Université de Montréal

Algorithme de branch-and-price-and-cut pour le problème de conception de réseaux avec coûts fixes, capacités et un seul produit

par
Ghalia Kéloufi

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

Décembre, 2015

RÉSUMÉ

De nombreux problèmes liés aux domaines du transport, des télécommunications et de la logistique peuvent être modélisés comme des problèmes de conception de réseaux. Le problème classique consiste à transporter un flot (données, personnes, produits, etc.) sur un réseau sous un certain nombre de contraintes dans le but de satisfaire la demande, tout en minimisant les coûts.

Dans ce mémoire, on se propose d'étudier le problème de conception de réseaux avec coûts fixes, capacités et un seul produit, qu'on transforme en un problème équivalent à plusieurs produits de façon à améliorer la valeur de la borne inférieure provenant de la relaxation continue du modèle.

La méthode que nous présentons pour la résolution de ce problème est une méthode exacte de branch-and-price-and-cut avec une condition d'arrêt, dans laquelle nous exploitons à la fois la méthode de génération de colonnes, la méthode de génération de coupes et l'algorithme de branch-and-bound. Ces méthodes figurent parmi les techniques les plus utilisées en programmation linéaire en nombres entiers.

Nous testons notre méthode sur deux groupes d'instances de tailles différentes (grandes et très grandes), et nous la comparons avec les résultats donnés par CPLEX, un des meilleurs logiciels permettant de résoudre des problèmes d'optimisation mathématique, ainsi qu'avec une méthode de branch-and-cut. Il s'est avéré que notre méthode est prometteuse et peut donner de bons résultats, en particulier pour les instances de très grandes tailles.

Mots clés: Problème de conception de réseaux à un seul produit, problème de conception de réseaux multi-produits, branch-and-bound, génération de colonnes, méthodes de coupes.

ABSTRACT

Many problems in transportation, telecommunications and logistics can be formulated as network design problems. The general problem consists in identifying a subgraph of the network on which facilities are to be built, and the quantities of products to transport on it, subject to a set of constraints, with the objective of minimizing total costs.

In this dissertation, the aim is to adress the capacitated network design problem with fixed costs and a single commodity, that we transform into a muticommodity network design problem in order to improve the value of the lower bound of the relaxed model.

The proposed method is a branch-and-price-and-cut algorithm with stopping criterion that combines column generation, cutting-plane and branch-and-bound methods.

Numerical tests are performed using two groups of instances with different sizes, large and very large. The results are compared to those given by CPLEX, one of the most efficient software tools for integer programming, and to a branch-and-cut algorithm. It is shown that the method we adopted is competitive in particular for very large instances.

Keywords : Single-commodity capacitated network design problem, multicommodity capacitated network design problem, branch-and-bound, column generation, cutting-plane methods.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1

INTRODUCTION

Le problème de conception de réseaux est un des problèmes les plus étudiés en recherche opérationnelle, étant donné son importance et ses différentes applications en logistique, télécommunications, transport et localisation. Le problème général consiste à transporter un flot (données, personnes, produits, etc.) sur un réseau sous un certain nombre de contraintes dans le but de satisfaire la demande, tout en minimisant le coût total englobant le coût de transport et le coût d'installation des équipements.

Le problème de conception de réseaux avec coûts fixes et capacités consiste à choisir un sous-ensemble d'arcs et de nœuds dans le graphe sous-jacent au réseau, à un coût minimum, afin de satisfaire la demande, tout en respectant les contraintes de capacités des équipements (arcs).

Ce problème est simple à formuler en programme linéaire en nombres entiers. Il fait toutefois partie des problèmes NP-difficiles. La plupart des méthodes proposées pour sa résolution sont soit basées sur des heuristiques, soit sur une combinaison d'heuristiques et de méthodes exactes, notamment pour les instances de très grande taille.

Dans ce mémoire, on se propose d'étudier le problème de conception de réseaux avec coûts fixes, capacités et un seul produit (SCND), qu'on transforme en un problème à plusieurs produits (MCND). Le but de cette transformation est d'améliorer la borne inférieure tirée de la relaxation continue du modèle. Cette transformation entraîne toutefois une augmentation de la taille du modèle, autant du nombre de variables que de contraintes. C'est pourquoi nous tenterons de résoudre le problème par une méthode spécialisée qui génère les variables et les contraintes de manière dynamique et en nombre restreint, tout en garantissant de résoudre la relaxation continue.

Nous proposons ainsi une méthode exacte de branch-and-price-and-cut, dans laquelle nous exploitons à la fois la méthode de génération de colonnes, la méthode de génération de coupes et l'algorithme général de branch-and-bound basé sur la relaxation continue. Nous ajoutons une condition d'arrêt qui permet d'arrêter ces méthodes avant l'obtention

de la valeur optimale de relaxation continue lorsque la solution obtenue ne s'améliore pas après un certain nombre d'itérations. Ceci permet d'améliorer davantage la performance de notre méthode en terme de temps de résolution de chaque nœud de l'arbre de recherche, afin de pouvoir résoudre efficacement des problèmes de très grande taille.

Notre mémoire est organisé comme suit. Dans le chapitre 2, nous exposerons quelques notions de base de la programmmation entière, notamment les définitions et les théorèmes qui en relèvent, en plus d'un descriptif des méthodes principales de résolution. Le chapitre 3 présentera une revue de la littérature sur les variantes du problème de conception de réseaux, ainsi que les méthodes de résolution. Dans le chapitre 4, nous décrirons en détail le problème étudié et les différentes étapes de l'algorithme proposé. Le chapitre 5 présentera les résultats obtenus et leurs interprétations.

CHAPITRE 2

NOTIONS DE BASE DE PROGRAMMATION LINÉAIRE EN NOMBRES ENTIERS

Dans ce chapitre, nous définissons et présentons les notions de base de la programmation linéaire en nombres entiers (PLNE). Nous abordons ensuite les stratégies fondamentales de résolution de ce type de problèmes, à savoir les méthodes de coupes, l'algorithme de branch-and-bound et la méthode de génération de colonnes. Nous nous inspirons des références suivantes : Hillier et Liberman [31], Michal et Deepankar [42], Wolsey [47], Nemhauser et Wolsey [38], Bazaraa et al. [8], Achterberg [1], Achterberg et al. [2], Atamtürk et Savelsbergh [4], et Chouman et al. [12].

2.1 Programmation linéaire en nombres entiers

Une multitude de quantités peuvent s'exprimer sous forme de nombres réels, issus d'un domaine continu. Au contraire, certaines décisions sont de nature discrète, et doivent être représentées à l'aide de nombres entiers. L'intégration des contraintes d'intégralité dans les modèles linéaires en modifie profondément la nature. Lorsqu'un problème est linéaire avec des variables continues et des variables entières, nous parlons de programmation linéaire mixte. Si les variables entières sont de valeurs 0 ou 1, nous parlons de programmation 0-1 (binaire). La définition formelle de ces modèles est donnée ci-après.

2.1.1 Formulation du PLNE

Soient :

- $A \in \mathbb{R}^{m \times n}$ et $B \in \mathbb{R}^{m \times p}$: des matrices de constantes ;
- $c \in \mathbb{R}^n$, $d \in \mathbb{R}^p$ et $e \in \mathbb{R}^m$: des vecteurs de constantes ;
- $x \in \mathbb{R}^n$: un vecteur des variables continues ;

- $y \in \mathbb{N}^p$: un vecteur des variables entières.

On peut formuler un PLNE comme suit :

$$\begin{aligned} \min \quad & c^T x + d^T y \\ \text{sujet à :} \quad & \\ & Ax + By \geq e \\ & x \in \mathbb{R}^n_+ \\ & y \in \mathbb{N}^p. \end{aligned}$$

2.1.2 Relaxation linéaire

La relaxation linéaire d'un PLNE consiste en la modification des contraintes (2.1) pour permettre aux variables entières de prendre des valeurs continues. Le modèle linéaire relaxé (PLR) est le suivant :

$$\begin{aligned} \min \quad & c^T x + d^T y \\ \text{sujet à :} \quad & \\ & Ax + By \geq e \\ & x \in \mathbb{R}^n_+ \\ & y \in \mathbb{R}^p_+ \end{aligned} \tag{2.1}$$

L'objectif de la relaxation linéaire est de transformer le PLNE en un PLR beaucoup plus facile à résoudre, et dont la solution optimale constitue une borne inférieure de la solution du PLNE. Si la solution optimale du PLR est entière, elle est la solution optimale du PLNE. Dans le cas où le domaine réalisable du PLR est vide, le domaine réalisable du PLNE l'est aussi.

2.1.3 Notion de dualité

La notion de dualité a été introduite dès les débuts de la programmation linéaire. Le concept consiste à montrer que chaque programme linéaire (PL) a un autre programme linéaire associé nommé *programme linéaire dual* (PL-D). Le PL original est appelé *pri-*

mal (PL-P).
Formellement, considérons le PL-P suivant :

$$\text{(PL-P)} : \{\min \ \ z_P = c^T x, Ax \geq b, x \geq 0\},$$

où $x, c \in \mathbb{R}^n, b \in \mathbb{R}^m$ et $A \in \mathbb{R}^{m \times n}$. Considérons également un vecteur $u = [u_1, ..., u_p]^T$ où u_i est un multiplicateur associé à la $i^{\text{ème}}$ contrainte du (PL-P). Le problème dual associé est :

$$\text{(PL-D)} : \{\max z_D = b^T u, A^T u \leq c, u \geq 0\}.$$

2.1.3.1 Théorème de dualité faible

Soient x réalisable du (PL-P) et u réalisable du (PL-D). Le théorème de dualité faible affirme que $z_D = b^T u \leq z_P = c^T x$.

2.1.3.2 Théorème de dualité forte

Si un des deux problèmes, primal ou dual, possède une solution optimale avec valeur finie, alors la même chose est vraie pour l'autre problème, et les valeurs optimales des deux problèmes sont égales ($z_D^* = b^T u^* = z_P^* = c^T x^*$). Si un des deux problèmes n'est pas borné, alors le domaine réalisable de l'autre problème est vide.

2.1.3.3 Théorème des écarts complémentaires

Un couple de solutions réalisables (x, u) des problèmes PL-P et PL-D est un couple de solutions optimales *si et seulement si*

- Si une contrainte de l'un des deux programmes (PL-P ou PL-D) n'est pas active (satisfaite à égalité), alors la variable associée à cette contrainte de l'autre programme est nulle.
- Si une variable de décision de l'un des deux programmes est strictement positive, alors la contrainte associée est active.

Formellement, les conditions de complémentarité des problèmes PL-P et PL-D sont :

$$\begin{cases} (Ax-b)^T u = 0 \\ (A^T u - c)^T x = 0 \end{cases}$$

2.2 Méthodes de résolution en programmation linéaire en nombres entiers

2.2.1 Branch-and-bound

Cette méthode repose sur une stratégie *diviser pour régner*, consistant à diviser le problème original en sous-problèmes plus simples à résoudre ayant des domaines de solutions réduits, résolus dans le cadre d'un arbre de recherche. La méthode de branch-and-bound est donné comme suit :

1. Initialisation :
   - $\bar{z} = +\infty$, meilleure borne entière connue.
   - $S = \{\text{PLR}\}$, ensemble des sous-problèmes à résoudre, contenant initialement le problème linéaire relaxé.
2. Résoudre PLR le problème relaxé du PLNE initial. Si la solution du PLR est entière alors arrêter.
3. Choisir une variable y_j non entière dont la valeur est α. Séparer alors le problème en deux sous-problèmes :
   - $\text{PLR}_1 = \text{PLR} \cup \{y_j \leq \lfloor \alpha \rfloor\}$
   - $\text{PLR}_2 = \text{PLR} \cup \{y_j \geq \lceil \alpha \rceil\}$

   Ajouter PLR_1 et PLR_2 à S.
4. Choisir un sous-problème à résoudre à partir de l'ensemble S, soit PLR_k.
   Si PLR_k n'est pas réalisable, retirer le sous-problème PLR_k de l'ensemble S.
   Aller à 6.

5. Soit z_k la valeur optimale de PLR_k :

   (a) Si $z_k \geq \bar{z}$, il n'est pas nécessaire d'explorer davantage le domaine réalisable de PLR_k. Retirer PLR_k de S. Aller à 6.

   (b) Sinon, si la solution optimale de PLR_k associée à la valeur z_k est entière, alors $\bar{z} = z_k$. Retirer PLR_k de S. Aller à 6.

   (c) Sinon (solution continue et possibilité d'amélioration), aller à 3.

6. Si $S \neq \emptyset$ aller à 4. Sinon, arrêter.

Figure 2.1 : Algorithme de branch-and-bound

Cette procédure est habituellement représentée sous forme d'un arbre binaire où, à chaque niveau, une partition du sommet père s'effectue suivant la règle décrite précédemment (étape 3). Il s'agit de parcourir cet arbre d'énumération afin de trouver la solution optimale. Pour implémenter l'algorithme de branch-and-bound, il faut spécifier une stratégie d'exploration de l'arbre (étape 4) et une règle de branchement permettant de choisir la variable sur laquelle brancher (étape 3).

2.2.1.1 Élagage

L'exploration d'un chemin de l'arbre de branch-and-bound peut prendre fin pour trois raisons :

- La solution devient entière (étape 5.b de l'algorithme branch-and-bound 2.1).
- Le domaine réalisable devient vide (étape 4 de l'algorithme branch-and-bound 2.1).
- La valeur de l'objectif correspondant à la solution optimale du problème relaxé est supérieure (moins bonne) à celle d'une solution admissible connue, obtenue à un autre sommet de l'arbre (étape 5.a de l'algorithme branch-and-bound 2.1).

Dans chacun de ces trois cas, on dit que le sommet est élagué, et il est inutile de pousser l'exploration plus loin dans cette direction. L'algorithme s'arrête lorsque tous les sommets sont élagués. La meilleure solution obtenue au cours du déroulement de l'algorithme est alors l'optimum global du PLNE.

2.2.1.2 Stratégies d'exploration de l'arbre

La règle de sélection du nœud spécifie quel sous-problème choisir dans l'ensemble S à chaque itération de l'algorithme. Cette règle a un impact en terme d'ordre de visite de chaque nœud et par conséquent, ceci a un impact sur l'évolution de l'algorithme et la qualité des bornes inférieures et supérieures obtenues. Nous décrivons ci-dessous les règles les plus connues.

2.2.1.2.1 Stratégie meilleur d'abord La recherche meilleur d'abord permet d'améliorer la borne inférieure en sélectionnant le nœud ayant la plus petite borne inférieure. Cette stratégie permet d'explorer en un nombre minimum de nœuds l'arbre de recherche, puisque sans connaître la valeur optimale du problème, elle peut permettre d'orienter l'exploration vers les régions de l'arbre de recherche où la borne inférieure est la plus petite. Elle utilise cependant beaucoup plus de mémoire que la recherche en profondeur puisqu'elle garde plus de nœuds actifs en mémoire et elle ne trouve généralement pas de solutions réalisables rapidement, étant donné qu'elle doit résoudre les problèmes à nouveau en raison de leur dispersion dans l'arbre de recherche.

2.2.1.2.2 Stratégie en profondeur Cette stratégie détermine l'ordre d'exploration de l'arbre a priori : on choisit un nœud fils du nœud précédent dans le dernier branchement. Ceci va permettre d'optimiser l'espace mémoire, puisque les problèmes dans les nœuds fils contiennent une contrainte additionnelle par rapport au problème du nœud père, et peuvent être réoptimisés par l'algorithme dual du simplexe. Si aucun branchement n'est effectué, après l'élagage d'un nœud, on sélectionne le nœud du même niveau ou du niveau supérieur dans l'arbre. Cette technique favorise l'amélioration de la borne supérieure (en contexte de minimisation) en visitant les nœuds les plus profonds, et permet en

conséquence d'identifier des solutions entières plus rapidement. Cependant, des nœuds superflus, qui auraient pu être élagués si une meilleure borne était connue, peuvent être explorés selon cette stratégie.

2.2.1.2.3 Stratégie en largeur Cette technique repose sur l'idée de choisir le nœud le plus proche de la racine. En d'autres mots, reculer le plus possible en arrière pour sélectionner le prochain nœud à évaluer, en y allant niveau par niveau. Comme pour la stratégie meilleur d'abord, le défaut principal de cette approche est la grande consommation de mémoire.

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

Le choix de la stratégie d'exploration de l'arbre est souvent heuristique. Tous les logiciels de PLNE offrent la possibilité d'implémenter et de tester une variété de règles d'exploration de l'arbre.

2.2.1.3 Règles de branchement

La règle de branchement vise à identifier la variable sur laquelle est effectué le branchement. Une méthode simple consiste à choisir la variable la plus fractionnaire, c'est-à-dire dont la partie décimale est la plus proche de 0.5, ou de façon équivalente la variable la moins fractionnaire (la plus proche d'être entière). Par exemple, 4.6 est plus fractionnaire que 1.3. Dans le but d'améliorer la borne inférieure plus rapidement, des règles plus sophistiquées permettent d'estimer l'impact du choix de branchement sur les valeurs de la borne inférieure. Nous décrivons ci-dessous quelques règles. Encore une fois, les logiciels de PLNE proposent une variété de règles de branchement parmi lesquelles l'utilisateur peut choisir.

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

Soient λ_i^0 et λ_i^1 les accroissements de la borne en fixant la valeur de la variable d'indice i (y_i) à 0 et à 1. On définit μ_i^0 et μ_i^1, les accroissements unitaires de la borne comme suit :

$$\mu_i^0 = \frac{\lambda_i^0}{\tilde{y}_i} \qquad ; \qquad \mu_i^1 = \frac{\lambda_i^1}{1-\tilde{y}_i} \tag{2.2}$$

où $\tilde{y}_i \in]0,1[$ est la valeur de la variable y_i dans la solution du problème courant.
Le pseudo-coût (p_i^0 , p_i^1) est ainsi défini comme la moyenne des accroissements unitaires de la borne :

$$p_i^0 = \frac{U_i^0}{N_i^0} \qquad ; \qquad p_i^1 = \frac{U_i^1}{N_i^1} \tag{2.3}$$

où U_i^0 et U_i^1 représentent la somme des accroissements unitaires μ_i^0 et μ_i^1 lorsqu'on branche à 0 et à 1 sur la variable y_i et N_i^0 et N_i^1 représentent le nombre de branchements effectués à 0 et à 1 sur la variable y_i.

Enfin, une fois l'estimation de la valeur des pseudo coûts 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 possible de maximiser la somme des variations sur les deux branches, c'est-à-dire choisir la variable d'indice i^* tel que

$$i^* = \underset{i \in N}{\operatorname{argmax}} \{\tilde{y}_i p_i^0 + (1-\tilde{y}_i) p_i^1\}.$$

Un autre critère consiste à maximiser la variation minimale sur les deux branches, ce qui revient à choisir la variable dont l'indice i^* est tel que

$$i^* = \underset{i \in N}{\operatorname{argmax}} \{\min\{\tilde{y}_i p_i^0, (1-\tilde{y}_i) p_i^1\}\}.$$

Finalement, un troisième critère consiste à sélectionner la variable d'indice

$$i^* = \underset{i \in N}{\operatorname{argmax}} \{\theta \min\{\tilde{y}_i p_i^0, (1-\tilde{y}_i) p_i^1\} + (1-\theta) \max\{\tilde{y}_i p_i^0, (1-\tilde{y}_i) p_i^1\}\},$$

où θ est un paramètre dans l'intervalle [0,1], souvent choisi proche de 1 afin de conserver l'arbre de recherche équilibré [2, 23].

2.2.1.3.2 Branchement fort Le branchement fort permet de déterminer la variable fractionnaire qui améliore le plus la valeur de la fonction objectif quand elle est fixée à une valeur entière. Le principe de cette méthode est de résoudre les relaxations linéaires de tous les problèmes résultants du branchement sur chacune des variables fractionnaires pour déterminer la meilleure variable sur laquelle il faut brancher. L'inconvénient est le long temps de calcul qu'elle requiert. Pour accélérer la méthode, une alternative possible est de ne considérer qu'un sous-ensemble de variables les plus prometteuses.

2.2.1.3.3 Pseudo-coûts avec initialisation par branchement fort Etant donné que la méthode de branchement fort est très coûteuse en terme de temps de calcul, elle n'est généralement pratiquée que pendant les premières itérations de la méthode de branch-and-bound pour initialiser le pseudo-coût. La méthode des pseudo-coûts est par la suite utilisée.

2.2.1.3.4 Branchement fiable Cette méthode de branchement est une généralisation du branchement basé sur les pseudo-coûts. Le branchement fort n'est pas effectué uniquement sur des variables ayant des pseudo-coûts non initialisés, mais aussi sur des variables avec des pseudo-coûts jugés non fiables. Un seuil de fiabilité η_f est défini et un pseudo-coût est alors jugé fiable lorsque $min\{N_i^0, N_i^1\} \geq \eta_f$.

2.2.2 Méthode de génération de colonnes

L'un des problèmes de la programmation linéaire en nombres entiers de grande taille est le grand nombre de variables. La génération de colonnes est l'une des principales

méthodes utilisées pour résoudre ce type de problèmes. Son principe consiste à ne pas considérer explicitement toutes les variables du problème linéaire initial, dit *problème maître* (MP), mais uniquement un ensemble réduit de variables et formuler ainsi le "problème maître restreint" (RMP). Des variables sont ajoutées au RMP au fur et à mesure des itérations, jusqu'à atteindre l'optimalité.

Soit $\mathbb{N} = \{1, .., n\}$ l'ensemble des indices des variables x_q du modèle 2.1, qui est alors considéré comme le problème maître (MP). Pour résoudre (MP), on considère un sous-ensemble de variables $\overline{\mathbb{N}} \subseteq \mathbb{N}$ sur lequel on résout le "problème maître restreint" suivant :

$$\begin{aligned} \min \quad & c^T x + d^T y \\ \text{sujet à :} \quad & \\ & Ax + By \geq e \\ & x_q \geq 0, \ \ q \in \overline{\mathbb{N}} \\ & x_q = 0, \ \ q \in \mathbb{N} \backslash \overline{\mathbb{N}} \\ & y \in \mathbb{R}^p_+ \end{aligned} \tag{2.4}$$

Lors de chaque itération, on applique l'algorithme du simplexe pour résoudre un "sous-problème" par rapport aux variables duales, en vue d'améliorer la valeur de l'objectif de (MP). Parmi les variables hors base, on choisit la variable $\bar{x}_q$ qui a le coût réduit le plus négatif pour l'ajouter au problème maître restreint.

Le coût réduit associé à la variable $\bar{x}_q$ est donné par :

$$\overline{c_q} = c_q - \overline{\pi} a_q \tag{2.5}$$

c_q est la nième composante du vecteur des coûts c, a_q est la nième colonne de la matrice A, et $\overline{\pi}$ est le vecteur des variables duales optimales associées aux contraintes (2.4) du problème maître restreint.

Si tous les coûts réduits sont non négatifs, on peut conclure que la solution optimale du problème maître restreint actuel l'est aussi pour le problème maître (MP).

Dans le cas contraire, le problème maître restreint (RMP) est réoptimisé après l'introduction d'une ou plusieurs variables ayant des coûts réduits négatifs. Nous invitons le

lecteur à consulter les références [22] et [23].

2.2.3 Méthode de branch-and-price

L'algorithme de branch-and-bound combiné avec la génération de colonnes est appelé algorithme de branch-and-price. Plus précisément, dans un algorithme de branch-and-price, l'exploration de l'arbre se fait de la même manière que dans un algorithme de branch-and-bound classique, mais au lieu de résoudre les relaxations linéaires à chaque nœud avec l'algorithme du simplexe, le PL est résolu en utilisant la génération de colonnes. Les colonnes générées peuvent être valides dans tout l'arbre ou seulement dans la branche courante. Selon [43], les deux principales difficultés dans la mise en œuvre d'un branch-and-price sont :

- Formuler le sous-problème et être capable de le résoudre rapidement (même difficulté que dans le cadre de la génération de colonnes) ;
- Trouver une règle de branchement adaptée qui ne perturbe pas le sous-problème (difficulté propre au branch-and-price).

2.2.4 Méthodes de coupes

Pour appliquer la méthode de branch-and-bound afin d'améliorer la formulation et d'obtenir de meilleures bornes inférieures sur la valeur optimale fournie par la relaxation linéaire, il faut ajouter des contraintes basées sur l'intégralité des variables. Le problème obtenu contient donc beaucoup plus de contraintes et est plus difficile à résoudre dans un temps raisonnable. On fait donc appel à la génération des coupes pour résoudre un problème de programmation linéaire en nombres entiers ayant un très grand nombre de contraintes. Il s'agit de commencer avec uniquement un sous-ensemble de contraintes, puis d'introduire itérativement les contraintes violées.

La méthode de génération de coupes consiste à résoudre itérativement le problème de séparation pour identifier des inégalités valides violées (voir la section 2.2.4.1) et

résoudre le nouveau problème obtenu après l'ajout de ces inégalités valides à la formulation. Ceci vise à réduire le domaine réalisable de la relaxation linéaire sans pour autant modifier celui du problème en nombres entiers. On arrête ce processus si l'une des situations est rencontrée :

- Il n'y a plus d'inégalités valides violées.
- La valeur optimale obtenue est supérieure ou égale à la borne supérieure (meilleure valeur optimale connue)
- La variation de la borne est inférieure à un seuil.

Les algorithmes de branch-and-bound et de génération de coupes combinés sont appelés algorithmes de branch-and-cut. La génération de coupes est appliquée à chaque nœud dans le but d'améliorer le temps de calcul.

2.2.4.1 Inégalités valides

Une inégalité valide (IV) ou coupe est une contrainte satisfaite par toutes les solutions du domaine réalisable du problème en nombres entiers. Dans cette section, nous présentons deux grandes classes d'IV pertinentes au problème de conception de réseaux dans le but d'améliorer les bornes obtenues à partir des relaxations linéaires : les inégalités de chemin de flot (*"flow path inequalities"*) et les inégalités de couverture de flot (*"flow cover inequalities"*).

Les techniques de base utilisées dans la dérivation des IV ont été beaucoup étudiées. Nous référons le lecteur intéressé à Nemhauser and Wolsey [38], pour une analyse détaillée.

La notion d'inégalité valide a été développée pour le problème de *sac-à-dos* qui consiste à déterminer un sous-ensemble optimal d'objets à mettre dans un sac sans dépasser une certaine capacité de ce sac d. En considérant les variables binaires y_j, pour tout $j = 1,..,n$, égales à 1 si l'objet i est mis dans le sac et 0 sinon, la contrainte de capacité s'écrit :

$$\sum_{j=1}^{n} u_j y_j \leq d,$$

où les coefficients u_j (poids des objets) et d sont des entiers (il est également naturel de supposer que $u_j \leq d$ pour tout $j = 1,..,n$). Le domaine réalisable est noté P.

Avant de présenter les différents classes d'inégalités, nous allons introduire quelques définitions.

Définition Un sous-ensemble $C \subset N = \{1,2,...n\}$, où N est l'ensemble des indices, représente une couverture de P si

$$\sum_{j \in C} u_j > d$$

Cette couverture est minimale si elle ne contient strictement aucune autre couverture

$$\sum_{j \in C \setminus \{k\}} u_j \leq d \quad \forall k \in C.$$

Proposition [46] Toute couverture C de P induit une inégalité de couverture, dont la forme est :

$$\sum_{j \in C} y_j \leq |C| - 1.$$

Cette inégalité est une inégalité valide de P.

2.2.4.1.1 Inégalités de couverture de flot Les inégalités de couverture de flot ont été développées initialement par Padberg et al. [41] dans le cadre d'une étude des propriétés polyédrales de la structure du nœud seul *("single node")*, identifiée dans les relaxations du problème de conception de réseaux défini par les variables binaires $y_j, j \in N$ (Atamtürk et al. [3], Wolsey [48] et Gabrel [24]). Les inégalités de couverture de flot sont énoncées comme suit :

Proposition [12] Si $C \subset N$ est une couverture de flot et $\lambda = \sum_{j \in C} u_j - d$ l'excès de capacité, alors

$$\sum_{j \in C} x_j \leq d - \sum_{j \in C} (u_j - \lambda)^+ (1 - y_j)$$

est une IV pour P_{SN}, où $P_{SN} = \{y \in \{0,1\}^n, x \in R_+^n, \sum_{j \in N} x_j \leq d, x_j \leq u_j y_j, \forall j \in N\}$ est le domaine réalisable de la structure du nœud seul, et $(x)^+ = \max(0,x)$.

2.2.4.1.2 Inégalités de chemin de flot Cette section est inspirée du travail de Christophel [15] et Atamtürk et al. [5].

Les coupes de chemin de flot ont été introduites pour la première fois par Van Roy et Wolsey [44, 45] qui ont essayé de dériver des inégalités valides pour le problème de conception de réseaux avec coûts fixes et sans capacité.

Considérons le graphe $G = (V \cup \{s_v, t_v\}, A)$, où s_v et t_v sont des nœuds source et puits, et $V = \{1, .., K\}$ est un ensemble des nœuds de chemin (nœuds de transfert, "path nodes"). Soit N_k^+ l'ensemble des arcs entrants au sommet k à partir de s_v et N_k^- l'ensemble des arcs sortant du sommet k vers t_v. Notons ces arcs par $l_k \in N_k^+ \cup N_k^-$. Notons également la demande au nœud k par b_k et considérons les variables de flot x_k sur l'arc l_k et les variables binaires y_k de conception de l'arc l_k. Nous définissons un chemin à coût fixe comme suit :

Définition 1 Un chemin à coût fixe est décrit par les contraintes suivantes :

$$-x_1 + \sum_{j \in N_1^+} x_j - \sum_{j \in N_1^-} x_j \leq b_1$$

$$+x_{k-1} - x_k + \sum_{j \in N_k^+} x_j - \sum_{j \in N_k^-} x_j \leq b_k \quad pour \quad k = 2, .., K-1$$

$$+x_{K-1} \qquad + \sum_{j \in N_K^+} x_j - \sum_{j \in N_K^-} x_j \leq b_K$$

$$x_k \geq 0, \quad k = 1, .., K-1$$

$$x_j \leq u_j y_j, \quad j \in \bigcup_{k=1}^{K} (N_k^+ \cup N_k^-)$$

$$y_j \in \{0,1\}, \quad j \in \bigcup_{k=1}^{K} (N_k^+ \cup N_k^-)$$

Deux types de coupes ont été décrites par Van Roy et Wolsey [45], des inégalités simples comme présentées dans la définition 2, et des inégalités étendues définies par la suite (définition 3).

Définition 2 Les inégalités simples de réseau *("simple network inequalities")* suivantes :

$$\sum_{k=1}^{K} \sum_{j \in C_k^+} x_j \leq \sum_{k=1}^{K} \sum_{j \in C_k^+} (\sum_{t=k}^{K} b_t^+) y_j + \sum_{k=1}^{K} \sum_{j \in N_k^-} x_j$$

où

$$C_k^+ \subseteq N_k^+ \quad pour \quad k = 1...K$$

est une couverture de N_k^+ et $b_t^+ = \max(0, b_t)$ sont des inégalités valides pour les chemins de coûts fixes comme définis précédemment.

Définition 3 Les inégalités étendues :

$$\sum_{k=1}^{K} \sum_{j \in C_k^+} x_j \leq \sum_{k=1}^{K} \sum_{j \in C_k^+} \sum_{t=k}^{K} (b_t + \sum_{i \in Q_t^-} u_t) y_j + \sum_{k=1}^{K} \sum_{j \in N_k^- \setminus Q_k^-} x_j$$

où

$$C_k^+ \subseteq N_k^+ \quad et \quad Q_k^- \subseteq N_k^- \quad pour \quad k = 1...K$$

sont également des inégalités valides pour les chemins à coûts fixes.
Les inégalités ainsi définies sont adaptées principalement aux classes de problèmes de conception de réseaux et de détermination des lots *("lot-sizing").*

2.2.5 Méthode de branch-and-price-and-cut

Lorsque les trois algorithmes, génération de colonnes, génération de coupes et branch-and-bound, sont combinés, le tout donne lieu à la méthode de branch-and-price-and-cut,

dans laquelle la relaxation linéaire définie à chaque nœud est résolue en faisant alterner les deux méthodes de décomposition afin d'aboutir à de meilleures bornes inférieures dans un temps limité.

CHAPITRE 3

REVUE DE LITTÉRATURE SUR LES PROBLÈMES DE CONCEPTION DE RÉSEAUX

Les problèmes de conception de réseaux sont parmi les plus étudiés en recherche opérationnelle, étant donné leurs différentes applications en télécommunications, transport et autres domaines. Le but de ce chapitre est de présenter le problème classique de conception de réseaux, ses variantes et une revue de littérature sur les méthodes de résolution en s'inspirant des principales réferences dans le domaine (voir les articles de synthèse de Magnanti et Wong [35], Minoux [36], Gavish [25], Balakrishnan et al. [6], Crainic [17] et Gendron [26] qui contiennent des exemples).

3.1 Problème de conception de réseaux

Soit un réseau représenté par un graphe orienté $G = (V,A)$, où V est l'ensemble des nœuds et A est l'ensemble des arcs. On associe à ce réseau un ensemble de produits $k \in K$ qui peuvent circuler dans le réseau.

Les nœuds se subdivisent en trois catégories pour chaque produit k : les origines $O(k)$, les destinations $D(k)$ et les nœuds de transfert. On associe à chaque origine $i \in O(k)$ une offre o_i^k du produit k et à chaque destination $i \in D(k)$ une demande d_i^k du produit k. Les offres et les demandes vérifient l'égalité suivante :

$$\sum_{i \in O(k)} o_i^k = \sum_{i \in D(k)} d_i^k$$

Le problème de conception de réseaux consiste à choisir un sous-ensemble d'arcs sur lesquels sont installés des équipements, et le flot de produits à y faire circuler de façon à satisfaire la demande de ces produits tout en respectant les contraintes du problème.

L'objectif est de minimiser les coûts fixes d'installation des équipements et les coûts variables de transport des produits. Pour modéliser le problème général de conception de réseaux, soient les variables continues x_{ij}^k représentant la quantité du flot du produit k à transporter sur l'arc (i,j) et les variables entières y_{ij}^l représentant le nombre d'équipements du type $l \in L$ à installer sur l'arc (i,j) (variables de conception).
Nous introduisons également la notation suivante pour désigner les voisins d'un noeud $i : V_i^+ = \{j \in V/(i,j) \in A\}$ et $V_i^- = \{j \in V/(j,i) \in A\}$.
Nous reprenons la formulation proposée par Larose [34] qui est basée sur les modèles de Magnanti et al. [35] et Gendron et al. [28].

Le modèle général de conception de réseaux s'énonce alors ainsi :

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

Sujet à

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

$$\sum_{k \in K} x_{ij}^k \leq \sum_{l \in L} u_{ij}^l y_{ij}^l, \qquad \forall (i,j) \in A, \tag{3.3}$$

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

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

$$y_{ij}^l \geq 0 \ \ et \ \ entier \qquad \forall (i,j) \in A, \ l \in L. \tag{3.6}$$

L'objectif du problème est de minimiser une fonction de coût $\phi(y,x)$ en terme des variables x et y.
Les contraintes (3.2) sont les contraintes de conservation de flot tandis que les inégalités (3.3) représentent les contraintes de capacités. Les contraintes (3.4) incluent toutes les autres contraintes qui peuvent figurer dans le problème telles que la contrainte de budget qui spécifie que le coût total d'installation des équipements ne doit pas dépasser un budget limité. Les contraintes (3.5) précisent que les variables de flot x_{ij}^k sont non négatives

et les contraintes (3.6) indiquent que les variables y_{ij}^{l} sont des variables entières non négatives.
Si le problème est avec coûts fixes, la fonction objectif s'écrit en fonction des deux coûts : un coût unitaire de transport du produit k à travers l'arc (i,j), c_{ij}^{k} et un coût fixe pour l'ouverture de l'installation l sur l'arc (i,j), f_{ij}^{l}.
La fonction objectif s'écrit sous la forme :

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

3.2 Variantes du problème de conception de réseaux

À partir du problème général du conception de réseaux, plusieurs problèmes ont été dérivés pour tenir compte d'autres composantes du modèle, comme le nombre de produits ou la capacité sur les arcs. On distingue donc le cas avec un seul produit du cas multi-produits, et le cas avec des arcs de capacité limitée ou de capacité infinie.

3.2.1 Problème de conception de réseaux avec un seul produit

Contrairement au problème de conception de réseaux multi-produits, celui avec un seul produit a été relativement moins étudié dans la littérature. Pour formuler ce problème, il suffit d'éliminer l'indice k de la formulation précédente (voir section 3.1). Le problème de conception de réseaux avec coûts fixes, capacités et un seul produit (SCND, *single-commodity capacitated network design*) est modélisé sous la forme d'un programme linéaire en nombres entiers. Il consiste à choisir les arcs à construire sur un réseau pour y transporter un flot d'un produit tout en satisfaisant la demande aux nœuds destinations à partir des nœuds origines à un coût minimum.
On définit les variables de décision x_{ij}, les variables de flot représentant la quantité de flot sur chaque arc (i,j) et les variables binaires y_{ij}, les variables de conception telles que

$$y_{ij} = \begin{cases} 1, \text{ si l'arc } (i,j) \text{ est utilisé,} \\ 0, \text{ sinon.} \end{cases}$$

Soient les ensembles :

O : ensemble des nœuds origines, $O = \{i \in V | S_i > 0\}$;

D : ensemble des nœuds destinations, $D = \{i \in V | S_i < 0\}$;

T : ensemble des nœuds de transfert, $T = \{i \in V | S_i = 0\}$.

S_i est alors l'offre (ou la demande) du produit au nœud i. De plus, pour le problème avec capacité, on impose à chaque arc $(i, j) \in A$ une capacité $u_{ij} > 0$.

De plus, les données du problème doivent satisfaire la relation d'équilibre suivante :

$$\sum_{i \in V} S_i = 0.$$

Le modèle est formulé comme suit :

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

Sujet à

$$\sum_{j \in V_i^+} x_{ij} - \sum_{j \in V_i^-} x_{ji} = \begin{cases} S_i, & \text{si } i \in O, \\ -S_i, & \text{si } i \in D, \\ 0, & \text{si } i \in T. \end{cases} \qquad \forall i \in V, \tag{3.9}$$

$$x_{ij} \leq u_{ij} y_{ij}, \qquad \forall (i, j) \in A, \tag{3.10}$$

$$x_{ij} \geq 0, \qquad \forall (i, j) \in A, \tag{3.11}$$

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

Les contraintes (3.9) sont les contraintes de conservation de flot, tandis que les inégalités (3.10) représentent les contraintes de capacité, où u_{ij} est la capacité de l'arc (i,j).

3.3 Méthodes de résolution du problème de conception de réseaux

Les méthodes de résolution du problème de conception de réseaux se subdivisent en deux grandes classes, à savoir les méthodes exactes et les méthodes heuristiques.

Nous présentons dans ce qui suit la littérature relative au problème avec un seul produit,

ainsi qu'au problème à plusieurs produits.

3.3.1 Revue de littérature du problème de conception de réseaux avec un seul produit

Ortega et Wolsey [40] ont étudié en 2003 le problème de conception de réseaux avec un seul produit, sans capacité et avec coûts fixes. Pour le résoudre, il ont proposé un algorithme de branch-and-cut.
Dans un article plus récent, Miranda et al. [37] ont traité le problème de conception de réseaux avec un seul produit, coûts fixes et demande incertaine. Ils ont proposé pour la résolution une heuristique à trois phases. La première phase est une phase de construction dans laquelle une solution initiale est construite après élimination d'un ensemble d'arcs. La phase 2 consiste à améliorer la solution initiale du graphe réduit en se basant sur l'exploration du voisinage. Enfin, la phase 3 consiste à améliorer la solution en considérant tout le graphe.

Cacchiani et al. [10] ont étudié le même problème, où il est question de choisir les arcs du réseau de manière à avoir une capacité suffisante pour couvrir la demande quel que soit le sénario. Ils l'ont formulé en programme linéaire en nombres entiers, renforcé par des inégalités valides dérivées des coupes de Chvátal-Gomory $\{0, 1/2\}$. Ils ont alors proposé de le résoudre par un algorithme de branch-and-cut.

3.3.2 Revue de littérature du problème de conception de réseaux avec plusieurs produits

Contrairement au cas avec un seul produit, le problème de conception de réseaux avec plusieurs produits (MCND) a été abondamment étudié dans la littérature. Plusieurs méthodes de résolution efficaces ont été développées dans ce contexte.

3.3.3 Problème de conception de réseaux multi-produits sans capacité

Pour le problème multi-produits sans capacité, Balakrishnan et al. [7] ont développé une famille d'algorithmes de montée duale. Les résultats obtenus étaient remarquables : un écart d'optimalité de 1 à 4 % pour des problèmes de grandes tailles.

Holmberg et Hellstrand [32] ont traité le cas d'une seule origine et une seule destination par produit. Ils ont proposé une heuristique lagrangienne combinée avec la méthode de branch-and-bound. L'idée était de relaxer les contraintes de conservation de flot, puis d'appliquer une heuristique primale pour aboutir à une solution réalisable. La méthode s'est montrée efficace pour la résolution des problèmes de grandes tailles.

Cruz et al. [21] ont proposé un algorithme de branch-and-bound simplifié où le branchement et la résolution se basent sur la relaxation lagrangienne, ce qui a donné un temps d'exécution nettement meilleur.

3.3.4 Problème de conception de réseaux multi-produits avec capacités

Plusieurs méthodes, exactes et heuristiques, ont été proposées dans la littérature pour résoudre ce type de problèmes. Parmi les méthodes exactes, on peut citer l'approche basée sur la relaxation lagrangienne. Dans ce contexte, Holmberg et Yuan [33] ont proposé la méthode du sous-gradient pour résoudre le dual lagrangien, pour ensuite intégrer l'heuristique lagrangienne dans l'algorithme de branch-and-bound.

Toujours en se basant sur la relaxation lagrangienne, Gendron et Crainic [27], Gendron et al. [28] et Crainic et al. [17] ont cherché à améliorer la qualité des bornes inférieures. Ils ont conclu que la même borne que celle obtenue par la relaxation linéaire de la formulation forte est obtenue en utilisant différents types de relaxations lagrangiennes.

Une autre méthode exacte se basant sur l'introduction d'inégalités valides (coupes) a été aussi utilisée par d'autres chercheurs. Costa et al. [16] ont comparé trois types

d'inégalités valides, à savoir les inégalités de Benders, les inégalites métriques et les inégalités "cutset". Ils ont aussi tenté d'améliorer la performance de la décomposition de Benders.

En incorportant cinq types d'inégalités valides dans le problème à travers l'algorithme de génération de coupes qu'ils ont intégré dans l'algorithme de branch-and-bound, Chouman et al. [13] ont atteint l'optimalité pour la majorité des instances.

Gendron et Larose [29] ont développé une méthode de branch-and-price-and-cut pour le problème. Ils ont obtenu de bons résultats pour des instances de grande taille.

Chouman et al. [14] ont proposé un algorithme de branch-and-cut qui inclut des méthodes permettant d'éliminer des valeurs de certaines variables (méthodes de filtrage).

Étant donné la difficulté du problème, plusieurs chercheurs se sont dirigés vers l'utilisation des heuristiques pour la résolution du problème de conception de réseaux avec capacités et coûts fixes. La méthode de recherche tabou a été adaptée par Crainic et al. [19].

Crainic et Gendreau [18] ont proposé une recherche tabou parallèle coopérative où l'aspect coopératif du parallélisme se manifeste par l'échange des informations. Les résultats obtenus confirment la supériorité de la méthode parallèle par rapport à la méthode séquentielle. Dans le même esprit, Crainic et al. [20] ont developpé une recherche coopérative multi-niveaux.

D'autres auteurs ont cherché à combiner la programmation mathématique avec les heuristiques. Hewitt [30] a développé une méthode qui s'appuie sur la recherche à base de voisinage. Pour obtenir les bornes inférieures, il a proposé d'utiliser la relaxation linéaire de la formulation dans l'espace des chemins renforcée par les coupes obtenues dans la phase de recherche à base de voisinage.

Enfin, Chouman et Crainic [11] ont proposé une combinaison d'une méthode exacte avec la métaheuristique tabou. Les solutions réalisables obtenues étaient de bonne qualité par rapport à celles données par les heuristiques existantes.

CHAPITRE 4

MÉTHODE DE BRANCH-AND-PRICE-AND-CUT

Contrairement aux problèmes de conception de réseaux multi-produits avec capacités, les problèmes avec un seul produit ont reçu moins d'attention. Nous décrivons dans ce chapitre la méthode que nous avons développée, ainsi que les différentes étapes de l'algorithme que nous proposons pour résoudre le problème de conception de réseaux avec coûts fixes, capacités et un seul produit, soit la méthode de branch-and-price-and-cut (BPC). Elle est basée sur la combinaison de la méthode de génération de colonnes, la méthode de génération de coupes et l'algorithme de branch-and-bound. La méthode a été inspirée d'un algorithme proposé par Gendron et Larose [29] pour le problème de conception de réseaux multi-produits avec coûts fixes et capacités. Nous adaptons cette méthode, en lui ajoutant des conditions d'arrêt, grâce à une reformulation des problèmes à un seul produit, qui les transforme en problèmes équivalents à plusieurs produits. Le but de cette transformation est d'améliorer la borne inferieure provenant de la relaxation continue du modèle pour ainsi réduire la taille de l'arbre de recherche. La méthode de branch-and-price-and-cut est bien adaptée pour traiter la grande taille du modèle qui résulte de cette transformation.

4.1 Formulations du problème de conception de réseaux avec capacités, coûts fixes et un seul produit

4.1.1 Modèle classique

Rappelons que le problème SCND se caractérise par un seul produit avec plusieurs origines, plusieurs destinations et des coûts fixes définis sur un graphe orienté $G = (V, A)$, où V est l'ensemble des nœuds et A l'ensemble des arcs. Pour chaque arc $(i, j) \in A$, on associe une capacité $u_{ij} > 0$, un coût unitaire de transport $c_{ij} \geq 0$ et un coût fixe $f_{ij} \geq 0$. À chaque nœud $i \in V$ du graphe G est associée une demande de flot S_i, positive si i est

un nœud origine, négative si i est une destination, et nulle si i est un nœud de transfert (on a $\sum_{i \in V} S_i = 0$). L'objectif est de satisfaire la demande au moindre coût aux nœuds destinations à partir des nœuds origines. Tel que présenté à la section 3.2.1, le problème peut être modélisé comme suit :

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

Sujet à

$$\sum_{j \in V_i^+} x_{ij} - \sum_{j \in V_i^-} x_{ji} = \begin{cases} S_i, & \text{si } i \in O, \\ -S_i, & \text{si } i \in D, \\ 0, & \text{si } i \in T. \end{cases} \qquad \forall i \in V, \tag{4.2}$$

$$x_{ij} \leq u_{ij} y_{ij}, \qquad \forall (i,j) \in A, \tag{4.3}$$

$$x_{ij} \geq 0, \qquad \forall (i,j) \in A, \tag{4.4}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A. \tag{4.5}$$

Les contraintes (4.2) sont les contraintes de conservation de flot, tandis que les contraintes (4.3) représentent les contraintes de capacité.

4.1.2 Reformulation du problème

Le problème à un seul produit peut être reformulé en un problème de conception de réseaux multi-produits (MCND). L'objectif est de bénéficier de la performance de certaines méthodes développées pour le cas multi-produits, notamment l'approche proposée par Gendron et Larose [29]. L'idée est de créer une super-origine (SO) connectée à toutes les origines par des arcs ayant des coûts de transport nuls et des capacités égales aux offres des origines initiales. Chaque destination est alors considérée comme un produit dont l'origine correspond à la super-origine et la destination correspond à la destination elle-même (voir la Figure 4.1).

Soit K l'ensemble de produits qui circulent dans le réseau. Chaque destination k (produit) a une demande associée $d^k > 0$, un nœud origine $O(k)$ correspondant à SO et un

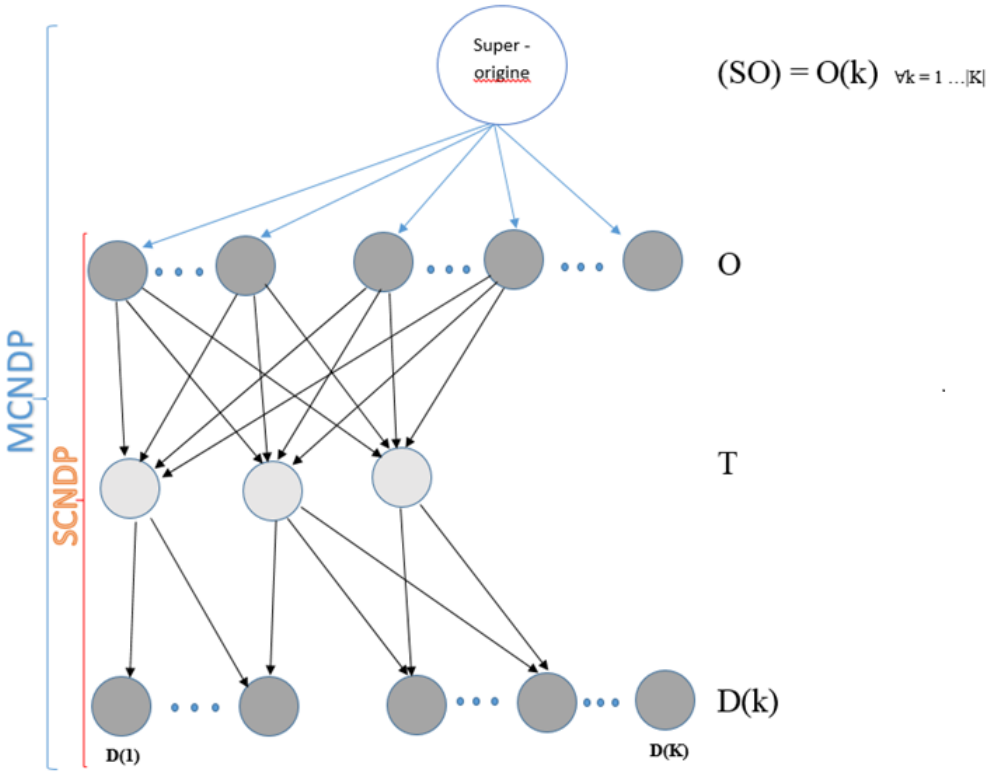

Figure 4.1 : Reformulation du SCND en MCND

nœud destination $D(k)$.

Soient les variables w_{ij}^k de reformulation du problème. On introduit les contraintes suivantes :

$$w_{ij}^k \in [0,1] \qquad \forall (i,j) \in A, \;\; k \in K, \tag{4.6}$$

$$\sum_{k \in K} d^k w_{ij}^k = x_{ij} \qquad \forall (i,j) \in A, \tag{4.7}$$

$$\sum_{j \in V_i^+} w_{ij}^k - \sum_{j \in V_i^-} w_{ji}^k = \begin{cases} 1, & \text{si } i = O(k), \\ -1, & \text{si } i = D(k), \\ 0, & \text{sinon.} \end{cases} \qquad \forall i \in V, \;\; k \in K, \tag{4.8}$$

$$w_{ij}^k \leq y_{ij}, \qquad \forall (i,j) \in A, \;\; k \in K. \tag{4.9}$$

En utilisant l'équation (4.7), nous remplaçons les variables x_{ij} par w_{ij}^k dans les contraintes (4.3). Nous obtenons ainsi la formulation donnée par :

$$\min \ \sum_{k\in K}\sum_{(i,j)\in A} c_{ij} w_{ij}^k + \sum_{(i,j)\in A} f_{ij} y_{ij} \tag{4.10}$$

sujet aux contraintes (4.5)-(4.6),(4.8)-(4.9) et

$$\sum_{k\in K} d^k w_{ij}^k \leq u_{ij} y_{ij}, \qquad \forall (i,j) \in A. \tag{4.11}$$

L'amélioration de la borne inférieure de la relaxation continue provient de l'ajout des contraintes (4.9). En effet, avant l'ajout de ces contraintes, la valeur de chaque variable y_{ij} dans la relaxation continue est donnée par $y_{ij} = \frac{x_{ij}}{u_{ij}}$. Après l'ajout de ces contraintes, la valeur de chaque variable y_{ij} dans la relaxation continue devient

$$y_{ij} = \max \left\{ \max_{k\in K} \{w_{ij}^k\}, \ \frac{\sum_{k\in K} d^k w_{ij}^k}{u_{ij}} \right\} \geq \frac{x_{ij}}{u_{ij}}.$$

4.2 Méthode de branch-and-price-and-cut pour la résolution du problème

Dans cette partie, nous décrivons en détail les différentes étapes de l'algorithme de branch-and-price-and-cut que nous proposons pour résoudre le problème de conception de réseaux avec coûts fixes, capacités et un seul produit, après l'avoir reformulé en un problème de conception de réseaux multi-produits avec coûts fixes et capacités.

4.2.1 Évaluation d'un nœud

À chaque nœud de l'arbre de recherche, nous résolvons la relaxation continue de la formulation du problème en appliquant la méthode de génération de colonnes sur les variables de flot, en alternance avec la génération de coupes, qui permet d'introduire les inégalités valides (4.9) violées pour augmenter la borne inférieure sur la valeur optimale du problème initial en nombres entiers. Afin d'accélérer la résolution du problème, nous faisons appel à la technique de fixation des variables en utilisant les coûts réduits des

variables de conception, ainsi qu'à une technique qui permet d'arrêter les méthodes de génération de colonnes et de génération de coupes avant d'obtenir la solution optimale, si la valeur de la borne ne s'améliore pas pour un certain nombre d'itérations. Nous présentons dans les sections suivantes ces différentes étapes.

4.2.1.1 Génération de colonnes

Le choix d'utiliser la méthode de génération de colonnes est justifié par le grand nombre de variables du problème. La méthode consiste à résoudre le problème relaxé (dit alors problème maître) en commençant par un ensemble restreint de variables de flot w_{ij}^k et en les générant de manière dynamique, constituant ainsi une suite de problèmes maîtres restreints dont la résolution aboutit à une solution de la relaxation continue du problème initial. À chaque itération, les variables à introduire dans le problème restreint sont déterminées suite à la résolution d'un ou plusieurs sous-problèmes.

Nous présentons dans ce qui suit chacune de ces composantes.

4.2.1.1.1 Problème maître Le problème maître correspond à la relaxation linéaire de la reformulation du problème SCND présentée à la section 4.1.2.

4.2.1.1.2 Problème maître restreint Pour initialiser la méthode de génération de colonnes, on ajoute des arcs artificiels entre l'origine $O(k)$ et la destination $D(k)$ de chaque produit k. Ces arcs n'ont aucune limite de capacité ($u_{O(k)D(k)}$ est infinie) et un coût fixe nul ($f_{O(k)D(k)} = 0$). Le coût de transport de flot par produit sur ces arcs $c_{O(k)D(k)}$ est pris égal à une très grande valeur, permettant ainsi d'éliminer toute solution contenant ces arcs dès les premières itérations de la méthode. De plus, à chaque itération, pour définir le problème maître restreint, on associe à chaque arc $(i, j) \in A_+$ un sous-ensemble de produits $\tilde{K_{ij}} \subseteq K$ correspondant aux variables de flots générées, où $A_+ = A \bigcup \{(O(k), D(k)), k \in K\}$ est l'ensemble de tous les arcs du réseau G ainsi que les arcs artificiels.

On définit également les sous-ensembles suivants :

$\tilde{A^k} = \{(i, j) \in A_+ | k \in \tilde{K_{ij}}\}, k \in K.$

$\tilde{V}_i^k(+) = \{j \in V | (i,j) \in \tilde{A^k}\}$ et $\tilde{V}_i^k(-) = \{j \in V | (j,i) \in \tilde{A^k}\}, i \in V, k \in K$, les ensembles respectifs des nœuds destinations et origines liant les arcs de $\tilde{A^k}$ pour tout $k \in K$.

De plus, on dénote par $\overline{K}_{ij} \subseteq \tilde{K_{ij}}$, le sous-ensemble d'inégalités valides de type (4.9) qui ont été générées tout au long de la méthode.
Le problème maître restreint (PMR) est présenté sous la forme suivante, en associant des variables duales aux contraintes de la formulation :

$$\min \ \sum_{k \in \tilde{K_{ij}}} \sum_{(i,j) \in A_+} c_{ij} w_{ij}^k + \sum_{(i,j) \in A_+} f_{ij} y_{ij} \tag{4.12}$$

Sujet à

$$\sum_{j \in \tilde{V}_i^k(+)} w_{ij}^k - \sum_{j \in \tilde{V}_i^k(-)} w_{ji}^k = \begin{cases} 1, & \text{si } i = O(k), \\ -1, & \text{si } i = D(k), \\ 0, & \text{autrement} \end{cases} \qquad \forall i \in V, k \in K, \qquad (\pi_i^k) \tag{4.13}$$

$$\sum_{k \in \tilde{K}_{ij}} d^k w_{ij}^k \leq u_{ij} y_{ij}, \forall (i,j) \in A_+, \qquad (\alpha_{ij}) \tag{4.14}$$

$$w_{ij}^k \leq y_{ij}, \forall (i,j) \in A_+, k \in \overline{K}_{ij} \subseteq \tilde{K_{ij}}, \qquad (\beta_{ij}^k) \tag{4.15}$$

$$y_{ij} \leq 1, \forall (i,j) \in A_+, \qquad (\gamma_{ij}) \tag{4.16}$$

$$w_{ij}^k \geq 0, \qquad \forall (i,j) \in A_+, k \in \tilde{K_{ij}}, \tag{4.17}$$

$$y_{ij} \geq 0, \qquad \forall (i,j) \in A_+. \tag{4.18}$$

À chaque itération de la génération de colonnes, les variables de flot avec coûts réduits négatifs sont ajoutées au problème maître restreint suite à la résolution d'un sous-problème. La solution actuelle (après l'ajout des nouvelles variables) n'est plus optimale, mais elle reste toujours réalisable pour le nouveau problème restreint. Ceci nous permet d'utiliser l'algorithme primal du simplexe pour résoudre le nouveau problème maître restreint.

4.2.1.1.3 Sous-problème Le sous-problème permet de déterminer les variables de flot ayant des coûts réduits négatifs $w_{ij}^k, k \notin \tilde{K_{ij}}, (i,j) \in A_+$, à partir du problème dual du problème maître restreint donné par :

$$\max \ \sum_{k \in K} (\pi_{O(k)}^k - \pi_{D(k)}^k) - \sum_{ij \in A} \gamma_{ij} \tag{4.19}$$

Sujet à

$$\pi_i^k - \pi_j^k - \alpha_{ij} - \beta_{ij}^k \leq c_{ij}, \qquad \forall (i,j) \in A, k \in K, \qquad (w_{ij}^k) \tag{4.20}$$

$$u_{ij}\alpha_{ij} + \sum_{k \in K} \beta_{ij}^k - \gamma_{ij} \leq f_{ij}, \qquad \forall (i,j) \in A, \qquad (y_{ij}) \tag{4.21}$$

$$\alpha_{ij} \geq 0, \qquad \forall (i,j) \in A, \tag{4.22}$$

$$\beta_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K, \tag{4.23}$$

$$\gamma_{ij} \geq 0, \qquad \forall (i,j) \in A, \tag{4.24}$$

À partir des contraintes (4.20), le coût réduit des variables w_{ij}^k est donné par :

$$c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} + \beta_{ij}^k, \qquad \forall (i,j) \in A, k \in K \tag{4.25}$$

Les variables w_{ij}^k qu'on ajoute au problème maître restreint pour améliorer la solution primale sont donc telles que :

$$c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} + \beta_{ij}^k < 0, \qquad \forall (i,j) \in A, k \in K \tag{4.26}$$

Puisque les variables β_{ij}^k du problème dual ne sont pas toutes connues, vu que certaines des contraintes correspondants (inégalités fortes (4.9)) ne sont pas encore introduites dans le problème maître restreint, nous utilisons les équations des écarts complémentaires (4.27)-(4.31) pour dériver des conditions permettant de contourner cette difficulté.

$$w_{ij}^k (c_{ij} - \pi_i^k + \pi_j^k + \alpha_{ij} + \beta_{ij}^k) = 0, \qquad \forall (i,j) \in A, k \in K, \tag{4.27}$$

$$y_{ij} (f_{ij} + \gamma_{ij} - u_{ij}\alpha_{ij} - \sum_{k \in K} \beta_{ij}^k) = 0, \qquad \forall (i,j) \in A, \tag{4.28}$$

$$\alpha_{ij}(u_{ij}y_{ij} - \sum_{k\in K} w_{ij}^k) = 0, \qquad \forall (i,j) \in A, \tag{4.29}$$

$$\beta_{ij}^k(y_{ij} - w_{ij}^k) = 0, \qquad \forall (i,j) \in A, k \in K, \tag{4.30}$$

$$\gamma_{ij}(1 - y_{ij}) = 0, \qquad \forall (i,j) \in A. \tag{4.31}$$

Soit $(\overline{w}, \overline{y})$ la solution optimale du PMR et $(\overline{\pi}, \overline{\alpha}, \overline{\beta}, \overline{\gamma})$ la solution optimale de son dual. Pour chaque PMR, la solution optimale satisfait $\overline{w}_{ij}^k = 0$ pour $(i,j) \in A$ et $k \notin \tilde{K}_{ij}$.

On distingue alors deux cas, selon que les valeurs $\overline{y}_{ij}$ sont positives ou nulles :

- **cas 1** : $\overline{y}_{ij} > 0$

  Dans ce cas, nous avons par l'équation (4.30),$\forall k \notin \tilde{K}_{ij}$:

  $$\overline{\beta}_{ij}^k(\underbrace{\overline{y}_{ij}}_{>0} - \underbrace{\overline{w}_{ij}^k}_{=0}) = 0 \Longrightarrow \overline{\beta}_{ij}^k = 0 \tag{4.32}$$

  D'où le coût réduit est :

  $$c_{ij} - \overline{\pi}_i^k + \overline{\pi}_j^k + \overline{\alpha}_{ij} \tag{4.33}$$

  Il s'agit donc d'ajouter au PMR les variables de flot w_{ij}^k, $k \notin \tilde{K}_{ij}$, telles que :

  $$c_{ij}^k - \overline{\pi}_i^k + \overline{\pi}_j^k + \overline{\alpha}_{ij} < 0, \qquad \forall (i,j) \in A, k \in K \tag{4.34}$$

- **cas 2** : $\overline{y}_{ij} = 0$

Dans ce cas, nous avons $\overline{w}_{ij}^k = 0, \forall k \in K$ et par l'équation (4.31) :

$$\overline{\gamma}_{ij}(1 - \underbrace{\overline{y}_{ij}}_{=0}) = 0 \Longrightarrow \overline{\gamma}_{ij} = 0 \tag{4.35}$$

En exploitant la contrainte (4.20) du dual, nous avons :

$$\overline{\beta}_{ij}^k \geq \overline{\pi}_i^k - \overline{\pi}_j^k - \overline{\alpha}_{ij} - c_{ij} \tag{4.36}$$

Par la contrainte (4.23) du dual ($\overline{\beta}_{ij}^k \geq 0$), nous obtenons l'inégalité suivante :

$$\overline{\beta}_{ij}^k \geq \max\{(0, \overline{\pi}_i^k - \overline{\pi}_j^k - \overline{\alpha}_{ij} - c_{ij})\} \qquad \forall k \in K \tag{4.37}$$

En exploitant la contrainte (4.21) du dual, nous avons aussi :

$$f_{ij} - u_{ij}\overline{\alpha}_{ij} \geq \sum_{k \in K} \overline{\beta}_{ij}^k \tag{4.38}$$

À partir des inégalités (4.37) et (4.38), nous obtenons la relation suivante :

$$f_{ij} - u_{ij}\overline{\alpha}_{ij} \geq \sum_{k \in K} \max\{(0, c_{ij}^k - \overline{\pi}_i^k + \overline{\pi}_j^k + \overline{\alpha}_{ij})\} \tag{4.39}$$

Nous introduisons alors dans le problème maître restreint les variables $w_{ij}^k, k \notin \tilde{K_{ij}}$, telles que cette inéquation est violée, c'est-à-dire :

$$f_{ij} - u_{ij}\overline{\alpha}_{ij} < \sum_{k \in K} \max\{(0, c_{ij}^k - \overline{\pi}_i^k + \overline{\pi}_j^k + \overline{\alpha}_{ij})\}, \tag{4.40}$$

mais seulement si leurs coûts réduits sont négatifs $c_{ij} - \overline{\pi}_i^k + \overline{\pi}_j^k + \overline{\alpha}_{ij} < 0$.

Le processus de génération de colonnes s'arrête quand aucune variable de flot n'a un coût réduit négatif, ce qui veut dire que les inégalités présentées dans les deux cas sont vérifiées pour tout $(i, j) \in A$ et $k \in K$.

À cette étape, la génération de coupes est appliquée, en vérifiant s'il existe des inégalités valides violées par la solution obtenue par la génération de colonnes. Dans ce qui suit, nous décrivons en détail la méthode de génération de coupes.

4.2.1.2 Génération de coupes

Les contraintes (4.9), constituant des inégalités valides pour le problème, permettent d'obtenir de meilleures bornes inférieures. Leur nombre est cependant très grand, ce qui nous amène à les générer de manière dynamique. Nous commençons alors par une formulation plus faible du problème, et à la fin de la génération de colonnes, les inégalités

(4.9) violées par la solution obtenue $(\overline{w}, \overline{y})$ sont introduites dans le modèle.
Le processus se fait de manière itérative, et ce jusqu'à ce qu'aucune inégalité ne soit violée. En effet, lorsque ces inégalités sont ajoutées au problème, il devient non réalisable, il est alors résolu par la méthode duale du simplexe, en partant de la dernière solution optimale connue.

Suite à l'ajout des inégalités valides, le processus de génération de colonnes est repris suite à l'application d'une technique de fixation de variables permettant d'accélérer l'algorithme. Nous la présentons dans la section suivante.

4.2.1.3 Fixation des variables

L'utilisation de la technique de fixation de variables de décision en utilisant leurs coûts réduits a pour but de réduire l'espace de recherche afin d'accélérer la résolution du problème [29]. Son principe consiste à vérifier si la variable de conception à introduire dans le problème améliore sa valeur optimale ou non [39].

Étant donné la borne inférieure Z^l donnée par la relaxation linéaire LP et la meilleure borne supérieure connue Z^*, on teste pour chaque arc (i, j) tel que $\overline{y}_{ij} \in \{0,1\}$, si $Z^l + |\overline{f}_{ij}| \geq Z^*$, avec $\overline{f}_{ij} = f_{ij} - u_{ij}\overline{\alpha}_{ij} - \sum_{k \in K} \overline{\beta}^k_{ij}$. On fixe dans ce cas y_{ij} à $\overline{y}_{ij}$.

- Si $\overline{y}_{ij} = 0$, alors par la relation (4.31), $\overline{\gamma}_{ij} = 0$. Si on ajoute la contrainte $y_{ij} \geq 1$ à laquelle correspond la variable duale $\lambda_{ij} \geq 0$, la contrainte duale (4.21) devient $\lambda_{ij} \leq f_{ij} - u_{ij}\alpha_{ij} - \sum_{k \in K} \beta^k_{ij}$. En posant $\lambda_{ij} = \overline{f}_{ij}$, on obtient une solution duale réalisable ; une borne inférieure est donc donnée par $Z^l + \overline{f}_{ij}$.
  Ainsi, si $Z^l + \overline{f}_{ij} = Z^l + |\overline{f}_{ij}| \geq Z^*$, on doit nécessairement avoir $y_{ij} = 0$.

- Si $\overline{y}_{ij} = 1$, alors $\gamma_{ij} = \max\{0, -\overline{f}_{ij}\}$ par (4.28) et (4.31). Si on ajoute la contrainte $y_{ij} \leq 0$ à laquelle correspond la variable duale $\mu_{ij} \geq 0$, alors la contrainte duale devient $-\mu_{ij} - \gamma_{ij} \leq f_{ij} - u_{ij}\alpha_{ij} - \sum_{k \in K} \beta^k_{ij}$. En posant $\mu_{ij} = -\overline{f}_{ij}$ et $\gamma_{ij} = 0$, on obtient une solution duale réalisable ; une borne inférieure est donc donnée par $Z^l - \overline{f}_{ij}$. Ainsi, si $Z^l - \overline{f}_{ij} = Z^l + |\overline{f}_{ij}| \geq Z^*$, on doit nécessairement avoir $y_{ij} = 1$.

4.2.1.4 Condition d'arrêt

Pour améliorer davantage la performance de l'algorithme de branch-and-price-and-cut en terme de temps de résolution de chaque nœud de l'arbre de recherche, nous introduisons une condition qui permet d'arrêter le processus d'évaluation du nœud lorsque l'écart entre la valeur réalisable obtenue après la génération de colonnes ou la génération de coupes et les valeurs antécédentes reste inférieure à une très petite valeur ε durant un certain nombre d'itérations, c'est-à-dire lorsque la valeur obtenue ne s'améliore pas pour quelques itérations.
Lorsqu'on arrête le processus d'évaluation d'un nœud, nous obtenons une solution non nécessairement optimale, mais qui représente une borne inférieure sur la valeur optimale du problème SCND.
Le nombre d'itérations et le taux d'approximation ε sont des paramètres choisis après plusieurs tests effectués sur des instances de grandes tailles (voir le chapitre 5).

4.2.1.5 Règle de branchement

La règle de branchement utilisée dans notre algorithme est celle du branchement fiable (voir section 2.2.1.3.4). Cette règle combine la minimisation du nombre total de nœuds générés et du temps nécessaire pour la sélection de la variable parmi les variables candidates pour le branchement. Elle permet de selectionner les meilleurs variables sur lesquelles brancher dans un temps limité et de manière à permettre l'élagage du plus grand nombre de nœuds.

4.2.1.6 Méthode de recherche

Au début de nos tests préliminaires, nous avons comparé dans notre expérimentation deux stratégies de recherche : la recherche meilleur d'abord et la recherche en profondeur. Cette dernière n'a pas donné des bons résultats, étant donné que toutes les instances testées dans notre étude sont de grande taille, et aucune d'entre elles n'a été résolue dans la limite de temps fixée. Nous utilisons donc seulement la méthode de recherche meilleur d'abord dans l'objectif d'obtenir de meilleures bornes inférieures dans les meilleurs dé-

lais, la solution optimale étant loin d'être trouvée.

4.2.2 Algorithme de branch-and-price-and-cut

L'algorithme de branch-and-price-and-cut est donné comme suit :

1. Initialiser une borne supérieure Z^* et une solution réalisable.
2. Initialiser $\mathscr{L}$, l'ensemble des nœuds à évaluer.
3. Soit Z^{lPrec} une borne inférieure initialisée à 0.
4. Soit $Cmp = 0$ un compteur du nombre d'itérations.
5. Soit *max* un nombre maximum d'itérations.
6. Soit ε l'écart de convergence.
7. *Sélection :* Sélectionner le nœud à évaluer dans $\mathscr{L}$ et le retirer de $\mathscr{L}$.
8. *Borne inférieure :* Résoudre la relaxation continue du nœud sélectionné :
   (a) Trouver une paire de solutions primale-duale $\overline{w}$, $\overline{y}$ et $\overline{\pi}$, $\overline{\alpha}$, $\overline{\beta}$, $\overline{\gamma}$ en résolvant le PMR avec la méthode duale du simplexe.
   Génération de colonnes :
   (b) *Sous-problème :* Pour chaque arc (i, j), si
   $\overline{y}_{ij} > 0$ ou $(\overline{y}_{ij} = 0$ et $f_{ij} - u_{ij}\overline{\alpha}_{ij} < \sum_{k \in K} \max\{0, -(c^k_{ij} - \overline{\pi}^k_i + \overline{\pi}^k_j + \overline{\alpha}_{ij})\})$,
   alors pour chaque $k \notin \tilde{K}_{ij}$ tel que $c^k_{ij} - \overline{\pi}^k_i + \overline{\pi}^k_j + \overline{\alpha}_{ij} < 0$, ajouter les variables de flot w^k_{ij} au PMR.
   (c) Si des variables de flot sont ajoutées au PMR, résoudre le nouveau PMR par le primal du simplexe pour obtenir une paire de solutions primale-duale $\overline{w}$, $\overline{y}$ et $\overline{\pi}$, $\overline{\alpha}$, $\overline{\beta}$, $\overline{\gamma}$; aller à l'étape 8b.
   (d) Soit Z^l la borne inférieure obtenue après la procédure de génération de colonnes,

i. Si $\bar{y}$ est entière et $Z^l < Z^*$, alors $Z^l = Z^*$ et sauvegarder $\bar{y}$ et $\overline{w}$ comme nouvelle solution optimale.

ii. Si $Z^l \geq Z^*$, aller à l'étape 10.

iii. Si $((Z^l - Z^{lPrec})/Z^l) < \varepsilon$, $Cmp = Cmp + 1$, sinon $Cmp = 0$.
$Z^{lprec} = Z^l$.

iv. Si (Cmp = max), aller à l'étape 10.

Génération de coupes :

(e) *Séparation :* Pour chaque arc (i, j) et pour chaque $k \in \tilde{K}_{ij}$ tel que $\overline{w}_{ij}^k > \bar{y}_{ij}$, ajouter les inégalites fortes correspondantes au PMR.

(f) Si des inégalités fortes sont ajoutées au PMR :

i. Résoudre le nouveau PMR par la méthode duale du simplexe pour obtenir $\overline{w}$, $\bar{y}$ et une borne inférieure Z^l.

ii. Si $\bar{y}$ est entière et $Z^l < Z^*$, alors $Z^l = Z^*$ et sauvegarder $\bar{y}$ et $\overline{w}$ comme nouvelle solution optimale.

iii. Si $Z^l \geq Z^*$, aller à l'étape 10.

iv. Si $((Z^l - Z^{lPrec})/Z^l) < \varepsilon$, $Cmp = Cmp + 1$, sinon $Cmp = 0$.
$Z^{lprec} = Z^l$.

v. Si $(Cmp = max)$, aller à l'étape 10.

vi. *Fixation des variables :* Pour chaque arc (i, j), si $Z^l + |\overline{f}_{ij}| \geq Z^*$, alors fixer y_{ij} à $\bar{y}_{ij}$; si des variables sont fixées, aller à l'étape 8a.

vii. Aller à l'étape 8b.

9. *Branchement :* Si $Z^l < Z^*$, appliquer la règle de branchement pour générer deux nœuds fils et les insérer dans $\mathscr{L}$.

10. Si $\mathscr{L} = \emptyset$, arrêter ; sinon aller à l'étape 7.

Pour commencer, l'étape 1 consiste à trouver une solution réalisable initiale et une borne supérieure en appliquant une heuristique de CPLEX au problème SCND.

À l'étape 2, l'ensemble $\mathscr{L}$ des nœuds à évaluer est initialisé au nœud racine. De l'étape 3 à 6, on initialise les paramètres Z^{lPrec} à 0, Cmp à 0, max, et ε. Dans l'étape 7, on évalue le prochain nœud et on le retire de l'ensemble $\mathscr{L}$. Dans l'étape 8, il s'agit d'appliquer à chaque nœud la procédure de génération de colonnes et de coupes. Une paire de solutions primale-duale est cherchée à l'étape 8a de l'algorithme. Ensuite, le sous-problème est résolu pour déterminer les variables de flot avec coûts réduits négatifs (étape 8b de l'algorithme). Une fois les variables de flot ajoutées au PMR, on continue la génération de colonnes, sinon une borne inférieure est obtenue. Les tests d'arrêt sont effectués à l'étape 8d pour comparer la nouvelle solution Z^l par rapport à la solution Z^* et par rapport à la solution Z^{lPrec}. L'étape 8e consiste à résoudre le problème de séparation pour générer les inégalités fortes. Dans l'étape 8f, il s'agit de résoudre le PMR avec les nouvelles coupes en utilisant la méthode duale du simplexe, suivie par des tests d'arrêt effectués sur la nouvelle borne inférieure. Ensuite, nous utilisons la technique de fixation des variables. À l'étape 9, la règle de branchement est appliquée selon la valeur de Z^l obtenue. Finalement, l'étape 10 est la condition d'arrêt de la méthode de BPCC.

CHAPITRE 5

RÉSULTATS

Dans ce chapitre, nous présentons les résultats expérimentaux de notre méthode sur des instances de grande taille. Les résultats sont ensuite comparés à ceux donnés par CPLEX, un des meilleurs logiciels disponibles de nos jours dans le domaine de la programmation mathématique.

5.1 Environnement

5.1.1 Instances

Notre expérimentation a été effectuée sur 72 instances variées avec un seul produit, générées par une méthode due à Gendron et Bisaillon, dont 36 instances sont avec capacité. Le générateur reçoit en entrée le nombre de sommets, le nombre d'arcs, les demandes et les offres du produit, et crée les arcs de manière aléatoire en liant les sommets. De plus, il crée les arcs artificiels, les coûts et les capacités des arcs. Les demandes sont générées aléatoirement selon une loi uniforme.
Ces instances ont été par la suite transformées pour s'adapter au cas multi-produits. Il en résulte des instances qui sont divisées en deux groupes, un groupe avec un seul produit et un groupe avec plusieurs produits. À chaque instance à un seul produit correspond une instance équivalente à plusieurs produits obtenue au moyen de la reformulation décrite à la section 4.1.2. Par exemple, à partir d'une instance avec 10 offres et 30 demandes pour un seul produit, on génère une instance à $|K| = 10 \times 30$ produits. La taille d'une instance est définie par le nombre de sommets, le nombre d'arcs et le nombre de produits dans le réseau pour les instances multi-produits tel que présenté dans le tableau 5.I. Ces instances sont de grande taille (G) ou de très grande taille (T).

Groupes d'instances	$\|V\|$	$\|K\|$	$\|A\|$	Nombre d'instances
G	100	10×30	400	3
		10×40	500	3
		10×50	600	3
	150	15×45	900	3
		15×60	1 225	3
		15×75	1 350	3
T	100	10×30	2 000	3
		10×40	3 000	3
		10×50	4 000	3
	150	15×45	4 500	3
		15×60	6 750	3
		15×75	9 000	3

Tableau 5.I : Instances utilisées

5.1.2 Environnement matériel et logiciel

Nous exécutons notre algorithme sur sungrid, une machine multiprocesseur 12 Cores Intel Xeon X5675 avec une cadence de 3.07GHZ et 94.5Go de RAM sous le système d'exploitation Linux.

La méthode proposée est programmée en langage C++ en utilisant la librairie logicielle *Solving Constraint Integer Programs* (SCIP) [1] développée en langage C dans le cadre de la thèse de doctorat de Achterberg [1] dans le but de faciliter l'implémentation de la méthode de branch-and-bound. SCIP est actuellement un des solveurs non commerciaux les plus efficaces pour la programmation en nombres entiers, notamment pour la méthode de branch-and-cut-and-price (BPC) et la méthode de branch-and-cut (BC). Les relaxations continues sont résolues par CPLEX 12.5.1.

5.2 Mesures de performance

Nous présentons d'abord les mesures de performance sur lesquelles nous nous sommes basés pour évaluer notre méthode et la comparer aux méthodes existantes.

Comme nous avons des instances de grande taille, nous avons appliqué la méthode de recherche meilleure d'abord (nommée bfs) pour obtenir une meilleure borne inférieure

[1] disponible à http ://scip.zip.de.

dans un meilleur délai.

Nous fixons la limite de temps à 3 heures pour toutes les instances non résolues, puisqu'une solution optimale n'a pas pu être déterminée dans les limites de temps fixées (initialement, 3 heures, 12 heures et 24 heures). Nous calculons par la suite l'écart relatif (GAP) entre les bornes supérieures et les bornes inferieures selon la formule suivante :

$$GAP = (\text{borne sup} - \text{borne inf}) / \text{ borne sup}$$

Pour les instances résolues, nous comparons le temps moyen de résolution en secondes, le nombre moyen de nœuds évalués par chaque algorithme de résolution et le nombre moyen de variables de flot w_{ij}^k générées par chaque méthode.
Une méthode est alors plus efficace qu'une autre si elle permet de résoudre les instances dans un temps plus court ou génère un gap plus petit par rapport à toutes les autres méthodes.

5.3 Tests à la racine

Avant d'effectuer nos expérimentations sur tout l'arbre de recherche, nous avons d'abord appliqué la génération de coupes et de colonnes à la racine de l'arbre de recherche. L'objectif est d'obtenir une idée préliminaire sur la complexité de notre méthode, ainsi que sur le temps de résolution.
Nous avons comparé, par la suite, toutes les méthodes à la racine en fixant le nombre de nœuds à évaluer à 1, la limite de temps étant fixée à 3 heures.

5.3.1 Tests effectués

Pour tester la performance de notre méthode, nous effectuons cinq types de comparaisons.

5.3.1.1 Méthode de branch-and-bound de CPLEX avec un seul produit

Nous comparons notre méthode avec la méthode de branch-and-bound de CPLEX (version 12.5.1) avec un seul produit (nommée CPLEX Single et identifiée par A dans les tableaux). Nous avons effectué deux types de tests : dans le premier, aucun paramètre de CPLEX n'est modifié, à l'exception du nombre de noeuds à explorer qui est fixé à 1 ($nodeLim = 0$) dans le cas des tests à la racine. CPLEX est alors libre d'utiliser toutes les coupes dont il dispose. Le second type de tests est similaire au précédent sauf que nous désactivons les coupes de chemin de flot (*"flow path cuts"*) et les coupes de couverture de flot (*"flow cover cuts"*) décrites dans le chapitre 2 et ce, dans le but d'assurer une comparaison plus juste avec notre méthode où ces inégalités ne sont pas considérées. En effet, selon Bixby et Rothberg [9], le fait de désactiver les coupes de chemin de flot induit une dégradation de la performance de CPLEX de 1.04. De manière similaire, le fait de désactiver celles de couverture de flot cause une dégradation de 1.22.
Nous notons ce dernier type de tests par « CPLEX Single-PCC » et le référons par AB dans les tableaux.

5.3.1.2 Méthode de branch-and-bound de CPLEX multi-produits

Nous comparons notre méthode avec la méthode de branch-and-bound de CPLEX appliquée au problème multi-produits (nommée CPLEX multi et identifiée par B dans les tableaux), après modifications de quelques paramètres et la désactivation des heuristiques dans l'objectif de trouver un bon résultat dans ce premier type de tests. Nous ajoutons itérativement tous les types de coupes pour résoudre la formulation. La résolution se termine lorsqu'aucune inégalité ne permet d'améliorer la borne inférieure. De manière similaire à la méthode précédente, on affecte la valeur 0 au paramètre *NodeLim*, afin d'obtenir le résultat de la méthode appliquée à la racine, ce qui correspond à la résolution de la relaxation continue du problème. Dans un second test, on désactive les deux coupes de chemin de flot et de couverture de flot de CPLEX avant de lancer l'exécution à la racine (CPLEX multi-PCC, notée BB dans les tableaux).

5.3.1.3 Méthode de branch-and-cut

Nous comparons notre méthode avec une version sans génération de colonnes où, à chaque nœud de l'arbre de branch-and-bound, la relaxation continue est résolue simplement par CPLEX, en rajoutant itérativement les inégalités valides au modèle. Cette méthode est nommée BC et est identifiée par C dans les tableaux.

5.3.1.4 Méthode de branch-and-price-and-cut sans condition d'arrêt

Nommée BPC et identifiée par D dans les tableaux, il s'agit de l'algorithme que nous avons décrit dans le chapitre précedent, mais sans condition d'arrêt, c'est-à-dire que nous permettons aux méthodes de génération de colonnes et de coupes de continuer tant qu'elles n'ont pas abouti à une solution optimale de la relaxation continue.

5.3.1.5 Méthode de branch-and-price-and-cut avec condition d'arrêt

Nous testons notre méthode avec la condition d'arrêt, en donnant les valeurs suivantes à l'écart de convergence ε (0.0001, 0.001, 0.01, 0.0005, 0.005, 0.05) et en choisissant le nombre maximum d'itérations (Max) dans l'ensemble discret $\{2, ..., 7\}$. Notre objectif est de trouver les bonnes valeurs des paramètres qui permettent d'améliorer le temps de résolution. Nous comparons les résultats de cette méthode avec toutes les méthodes précédentes, d'abord à la racine (section 5.4), puis sur tout l'arbre de recherche (section 5.5). Nous donnons le nom BPCC à cette méthode et l'identifions par E dans les tableaux.

Par souci de clarté, un résumé des notations des différentes méthodes testées est rapporté au tableau 5.II.

5.4 Analyse des résultats à la racine

Les résultats des tests à la racine sont rapportés dans les tableaux de 5.III à 5.VIII. La comparaison des différentes méthodes est présentée dans le tableau 5.III pour les grandes

Méthode	Nomination	Notation
Branch-and-bound de CPLEX avec un seul produit	CPLEX Single	A
Branch-and-bound de CPLEX avec les IV désactivée (cuts pathcut -1 et cuts flowcovers -1)	CPLEX Single-PCC	AB
Branch-and-bound de CPLEX multiproduits	CPLEX Multi	B
Branch-and-bound de CPLEX avec les IV désactivées (cuts pathcut -1 et cuts flowcovers -1)	CPLEX Multi-PCC	BB
Branch-and-cut	BC	C
Branch–and-price-and-cut sans condition d'arrêt	BPC	D
Branch–and-price-and-cut avec condition d'arrêt	BPCC	E

Tableau 5.II : Résumé des méthodes testées

instances (groupe G) et dans le tableau 5.VI pour les très grandes instances (groupe T). Les résultats de notre méthode BPCC avec les differentes valeurs des paramètres Max et ε (notée BPCC(Max-ε)) sont présentés dans le tableau 5.IV pour le groupe G et dans le tableau 5.VII pour le groupe T. La comparaison des différentes méthodes avec les bons paramètres pour chaque instance du problème est présentée dans le tableau 5.V pour le groupe G et le tableau 5.VIII pour le groupe T.

Méthode	**CPU(s)**	**GAP (%)**	
		Meilleure BI	**BI initiale**
A	0,40	1,86	9,27
AB	1,50	2,27	9,27
B	3,88	2,41	-
BB	3,65	2,46	-
C	4,45	3,78	-
D	3,95	3,78	-

Tableau 5.III : Comparaison des différentes méthodes à la racine - Groupe G

Nous comparons initialement toutes les méthodes décrites à la section 5.3.1 : "A", "AB", "B", "BB", "C", "D" et "E" pour les deux groupes d'instances G et T.
Nous remarquons, dans les tableaux 5.III et 5.V pour le groupe G et dans les tableaux 5.VI et 5.VIII pour le groupe T, que pour un seul produit, le meilleur temps de résolution moyen pour toutes les instances est de 0,4s et 2,91s, respectivement, obtenu par la méthode CPLEX Single, qui est beaucoup plus rapide sur les instances de grande taille où le nombre d'arcs est moins élevé (de 400 à 1125). Elle fournit également les meilleurs gaps moyens (1,86% et 2,25% pour les instances G et T, respectivement). Ces valeurs sont cependant obtenues avec la meilleure borne inférieure après la génération de coupes de CPLEX puisque la méthode CPLEX Single applique plusieurs types de coupes. Par contre, lorsque le gap est calculé en utilisant la borne inférieure initiale, la méthode fournit les pires gaps (9,27% et 8,24% pour les groupes G et T, respectivement). La méthode CPLEX Single-PCC, permet d'avoir des gaps comparables à ceux de CPLEX Single (2,27% et 2,85% pour les instances G et T, respectivement), mais dans des durées plus longues (1,5s et 11,16s).
Dans le cas des instances multi-produits, la méthode CPLEX Multi vient en première

position et donne des gaps moyens meilleurs (2,41% et 3,55% pour les instances G et T, respectivement) dans un temps moyen meilleur (3,88s et 24,37s pour les instances G et T, respectivement).

Comme prévu, lorsque les coupes de chemin de flot et de couverture de flot sont désactivées, on observe une augmentation légère des gaps (2,46% et 3,59% pour les instances G et T, respectivement) et des durées de résolution (3,65s et 29,8s pour les instances G et T, respectivement) par rapport à la méthode CPLEX Multi, où ces coupes sont générées. On observe que le même écart est obtenu par les trois méthodes C (BC), D (BPC) et E (BPCC) pour les instances de très grande taille, avec et sans génération des colonnes (3,78% pour les instances du groupe G et 3.71% pour les instances du groupe T), mais notre méthode E (BPCC) assure cependant le meilleur temps de résolution pour toutes les instances (3,77s et 51,06s en moyenne pour les groupes G et T, respectivement) par rapport aux deux autres méthodes (BC et BPC).

Tel que démontré dans les tableaux 5.IV et 5.VII, la méthode BPCC avec condition d'arrêt donne un gap moyen comparable aux autres méthodes (3,82 % et 3,71 % pour les instances G et T, respectivement) lorsque le nombre maximum d'itérations est fixé à 7 et l'écart de convergence ε à 0.0001. Nous observons aussi que plus le nombre d'itérations augmente et la valeur de ε diminue, plus le gap diminue. Si la valeur de ε est dans l'intervalle entre 0.05 et 0.005 avec un nombre d'itérations petit, le gap est plus élevé.

Notre méthode nécessite également un temps d'éxecution moyen meilleur (3,77s et 51,06s pour les groupes G et T, respectivement) puisque la condition d'arrêt permet d'arrêter la résolution à une étape précoce lorsque la solution cesse de s'améliorer, et nécessite de ce fait moins de temps de résolution.

Max	ε	**CPU (s)**	**GAP (%)**
2	0,05	2,02	5,41
	0,01	2,80	4,16
	0,005	3,12	4,04
	0,001	3,45	3,90
	0,0005	3,69	3,87
	0,0001	3,53	3,83
3	0,05	2,14	4,72
	0,01	3,09	4,02
	0,005	3,17	3,95
	0,001	3,71	3,85
	0,0005	3,75	3,84
	0,0001	4,03	3,82
4	0,05	2,62	4,31
	0,01	3,30	3,93
	0,005	3,51	3,88
	0,001	3,91	3,84
	0,0005	4,00	3,83
	0,0001	3,93	3,82
5	0,05	3,04	4,11
	0,01	3,39	3,89
	0,005	3,53	3,86
	0,001	3,78	3,84
	0,0005	3,98	3,82
	0,0001	4,06	3,82
6	0,05	3,26	3,99
	0,01	3,49	3,87
	0,005	3,63	3,85
	0,001	4,005	3,83
	0,0005	4,05	3,82
	0,0001	4,09	3,82
7	0,05	3,49	3,91
	0,01	3,69	3,86
	0,005	3,82	3,84
	0,001	3,94	3,82
	0,0005	4,05	3,82
	0,0001	3,77	3,82

Tableau 5.IV : BPCC à la racine - Groupe G

Instances	A			AB		B		BB		C		D		E (7-0.0001)	
	CPU	GAP		CPU	GAP	CPU	GAP	CPU	GAP	CPU	GAP	CPU	GAP	CPU	GAP
		Meilleure BI	BI initiale												
o10_d30_a400	0,33	1,83	10,37	0,55	2,31	0,76	2,80	0,70	2,96	0,48	3,65	0,46	3,65	0,44	3,65
o10_d30_a500	0,39	2,66	13,05	0,60	3,53	1,28	3,72	1,33	3,75	1,81	5,64	1,47	5,64	1,37	5,64
o10_d30_a600	0,41	1,67	8,10	0,78	1,86	1,66	2,43	1,26	2,41	0,86	4,97	0,70	4,97	0,71	4,97
o10_d40_a400	0,25	2,26	10,26	0,54	2,75	1,46	3,02	1,11	3,32	1,65	4,63	1,57	4,63	1,48	4,63
o10_d40_a500	0,30	1,45	7,86	0,66	1,86	1,40	1,99	1,37	1,89	0,91	2,87	0,76	2,87	0,72	2,87
o10_d40_a600	0,24	1,67	8,80	1,22	2,01	1,92	2,18	1,71	2,33	1,65	3,12	1,53	3,12	1,44	3,12
o10_d50_a400	0,16	1,76	9,53	0,43	1,84	1,46	1,52	1,39	1,61	2,56	3,13	2,80	3,13	2,86	3,13
o10_d50_a500	0,24	1,42	7,97	0,76	1,58	1,39	1,36	1,31	1,30	3,36	2,83	2,83	2,83	2,58	2,83
o10_d50_a600	0,16	1,25	6,58	0,88	1,42	1,34	1,75	1,30	1,79	1,03	2,31	0,98	2,31	0,91	2,31
Moy 100	0,28	1,78	9,17	0,71	2,13	1,41	2,31	1,28	2,37	1,59	3,68	1,46	3,68	1,39	3,68
o15_d45_a900	0,61	2,93	16,99	2,53	3,28	8,55	4,17	7,46	4,27	18,00	7,08	13,32	7,08	13,27	7,08
o15_d45_a1225	0,61	2,15	10,08	2,32	2,84	5,81	3,33	6,11	3,26	5,21	4,47	5,40	4,47	4,86	4,47
o15_d45_a1350	0,62	2,40	10,55	2,94	3,46	8,95	3,20	7,71	3,38	8,27	4,92	5,68	4,92	5,33	4,92
o15_d60_a900	0,40	2,00	8,61	1,69	2,40	5,40	2,49	4,02	2,55	3,02	3,41	3,25	3,41	3,06	3,41
o15_d60_a1225	0,52	1,63	7,86	2,26	2,01	4,69	2,00	4,31	2,01	5,40	2,83	4,97	2,83	4,60	2,83
o15_d60_a1350	0,58	1,58	8,05	2,55	1,93	7,85	2,17	6,73	2,23	8,25	3,10	8,15	3,10	7,46	3,10
o15_d75_a900	0,36	1,32	6,96	1,54	1,52	4,63	1,73	4,25	1,67	4,57	2,66	4,89	2,66	4,60	3,35
o15_d75_a1225	0,41	1,51	7,05	1,92	1,74	5,04	1,39	5,70	1,37	5,33	2,88	5,38	2,88	5,20	2,88
o15_d75_a1350	0,58	1,91	8,12	2,79	2,44	6,29	2,21	7,91	2,16	7,66	3,59	6,96	3,59	6,89	3,59
Moy 150	0,52	1,94	9,36	2,28	2,40	6,36	2,52	6,02	2,54	7,30	3,88	6,44	3,88	6,14	3,96
Moy totale (G)	0,40	1,86	9,27	1,50	2,27	3,88	2,41	3,65	2,46	4,45	3,78	3,95	3,78	3,77	3,82

Tableau 5.V : Comparaison des differentes méthodes à la racine avec les bons paramètres- Groupe G (CPU en s et GAP en %)

Méthode	CPU(s)	GAP (%)	
		Meilleure BI	BI initiale
A	2,91	2,25	8,24
AB	11,16	2,85	8,24
B	24,37	3,55	-
BB	29,80	3,59	-
C	78,27	3,71	-
D	54,86	3,71	-

Tableau 5.VI : Comparaison des différentes méthodes à la racine - Groupe T

Concernant le nombre de variables, la méthode CPLEX Single en génère moins dans un temps d'exécution concurrentiel avec une meilleure borne inférieure.
En conclusion, la méthode CPLEX Single à la racine avec toutes les coupes reste la plus perfomante pour les instances de grande taille et de très grande taille. Notre méthode est également prometteuse et elle permet de résoudre toutes les instances à la racine dans la limite de temps de 3 heures, ce qui nous motive à généraliser nos tests à tout l'arbre de recherche.

5.5 Comparaisons des méthodes sur tout l'arbre de recherche

Rappelons d'abord que nous utilisons la règle de recherche meilleur d'abord et que nous donnons à toutes les méthodes la même borne supérieure initiale utilisée dans les tests à la racine avant de les lancer, la limite de temps étant fixée à 3 heures. Nous comparons les différentes méthodes en tenant compte du temps de résolution et des gaps entre les bornes supérieures et inférieures pour chaque instance, ainsi que de l'amélioration de la borne inférieure. Comme les instances sont de grandes et de très grandes tailles, les résultats des tests sur tout l'arbre de recherche sont séparés selon deux groupes G et T, rapportés dans les tableaux 5.XI à 5.XIV.

Max	ε	CPU (s)	GAP (%)
2	0,05	8,75	11,73
	0,01	20,51	5,63
	0,005	27,32	4,73
	0,001	35,92	4,04
	0,0005	40,22	3,88
	0,0001	43,95	3,77
3	0,05	14,29	7,63
	0,01	25,55	4,70
	0,005	28,93	4,38
	0,001	40,15	3,86
	0,0005	43,51	3,78
	0,0001	52,36	3,73
4	0,05	16,56	6,45
	0,01	30,38	4,39
	0,005	33,08	4,12
	0,001	48,74	3,81
	0,0005	51,93	3,76
	0,0001	48,89	3,72
5	0,05	21,50	5,85
	0,01	33,48	4,19
	0,005	37,59	4,01
	0,001	49,70	3,77
	0,0005	51,68	3,75
	0,0001	52,88	3,71
6	0,05	22,02	5,14
	0,01	34,91	4,10
	0,005	40,55	3,91
	0,001	45,90	3,76
	0,0005	48,67	3,74
	0,0001	51,06	3,71
7	0,05	23,95	4,91
	0,01	37,83	3,99
	0,005	41,30	3,87
	0,001	45,94	3,75
	0,0005	48,08	3,74
	0,0001	52,96	3,71

Tableau 5.VII : BPCC à la racine - Groupe T

Instances	A			AB		B		BB		C		D		E (6-0.0001)	
	CPU	GAP		CPU	GAP	CPU	GAP	CPU	GAP	CPU	GAP	CPU	GAP	CPU	GAP
		Meilleure BI	BI initiale												
o10_d30_a2000	0,97	2,68	8,90	3,05	3,25	2,61	4,34	2,69	4,33	5,96	4,37	4,82	4,37	4,74	4,37
o10_d30_a3000	2,14	2,24	10,99	3,05	3,18	8,11	4,85	10,04	4,99	18,53	5,4	10,88	5,4	10,49	5,4
o10_d30_a4000	2,74	2,72	7,83	5,16	2,97	8,22	5.0	10,06	4,93	14,34	5,02	9,02	5,02	8,93	5,02
o10_d40_a2000	0,75	1,60	7,32	2,94	1,85	2,94	2,63	3,51	2,65	5,57	2,72	5,26	2,72	5,29	2,72
o10_d40_a3000	1,12	2,09	9,07	6,69	2,32	8,62	3,09	9,62	3,03	26,09	3,19	12,74	3,19	11,67	3,19
o10_d40_a4000	2,11	2,10	7,97	6,64	2,48	11,74	3,17	13,05	3,17	39,53	3,35	22,45	3,35	21,08	3,35
o10_d50_a2000	0,93	2,30	9,96	4,5	2,62	4,6	3,29	11,51	2,29	28,58	3,4	17,65	3,40	16,4	3,41
o10_d50_a3000	1,08	1,95	7,24	4,25	2,35	8,47	2,47	4,81	3,26	18,5	2,54	15,54	2,54	15,04	2,54
o10_d50_a4000	1,34	1,56	5,85	5,06	2,33	10,3	2,28	9,2	2,46	37,4	2,25	20,34	2,25	20,23	2,25
Moy 100	**1,46**	**2,14**	**8,35**	**4,59**	**2,59**	**7,29**	**3,46**	**8,28**	**3,46**	**21,61**	**3,58**	**13,19**	**3,58**	**12,65**	**3,58**
o15_d45_a4500	2,53	3,17	9,74	11,15	3,82	25,37	5,34	29,41	5,44	53,3	5,52	44,07	5,52	37,35	5,56
o15_d45_a6750	3,28	2,79	7,41	11,43	3,38	29,0	4,27	33,06	4,33	82,44	4,63	48,09	4,63	48,69	4,63
o15_d45_a9000	8,92	2,53	8,42	22,46	3,69	41,8	4,07	44,36	4,29	89,69	4,36	62,13	4,36	56,73	4,38
o15_d60_a4500	2,58	1,71	7,37	14,18	2,21	23,43	2,33	26,9	2,57	60,74	2,82	32,83	2,82	31,01	2,82
o15_d60_a6750	4,22	2,27	8,22	21,19	3,14	33,01	3,42	34,96	3,55	79,4	3,83	48,09	3,83	46,73	3,83
o15_d60_a9000	3,54	1,51	5,70	15,24	1,74	43,37	2,61	67,6	2,54	85,72	2,78	80,59	2,78	71,53	2,78
o15_d75_a4500	4,18	2,62	9,95	18,01	3,65	51,67	3,85	79,40	3,78	214,02	3,88	156,66	3,88	155,31	3,88
o15_d75_a6750	4,85	2,49	8,12	17,43	3,71	56,68	3,79	56,86	3,82	348,91	3,56	198,99	3,56	178,68	3,56
o15_d75_a9000	5,15	2,20	8,31	28,37	2,94	68,67	3,12	89,31	3,14	200,07	3,15	197,29	3,15	179,20	3,17
Moy 150	**4,36**	**2,36**	**8,14**	**17,72**	**3,14**	**41,44**	**3,64**	**51,32**	**3,72**	**134,92**	**3,84**	**96,53**	**3,84**	**89,47**	**3,85**
Moy totale (T)	**2,91**	**2,25**	**8,24**	**11,16**	**2,85**	**24,37**	**3,55**	**29,8**	**3,59**	**78,27**	**3,71**	**54,86**	**3,71**	**51,06**	**3,71**

Tableau 5.VIII : Comparaison des differentes méthodes à la racine avec les bons paramètres- Groupe T (CPU en s et GAP en %)

La comparaison des differentes méthodes pour chaque instance du problème est présentée dans le tableau 5.XI pour le groupe G et le tableau 5.XIV pour le groupe T. La comparaison des différentes méthodes est présentée dans le tableau 5.IX pour le groupe G et le tableau 5.XII pour le groupe T. Les résultats de notre méthode BPCC avec différentes valeurs des paramètres d'arrêt sont présentés dans le tableau 5.X pour le groupe G et le tableau 5.XIII pour le groupe T.
Pour G et T, nous comparons à la fois les méthodes CPLEX Single, CPLEX Multi, BC, BPC, et BPCC.

Méthode	Nœuds	Variables	GAP%
CPLEX Single	17 622 675	1 692	0,93
CPLEX Multi	1 577 823	47 996	1,50
BC	99 577	48 045	1,49
BPC	119 210	11 696	1,46

Tableau 5.IX : Comparaison des différentes méthodes sur tout l'arbre de recherche - Groupe G

Méthode	Nœuds	Variables	GAP%
CPLEX Single	7 557 440	9 571	1,86
CPLEX Multi	245 113	262 214	2,34
BC	8 272	262 263	2,20
BPC	9 846	38 207	2,18

Tableau 5.X : Comparaison des différentes méthodes sur tout l'arbre de recherche - Groupe T

Nous remarquons que les instances de très grande taille (T) sont toutes non résolues car les méthodes atteignent la limite de temps avant de trouver une solution optimale. Par contre, pour les instances du groupe G (100 nœuds) 5 instances sur l'ensemble des 18 instances ont été résolues par la méthode CPLEX Single dans un temps de résolution moyen de 4 325s (voir tableau 5.XI). 13 instances n'ont cependant pas pu être résolues. La méthode fournit quand même les meilleurs gaps moyens pour les groupes d'instances G et T (0,93% et 1,86%, respectivement). Nous pouvons aussi remarquer dans les tableaux 5.X et 5.XIV que le nombre de nœuds évalués est très grand par rapport aux autres méthodes (17 622 675 et 7 557 440 pour les instances de G et T, respectivement).

Instances	A				B			C			D		
	Nœuds	Var	CPU	GAP	Nœuds	Var	GAP	Nœuds	Var	GAP	Nœuds	VAR	GAP
o10_d30_a400	37 273 422	813	10 800	0,24	4 058 587	13 734	1,70	307 399	13 763	0,93	376 973	7 190	0,92
o10_d30_a500	26 235 059	1 052	10 800	1,81	2 421 378	16 741	2,11	119 749	16 770	1,43	116 447	8 117	1,47
o10_d30_a600	2 412 586	1 260	850	0,01	2 493 900	19 841	1,11	235 343	19 870	0,80	288 176	7 636	0,74
o10_d40_a400	31 637 751	876	10 800	0,30	2 477 242	18 533	1,69	162 837	18 572	1,28	165 626	9 375	1,26
o10_d40_a500	26 367 588	1 080	8 828	0,01	2 345 076	22 592	1,02	122 676	26 690	0,91	231 323	9 015	0,81
o10_d40_a600	28 313 094	1 278	10 800	0,47	1 796 407	26 651	1,29	217 087	22 631	0,85	162 955	9 392	0,83
o10_d50_a400	15 493 291	895	4 422	0,01	2 225 057	23 461	0,76	112 128	23 510	0,96	110 007	10 700	0,95
o10_d50_a500	10 854 413	1 096	3 541	0,01	2 363 333	28 561	0,41	124 441	28 610	0,50	184 302	11 299	0,45
o10_d50_a600	11 048 435	1 298	3 984	0,01	2 511 127	33 661	0,94	178 719	33 710	0,71	240 603	9 624	0,68
Moy ins résolues	**13 235 263**	**1 126**	**4 325**	**0,01**									
Moy 100	**21 070 627**	**1 072**	**7 203**	**0,32**	**2 521 345**	**22 642**	**1,23**	**175 598**	**22 681**	**0,93**	**208 490**	**9 150**	**0,90**
o15_d45_a900	15 802 018	1 888	10 800	2,35	665 591	44 161	2,86	7 764	44 205	3,72	10 746	14 766	3,61
o15_d45_a1225	14 505 741	2 340	10 800	1,77	668 760	54 511	1,99	32 098	54 555	2,32	35 303	12 198	2,23
o15_d45_a1350	13 102 284	2 790	10 800	2,13	491 238	64 861	2,43	16 183	64 905	2,99	22 121	12 787	3,01
o15_d60_a900	15 601 021	1 918	10 800	1,65	805 085	59 476	1,94	32 348	59 535	1,90	38 282	13 180	1,88
o15_d60_a1225	15 129 094	2 370	10 800	1,32	618 357	73 201	1,41	31 444	73 260	1,46	34 885	11 154	1,42
o15_d60_a1350	13 066 552	2 820	10 800	1,44	483 041	86 926	1,55	17 985	86 985	1,69	18 694	16 015	1,69
o15_d75_a900	16 528 272	1 946	10 800	1,13	719 049	75 241	1,28	32 616	75 315	1,17	43 858	14 091	1,16
o15_d75_a1225	14 844 067	2 400	10 800	1,31	721 785	92 341	0,95	23 468	92 415	1,43	42 011	16 696	1,37
o15_d75_a1350	12 942 137	2 850	10 800	1,66	535 809	109 441	1,53	18 100	109 515	1,86	23 466	17 286	1,86
Moy 150	**14 613 465**	**2 369**	**10 800**	**1,64**	**634 302**	**73 351**	**1,77**	**23 556**	**73 410**	**2,06**	**29 930**	**14 241**	**2,03**
Moy totale (G)	**17 622 675**	**1 692**	**8 779**	**0,93**	**1 577 823**	**47 996**	**1,50**	**99 577**	**48 045**	**1,49**	**119 210**	**11 696**	**1,46**

Tableau 5.XI : Comparaison des differentes méthodes sur tout l'arbre de recherche - Groupe G (CPU en s et GAP en %)

Max	ε	**Nœuds**	**Variables**	GAP (%)
	0,05	114 564	11 567	1,49
	0,01	116 729	11 452	1,48
2	0,005	116 469	11 480	1,47
	0,001	117 539	11 507	1,48
	0,0005	119 755	11 509	1,47
	0,0001	118 861	11 483	1,47
	0,05	113 793	11 488	1,48
	0,01	114 063	11 564	1,47
3	0,005	108 754	11 480	1,48
	0,001	112 019	11 489	1,48
	0,0005	112 658	11 428	1,48
	0,0001	111 337	11 483	1,47
	0,05	101 515	11 408	1,49
	0,01	102 873	11 331	1,49
4	0,005	101 675	11 375	1,49
	0,001	102 671	11 380	1,48
	0,0005	108 198	11 511	1,48
	0,0001	103 189	11 444	1,49
	0,05	101 702	11 423	1,48
	0,01	101 837	11 395	1,48
5	0,005	99 664	11 413	1,49
	0,001	97 632	11 339	1,50
	0,0005	109 184	11 442	1,48
	0,0001	96 605	11 347	1,50
	0,05	102 417	11 455	1,48
	0,01	98 460	11 340	1,49
6	0,005	98 877	11 109	1,49
	0,001	114 114	11 537	1,47
	0,0005	102 250	11 452	1,45
	0,0001	104 035	11 500	1,48
	0,05	100 385	11 394	1,49
	0,01	99 406	11 370	1,48
7	0,005	102 203	11 425	1,49
	0,001	100 870	11 433	1,49
	0,0005	103 385	11 448	1,48
	0,0001	104 606	11 439	1,48

Tableau 5.XII : BPCC sur tout l'arbre de recherche - Groupe G

Max	ε	**Nœuds**	**Variables**	GAP (%)
	0,05	12 072	38 244	2,45
	0,01	12 146	38 038	2,27
2	0,005	11 806	37 854	2,25
	0,001	12 100	37 977	2,21
	0,0005	12 128	38 011	2,20
	0,0001	12 252	37 915	2,19
	0,05	10 765	38 558	2,25
	0,01	11 352	38 073	2,22
3	0,005	10 823	38 025	2,21
	0,001	11 309	38 060	2,19
	0,0005	11 326	38 108	2,18
	0,0001	11 399	38 661	2,18
	0,05	10 650	38 338	2,20
	0,01	10 124	38 334	2,18
4	0,005	10 781	38 284	2,18
	0,001	10 338	38 048	2,17
	0,0005	10 862	38 132	2,17
	0,0001	10 654	38 137	2,17
	0,05	10 453	38 090	2,19
	0,01	10 390	38 055	2,18
5	0,005	10 145	38 090	2,18
	0,001	10 379	38 116	2,17
	0,0005	10 737	38 154	2,16
	0,0001	10 731	38 141	2,17
	0,05	10 016	38 321	2,18
	0,01	10 159	38 296	2,18
6	0,005	9 994	38 066	2,18
	0,001	10 167	38 154	2,16
	0,0005	10 141	38 173	2,17
	0,0001	10 078	38 147	2,17
	0,05	10 152	38 319	2,17
	0,01	10 107	38 143	2,18
7	0,005	9 855	38 617	2,18
	0,001	10 138	37 986	2,18
	0,0005	9 920	38 152	2,17
	0,0001	10 013	38 037	2,17

Tableau 5.XIII : BPCC sur tout l'arbre de recherche - Groupe T

Instances	A			B			C			D			E (5-0,0005)		
	Nœuds	Var	GAP	Nœuds	Var	GAP	Nœuds	Var	GAP	Nœuds	VAR	GAP	Nœuds	Var	GAP
o10_d30_a2000	10 975 718	4 000	1,91	623 437	62 311	2,51	38383	62340	1,91	38 256	12 816	1,90	42 017	12 706	1,87
o10_d30_a3000	8 142 219	6 000	1,79	400 775	93 311	2,53	13021	93340	2,41	15 471	17 857	2,37	16 884	17 877	2,32
o10_d30_a4000	5 698 895	7 964	2,30	353 074	123 753	2,74	12 107	123 782	2,25	16 009	22 608	2,20	17 743	22 590	2,15
o10_d40_a2000	10 832 750	4 000	1,27	582 561	82 411	1,54	23 485	82 450	1,25	27 585	13 074	1,27	29 301	13 155	1,25
o10_d40_a3000	8 175 398	5 988	1,77	394 670	123 165	2,11	8 618	123 204	1,81	9 647	20 661	1,84	13 014	20 806	1,77
o10_d40_a4000	6 390 642	7 856	1,45	288 316	161 459	2,15	6 495	161 498	1,97	9 929	30 557	1,89	11 487	29 971	1,91
o10_d50_a2000	11 513 173	3 996	1,82	426 661	102 409	1,80	12 369	102 458	1,50	15 882	17 631	1,47	16 488	17 697	1,47
o10_d50_a3000	8 344 199	5 928	1,42	331 900	151 675	1,80	9 160	151 724	1,45	11 123	24 529	1,44	10 961	24 341	1,44
o10_d50_a4000	7 395 093	7 240	0,97	302 912	185 131	1,35	8 062	185 180	1,24	10 998	30 536	1,20	12 269	30 135	1,20
Moy 100	**8 607 565**	**5 886**	**1,63**	**411 590**	**120 625**	**2,06**	**14 633**	**120 664**	**1,76**	**17 211**	**21 141**	**1,73**	**18 907**	**21 031**	**1,71**
o15_d45_a4500	4 923 198	9 000	2,83	142 809	207 691	4,35	3 385	207 735	3,48	4 614	25 294	3,44	5 604	25 542	3,42
o15_d45_a6750	2 404 055	13 492	2,47	114 062	311 007	3,00	2 521	311 051	3,29	3 814	39 454	3,23	4 137	39 257	3,20
o15_d45_a9000	2 189 286	17 944	2,21	63 420	413 403	2,69	2 172	413 447	3,11	2 733	58 243	3,11	2 967	58 371	3,05
o15_d60_a4500	5 135 556	9 000	1,42	130 304	275 416	1,75	3 479	275 475	1,97	5 057	34 058	2,00	5 982	33 472	1,98
o15_d60_a6750	2 861 139	13 496	2,06	51 322	412 544	2,26	2 440	412 603	2,43	2 584	49 566	2,44	2 851	49 307	2,44
o15_d60_a9000	2 386 843	17 652	1,27	41 421	539 302	1,91	1 777	539 361	1,79	1 896	75 157	1,76	1 887	74 866	1,80
o15_d75_a4500	33 188 584	9 000	2,36	57 720	343 141	2,88	440	343 215	2,94	547	56 892	2,89	716	56 987	2,91
o15_d75_a6750	2 993 196	13 356	2,18	65 023	508 669	2,40	519	508 743	2,43	570	68 593	2,42	687	68 940	2,44
o15_d75_a9000	2 483 968	16 366	1,97	41 638	623 049	2,41	455	623 123	2,34	515	90 193	2,34	520	90 363	2,35
Moy 150	**6 507 314**	**13 256**	**2,08**	**78 635**	**403 802**	**2,63**	**1 910**	**403 861**	**2,64**	**2 481**	**55 272**	**2,62**	**2 817**	**55 234**	**2,62**
Moy totale (T)	**7 557 440**	**9 571**	**1,86**	**245 113**	**262 214**	**2,34**	**8 272**	**262 263**	**2,20**	**9 846**	**38 207**	**2,18**	**10 862**	**38 132**	**2,16**

Tableau 5.XIV : Comparaison des differentes méthodes sur tout l'arbre de recherche avec les bons paramètres- Groupe T (CPU en s et GAP en %)

Pour les quatre autres méthodes, toutes les instances pour les deux groupes G et T sont non résolues (tableaux 5.XI, 5.IX, 5.XII et 5.XIV), donnant des gaps moyens de 1,50% et 2,34% pour G et T, respectivement, pour la méthode CPLEX Multi, 1,49% et 2,20% pour G et T, respectivement, pour la méthode BC, 1,46% et 2,18% pour G et T, respectivement, pour la méthode BPC et 1,45% et 2,16% pour G et T, respectivement, pour la méthode BPCC avec de bonnes valeurs des paramètres de la condition d'arrêt (Max = 6, $\varepsilon = 0.0005$ et Max = 5, $\varepsilon = 0.0005$), tel que montré aux tableaux 5.XII et 5.XIII.

Nous avons aussi remarqué que les paramètres d'arrêt qui génèrent de meilleurs résultats alternent généralement entre Max = 4 ou 5, avec ε de $0,0005$ à $0,0001$ pour les instances T et Max = 6, avec $\varepsilon = 0,0005$ pour les instances G.

Nous pouvons affirmer que notre méthode BPCC démeure compétitive, notamment pour les instances de très grande taille (T). Elle permet de réduire le temps d'exécution et elle donne aussi un gap moyen comparable à celui de CPLEX Single pour le groupe d'instances de très grande taille après 3h. Ceci est d'autant plus vrai lorsque la limite de temps est fixée à 12 et à 24 heures (tableaux 5.XV et 5.XVI), où on obtient une amélioration de la borne inférieure. Par contre, on observe que pour la méthode CPLEX Single aucune amélioration significative n'est obtenue après 3h (figure 5.1). Elle reste la meilleure pour la majorité des instances. Toutefois, pour quelques instances de très grande taille, notre méthode a donné un gap meilleur que celui de CPLEX Single pour une durée d'exécution de 3h et plus.

Nous concluons que les coupes générées par CPLEX sont plus efficaces que les inégalités valides que nous avons ajoutées à chaque itération de notre méthode. De plus, le temps requis pour la résolution de la reformulation multi-produits est plus élevé. Finalement, nous soulignons que la reformulation du problème augmente considérablement le nombre de variables, mais l'importance du choix de notre méthode BPCC réside dans le fait que, quel que soit la condition d'arrêt appliquée, elle permet de générer à chaque itération un nombre de variables réduit par rapport aux méthodes CPLEX Multi et BC pour les instances multi-produits, comme le montre les tableaux 5.IX-5.XIII.

Instances	Cplex Single	BPCC (4-0,0005)	BPCC (5-0,0005)
n100_o10_d30_a2000	1,91	1,87	1,89
n100_o10_d30_a3000	1,79	2,32	2,38
n100_o10_d30_a4000	2,30	2,15	2,21
n100_o10_d40_a2000	1,27	1,25	1,26
n100_o10_d40_a3000	1,77	1,77	1,78
n100_o10_d40_a4000	1,45	1,91	1,93
n100_o10_d50_a2000	1,82	1,47	1,44
n100_o10_d50_a3000	1,42	1,44	1,44
n100_o10_d50_a4000	0,97	1,20	1,21
Moyenne 100	**1,63**	**1,71**	**1,72**

Tableau 5.XV : Comparaison de la méthode BPCC à la méthode CPLEX Single sur tout l'arbre de recherche - Groupe T (GAP (%) après 3 heures).

Instances	CPLEX Single			BPCC		
				5-0,0005	5-0,0001	
	3h	12h	24h	3h	12h	24h
n100_o10_d30_a2000	1,91	1,39	1,12	1,89	1,59	1,47
n100_o10_d30_a3000	1,79	1,79	1,79	2,38	2,06	1,93
n100_o10_d30_a4000	2,30	2,07	1,68	2,21	1,90	1,78
n100_o10_d40_a2000	1,27	0,92	0,71	1,26	1,08	0,99
n100_o10_d40_a3000	1,77	1,36	1,08	1,78	1,61	1,53
n100_o10_d40_a4000	1,45	1,22	0,98	1,93	1,71	1,61
n100_o10_d50_a2000	1,82	1,28	0,77	1,44	1,27	1,19
n100_o10_d50_a3000	1,42	1,15	0,99	1,44	1,30	1,24
n100_o10_d50_a4000	0,97	0,84	0,72	1,21	1,08	1,02
Moy 100	**1,63**	**1,34**	**1,09**	**1,72**	**1,51**	**1,42**
n150_o15_d45_a4500	2,83	2,83	2,83	3,40	3,09	2,97
n150_o15_d45_a6750	2,47	2,47	2,47	3,20	2,97	2,87
n150_o15_d45_a9000	2,21	2,21	2,21	3,04	2,82	2,70
n150_o15_d60_a4500	1,42	1,42	1,42	1,95	1,87	1,81
n150_o15_d60_a6750	2,06	2,06	2,06	2,45	2,25	2,19
n150_o15_d60_a9000	1,27	1,27	1,27	1,77	1,66	1,60
n150_o15_d75_a4500	2,36	2,36	2,36	2,87	2,62	2,52
n150_o15_d75_a6750	2,18	2,18	2,18	2,39	2,19	2,12
n150_o15_d75_a9000	1,97	1,97	1,97	2,36	2,18	2,12
Moy 150	**2,08**	**2,08**	**2,08**	**2,60**	**2,40**	**2,32**
Moy totale	**1,86**	**1,71**	**1,59**	**2,16**	**1,96**	**1,85**

Tableau 5.XVI : Amélioration de BPCC par rapport à CPLEX Single sur tout l'arbre de recherche - Gaps (%) obtenus après 3, 12 et 24 heures

Figure 5.1 : Amélioration du gap (%) dans le temps entre BPCC et CPLEX Single

5.5.1 Impact de la condition d'arrêt sur la méthode de branch-and-price-and-cut

Nous avons aussi étudié l'impact de l'implémentation de la méthode de branch-and-price-and-cut avec les conditions d'arrêt (max et valeur d'approximation ε) sur la qualité de la borne inférieure et sur le temps global d'exécution. D'après les résultats, nous observons que notre méthode BPCC performe mieux que BPC sur les instances de très grande taille (de 300 à 1125 produits et de 2000 à 9000 arcs). On montre également que le nombre de nœuds évalués est plus élevé pour BPCC par rapport à BPC (tableau 5.XIV), mais peu importe la condition d'arrêt appliquée, les deux méthodes (BPC et BPCC) génèrent presque le même nombre de variables en raison de leurs résultats très proches (tableaux 5.IX, 5.XII pour le groupe G et tableaux 5.X, 5.XIII pour le groupe T).

La méthode BPCC (avec condition d'arrêt) est alors plus performante, puisque la condition d'arrêt donne un meilleur temps d'exécution, et permet en plus d'évaluer un nombre plus important de nœuds.

CHAPITRE 6

CONCLUSION

Les problèmes de conception de réseaux font partie des problèmes les plus importants en recherche opérationnelle. Leurs applications sont diverses et sont essentiellement dans les domaines du transport, de la logistique et des télécommunications.

Dans ce mémoire, nous avons proposé une méthode exacte, utilisant des techniques de la programmation en nombres entiers pour résoudre une variante de ce problème. Nous avons appliqué la méthode de branch-and-price-and-cut pour la résolution du problème de conception de réseaux avec coûts fixes, capacités et un seul produit avec condition d'arrêt. Cette méthode repose sur l'algorithme de branch-and-bound, la méthode de génération de colonnes et la méthode de génération de coupes. Nous avons, dans un premier temps, présenté la formulation du problème avec un seul produit. Nous avons ensuite reformulé le problème en un problème multi-produits. Pour la résolution, et en vue d'améliorer la performance de l'algorithme de branch-and-price-and-cut, nous avons aussi introduit une approche permettant d'arrêter le processus de calcul lorsque la solution obtenue ne s'améliore plus durant un certain nombre itérations. Des conditions d'arrêt ont alors été mises en place dans le but d'obtenir de meilleurs temps de résolution.

Pour évaluer nos résultats, nous les avons comparé avec ceux donnés par CPLEX, un des logiciels les plus puissants pour la programmation en nombres entiers, et qui est d'ailleurs doté d'un module de génération d'inégalités valides. Une comparaison avec la méthode de branch-and-cut (sans génération de colonnes) a été également effectuée. Il s'est avéré que notre méthode est prometteuse et peut donner de bons résultats même pour les instances de très grande taille.

Pour améliorer les résultats expérimentaux, plusieurs idées peuvent être envisagées. Il s'agit par exemple d'introduire de nouvelles inégalités valides permettant de diminuer le saut d'intégralité et le temps de calcul. Une autre proposition consiste à utiliser une méthode heuristique pour limiter le nombre de nœuds générés dans l'arbre de recherche.

BIBLIOGRAPHIE

[1] T. Achterberg. Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

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

[3] A. Atamtürk, A. Gomez et S. Küçükyavuz. Three-partition inequalities for constant capacity capacitated fixed-charge network flow problems. Rapport technique, BCOL.15.02, IEOR, University of California-Berkeley, 2015.

[4] A. Atamtürk et M.W.P. Savelsbergh. Integer programming software systems. *Annals of Operations Research*, 140(1):67–124, 2005.

[5] A. Atamtürk, B. Tezel et S. Kucukyavuz. Submodular path inequalities for the capacitated fixed charge network flow problem. Rapport technique, BCOL.15.03, IEOR, University of California-Berkeley, 2015.

[6] A. Balakrishnan, T.L. Magnanti, A. Shulman et R.T. Wong. Models for planning capacity expansion in local access telecommunication networks. *Annals of Operations Research*, 33(4):237–284, 1991.

[7] A. Balakrishnan, T.L. Magnanti et R.T. Wong. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37(5):716–740, 1989.

[8] M.S. Bazaraa, J.J. Jarvis et H.D. Sherali. *Linear Programming and Network Flows*. John Wiley and Sons, 2ème édition, 1990.

[9] R. Bixby et E. Rothberg. Progress in computational mixed integer programming-a look back from the other side of the tipping point. *Annals of Operations Research*, 149:37–41, 2007.

[10] V. Cacchiani, M. Jaunger, F. Liers, A. Lodi et D.R. Schmidt. Single-commodity robust network design with finite and hose demand sets. Technical report, Publication OR-14-11, University of Bologna, Italy, 2015.

[11] M. Chouman et T.G. Crainic. A MIP-tabu search hybrid framework for multicommodity capacitated fixed-charge network design. Rapport technique, publication CIRRELT-2010-31, 2010.

[12] M. Chouman, T.G. Crainic et B. Gendron. Revue des inégalités valides pertinentes aux problèmes des conception de réseaux. *INFOR-Information Systems and Operational Research*, 41(1):5–34, 2003.

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

[14] M. Chouman, T.G. Crainic et B. Gendron. The impact of filtering in a branch-and-cut algorithm for multicommodity capacitated fixed charge network design. Rapport technique, publication CIRRELT-2014-35, 2014.

[15] P.M. Christophel. *Separation algorithms for cutting planes based on mixed integer row relaxations : Implementation and evaluation in the context of mixed integer programming solver software*. Thèse de doctorat, Universität Paderborn, 2009.

[16] A.M. Costa, J.F. Cordeau et B. Gendron. Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42(3):371–392, 2009.

[17] 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.

[18] T.G. Crainic et M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8(6):601–627, 2002.

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

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

[21] F.R.B. Cruz, J.M.G. Smith et G.R. Mateus. Solving to optimality the uncapacitated fixed-charge network flow problem. *Computers & Operations Research*, 25(1):67 – 81, 1998.

[22] 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, 2005.

[23] S. El Filali. Méthode de génération de colonnes pour les problèmes de conception de réseaux avec coûts d'ajout de capacité. Mémoire de maîtrise, Université de Montréal, 2014.

[24] V. Gabrel. *Résolution de programmes linéaires en nombres entiers de grande taille : décomposition et renforcement pour une résolution exacte et approchée*. Thèse de doctorat, Université Paris Dauphine, 2005.

[25] B. Gavish. Topological design of telecommunication networks-local access design methods. *Annals of Operational Research*, 33(1):17 – 71, 1991.

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

[27] B. Gendron et T.G. Crainic. Relaxations for multicommodity capacitated network design problems. Rapport technique, Publication CRT-965, 1994.

[28] B. Gendron, T.G. Crainic et A. Frangioni. Multicommodity capacitated network design. Dans B. Sansò et P. Soriano, éditeurs, *Telecommunications Network Planning*, chaptitre 1, pages 1–19. Springer, 1999.

[29] B. Gendron et M. Larose. Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *EURO Journal on Computational Optimization*, 2(1):55–75, 2014.

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

[31] F.S. Hillier et G.J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 7 édition, 2004.

[32] K. Holmberg et J. Hellstrand. Solving the uncapacitated network design problem by a lagrangean heuristic and branch-and-bound. *Operations Research*, 46(2):247–259, 1998.

[33] 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.

[34] 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.

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

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

[37] E.A. Miranda, V. Cacchiani, A. Lodi, T. Parriani et D.R. Schmidt. Single-commodity robust network design problem : Complexity, instances and heuristic solutions. *European Journal of Operational Research*, 238(3):711–723, 2014.

[38] G.L. Nemhauser et L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1999.

[39] C.D. Oliva San Martin. *Hybrid constraint programming and mathematical programming methods*. Thèse de doctorat, Université d'Avignon et des Pays de Vaucluse, 2004.

[40] F. Ortega et L.A. Wolsey. A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, 41(3):143 – 158, 2003.

[41] M.W. Padberg, T.J. Van Roy et L.A. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33(4):842–861, 1985.

[42] M. Pioro et D. Medhi. *Routing, flow, and capacity design in communication and computer networks*. The Morgan Kaufmann Series in Networking. Morgan Kaufmann Publishers, 2004.

[43] H. Toussaint. Introduction au Branch Cut and Price et au Solveur SCIP (Solving Constraint Integer Programs). Rapport technique, LIMOS/RR-13-07, 2013.

[44] T.J. Van Roy et L.A. Wolsey. Valid inequalities and separation for uncapacitated fixed charge networks. *Operations Research Letters*, 4(3):105 – 112, 1985.

[45] T.J. Van Roy et L.A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35(1):45–57, 1987.

[46] L.A. Wolsey. Faces for a linear inequality in 0–1 variables. *Mathematical Programming*, 8(1):165–178, 1975.

[47] L.A. Wolsey. *Integer programming*. John Wiley and Sons, 1998.

[48] L.A. Wolsey. Strong formulations for mixed integer programs : valid inequalities and extended formulations. *Mathematical programming*, 97:423–447, 2003.