag{4.13}$$

$$x_{ij}^{ks} \geqslant 0, \qquad k \in K, \ (i,j) \in A, \ s \in S_{ij} \tag{4.14}$$

$$y_{ij}^{s} \geqslant 0, \qquad (i,j) \in A, \ s \in S_{ij} \tag{4.15}$$

$$y_{ij}^{s} \ \ entier, \quad (i,j) \in A, \ s \in S_{ij}. \tag{4.16}$$

Remarquons que les contraintes $y_{ij}^{s} \leqslant 1$ sont implicitement représentées par les inégalités (4.13).

Le problème ainsi modélisé peut être vu comme un problème avec équipements multiples, chacun ayant sa propre fonction de coût et sa capacité. Intuitivement, en effectuant cette transformation, nous reformulons l'objectif du problème de "déterminer le nombre $s$ d'équipements à installer sur un arc" à "déterminer si l'équipement de type $s$ serait installée sur l'arc". Nous retrouvons, sous quelques hypothèses, le modèle général présenté dans le chapitre précédent (section 3.2.1).

Pour renforcer le modèle, des inégalités valides dites "inégalités étendues" (*extended linking inequalities*) sont introduites.

$$x_{ij}^{ks} \leqslant y_{ij}^{s}, \qquad k \in K, \ (i,j) \in A, \ s \in S_{ij} \tag{4.17}$$

Ces inégalités sont semblables aux inégalités fortes développées pour le problème à coûts fixes et capacités. Nous référons alors, par abus de langage, à la formulation (4.10)-(4.16) renforcée des inégalités (4.17) par la formulation "forte" du problème en variables binaires.

### 4.1.3 Revue de littérature sur le problème

Citons d'abord le travail récent de Frangioni et Gendron [30], où ils ont proposé la reformulation du problème en variables 0-1 comme moyen pour améliorer le temps de résolution, et ont testé une méthode de génération de lignes et de colonnes (*column and row generation*) intégrée à une heuristique sur le modèle, et obtenu de bon résultats. D'autres efforts de recherche, quoique peu nombreux, ont été déployés pour résoudre

des problèmes similaires. Nous présentons dans cette section une revue de la littérature sur les principales méthodes proposées.

Certains travaux se sont concentrés sur l'amélioration de la borne inférieure du problème et la dérivation de nouvelles inégalités valides pouvant notamment remplacer les inégalités fortes pour la première formulation du problème. Dans ce cadre, Magnanti et al. [51] ont proposé les inégalités de capacité résiduelle (*residual capacity inequalities*) comme généralisation et alternative aux inégalités fortes.
Ces inégalités ont été par la suite étudiées par Atamtürk et Rajan [8] et Atamtürk et Günlük [7]. Vu leur nombre exponentiel, elles ont été utilisées dans le cadre des méthodes de coupes.

D'autres travaux ont étudié différentes applications et spécifications du problème. Agarwal [3] a présenté une version similaire du problème utilisée pour modéliser des systèmes de télécommunication, qu'il a proposé de résoudre avec une méthode heuristique de recherche locale.

En 1989 déjà, Minoux [54] avait présenté le problème sous le nom "*Optimum rented lines network problem*", et avait fait référence à la méthode de résolution proposée par Hansen [39] sous certaines hypothèses, dont l'absence de contraintes de capacité. Il avait affirmé que sous ces conditions, le problème pouvait être formulé en problème de flot à coût minimum multi-produit.

Plus tard, Magnanti et al. [52] ont étudié une version du problème avec deux types d'équipements (*Network loading problem*). Ils ont calculé des bornes inférieures au problème en utilisant la relaxation lagrangienne, ainsi que des inégalités de coupure, et ont été capables de résoudre à l'optimum des instances de 15 nœuds.

Par la suite, Gabrel et Minoux [31] ont présenté une approche pour améliorer les bornes inférieures pour une version plus générale de la fonction de coût. Dahl et Stoer [24] ont également proposé une méthode de coupes initialisée par une heuristique pour une version particulière du problème (*survivable network*), apparaissant dans le domaine des réseaux informatiques, avec près de 100 nœuds et 4 types d'équipements.

Des travaux plus récents comprennent le développement de méthodes heuristiques [13, 35] et exactes [6, 38, 55].

## 4.2 Méthode de génération de colonnes pour la résolution du problème

Nous présentons dans cette partie une méthode de résolution développée pour les problèmes de conception de réseaux avec coût d'ajout de capacité. La méthode est basée sur la génération de colonnes, introduite dans un schéma général de *branch-and-bound*. Du moment que nous utilisons la formulation "forte" du problème en variables binaires, nous faisons appel aussi aux méthodes de génération de coupes.

Puisque la méthode combine ces trois algorithmes, nous l'appelons "méthode de *branch-and-price-and-cut*". Notons que cette méthode s'inspire d'une approche similaire proposée par Gendron et Larose pour les problèmes à coûts fixes et capacités [34].

### 4.2.1 Hypothèses et limites

Comme le montre le modèle (4.10)-(4.17), nous supposons que la fonction de coût pour l'ajout de capacité est linéaire de la forme :

$$f(y_{ij}^s) = s f_{ij} y_{ij}^s, \qquad (i,j) \in A, \;\; s \in S_{ij} \tag{4.18}$$

Rappelons aussi que nous avons choisi de résoudre la formulation "forte" en variables 0-1 du problème (4.10)-(4.17).

Pour la méthode que nous présentons, nous optons pour une approche exacte de résolution qui combine génération de colonnes, vu le très grand nombre de variables résultant de la désagrégation des variables $y$, et génération de coupes, expliquée par le très grand nombre de contraintes additionnelles (4.17). Le tout est introduit dans l'algorithme classique de *branch-and-bound* appliqué à la relaxation linéaire.

Il est aussi possible d'initialiser notre méthode par des bornes supérieures obtenues par d'autres méthodes exactes ou heuristiques, afin d'améliorer le temps d'exécution, notamment pour les instances de grande taille.

### 4.2.2 Résolution par *branch-and-price-and-cut*

Nous présentons d'abord la manière dont l'arbre de recherche est construit et exploré. Nous présentons par la suite la méthode par laquelle la relaxation linéaire est résolue à chaque nœud, en utilisant la génération de colonnes et de coupes.

#### 4.2.2.1 Constitution de l'arbre de recherche

**4.2.2.1.1 Règle de branchement** Pour la dérivation des nœuds fils, nous choisissons la méthode dichotomique et branchons sur une variable binaire à la fois, ce qui revient à fixer, à chaque itération, une variable $y$ à 0 ou à 1. Quant à la sélection de la variable sur laquelle il faut brancher, nous utilisons la règle de branchement fiable (*reliable branching* [1]) dans l'objectif de faire le meilleur choix de variable dans un temps raisonnable.

**4.2.2.1.2 Méthode de recherche** Parmi les stratégies de recherche présentées dans le premier chapitre, nous optons pour la méthode de recherche meilleur d'abord. Notre motivation est de pouvoir obtenir de meilleures bornes inférieures dans un temps raisonnable, notamment pour des instances de grande taille, pour lesquelles une solution exacte est loin d'être trouvée.

#### 4.2.2.2 Évaluation d'un nœud

À chaque nœud de l'arbre de recherche, nous résolvons un sous-problème linéaire correspondant à la relaxation linéaire du problème initial, où certaines variables $y$ sont forcées à prendre des valeurs entières.

Pour ce faire, nous appliquons la méthode de génération de colonnes aux variables $x$ dans la formulation forte du sous-problème. La valeur optimale obtenue constitue une borne inférieure sur la valeur optimale du problème initial. Nous utilisons par la suite la génération de coupes pour améliorer cette borne inférieure en générant les inégalités "étendues" violées itérativement.

Nous faisons appel aussi à la technique de fixation basée sur les coûts réduits, que nous appliquons aux variables $y$ afin d'améliorer davantage la performance de l'algo-

rithme de *branch-and-bound*.

#### 4.2.2.2.1 Génération de colonnes

La génération de colonnes est appliquée aux variables de flot $x$ qui sont générées de manière dynamique jusqu'à obtenir une solution optimale au sous-problème linéaire résolu à un nœud quelconque de l'arbre de recherche, lequel est appelé problème maître. Soulignons qu'aucune reformulation du problème maître n'est nécessaire pour appliquer la méthode.

À chaque itération de la génération de colonnes, un problème maître restreint, obtenu en éliminant certaines variables de flot du problème maître, est résolu, et les variables à y introduire sont déterminées suite à la résolution d'un sous problème de *pricing*.

**Initialisation** Pour initialiser la méthode de génération de colonnes et obtenir un premier problème maître restreint, nous ajoutons des arcs artificiels liant les nœuds origine $O(k)$ et destination $D(k)$ de chaque produit $k$. Ces arcs ont un coût d'ajout de capacité nul ($f_{O(k)D(k)} = 0$) et leur capacité $u_{O(k)D(k)}$ est infinie. De plus, nous affectons à chaque arc artificiel des coûts de transport de flot par produit $c^k_{O(k)D(k)}$ très élevés.

Cette modification permet d'une part d'assurer la réalisabilité du problème maître restreint initial (qui ne contient que les arcs artificiels), et d'autre part, les coûts très élevés associés permettront d'éliminer toute solution contenant ces arcs dès les premières itérations.

**Problème maître restreint** Pour définir le problème maître restreint, nous considérons un sous-ensemble de produits par arc $\tilde{K}_{ij} \subseteq K$, pour lesquels il existe au moins un $s \in S_{ij}$ tel que la variable $x^{ks}_{ij}$ est présente dans le modèle. De plus, seules les inégalités valides qui ont été générées sont représentées dans le modèle. Nous les définissons sur le sous-ensemble $\bar{K}_{ij} \subseteq \tilde{K}_{ij}$.
Les sous-ensembles suivants sont aussi définis :

- $\tilde{A}^k = \{(i,j) \in A / k \in \tilde{K}_{ij}\}$, $k \in K$, sous-ensemble d'arcs par lesquels il est possible

qu'un flot du produit $k$ passe (c'est-à-dire, il existe $s$ tel que $x_{ij}^{ks}$ est dans le modèle). Rappelons que $A$ est l'ensemble d'arcs potentiels du réseau $G$ augmenté des arcs artificiels.

- $\tilde{N}_+^k(i) = \{j \in N/(i,j) \in \tilde{A}^k\}$ et $\tilde{N}_-^k(i) = \{j \in N/(j,i) \in \tilde{A}^k\}$, $i \in N$, $k \in K$, représentant les ensembles respectifs des nœuds origines et destinations liant les arcs de $\tilde{A}^k$ pour chaque produit $k$.

Notons qu'au fur et à mesure que nous avançons dans l'arbre de recherche, le problème maître, et par conséquent le problème restreint, intègrent plus de contraintes additionnelles, qui sont les inégalités "étendues" générées tout au long de la méthode. Le problème maître restreint se présente alors comme suit.

$$min \sum_{(i,j)\in A} \sum_{k\in\tilde{K}_{ij}} \sum_{s\in S_{ij}} d^k c_{ij}^k x_{ij}^{ks} + \sum_{(i,j)\in A} \sum_{s\in S_{ij}} s f_{ij} y_{ij}^s \tag{4.19}$$

*sujet à :*

$$\sum_{j\in\tilde{N}_+^k(i)} \sum_{s\in S_{ij}} x_{ij}^{ks} - \sum_{j\in\tilde{N}_-^k(i)} \sum_{s\in S_{ji}} x_{ji}^{ks} = \begin{cases} 1 & i = O(k), \\ -1 & i = D(k), \\ 0 & autrement \end{cases} \quad i \in N,\ k \in K \quad (\pi_i^k) \tag{4.20}$$

$$\sum_{k\in\tilde{K}_{ij}} d^k x_{ij}^{ks} \leqslant s u_{ij} y_{ij}^s, \quad (i,j) \in A,\ s \in S_{ij} \quad (\alpha_{ij}^s) \tag{4.21}$$

$$\sum_{s\in S_{ij}} y_{ij}^s \leqslant 1, \quad (i,j) \in A \quad (\gamma_{ij}) \tag{4.22}$$

$$x_{ij}^{ks} \leqslant y_{ij}^s, \quad k \in \bar{K}_{ij},\ (i,j) \in A,\ s \in S_{ij} \quad (\beta_{ij}^{ks}) \tag{4.23}$$

$$x_{ij}^{ks} \geqslant 0, \quad (i,j) \in A,\ s \in S_{ij},\ k \in \tilde{K}_{ij} \tag{4.24}$$

$$y_{ij}^s \geqslant 0, \quad (i,j) \in A,\ s \in S_{ij} \tag{4.25}$$

Nous associons à chaque contrainte du problème maître restreint une variable duale. Ces variables nous seront utiles pour la formulation et la résolution du sous-problème de *pricing*. Leurs valeurs sont calculées en résolvant le problème dual suivant :

$$Dual: \quad max \sum_{k \in K} (\pi^k_{O(k)} - \pi^k_{D(k)}) - \sum_{(i,j) \in A} \gamma_{ij} \tag{4.26}$$

$$\pi^k_i - \pi^k_j - d^k \alpha^s_{ij} - \beta^{ks}_{ij} \leqslant d^k c^k_{ij} \qquad k \in K, \ (i,j) \in A, \ s \in S_{ij} \tag{4.27}$$

$$su_{ij} \alpha^s_{ij} - \gamma_{ij} + \sum_{k \in K} \beta^{ks}_{ij} \leqslant sf_{ij} \quad (i,j) \in A, \ s \in S_{ij} \tag{4.28}$$

$$\alpha^s_{ij} \geqslant 0, \qquad (i,j) \in A, \ s \in S_{ij} \tag{4.29}$$

$$\gamma_{ij} \geqslant 0, \qquad (i,j) \in A \tag{4.30}$$

$$\beta^{ks}_{ij} \geqslant 0, \qquad k \in K, \ (i,j) \in A, \ s \in S_{ij}. \tag{4.31}$$

**Sous-problème** Nous cherchons à déterminer les variables de flot non encore générées $x^{ks}_{ij}, (i,j) \in A, s \in S_{ij}, k \notin \tilde{K}_{ij}$, ayant des coûts réduits négatifs, et dont l'introduction dans le problème maître restreint permettra d'améliorer la solution courante.

En utilisant le dual du problème maître, formulé en (4.26)-(4.31), nous déduisons par la contrainte (4.27) la formule des coûts réduits des variables $x^{ks}_{ij}$ :

$$\bar{\bar{c}}^{ks}_{ij} = d^k c^k_{ij} - \pi^k_i + \pi^k_j + d^k \alpha^s_{ij} + \beta^{ks}_{ij}, \quad k \in K, \ (i,j) \in A, \ s \in S_{ij} \tag{4.32}$$

Les variables $x^{ks}_{ij}$ à introduire dans le problème maître restreint sont donc telles que $\bar{\bar{c}}^{ks}_{ij} < 0, (i,j) \in A, s \in S_{ij}, k \notin \tilde{K}_{ij}$. Afin de calculer ces coûts réduits, nous utilisons le dual de la relaxation linéaire du modèle (4.10)-(4.17).

Le calcul de $\bar{\bar{c}}^{ks}_{ij}$ est problématique. En effet, du moment que toutes les inégalités valides n'ont pas été forcément générées, les valeurs des variables duales associées $\beta^{ks}_{ij}$ ne sont pas toutes connues. Nous proposons alors de contourner leur calcul en se basant sur les équations d'écarts complémentaires.

$$x^{ks}_{ij}(d^k c^k_{ij} - \pi^k_i + \pi^k_j + d^k \alpha^s_{ij} + \beta^{ks}_{ij}) = 0 \quad k \in K, \ (i,j) \in A, \ s \in S_{ij} \tag{4.33}$$

$$y^s_{ij}(sf_{ij} - su_{ij} \alpha^s_{ij} + \gamma_{ij} - \sum_{k \in K} \beta^{ks}_{ij}) = 0 \quad (i,j) \in A, \ s \in S_{ij} \tag{4.34}$$

$$\alpha_{ij}^{s}(su_{ij}y_{ij}^{s} - \sum_{k\in K} d^{k}x_{ij}^{ks}) = 0 \quad (i,j)\in A,\ \ s\in S_{ij} \tag{4.35}$$

$$\gamma_{ij}(1 - \sum_{s\in S_{ij}} y_{ij}^{s}) = 0 \quad (i,j)\in A \tag{4.36}$$

$$\beta_{ij}^{ks}(y_{ij}^{s} - x_{ij}^{ks}) = 0 \qquad k\in K,\ \ (i,j)\in A,\ \ s\in S_{ij} \tag{4.37}$$

Nous dénotons par ($\bar{x}$,$\bar{y}$) la solution optimale du problème maître restreint, et par $(\bar{\pi},\bar{\alpha},\bar{\gamma},\bar{\beta})$ la solution duale correspondante. Cette solution peut être complétée en posant $\bar{x}_{ij}^{ks} = 0$ pour tout $(i,j)\in A$, $s\in S_{ij}$, $k\notin \tilde{K}_{ij}$.

À partir de là, nous pouvons distinguer deux cas, selon que les variables $\bar{y}_{ij}^{s}$ sont positives ou nulles.

- **Cas 1** : $\bar{y}_{ij}^{s} > 0$.

  Dans ce cas, par la relation (4.37), nous avons $\forall k\notin \tilde{K}_{ij}$ :

  $$\bar{\beta}_{ij}^{ks}(\underbrace{\bar{y}_{ij}^{s}}_{>0} - \underbrace{\bar{x}_{ij}^{ks}}_{=0}) = 0 \quad \Rightarrow \quad \bar{\beta}_{ij}^{ks} = 0$$

  La formule du coût réduit devient alors :

  $$\bar{\bar{c}}_{ij}^{ks} = d^{k}c_{ij}^{k} - \bar{\pi}_{i}^{k} + \bar{\pi}_{j}^{k} + d^{k}\bar{\alpha}_{ij}^{s} \tag{4.38}$$

  Il suffit donc d'introduire dans le problème maître restreint les variables $x_{ij}^{ks}$, $k\notin \tilde{K}_{ij}$, telles que :

  $$d^{k}c_{ij}^{k} - \bar{\pi}_{i}^{k} + \bar{\pi}_{j}^{k} + d^{k}\bar{\alpha}_{ij}^{s} < 0 \tag{4.39}$$

- **Cas 2** : $\bar{y}_{ij}^{s} = 0$.

  Dans ce cas, par la contrainte (4.27) du dual, nous avons :

  $$\bar{\beta}_{ij}^{ks} \geqslant \bar{\pi}_{i}^{k} - \bar{\pi}_{j}^{k} - d^{k}\bar{\alpha}_{ij}^{s} - d^{k}c_{ij}^{k} \tag{4.40}$$

  Puisque $\bar{\beta}_{ij}^{ks} \geq 0$ (par la contrainte (4.31) du dual), alors la relation suivante devrait

être vérifiée $\forall k \in K$ :

$$\bar{\beta}_{ij}^{ks} \geqslant max\{0, \bar{\pi}_i^k - \bar{\pi}_j^k - d^k \bar{\alpha}_{ij}^s - d^k c_{ij}^k\} \tag{4.41}$$

En exploitant la contrainte (4.28) du dual, nous avons aussi :

$$sf_{ij} - su_{ij}\bar{\alpha}_{ij}^s - \sum_{k \in K} \bar{\beta}_{ij}^{ks} \geqslant -\bar{\gamma}_{ij} \tag{4.42}$$

En combinant les relations (4.41) et (4.42), nous obtenons l'inégalité suivante :

$$\bar{\omega}_{ij}^s = sf_{ij} - su_{ij}\bar{\alpha}_{ij}^s - \sum_{k \in K} max\{0, \bar{\pi}_i^k - \bar{\pi}_j^k - d^k \bar{\alpha}_{ij}^s - d^k c_{ij}^k\} \geqslant -\bar{\gamma}_{ij} \tag{4.43}$$

Puisque pour chaque arc $(i, j) \in A$, cette relation doit être vérifiée pour tout $s \in S_{ij}$, on ne génère de nouvelles variables que lorsque $\min_{s \in S_{ij}} \bar{\omega}_{ij}^s < -\bar{\gamma}_{ij}$, et uniquement pour $s^* = \underset{s \in S_{ij}}{\operatorname{argmin}} \bar{\omega}_{ij}^s$.
Les variables correspondantes $x_{ij}^{ks}$, $k \notin \tilde{K}_{ij}$ sont alors introduites dans le problème maître restreint, mais seulement si leurs coûts réduits sont négatifs.

En résumé, les variables $x_{ij}^{ks}$, $k \notin \tilde{K}_{ij}$ à introduire dans le problème maître restreint sont telles que :

1. $\bar{y}_{ij}^s > 0 \quad et \quad d^k c_{ij}^k - \bar{\pi}_i^k + \bar{\pi}_j^k + d^k \bar{\alpha}_{ij}^s < 0, \ \ \forall (i, j) \in A, s \in S_{ij}.$

2. $\bar{y}_{ij}^{s^*} = 0 \quad pour \quad s^* = \underset{s \in S_{ij}}{\operatorname{argmin}} \{\bar{\omega}_{ij}^s : \ \bar{\omega}_{ij}^s < -\bar{\gamma}_{ij}\}$
   $et \quad d^k c_{ij}^k - \bar{\pi}_i^k + \bar{\pi}_j^k + d^k \bar{\alpha}_{ij}^{s^*} < 0, \ \ \forall (i, j) \in A.$

Une fois les nouvelles variables introduites dans le problème maître restreint, il est résolu à nouveau, et ainsi de suite jusqu'à obtenir une solution optimale au problème. Lorsque nous nous arrêtons à l'optimalité, nous avons une borne inférieure $\underline{z}$ sur la valeur optimale du problème en nombres entiers (4.10)-(4.17). Si $\underline{z}$ est entière et inférieure à la valeur de la meilleur solution connue dans le cadre de la méthode de *branch-and-bound*,

alors cette meilleure solution est mise à jour par la solution correspondante à $\underline{z}$. Si, par contre, $\underline{z}$ est supérieure à la valeur de la meilleure solution connue, le nœud en cours est directement élagué sans passer à la génération de coupes. Lorsqu'aucun de ces deux cas ne se présente, nous appliquons la génération de coupes pour améliorer la borne inférieure $\underline{z}$.

**4.2.2.2.2 Génération de coupes** Le problème résolu initialement est une formulation faible de la relaxation linéaire que nous souhaitons renforcer pour obtenir de meilleures bornes inférieures. Les inégalités "étendues" sont alors introduites dans le problème pour obtenir une formulation forte. Comme leur nombre est exponentiel, elles sont générées de manière dynamique dans le cadre des méthodes de coupes.

La procédure consiste initialement à introduire dans le problème linéaire les inégalités $x_{ij}^{ks} \leqslant y_{ij}^{s}$ violées par la solution optimale $(\bar{x}, \bar{y})$ obtenue par génération de colonnes, c'est-à-dire telle que :

$$\bar{x}_{ij}^{ks} > \bar{y}_{ij}^{s}, \;\; k \in K, (i,j) \in A, s \in S_{ij} \tag{4.44}$$

Par la suite, puisque la solution courante devient non réalisable, le problème est réoptimisé en utilisant la méthode duale du simplexe. La procédure est ainsi réitérée jusqu'à ce qu'aucune inégalité valide ne soit violée.

Encore une fois, dans le cadre de la méthode de *branch-and-bound*, si la génération de coupes se termine par une solution optimale entière dont la valeur est inférieure à celle de la borne supérieure, cette borne est alors mise à jour et le nœud est élagué par optimalité. Si, par contre, cette valeur est supérieure à la valeur de la meilleure solution connue, le nœud correspondant est élagué par dominance.

**4.2.2.2.3 Fixation des variables** Nous appliquons à chaque nœud de l'arbre de recherche la technique de fixation de variables basée sur les coûts réduits. La méthode consiste à fixer les variables à l'une ou l'autre de leurs bornes, afin de réduire davantage l'espace de recherche et d'accélérer la résolution.

Étant donnée $\underline{z}$ la valeur optimale de la relaxation linéaire du problème en cours, $z_{opt}$ la valeur de la meilleure solution optimale connue, et $\bar{\bar{f}}^s_{ij} = sf_{ij} - su_{ij}\bar{\alpha}^s_{ij} + \bar{\gamma}_{ij} - \sum_{k\in K}\bar{\beta}^{ks}_{ij}$ le coût réduit de la variable $y^s_{ij}$ lorsque cette dernière est dans l'intervalle ]0,1[. Nous effectuons le test suivant : Si

$$\underline{z} + |\bar{\bar{f}}^s_{ij}| \geqslant z_{opt}, \tag{4.45}$$

alors $y^s_{ij}$ peut être fixée à la valeur $\bar{y}^s_{ij}$ correspondant à la solution optimale de la relaxation linéaire. Cette solution est nécessairement entière ($\bar{y}^s_{ij} \in \{0,1\}$).

Une fois les variables $y$ vérifiant la condition (4.45) fixées à l'une ou l'autre de leurs bornes, le problème résultant, auquel sont ajoutées les contraintes de branchement, donne lieu à de nouveaux sous-problèmes résolus dans les nœuds fils par génération de colonnes et de coupes, en prenant en considération la stratégie de recherche adoptée et leur ordre dans la liste des nœuds.

# CHAPITRE 5

# RÉSULTATS

Nous présentons dans ce chapitre les expérimentations que nous avons effectuées pour tester l'efficacité de la méthode proposée ainsi que les résultats que nous avons obtenus. Nous les comparons aux résultats obtenus par d'autres méthodes pour conclure à l'efficacité de la nôtre.

## 5.1 Environnement

### 5.1.1 Instances

Dans l'absence de données réelles appropriées, nous avons effectué nos expérimentations sur 32 instances variées générées par Frangioni et Gendron [30] dans le cadre de leur travail sur la reformulation du problème de conception de réseaux avec coûts d'ajout de capacité en variables binaires.

Le générateur reçoit en entrée le nombre de sommets, d'arcs et de produits, et crée les arcs en connectant les sommets de manière aléatoire sans permettre d'arcs parallèles. Les origines et destinations des produits, ainsi que les coûts, les capacités des arcs et les demandes sont également générés de manière aléatoire selon une loi uniforme.

Les instances peuvent être catégorisées en quatre groupes tels que présentés dans le tableau 5.I. Chaque catégorie a le même nombre de sommets et de produits, seul le nombre d'arcs est différent. On distingue des instances de petite (P), moyenne (M), grande (G) ou très grande (TG) taille. Les coûts variables et fixes diffèrent et peuvent dépasser les uns les autres. De même, les capacités des arcs sont parfois grandes et parfois petites et donc plus contrariantes.

### 5.1.2 Environnement matériel et logiciel

Pour tester notre méthode sur les instances présentées précédemment nous avons utilisé un processeur Intel Xeon X5675 avec une cadence de 3.07 GHz et 96 Go de

| Groupe | \|N\| | \|K\| | \|A\| | Nombre |
|---|---|---|---|---|
| P | 20 | 40 | 230 | 4 |
| | | | 289 | 4 |
| M | 30 | 100 | 517 | 4 |
| | | | 669 | 4 |
| G | 20 | 200 | 229 | 4 |
| | | | 287 | 4 |
| TG | 30 | 400 | 519 | 4 |
| | | | 688 | 4 |

Tableau 5.I – Groupes d'instances.

RAM. Le système d'exploitation utilisé est RHEL 6.4.

La méthode proposée a été programmée en C++. Nous nous sommes basés sur la librairie SCIP 3.0.1 (*Solving Constraint Integer Programs*)[1] pour implémenter le *branch-and-bound*, la génération de colonnes et la génération de coupes.

SCIP est en effet une librairie logicielle Développée en langage C dans le cadre de la thèse de doctorat de T. Achterberg [1], ayant pour objectif de faciliter le développement d'algorithmes de *branch-and-bound*.

Que ce soit pour notre méthode ou pour la méthode heuristique de "*branch-and-bound* avec *price-and-cut* à la racine" développée par Frangioni et Gendron [30] à laquelle nous nous comparons, toutes les relaxations linéaires ont été résolues via CPLEX 12.4.0.

## 5.2 Tests effectués

Pour tester la performance de note méthode nous effectuons quatre types d'expérimentations. D'abord nous testons notre méthode avec et sans génération de colonnes, afin de confirmer notre hypothèse sur l'impact de la génération de colonnes sur la réduction du temps de résolution. Nous testons ensuite l'efficacité de la méthode de *pricing* choisie en se comparant au cas où l'on génère tous les niveaux de capacité par arc (noté "B&P&C I"), au lieu de choisir un seul niveau de capacité tel que décrit dans la section 4.2.2.2.2 (cas noté "B&P&C II", correspondant à notre méthode).

[1] `http://scip.zib.de`

Nous comparons par la suite notre méthode à celles déjà existantes, à savoir la méthode de *branch-and-bound* de CPLEX, et la méthode heuristique de *branch-and-bound* avec *price-and-cut* à la racine [30] (notée "HeurB&B + P&C").

Nous présentons dans les prochaines sous-sections les tests effectués sur nos instances en utilisant les méthodes comparées.

### 5.2.1 Tests à la racine

Avant d'effectuer nos expérimentations sur tout l'arbre de recherche, nous avons préféré d'abord le faire à la racine uniquement, en fixant la limite des nœuds à explorer à un, afin d'avoir une idée sur la taille et la complexité du problème ainsi que sur le temps de résolution requis.

Nous avons donc comparé les trois principales méthodes à la racine en fixant la limite de temps à 3 heures, et en utilisant comme bornes supérieures initiales les meilleures entres celles obtenues après avoir tourné toutes les méthodes comparées pendant plus de 12 heures.

Nous avons effectué deux types de tests avec CPLEX. Dans le premier, nous lui avons communiqué la formulation faible du modèle 0-1, et sans changer les paramètres par défaut, nous lui avons demandé de le résoudre (la méthode est notée "CPLEX sans IF"). CPLEX était alors libre d'utiliser tous les types de coupes qu'il possède. Dans le deuxième test, nous lui avons donné la formulation forte avant de lancer l'exécution ("CPLEX avec IF").

Pour la méthode que nous avons proposé, en plus de la méthode de *pricing* adoptée, nous avons aussi considéré le cas où tous les niveaux de capacités sont générés.

Finalement, pour la méthode heuristique de *branch-and-bound* avec *price-and-cut* à la racine, nous avons désactivé l'heuristique afin d'obtenir le résultat de la méthode appliquée à la racine.

### 5.2.2 Test général

Après les tests à la racine, nous avons généralisé nos expérimentations à l'ensemble de l'arbre de recherche. Rappelons que la technique d'exploration de l'arbre que nous avons retenu est la recherche meilleur d'abord. L'objectif est d'améliorer la borne inférieure afin d'approcher la solution optimale, notamment dans le cas des grandes instances.

Nous avons, dans un premier temps, testé notre méthode sans génération de colonnes (ce qui a donné lieu à la méthode "B&C"), puis avec génération de colonnes selon les deux modes de *pricing* : "B&P&C I" et "B&P&C II", et ce dans une limite de temps de 3 heures et avec les mêmes bornes supérieures utilisées dans les tests à la racine.

Ensuite, pour réaliser nos expérimentations avec CPLEX, nous avons repris les mêmes tests effectués à la racine et les avons généralisés sur tout l'arbre de recherche. Dans ce cas aussi, comme il est difficile de trouver des solutions exactes dans un horizon proche pour les instances de grande taille, nous avons fixé la limite de temps à 3 heures afin de comparer le gap entre les bornes supérieures et inférieures.

Enfin, nous avons testé la méthode "HeurB&B + P&C" sans lui donner de borne supérieure. La technique de polissage (*polishing*) a été exécutée pendant une heure afin d'améliorer la qualité de la solution réalisable. La limite de temps a été fixée à 3 heures pour tous les groupes d'instances, même si la méthode n'est pas capable de terminer la résolution à la racine et d'atteindre la phase de polissage.

Le *polishing* est en effet une méthode heuristique proposée par CPLEX qui permet de résoudre des problèmes linéaires en nombres entiers mixtes pour lesquels il est peu probable de trouver une solution optimale. Elle priorise donc l'amélioration de la meilleure solution réalisable existante.[2]

Notons qu'à chaque fois que la méthode "B&P&C II" a été comparée à l'une des autres méthodes, les mêmes limites de temps et bornes supérieures initiales ont été respectées.

[2] `http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r4`

## 5.3 Analyse des résultats

Avant de procéder à l'analyse des résultats des tests effectués, nous présentons d'abord les mesures de performance que nous avons adoptées pour évaluer notre méthode.

### 5.3.1 Mesures de performance

Pour comparer notre méthode à celles déjà existantes dans la littérature ainsi qu'à CPLEX, nous avons défini deux principales mesures de performance comme base de comparaison.

Dans le cas des petites et certaines moyennes instances qui ont été résolues, nous comparons le temps de résolution global (CPU).

Dans le cas des instances de plus grandes tailles, pour lesquelles une solution optimale n'a pas pu être déterminée dans la limite de temps fixée, nous comparons l'écart relatif entre les bornes supérieures et inférieures (GAP) calculé selon la formule (5.1).

$$GAP = \frac{UB - LB}{UB} \times 100 \tag{5.1}$$

Dans ce sens, une méthode sera dite plus performante qu'une autre si elle permet de résoudre les instances dans un temps plus court, ou si elle génère des gaps plus petits pour toutes ou au moins un nombre plus grand d'instances.

### 5.3.2 Analyse à la racine

Les résultats des tests à la racine sont rapportés dans les tableaux 5.II et 5.III. Nous comparons à la fois les méthodes "B&P&C I", "B&P&C II", "CPLEX avec IF", "CPLEX sans IF" et "HeurB&B + P&C", en tenant compte du temps de résolution et des gaps entre les bornes supérieures et inférieures pour chaque instance du problème.

L'observation des durées de résolution moyennes requises ainsi que les gaps moyens obtenus par chacune des méthodes (il s'agit des moyennes arithmétiques) montre que le meilleur gap moyen pour les instances de petite taille est obtenu avec "CPLEX avec

| Groupe | \|A\| | Instance | B&P&C I | | B&P&C II | | CPLEX avec IF | | CPLEX sans IF | | HeurB&B+P&C | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | CPU (s) | GAP (%) | CPU (s) | GAP (%) | CPU (s) | GAP (%) | CPU (s) | GAP (%) | CPU (s) | GAP (%) |
| P | 230 | p33 | 6,7 | 0,0 | 1,6 | 0,0 | 1,6 | 0,0 | 108,8 | 0,8 | 0,2 | 0,0 |
| | | p34 | 11,3 | 0,0 | 1,7 | 0,0 | 2,3 | 0,0 | 77,2 | 2,3 | 0,4 | 0,0 |
| | | p35 | 20,0 | 0,2 | 3,2 | 0,2 | 9,6 | 0,0 | 176,5 | 1,3 | 0,3 | 0,2 |
| | | p36 | 30,0 | 0,9 | 4,0 | 0,9 | 1 1,8 | 0,5 | 207,7 | 3,8 | 0,3 | 0,9 |
| | 289 | p41 | 6,7 | 0,1 | 1,4 | 0,1 | 3,2 | 0,0 | 248,0 | 1,0 | 0,3 | 0,1 |
| | | p42 | 18,0 | 0,5 | 2,0 | 0,5 | 5,0 | 0,1 | 148,8 | 3,4 | 0,3 | 0,5 |
| | | p43 | 21,8 | 0,5 | 3,0 | 0,5 | 10,9 | 0,2 | 243,7 | 1,9 | 0,3 | 0,5 |
| | | p44 | 27,9 | 1,2 | 3,2 | 1,2 | 65,5 | 0,7 | 560,3 | 4,5 | 0,3 | 1,2 |
| Moyenne P | | | 17,8 | 0,4 | 2,5 | 0,4 | 13,7 | 0,2 | 221,4 | 2,4 | 0,3 | 0,4 |
| M | 517 | p49 | 4 357,0 | 0,7 | 76,9 | 0,7 | 629,1 | 0,7 | 10 758,6 | 12,5 | 24,8 | 0,7 |
| | | p50 | 10 809,0 | 5,2 | 755,0 | 5,0 | 10 767,9 | 100,0 | 10 763,2 | 21,7 | 685,0 | 5,0 |
| | | p51 | 4 150,1 | 1,0 | 76,3 | 1,0 | 460,4 | 0,8 | 10 762,1 | 12,7 | 7,8 | 1,0 |
| | | p52 | 10 803,6 | 4,4 | 431,3 | 4,0 | 10 767,8 | 100,0 | 10 755,1 | 20,1 | 490,2 | 4,0 |
| | 669 | p57 | 1 844,1 | 0,0 | 46,9 | 0,0 | 275,8 | 0,0 | 10 763,6 | 14,3 | 11,4 | 0,0 |
| | | p58 | 9 459,7 | 1,1 | 230,9 | 1,1 | 4 256,6 | 1,1 | 10 752,0 | 18,3 | 173,8 | 1,1 |
| | | p59 | 2 301,3 | 1,0 | 61,3 | 1,0 | 535,0 | 0,8 | 10 745,4 | 15,1 | 6,9 | 1,0 |
| | | p60 | 7 255,2 | 1,6 | 153,0 | 1,6 | 5 070,9 | 1,6 | 10 711,1 | 16,1 | 49,2 | 1,6 |
| Moyenne M | | | 6 372,5 | 1,9 | 229,0 | 1,8 | 4 095,4 | 25,6 | 10 751,4 | 16,3 | 181,1 | 1,8 |

Tableau 5.II – Comparaison des performances des différentes méthodes à la racine - Groupes P et M.

| Groupe | \|A\| | Instance | B&P&C I | | B&P&C II | | CPLEX avec IF | | CPLEX sans IF | | HeurB&B+P&C | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | CPU (s) | GAP (%) | CPU (s) | GAP (%) | CPU (s) | GAP (%) | CPU (s) | GAP (%) | CPU (s) | GAP (%) |
| G | 229 | p37 | 10 808,6 | 6,2 | 379,6 | 5,7 | 10 917,1 | 5,7 | 10 765,8 | 13,8 | 1 071,0 | 5,7 |
| | | p38 | 10 802,8 | 5,9 | 548,8 | 5,4 | 10 761,0 | 100,0 | 10 784,3 | 25,2 | 1 889,3 | 5,4 |
| | | p39 | 10 800,0 | 6,8 | 631,8 | 5,9 | 10 741,0 | 100,0 | 10 767,5 | 17,4 | 1 126,0 | 5,9 |
| | | p40 | 10 800,0 | 7,0 | 486,2 | 6,7 | 10 767,8 | 100,0 | 10 754,7 | 17,2 | 1 895,8 | 6,5 |
| | 287 | p45 | 10 820,9 | 7,0 | 573,9 | 4,7 | 10 768,2 | 100,0 | 10 766,4 | 26,5 | 1 043,0 | 4,7 |
| | | p46 | 10 800,1 | 7,0 | 976,9 | 6,1 | 10 768,4 | 100,0 | 10 768,7 | 25,9 | 2 269,2 | 6,1 |
| | | p47 | 10 814,4 | 6,7 | 812,8 | 5,2 | 10 768,6 | 100,0 | 10 755,4 | 26,0 | 790,7 | 5,2 |
| | | p48 | 10 812,0 | 7,4 | 980,0 | 6,0 | 10 767,4 | 100,0 | 10 884,3 | 27,4 | 3 602,1 | 6,0 |
| Moyenne G | | | 10 807,4 | 6,7 | 673,7 | 5,7 | 10 782,4 | 88,2 | 10 780,9 | 22,4 | 1 710,9 | 5,7 |
| TG | 519 | p53 | 10 884,4 | 25,1 | 4 290,0 | 5,5 | 10 780,5 | 100,0 | 10 768,8 | 31,1 | 10 804,2 | 5,5 |
| | | p54 | 10 850,3 | 27,8 | 7 200,4 | 7,9 | 10 753,1 | 100,0 | 6 296,3 | 32,5 | 10 886,8 | 7,9 |
| | | p55 | 10 859,2 | 24,1 | 3 920,3 | 4,0 | 10 766,1 | 100,0 | 6 633,0 | 26,2 | 10 842,3 | 4,0 |
| | | p56 | 10 878,9 | 27,7 | 6 258,8 | 6,6 | 10 756,7 | 100,0 | 9 069,1 | 28,1 | 10 939,4 | 6,6 |
| | 688 | p61 | 10 878,0 | 36,0 | 9 459,0 | 7,2 | 10 514,2 | 100,0 | 9 591,3 | 36,8 | 10 804,1 | 7,2 |
| | | p62 | 10 900,7 | 38,1 | 10 875,6 | 9,9 | 10 727,8 | 100,0 | 10 773,5 | 38,6 | 10 836,3 | 9,9 |
| | | p63 | 10 799,7 | 30,9 | 5 903,7 | 5,5 | 10 779,7 | 100,0 | 10 770,6 | 31,8 | 10 812,5 | 5,5 |
| | | p64 | 10 853,6 | 33,4 | 9 363,9 | 8,0 | 10 788,6 | 100,0 | 10 752,5 | 33,8 | 10 969,9 | 8,0 |
| Moyenne TG | | | 10 863,1 | 30,4 | 7 159,0 | 6,8 | 10 682,6 | 100,0 | 9 048,9 | 31,7 | 10 861,9 | 6,8 |

Tableau 5.III – Comparaison des performances des différentes méthodes à la racine - Groupes G et TG.

IF" suivi par les méthodes "B&P&C II" et "HeurB&B + P&C". Cette dernière assure cependant le meilleur temps de résolution pour toutes les instances tandis que notre méthode vient en deuxième position. Le pire temps d'exécution est celui de "CPLEX sans IF".

Pour les instances de taille moyenne, le meilleur gap moyen (1,8%) est obtenu par la méthode "B&P&C II" ainsi que la méthode "HeurB&B + P&C", qui permet toutefois un temps d'exécution moyen légèrement meilleur (181s versus 229s pour notre méthode). Les autres méthodes restent peu compétitives avec des temps d'exécution moyens entre 4000 secondes et 3 heures.

Enfin, pour les instances de grande et très grande taille, les deux méthodes "B&P&C II" et "HeurB&B + P&C" permettent encore une fois des gaps moyens minimaux de 5,7% et 6,8% pour les groupes G et TG respectivement. Cependant, notre méthode permet d'atteindre ces gaps dans un temps moyen meilleur : 674s versus 1711s pour le groupe G et 7159s versus 3h pour le groupe TG (qui correspond à la limite de temps fixée).

En fait, les deux méthodes "B&P&C II" et "HeurB&B + P&C" performent la génération de colonnes et de coupes à la racine, le fait que "HeurB&B + P&C" prenne plus de temps nous amène à conclure que notre choix de la méthode de performer le *price-and-cut* est meilleur pour les grandes instances.

Concernant CPLEX, nous constatons que le fait de lui donner la formulation forte lui permet de résoudre les petites et certaines instances de taille moyenne plus rapidement. Cependant, lorsque la taille des instances augmente (groupes G et TG), les inégalités fortes augmentent considérablement la taille des problèmes. "CPLEX avec IF" n'est alors capable de fournir aucune solution initiale aux instances (d'où le gap de 100%), tandis que "CPLEX sans IF" permet d'avoir des gaps moyens de 22,4% et 31,7% pour les groupes G et TG respectivement.

Pour la méthode "B&P&C I", elle permet d'avoir des gaps moyens comparables à ceux de "B&P&C II" pour les instances des groupes P, M et G, mais dans des durées plus longues. Elle donne aussi un gap moyen comparable à celui de "CPLEX sans IF" pour le groupe TG après 3h d'exécution. Nous constatons donc que la méthode génère

beaucoup trop de variables, ce qui ralentit son exécution.

Nous pouvons finalement conclure que notre méthode est prometteuse : elle est plus performante pour les instances de grande et très grande taille, et assez compétitive pour les instances de tailles petites et moyennes. Elle a de plus permis de résoudre toutes les instances à la racine dans la limite de temps fixée. Ces résultats nous encouragent donc à généraliser nos tests à tout l'arbre de recherche.

### 5.3.3 Comparaison avec les autres méthodes

Nous présentons les résultats de la comparaison entre la méthode proposée et les deux autres. Nous étudions aussi l'impact de l'implémentation de la technique de génération de colonnes sur la qualité de la borne inférieure et le temps global d'exécution.

#### 5.3.3.1 Impact de la génération de colonnes

Étant donné que la reformulation du problème en variables binaires augmente considérablement le nombre de variables, nous souhaitons d'abord évaluer l'impact de l'utilisation de la méthode de génération de colonnes sur le temps d'exécution et l'amélioration de la borne inférieure. Nous souhaitons par la même occasion souligner l'importance du choix de la méthode de *pricing* et la réduction du nombre de variables générées à chaque itération pour la performance de notre méthode.

Pour ce faire, nous comparons les trois méthodes "B&P&C II", "B&P&C I" et "B&C" (qui correspond à la méthode "B&P&C II" où la génération de colonnes est désactivée). Les résultats sont rapportés dans le tableau 5.IV. Rappelons que la limite de temps est fixée à 3 heures, et que l'on donne une borne supérieure aux trois méthodes avant de les lancer.

Une lecture rapide du tableau permet de constater que la méthode "B&P&C II" est bien plus performante que les deux autres méthodes. Pour les instances du groupe P, résolues par les trois méthodes, elle permet d'obtenir une solution optimale après près d'une minute en moyenne, tandis que les méthodes "B&P&C I" et "B&C" prennent en moyenne près de 7 et 11 minutes respectivement.

| Groupe | \|A\| | Instance | B&C | | B&P&C I | | B&P&C II | |
|---|---|---|---|---|---|---|---|---|
| | | | CPU (s) | GAP (%) | CPU (s) | GAP (%) | CPU (s) | GAP (%) |
| P | 230 | p33 | 8,5 | 0,0 | 6,9 | 0,0 | 1,6 | 0,0 |
| | | p34 | 14,9 | 0,0 | 10,8 | 0,0 | 1,7 | 0,0 |
| | | p35 | 82,9 | 0,0 | 45,7 | 0,0 | 9,9 | 0,0 |
| | | p36 | 5 767,8 | 0,0 | 1 498,2 | 0,0 | 145,4 | 0,0 |
| | 289 | p41 | 18,6 | 0,0 | 8,4 | 0,0 | 1,5 | 0,0 |
| | | p42 | 73,2 | 0,0 | 33,0 | 0,0 | 4,5 | 0,0 |
| | | p43 | 505,2 | 0,0 | 405,9 | 0,0 | 58,4 | 0,0 |
| | | p44 | 1 516,6 | 0,0 | 1 146,4 | 0,0 | 225,6 | 0,0 |
| Moyenne P | | | 675,2 | 0,0 | 394,4 | 0,0 | 56,1 | 0,0 |
| M | 517 | p49 | 10 800,0 | 0,7 | 10 803,7 | 0,3 | 3 081,7 | 0,0 |
| | | p50 | 10 800,3 | 5,1 | 10 806,3 | 5,1 | 10 800,0 | 3,7 |
| | | p51 | 10 805,6 | 1,0 | 10 800,1 | 0,7 | 10 800,1 | 0,3 |
| | | p52 | 10 800,0 | 4,7 | 10 811,4 | 4,3 | 10 800,0 | 3,0 |
| | 669 | p57 | 7 527,5 | 0,0 | 2 149,5 | 0,0 | 55,4 | 0,0 |
| | | p58 | 10 800,0 | 1,3 | 10 800,2 | 1,1 | 10 800,0 | 0,1 |
| | | p59 | 10 800,2 | 1,0 | 10 804,6 | 0,8 | 10 800,2 | 0,6 |
| | | p60 | 10 800,3 | 1,8 | 10 800,0 | 1,4 | 10 800,0 | 1,0 |
| Moyenne M | | | 10 146,2 | 2,1 | 9 722,0 | 1,7 | 8 492,2 | 1,1 |
| G | 229 | p37 | 10 802,2 | 6,1 | 10 814,5 | 6,2 | 10 800,0 | 3,9 |
| | | p38 | 10 807,4 | 5,6 | 10 800,0 | 5,6 | 10 800,0 | 3,7 |
| | | p39 | 10 820,6 | 6,6 | 10 812,9 | 7,4 | 10 800,0 | 4,3 |
| | | p40 | 10 801,4 | 7,0 | 10 801,0 | 7,0 | 10 801,4 | 5,4 |
| | 287 | p45 | 10 813,0 | 6,3 | 10 800,3 | 6,5 | 10 801,4 | 3,5 |
| | | p46 | 10 804,3 | 6,8 | 10 800,1 | 7,0 | 10 800,8 | 4,9 |
| | | p47 | 10 821,4 | 6,6 | 10 809,6 | 7,0 | 10 801,2 | 4,1 |
| | | p48 | 10 800,0 | 7,4 | 10 800,0 | 7,2 | 10 801,4 | 5,0 |
| Moyenne G | | | 10 807,1 | 6,5 | 10 804,8 | 6,7 | 10 800,8 | 4,4 |
| TG | 519 | p53 | 10 898,2 | 24,2 | 10 884,2 | 24,4 | 10 802,9 | 5,4 |
| | | p54 | 10 920,3 | 27,2 | 10 811,0 | 32,4 | 10 804,3 | 7,8 |
| | | p55 | 10 852,2 | 26,2 | 10 800,9 | 25,7 | 10 805,3 | 3,8 |
| | | p56 | 10 899,3 | 28,1 | 10 933,2 | 27,7 | 10 800,0 | 6,5 |
| | 688 | p61 | 10 866,3 | 36,4 | 10 895,7 | 35,6 | 10 814,5 | 7,2 |
| | | p62 | 10 804,9 | 34,3 | 10 947,7 | 35,8 | 10 804,4 | 9,8 |
| | | p63 | 10 826,9 | 31,8 | 10 808,4 | 30,9 | 10 800,0 | 5,4 |
| | | p64 | 10 817,9 | 33,8 | 10 822,3 | 33,4 | 10 801,3 | 7,9 |
| Moyenne TG | | | 10 860,7 | 30,3 | 10 862,9 | 30,7 | 10 804,1 | 6,7 |

Tableau 5.IV – Impact du choix de la génération de colonnes sur l'amélioration des résultats.

Pour le groupe M, notre méthode donne un gap moyen de 1%, et permet de résoudre deux instances. Les deux autres méthodes donnent des gaps moyens de 1,7% pour la méthode "B&P&C I" et 2,1% pour la méthode "B&C" et permettent de résoudre une seule instance.

Finalement, pour les groupes G et TG, les trois méthodes arrivent à la limite de temps avant de trouver une solution. Les meilleurs gaps sont obtenus par la méthode "B&P&C II".

Nous remarquons toutefois est que la méthode sans génération de colonnes (qui donne des gaps assez grands relativement à notre méthode, notamment pour le groupe TG) est légèrement meilleure que la méthode "B&P&C I".

Nous pouvons conclure que la méthode de génération de colonnes permet effectivement, via la décomposition du problème, de réduire le temps d'exécution et d'obtenir de meilleures bornes inférieures, mais seulement si le choix de la méthode de *pricing* est bon. Si ce n'est pas la cas, comme pour la méthode "B&P&C I" où le nombre de variables générées est trop grand, l'utilisation de la génération de colonnes risque de donner de mauvais résultats, voire pire que si elle n'a pas été utilisée.

#### 5.3.3.2 Comparaison de notre méthode à la méthode de *branch-and-bound* de CPLEX

Nous comparons dans cette section la méthode proposée à CPLEX, auquel nous donnons dans un premier temps la formulation faible. Dans un deuxième temps, nous lui donnons la formulation forte. Nous rapportons les résultats dans le tableau 5.V. La limite de temps est fixée à 3 heures, et on donne aux deux méthodes la même borne supérieure initiale.

D'après les résultats, nous remarquons que les instances de petite taille ont été toutes résolues par la méthode "B&P&C II" et CPLEX, mais uniquement lorsque les inégalités fortes sont dans le modèle ("CPLEX avec IF"). Pour la méthode "CPLEX sans IF", seule la moitié des instances a été résolue. De plus, le temps requis pour la résolution du problème est considérablement élevé. CPLEX nécessite dans ce cas plus de temps pour trouver les inégalités valides à introduire.

| Groupe | \|A\| | Instance | B&P&C II | | CPLEX sans IF | | CPLEX avec IF | |
|---|---|---|---|---|---|---|---|---|
| | | | CPU (s) | GAP (%) | CPU (s) | GAP (%) | CPU (s) | GAP (%) |
| P | 230 | p33 | 1,6 | 0,0 | 430,7 | 0,0 | 1,9 | 0,0 |
| | | p34 | 1,7 | 0,0 | 373,9 | 0,0 | 2,0 | 0,0 |
| | | p35 | 9,9 | 0,0 | 1 867,0 | 0,0 | 15,5 | 0,0 |
| | | p36 | 145,4 | 0,0 | 10 858,9 | 0,6 | 136,6 | 0,0 |
| | 289 | p41 | 1,5 | 0,0 | 729,5 | 0,0 | 3,4 | 0,0 |
| | | p42 | 4,5 | 0,0 | 10 797,5 | 0,5 | 6,7 | 0,0 |
| | | p43 | 58,4 | 0,0 | 10 757,9 | 1,1 | 59,4 | 0,0 |
| | | p44 | 225,6 | 0,0 | 10 850,7 | 1,2 | 105,8 | 0,0 |
| Moyenne P | | | 56,1 | 0,0 | 5 833,3 | 0,4 | 41,4 | 0,0 |
| M | 517 | p49 | 3 081,7 | 0,0 | 10 800,0 | 13,6 | 10 800,1 | 0,3 |
| | | p50 | 10 800,0 | 3,7 | 10 800,2 | 18,5 | 10 801,1 | 100,0 |
| | | p51 | 10 800,1 | 0,3 | 10 810,5 | 13,0 | 10 765,8 | 0,7 |
| | | p52 | 10 800,0 | 3,0 | 10 800,2 | 20,2 | 10 761,3 | 100,0 |
| | 669 | p57 | 55,4 | 0,0 | 10 835,7 | 14,9 | 494,7 | 0,0 |
| | | p58 | 10 800,0 | 0,1 | 10 802,8 | 18,7 | 10 801,8 | 1,0 |
| | | p59 | 10 800,2 | 0,6 | 10 800,3 | 15,1 | 10 800,2 | 0,8 |
| | | p60 | 10 800,0 | 1,0 | 10 800,3 | 16,6 | 10 800,8 | 1,5 |
| Moyenne M | | | 8 492,2 | 1,1 | 10 806,3 | 16,3 | 9 503,2 | 25,5 |
| G | 229 | p37 | 10 800,0 | 3,9 | 10 800,3 | 18,1 | 10 645,2 | 100,0 |
| | | p38 | 10 800,0 | 3,7 | 10 800,5 | 25,2 | 10 769,9 | 100,0 |
| | | p39 | 10 800,0 | 4,3 | 10 800,3 | 17,4 | 10 801,0 | 5,9 |
| | | p40 | 10 801,4 | 5,4 | 10 800,3 | 15,9 | 10 770,0 | 100,0 |
| | 287 | p45 | 10 801,4 | 3,5 | 10 800,2 | 27,1 | 10 764,5 | 100,0 |
| | | p46 | 10 800,8 | 4,9 | 10 800,2 | 28,1 | 10 771,1 | 100,0 |
| | | p47 | 10 801,2 | 4,1 | 10 800,2 | 26,4 | 10 761,5 | 100,0 |
| | | p48 | 10 801,4 | 5,0 | 10 805,6 | 27,4 | 10 763,1 | 100,0 |
| Moyenne G | | | 10 800,8 | 4,4 | 10 800,9 | 23,2 | 10 755,8 | 88,2 |
| TG | 519 | p53 | 10 802,9 | 5,4 | 10 801,0 | 31,1 | 10 777,5 | 100,0 |
| | | p54 | 10 804,3 | 7,8 | 10 800,9 | 32,5 | 10 781,8 | 100,0 |
| | | p55 | 10 805,3 | 3,8 | 10 801,1 | 26,2 | 10 783,7 | 100,0 |
| | | p56 | 10 800,0 | 6,5 | 10 800,6 | 28,1 | 10 785,9 | 100,0 |
| | 688 | p61 | 10 814,5 | 7,2 | 10 801,1 | 36,8 | 10 785,3 | 100,0 |
| | | p62 | 10 804,4 | 9,8 | 10 803,8 | 38,6 | 10 787,6 | 100,0 |
| | | p63 | 10 800,0 | 5,4 | 10 803,2 | 31,8 | 10 808,1 | 100,0 |
| | | p64 | 10 800,0 | 5,4 | 10 803,2 | 31,8 | 10 808,1 | 100,0 |
| Moyenne TG | | | 10 804,1 | 6,7 | 10 802,0 | 32,4 | 10 786,9 | 100,0 |

Tableau 5.V – Comparaison des performances de la méthode proposée et de CPLEX.

En effet, lorsque nous donnons les inégalités fortes à CPLEX, la formulation est meilleure et ce dernier est capable de se concentrer sur la résolution du problème plus que sur la génération des coupes. D'autant plus que la petite taille des instances n'est pas très affectée par le nombre d'inégalités fortes introduites. Pour cette raison, les deux méthodes "B&P&C II" et "CPLEX avec IF" nécessitent à peu près les mêmes durées d'exécution, sauf pour une seule instance où CPLEX prend la moitié du temps requis par notre méthode.

Concernant les instances de taille moyenne, la méthode "B&P&C II" performe considérablement mieux que CPLEX. Celui-ci donne des gaps très élevés avec la méthode "CPLEX sans IF" (16,3% en moyenne), et est incapable de trouver une borne inférieure (qui est alors à zéro) dans deux cas avec la méthode "CPLEX avec IF". Les gaps sont donc de 100%. Dans ces cas, l'introduction des inégalités fortes alourdit le modèle et rend impossible à CPLEX de trouver une solution initiale à la relaxation linéaire. Pour le reste des instances, nous pouvons affirmer que "CPLEX avec IF" est assez compétitif avec notre méthode.

Enfin, pour les instances de plus grandes tailles (groupes G et TG), "CPLEX avec IF" est encore une fois incapable, dans la majorité des cas (15 sur 16), de trouver une solution réalisable initiale aux problèmes relaxés. Pour les deux autres méthodes, la méthode "B&P&C II" performe de loin mieux que "CPLEX sans IF" : nous constatons une amélioration du gap de 79%[3]. Nous en concluons que les coupes générées par CPLEX ne sont pas aussi efficaces que les inégalités valides que nous générons à chaque itération de notre méthode.

#### 5.3.3.3 Comparaison de notre méthode à la méthode heuristique de *branch-and-bound* avec *price-and-cut* à la racine

Pour comparer notre méthode ("B&P&C II") à la méthode heuristique de *branch-and-bound* avec *price-and-cut* à la racine ("HeurB&B + P&C"), nous nous basons à la fois sur le gap et la durée d'exécution. Étant donné que la méthode "HeurB&B + P&C"

[3] Il s'agit de l'écart relatif entre le gap moyen obtenu par notre méthode et celui obtenu par "CPLEX sans IF", calculé par la formule : $(GAP_{CPLEX sans IF} - GAP_{B\&P\&C II})/GAP_{CPLEX sans IF}$.

utilise une heuristique, nous n'utilisons aucune borne supérieure initiale pour résoudre les instances. Nous faisons de même pour notre méthode pour assurer la justesse de la comparaison.

Cependant, pour comparer les bornes inférieures obtenues, le calcul du gap se fait en se basant sur les bornes supérieures dont le calcul a été décrit dans la section 5.2.1.

Les résultats sont rapportés dans le tableau 5.VI. Notons que la méthode "HeurB&B + P&C" se termine dans certains cas avant que le gap ne soit nul même si la limite de temps n'est pas atteinte, et ce vu son caractère heuristique.

En fait, le temps d'exécution de cette méthode ("HeurB&B + P&C") correspond au temps que prend la résolution à la racine plus une heure d'exécution de l'heuristique de *branch-and-bound* de CPLEX et du polissage. Comme la méthode prend moins d'une heure pour résoudre les instances des groupes P, M et G à la racine, le temps d'exécution total pour ces trois groupes d'instances ne dépasse pas 2 heures. Contrairement à cela, la méthode "B&P&C II" atteint la limite de 3 heures pour les instances du groupe G et la majorité des instances du groupe M, et ce malgré qu'elle ne dépasse pas 1000 secondes à la racine. Ceci peut se justifier par le fait que notre méthode cherche une solution optimale exacte au problème, et ne se serait arrêtée qu'une fois cette solution trouvée, si aucune limite de temps n'avait été fixée.

Pour ces trois groupes d'instances (P, M et G), les résultats montrent que la méthode heuristique prend un temps remarquablement meilleur pour donner une solution approximative aux problèmes. La méthode "B&P&C II" permet par contre d'obtenir des gaps meilleurs avant ou à la fin de la limite de temps, en plus de résoudre exactement cinq fois plus d'instances que la méthode "HeurB&B + P&C" (10 versus 2 instances résolues).

Pour les instances de très grande taille, les deux méthodes arrivent à la limite de temps. Cependant, tandis que notre méthode arrive à explorer d'autres nœuds de l'arbre de recherche (vu que la résolution à la racine prend moins de 3 heures), la méthode "HeurB&B + P&C" s'arrête à la racine et reste donc incapable d'exécuter l'heuristique de polissage. Les gaps obtenus par les deux méthodes sont pratiquement les mêmes, à une décimale près.

En conclusion nous pouvons affirmer que notre méthode demeure compétitive, voire

| Groupe | \|A\| | Instance | B&P&C II | | HeurB&B + P&C | |
|---|---|---|---|---|---|---|
| | | | CPU (s) | GAP (%) | CPU (s) | GAP (%) |
| P | 230 | p33 | 1,2 | 0,00 | 0,4 | 0,00 |
| | | p34 | 1,3 | 0,00 | 0,5 | 0,00 |
| | | p35 | 38,9 | 0,00 | 0,4 | 0,17 |
| | | p36 | 613,3 | 0,00 | 0,5 | 0,93 |
| | 289 | p41 | 2,2 | 0,00 | 0,4 | 0,09 |
| | | p42 | 7,2 | 0,00 | 0,4 | 0,51 |
| | | p43 | 290,2 | 0,00 | 0,4 | 0,47 |
| | | p44 | 497,8 | 0,00 | 0,4 | 1,18 |
| Moyenne P | | | 181,5 | 0,00 | 0,4 | 0,42 |
| M | 517 | p49 | 7 503,1 | 0,00 | 27,8 | 0,72 |
| | | p50 | 10 800,2 | 3,84 | 4 353,1 | 4,99 |
| | | p51 | 10 800,4 | 0,50 | 9,0 | 0,98 |
| | | p52 | 10 800,0 | 3,22 | 4 177,6 | 4,02 |
| | 669 | p57 | 93,5 | 0,00 | 12,8 | 0,05 |
| | | p58 | 10 800,5 | 0,39 | 3 770,4 | 1,10 |
| | | p59 | 10 800,2 | 0,77 | 7,7 | 0,96 |
| | | p60 | 10 800,3 | 1,15 | 49,2 | 1,62 |
| Moyenne M | | | 9 049,8 | 1,23 | 1 551,0 | 1,81 |
| G | 229 | p37 | 10 800,6 | 4,46 | 4 840,0 | 5,71 |
| | | p38 | 10 800,0 | 4,14 | 5 199,8 | 5,41 |
| | | p39 | 10 800,0 | 4,63 | 4 923,2 | 5,93 |
| | | p40 | 10 800,0 | 5,66 | 5 632,0 | 6,54 |
| | 287 | p45 | 10 800,7 | 3,84 | 4 881,1 | 4,70 |
| | | p46 | 10 800,0 | 5,17 | 5 631,9 | 6,10 |
| | | p47 | 10 800,0 | 5,17 | 5 631,9 | 6,10 |
| | | p48 | 10 800,0 | 5,25 | 5 871,1 | 6,02 |
| Moyenne G | | | 10 800,2 | 4,68 | 5 174,1 | 5,69 |
| TG | 519 | p53 | 10 800,1 | 5,37 | 11 041,3 | 5,49 |
| | | p54 | 10 800,7 | 7,75 | 11 375,1 | 7,87 |
| | | p55 | 10 804,5 | 3,92 | 11 006,0 | 4,04 |
| | | p56 | 10 806,8 | 6,54 | 11 281,6 | 6,62 |
| | 688 | p61 | 10 809,1 | 7,10 | 11 202,2 | 7,15 |
| | | p62 | 10 801,1 | 9,90 | 11 508,6 | 9,90 |
| | | p63 | 10 800,0 | 5,46 | 11 134,2 | 5,46 |
| | | p64 | 10 801,2 | 8,01 | 11 450,0 | 8,01 |
| Moyenne TG | | | 10 802,9 | 6,76 | 11 249,9 | 6,82 |

Tableau 5.VI – Comparaison des performances de la méthode proposée et de la méthode heuristique de *branch-and-bound* avec *price-and-cut* à la racine.

meilleure, notamment pour les instances de très grande taille. Ces résultats sont assez satisfaisants comme il s'agit d'une méthode exacte comparée à une méthode heuristique.

#### 5.3.3.4 Conclusion sur la performance de la méthode proposée

À partir des tests que nous avons effectué sur l'échantillon d'instances dont nous disposons (que nous considérons assez diversifié), nous constatons que la méthode que nous proposons pour résoudre le problème de conception de réseaux avec coûts d'ajout de capacité est performante. D'une part, elle performe considérablement mieux que la méthode de *branch-and-bound* de CPLEX pour toutes les instances, et d'autre part, elle reste très compétitive par rapport à la méthode heuristique de *price-and-cut* à la racine proposée par Frangioni et Gendron [30], notamment pour les instances de très grande taille.

# CHAPITRE 6

# CONCLUSION

Les problèmes de conception de réseaux reviennent souvent dans diverses applications liées aux domaines du transport et des télécommunications. Ils constituent une classe de problèmes où les décisions stratégiques et opérationnelles sont confrontées, et nécessitent d'effectuer les bons choix et de trouver les meilleurs compromis dans l'objectif de réduire les coûts globaux ou les dépenses encourues.

Dans ce mémoire, nous avons présenté une méthode exacte, utilisant des techniques de la programmation en nombres entiers, pour résoudre un problème de conception de réseaux avec coûts d'ajout de capacité. Nous nous sommes basés sur les méthodes de génération de colonnes et de génération de coupes, que nous avons imbriquées dans un algorithme de *branch-and-bound* basé sur la relaxation continue.

Nous avons comparé notre méthode à CPLEX, un des logiciels d'optimisation les plus connus, ainsi qu'à une méthode existante dans la littérature, qui combine génération de colonnes et de coupes, et utilise une heuristique de *polishing* de CPLEX pour limiter et améliorer le temps d'exécution.

Les résultats ont montré que dans les deux cas, notre méthode a donné des résultats meilleurs que ceux obtenus par CPLEX ou par la méthode heuristique, et ce principalement pour les instances de grande et très grande taille (avec 200 et 400 produits).

Ce travail étant réalisé, plusieurs modifications peuvent être envisagées pour améliorer davantage les résultats obtenus. Nous proposons dans un premier temps d'introduire un critère d'arrêt permettant de suspendre la méthode de génération de colonnes lorsque la solution ne s'améliore plus durant un certain nombre d'itérations. Nous proposons également d'effectuer la génération de colonnes sur les variables $x$ et $y$, en ne considérant qu'un sous-ensemble de segments arcs - niveaux de capacité $((i,j),s)$ initialement.

Enfin, nous suggérons de combiner notre méthode à des méthodes heuristiques permettant, par exemple, de limiter le nombre de nœuds visités dans le cadre de la méthode de *branch-and-bound*. Ceci permettra, entre autres, de tester notre méthode sur des ins-

tances de tailles réelles.

## 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*, 33:42–54, 2005.

[3] Y. K. Agarwal. Design of capacitated multicommodity networks with multiple facilities. *Operations Research*, 50:333–344, 2002.

[4] R. K. Ahuja, T. L. Magnanti et M.R. Reddy. Applications of network optimization. Dans M.O. Ball et al., éditeurs, *Network Models, Handbooks in Operations Research and Management Science*, volume 7, chaptitre 1, pages 1–83. Elsevier Science B.V., 1995.

[5] R. K. Ahuja, T. L. Magnanti et J. B. Orlin. *Network Flows : Theory, Algorithms and Applications*. Prentice Hall, 1993.

[6] A. Atamtürk. On capacitated network design cut-set polyhedra. *Mathematical Programming*, 92:425–437, 2002.

[7] A. Atamtürk et O. Günlük. Network design arc-set with variable upper bounds. *Networks*, 50:17–28, 2007.

[8] A. Atamtürk et D. Rajan. On splittable and unsplittable flow capacitated network design arc-set polyhedra. *Mathematical Programming*, 92:315–333, 2002.

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

[10] A. Balakrishnan, T. L. Magnanti et R. T. Wong. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37:716–740, 1989.

[11] M. S. Bazaraa, J. J. Javis et H. D. Sherali. *Linear programming and network flows*. John Wiley and Sons, 2ème édition, 1990.

[12] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Computational Management Science*, 2(1):3–19, 2005.

[13] D. Berger, B. Gendron, J. Y. Potvin, S. Raghavan et P. Soriano. Tabu search for a network loading problem with multiple facilities. *Journal of Heuristics*, 6:253–267, 2000.

[14] D. P. Bertsekas. A unified framework for primal-dual methods in minimum cost network flow problems. *Mathematical Programming*, 32:125–145, 1985.

[15] J. Billheimer et P. Gray. Network design with fixed and variable cost elements. *Transportation Science*, 7:49–74, 1973.

[16] D. E. Boyce, A. Farhi et R. Weischedel. Optimal network problem : a branch-and-bound algorithm. *Environment and Planning*, 5(4):519–533, 1973.

[17] M. Chouman, T. G. Crainic et B. Gendron. A cutting-plane algorithm for multicommodity capacitated fixed-charge network design. Publication CIRRELT-2009-20, 2009.

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

[19] A. M. Costa, J. F. Cordeau, B. Gendron et G. Laporte. Accelerating benders decomposition approach with heuristic master problem solutions. *Pesquisa Operacional*, 32(1):3–20, 2012.

[20] T. G. Crainic. Service network design in freight transportation. *European journal of operational research*, 122:272–288, 2000.

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

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

[23] 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.

[24] G. Dahl et M. Stoer. A cutting plane algorithm for multicommodity survivable network design problems. *INFORMS Journal on Computing*, 10(1):1–11, 1998.

[25] G. B. Dantzig et P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29(4):767–778, 1961.

[26] J. Desrosiers et M. E. Lübbecke. A primer in column generation. Dans G. Desaulniers, J. Desrosiers et M. M. Solomon, éditeurs, *Column Generation*, chaptitre 1, pages 1–32. Springer US, 2005.

[27] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[28] R. Dionne et M. Florian. Exact and approximate algorithms for optimal network design. *Networks*, 9(1):37–59, 1979.

[29] M. Florian, G. Brushel, J. Ferland, G. G. Guerin et L. Nastansky. The engine scheduling problem in a railway network. *INFOR*, 14(2):121–138, 1976.

[30] A. Frangioni et B. Gendron. 0-1 reformulations of the multicommodity capacitated network design problem. *Discrete Applied Mathematics*, 157:1229–1241, 2009.

[31] V. Gabrel et M. Minoux. LP relaxations better than convexification for multicommodity network optimization problems with step increasing cost functions. *ACTA Mathematica Vietnamica*, 22(1):123–145, 1997.

[32] B. Gendron. Decomposition methods for network design. *Procedia Social and Behavioral Sciences*, 20:31–37, 2011.

[33] B. Gendron et T. G. Crainic. Relaxations for multicommodity capacitated network design problems. Publication CRT-945, 1994.

[34] B. Gendron et M. Larose. Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. Publication CIRRELT-2012-74, 2012.

[35] B. Gendron, J. Y. Potvin et P. Soriano. Diversification strategies in local search for a nonbifurcated network loading problem. *European journal of operational research*, 142:231–241, 2002.

[36] A. M. Geoffrion et G. W. Graves. Multicommodity distribution system design by benders decomposition. *Management Science*, 20(5):822–844, 1974.

[37] 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.

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

[39] P. Hansen. The optimum rented lines network problem. Dans *Symposium "Operations Research in Telecommunications" held at Rutgers University*. Rutgers Center for Operations Research, Novembre 1984.

[40] J. Hellstrand, T. Larsson et A. Migdalas. A characterization of the uncapacitated network design polytope. *Operations Research Letters*, 12:159–163, 1992.

[41] 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*, 22:314–325, 2010.

[42] F. S. Hillier et G. J. Lieberman. *Introduction to Operations Research*, chaptitre 12. McGraw-Hill, 7ème édition, 2001.

[43] H. H. Hoang. A computational approach to the selection of an optimal network. *Management Science*, 19(5):488–498, 1973.

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

[45] D. Kim et C. Barnhart. Transportation service network design : Models and algorithms. Dans *Computer-Aided Transit Scheduling*, volume 471 de *Lecture Notes in Economics and Mathematical Systems*, pages 259–283, 1999.

[46] D. Kim, C. Barnhart et K. Ware. Multimodal express package delivery : A service network design application. *Transportation Science*, 33:391–407, 1999.

[47] G. Kliewer et L. Timajev. Relax-and-cut for capacitated network design. Dans *Algorithms-ESA 2005, 13th Annual European Symposium*, volume 3369 de *Lecture Notes in Computer Science*, pages 47–58, 2005.

[48] M. Larose. Développement d'un algorithme de branch-and-price-and-cut pour le problème de conception de réseau avec coûts fixes et capacités. Mémoire de maîtrise, Université de Montréal, 2011.

[49] M. Los et C. Lardinois. Combinatorial programming, statistical optimization and the optimal transportation network problem. *Transportation Research Part B : Methodological*, 16(2):89–124, 1982.

[50] T. L. Magnanti et R. T. Wong. Network design and transportation planning : models and algorithms. *Transportation Science*, 18(1):1–55, 1984.

[51] T. L. Magnanti, P. Mirchandani et R. Vachani. The convex hull of two core capacitated network design problems. *Mathematical Programming*, 60:233–250, 1993.

[52] T. L. Magnanti, P. Mirchandani et R. Vachani. Modeling and solving the two-facility capacitated network loading problem. *Operations Research*, 43:142–157, 1995.

[53] T. L. Magnanti, P. Mireault et R. T. Wong. Tailoring Benders decomposition for network design. *Mathematical Programming Studies*, 26:112–154, 1986.

[54] M. Minoux. Network synthesis and optimum network design problems : Models, solution methods and applications. *Network*, 19:313–360, 1989.

[55] P. Mirchandani. Projections of the capacitated network loading problem. *European Journal of Operational Research*, 122:534–560, 200.

[56] G. L. Nemhauser et L. A. Wolsey. *Integer and combinatorial optimization*. John Wiley and Sons, 1999.

[57] R. L. Radin et U. Choe. Tighter relaxations of fixed charge network flow problems. Rapport technique, School of Industrial and Systems Engineering, The Georgia Institute of Technology, Atlanta, Georgia., 1979.

[58] R. Richardson. An optimization approach to routing aircraft. *Transportation Science*, 10(1):52–71, 1976.

[59] M. Sellmann, G. Kliewer et A. Koberstein. Lagrangian cardinality cuts and variable fixing for capacitated network design. Dans R. Möhring et R. Raman, éditeurs, *Algorithms-ESA 2002, 10th Annual European Symposium*, volume 2461 de *Lecture Notes in Computer Science*, pages 845–858, 2002.

[60] L. A. Wolsey. *Integer Programming*. John Wiley and Sons, 1998.