Résolution exacte de problèmes de couverture par arborescences sous contraintes de capacité

Jérémy Guillot

université de BORDEAUX

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR DE

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

SPÉCIALITÉ MATHÉMATIQUES APPLIQUÉES ET CALCUL SCIENTIFIQUE

Par Jérémy GUILLOT

Résolution exacte de problèmes de couverture par arborescences sous contraintes de capacité

Sous la direction de: François CLAUTIAUX
et Pierre PESNEAU

Soutenue le 18 décembre 2018

Membres du jury :

M.	STAUFFER, Gautier	Professeur, Kedge Business School	Président
Mme.	ELLOUMI, Sourour	Professeure, ENSTA	Rapporteur
M.	FOUILHOUX, Pierre	Maître de Conférences, Paris 6	Rapporteur
Mme.	GICQUEL, Céline	Maître de Conférences, IUT Orsay	Examinateur
M.	OMER, Jérémy	Maître de Conférences, INSA Rennes	Examinateur
M.	VANDERBECK, François	Professeur, Université de Bordeaux	Examinateur
M.	CLAUTIAUX, François	Professeur, Université de Bordeaux	Directeur
M.	PESNEAU, Pierre	Maître de Conférence, Université de Bordeaux	Directeur

Résumé : Dans ce document, nous étudions deux problèmes de sectorisation et proposons plusieurs méthodes de résolution exactes basées sur la décomposition de Dantzig-Wolfe et la génération de colonnes. Nous proposons deux modélisations en fonction de la manière d'appréhender l'objectif du problème qui consiste à obtenir des secteurs compacts. Pour chacune des modélisations, nous comparons des approches de résolution exactes basées sur des formulations compactes ou sur des formulations étendues obtenues par la décomposition de Dantzig-Wolfe. Le premier type de modèles proposé définit la fonction objectif à la manière d'un problème de p-median. Concernant les méthodes de résolution pour ce type de modèle, l'accent est mis sur l'accélération de la convergence de l'algorithme de génération de colonnes en mettant en place des techniques d'agrégation de contraintes afin de réduire la dégénérescence de l'algorithme du simplexe. Les expérimentations numériques montrent que la méthode d'agrégation de contraintes proposée permet effectivement de réduire le nombre d'itérations dégénérées. Cependant, elle ne suffit pas à accélérer l'algorithme de branch-and-price. Le choix d'utilisation de la formulation compacte ou de la formulation étendue dépend du type d'instances résolu. Le second type de modèles formule l'objectif d'une manière assez proche de celui des problèmes de p-centre. L'utilisation d'un tel objectif complexifie la résolution des sous-problèmes de génération de colonnes. L'accent est donc mis sur la conception d'algorithmes de branch-and-bound et de programmation dynamique pour les résoudre efficacement. Les expériences montrent que l'algorithme de branch-and-price surpasse les approches de résolution utilisant une formulation compacte du problème.

Mots clés : *sectorisation, formulations étendues, décomposition de Dantzig-Wolfe, génération de colonnes, agrégation de contraintes*

Abstract : In this document, we study two districting problems and propose several exact methods, based on Dantzig-Wolfe decomposition and column generation, to solve them. For each model, we compare exact approaches based either on compact formulations or on extended formulations obtained using Dantzig-Wolfe decomposition. The first type of model that we propose defines the objective function in a p-median problem fashion. Regarding the methods used to solve that kind of model, we emphasize accelerating the convergence of the column generation algorithm by designing constraint aggregation techniques in order to reduce the degeneracy in the simplex algorithm. Numerical experiments show that this constraint aggregation method indeed reduces the proportion of degenerated iterations. However, it is not enough to speed up the branch-and-price algorithm. Choosing to tackle the problem through either a compact formulation or an extended formulation depends on the structure of the instances to solve. The second type of model formulates the objective function in a way quite similar to that of p-centre problems. Using such an objective function induces complex column generation subproblems. We focus on designing branch-and-bound and dynamic programming algorithms in order to solve them efficiently. Experiments show that the branch-and-price approach surpasses any proposed method based on compact formulations of the problem.

Keywords : *districting, extended formulations, Dantzig-Wolfe decomposition, column generation, constraint aggregation*

Remerciements

Experiments presented in this manuscript were carried out using the PLAFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities : Conseil Régional d'Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the programme d'investissements d'Avenir (see https://www.plafrim.fr/).

Table des matières

Introduction

Notre objectif est d'étudier les limites des approches de décomposition en programmation linéaire pour résoudre deux problèmes difficiles d'optimisation combinatoire inspirés d'un problème pratique.

Notre problématique a été amenée par une entreprise française qui développe des solutions logicielles pour la géolocalisation et l'optimisation des tournées de véhicules de collecte de déchets ménagers. Celle-ci consiste à diviser une zone géographique, telle qu'une ville, en secteurs, chaque secteur étant ensuite traité par un véhicule de collecte. L'entreprise souhaitait pouvoir générer des secteurs qui puissent être traités facilement par la flotte de véhicule, ce qui signifie que les secteurs sont limités par une quantité de travail (temps de collecte et quantité de déchets ménagers) et doivent être « beaux », ce que nous pouvons interpréter par les notions plus formelles de *connexité*, de *compacité*, et de *convexité*. Ces contraintes sont transposables aux modèles de la littérature scientifique concernant les problèmes de sectorisation.

Bien que les applications industrielles associées aux problèmes de sectorisation soient particulièrement diverses, allant des découpages politiques à l'établissement de territoires de ventes en passant par l'optimisation de services publiques, la plupart des modèles partagent une structure commune. En reprenant le lexique utilisé dans ce domaine, les secteurs générés doivent être *équilibrés*, *connexes* et *compacts*.

La notion d'équilibre est généralement représentée à la manière d'un problème de sac à dos doublement contraint. Étant donné une certaine ressource à répartir de manière équilibrée entre les secteurs, la charge de chaque secteur doit être similaire ; autrement dit, elle doit appartenir à un intervalle défini par une charge minimale et une charge maximale. Dans notre cas, nous avons deux ressources à considérer : le temps de traitement d'un point de collecte (indépendamment du temps que met le véhicule à s'y rendre), et la quantité de déchets qui y est récupérée. Tout secteur est alors limité par une capacité maximale sur chacune des deux ressources, afin de garantir qu'il ne représente pas une charge de travail excessive.

La notion de connexité, quant à elle, est intuitive — on impose que les secteurs soient dessinés d'un seul bloc, sans être traversés par un autre secteur — mais complexe à traiter. La difficulté est qu'une contrainte de connexité fidèle doit pouvoir tenir compte de chaque paire de sommets appartenant au même secteur, et de chaque chemin pouvant les relier. Généralement, cela nécessite un nombre polynomial mais conséquent de contraintes linéaires pour pouvoir être formulé dans un programme mathématique. À partir de ce constat, plusieurs auteurs ont privilégié une formulation plus restrictive basée sur le concept d'arbres des plus courts chemins.

Enfin, la notion de compacité est représentée par une contrainte souple, définie par

la fonction objectif du problème. C'est ce point, précisément, qui différencie les deux modèles que nous étudions. L'un des modèles utilise une fonction objectif proche du problème de p-median, tandis que le second se rapproche plus du problème de p-centre. Les problèmes du p-median et du p-centre sont des problèmes de sectorisation classiques de la littérature scientifique.

En revanche, la notion de convexité, évoquée plus haut, n'est pas particulièrement représentée dans la recherche scientifique sur les problèmes de sectorisation. Elle n'est pas non plus représentée explicitement dans nos modèles, mais a tout de même orienté nos choix quant à la manière de modéliser la connexité et la compacité. Cela se traduit par une modélisation fortement basée sur le concept de plus courts chemins.

Une autre spécificité de notre problème est que nous considérons une couverture de la zone géographique, et non une partition des points de collecte. De ce fait, un point de collecte peut appartenir à deux secteurs différents. Bien que les problèmes de sectorisation soient en grande majorité des problèmes de partitionnement, l'utilisation de contraintes de couverture nous offre plus de flexibilité dans la construction des solutions, sans pour autant changer radicalement la structure des solutions optimales générées. En effet, puisque les secteurs sont limités par des capacités maximales, et que tout point de collecte implique un coût, un point ne sera couvert par plusieurs secteurs que si cela représente un avantage significatif pour le traitement de la zone toute entière.

D'un point de vue méthodologique, nos approches de résolution découlent de formulations étendues obtenues par la décomposition de Dantzig-Wolfe. Les enjeux essentiels de ces méthodes sont la résolution des sous-problèmes de pricing issus de cette décomposition, et la convergence du problème maître.

Les sous-problèmes de pricing sont des extensions du problème de sac à dos avec contraintes de précédence en arborescence (*Tree Knapsack Problem*, ou TKP). Nous adaptons à nos besoins des méthodes de résolution de la littérature, basées sur les approches par séparation et évaluation (branch-and-bound) et sur la programmation dynamique. Nous étudions également l'application de techniques basées sur le principe d'agrégation de contraintes afin de diminuer le phénomène de dégénérescence inhérent aux approches par génération de colonnes.

Ce document est organisé de la manière suivante. Dans le chapitre 1, nous présentons l'état de l'art de la recherche scientifique concernant des domaines fondamentaux pour les chapitres ultérieurs, à savoir la programmation mathématique en nombres entiers, les méthodes d'agrégation de contraintes, les problèmes de sectorisation et un problème de sac à dos avec contraintes de précédence. Le chapitre 2 est dédié à une première modélisation de notre problème de sectorisation sous la forme d'un problème de p-median. Nous y présentons les modèles mathématiques du problème maître et des sous-problèmes obtenus par la décomposition de Dantzig-Wolfe, ainsi que deux oracles. Nous étudions également plusieurs manières d'adapter des schémas d'agrégation de contraintes de la littérature à notre problème. La seconde modélisation proposée, traitée dans le chapitre 3, se rapproche des problèmes de p-centre. Ce problème étant plus complexe à résoudre, nous nous concentrons essentiellement sur la mise au point d'oracles pour la génération de colonnes et les performances qu'ils nous permettent d'obtenir.

Dans le premier chapitre, nous nous proposons de présenter l'état de l'art concernant plusieurs problèmes d'optimisation et méthodes de résolution qui seront par la suite

utilisés tout au long du document. Cet état de l'art traite de quatre sujets principaux : deux méthodologiques (programmation linéaire et agrégation), et deux applicatifs (sectorisation et sac à dos). La section 1.1 est dédiée à la programmation linéaire en nombres entiers. Nous y présentons les bases de la programmation linéaire classique, ainsi que certains concepts qui y sont liés, tels que les méthodes de sous-gradient, la relaxation Lagrangienne, et la décomposition de Dantzig-Wolfe. Nous abordons également quelques méthodes permettant la résolution de programmes linéaires en variables entières. Nous nous intéressons dans la section 1.2 à une famille de méthodes visant à réduire le phénomène de dégénérescence dans l'algorithme du simplexe primal. En particulier, nous y rappelons les principes de trois méthodes d'agrégation de contraintes et tentons de mettre en valeur leurs similarités et leurs différences. La section 1.3 traite de deux problèmes classiques de sectorisation : le problème du p-median et celui du p-centre. Nous y rapportons les modèles mathématiques classiques de la littérature scientifique. Enfin, la section 1.4 est consacrée à une variante du problème du sac à dos sous contraintes de précédence. Notamment, nous y présentons deux méthodes de résolution ; dans les chapitres suivants, nous adaptons ces méthodes à nos besoins.

Le deuxième chapitre est dédié à un problème de sectorisation consistant à couvrir un graphe à l'aide de secteurs, chaque secteur devant vérifier deux contraintes de sac à dos. La fonction objectif de ce problème est similaire à celle du problème classique du p-median. À partir d'une formulation compacte naturelle, nous construisons une formulation étendue par la décomposition de Dantzig-Wolfe. Nous obtenons ainsi un sous-problème de pricing pour chaque sommet du graphe, un sous-problème étant associé à un ensemble de secteurs de même centre. Ces sous-problèmes sont des extensions à deux dimensions du problème de sac à dos avec contraintes de précédence en arborescences ; nous adaptons un algorithme de branch-and-bound et un programme dynamique issus de la littérature scientifique pour les résoudre. Nous proposons d'adapter une méthode d'agrégation de contraintes, originellement prévue pour le problème de *set-partitioning*, afin de réduire le nombre d'itérations dégénérées dans la génération de colonnes. Les algorithmes proposés sont comparés en utilisant deux jeux d'instance, l'un étant dérivé de la CVRPLIB, une bibliothèque d'instances classiques pour les problèmes de *vehicle routing*, et l'autre étant composé d'instances réelles générées par une entreprise travaillant dans le domaine de la collecte de déchets ménagers.

Dans le troisième chapitre, nous étudions une autre variante du problème de sectorisation. La fonction objectif de ce problème est similaire à celle du problème classique du p-centre. Nous présentons trois modèles de programmation linéaire en nombres entiers, deux compacts et un basé sur une reformulation de Dantzig-Wolfe. Pour ce dernier, une attention particulière est portée aux sous-problèmes de génération de colonnes, qui sont considérés comme des extensions du problème de sac à dos avec contraintes de précédence en arborescences (TKP), auquel nous ajoutons une dimension de sac à dos ainsi que la notion de rayon. Bien que les contraintes de précédence et la notion de rayon ne puissent pas être intégrées simultanément, nous adaptons un algorithme de branch-and-bound et un programme dynamique issus de la littérature du TKP pour résoudre les problèmes de pricing. Les algorithmes sont comparés sur les mêmes jeux d'essai que le problème précédent.

Le dernier chapitre propose des conclusions à ce travail, et ouvre des perspectives

pour des travaux futurs.

Chapitre 1

État de l'art

Dans ce chapitre, nous nous proposons de présenter l'état de l'art concernant plusieurs problèmes d'optimisation et méthodes de résolution qui seront par la suite utilisés tout au long du document. Cet état de l'art traite de quatre sujets principaux : deux méthodologiques (programmation linéaire et agrégation), et deux applicatifs (sectorisation et sac à dos). La section 1.1 est dédiée à la programmation linéaire en nombres entiers. Nous y présentons les bases de la programmation linéaire classique, ainsi que certains concepts qui y sont liés, tels que les méthodes de sous-gradient, la relaxation Lagrangienne, et la décomposition de Dantzig-Wolfe. Nous abordons également quelques méthodes permettant la résolution de programmes linéaires en variables entières. Nous nous intéressons dans la section 1.2 à une famille de méthodes visant à réduire le phénomène de dégénérescence dans l'algorithme du simplexe primal. En particulier, nous y rappelons les principes de trois méthodes d'agrégation de contraintes et proposons une analyse de leurs similarités et leurs différences. La section 1.3 traite de deux problèmes classiques de sectorisation : le problème du p-median et celui du p-centre. Nous y rapportons les modèles mathématiques classiques de la littérature scientifique. Enfin, la section 1.4 est consacrée à une variante du problème du sac à dos sous contraintes de précédence. Notamment, nous y présentons deux méthodes de résolution que nous adaptons à nos besoins dans les chapitres suivants.

1.1 Programmation linéaire en nombres entiers

Dans cette section, nous introduisons les notions de programmation linéaire en nombres entiers que nous utilisons dans la suite du document. Nous commençons par rappeler les bases de la programmation linéaire, qui sont fondamentales aux techniques de génération de colonnes et de réduction de la dégénérescence proposées dans ce travail. Bien que nous considérions essentiellement la programmation linéaire du point de vue de l'algorithme du simplexe primal, nous décrivons également les éléments de base des méthodes de sous-gradient. Dans les chapitres suivants, nous utilisons ces dernières pour résoudre des problèmes non-linéaires ou pour mettre au point des heuristiques rapides en alternative à l'algorithme du simplexe. Ensuite, nous présentons les principes de la relaxation Lagrangienne et de la décomposition de Dantzig-Wolfe, deux méthodes génériques permettant, entre autres, de calculer des bornes duales plus fortes que la relaxation continue d'un problème linéaire en nombres entiers. Enfin, nous abordons les méthodes de

branch-and-bound et de programmation dynamique, qui sont des approches de résolution exactes de type diviser-pour-régner.

1.1.1 Programmation linéaire

Le terme de programmation linéaire désigne l'étude des problèmes d'optimisation linéaire, c'est-à-dire des problèmes consistant à minimiser une fonction objectif linéaire tout en vérifiant un ensemble de contraintes, linéaires également. Ce type de problème se formule donc naturellement comme un système d'équations combiné à une fonction objectif. La programmation linéaire moderne est due en grande partie aux travaux de Dantzig, qui a présenté, notamment lors d'une conférence en 1948 [Dan49], le concept de programme linéaire et la méthode du simplexe permettant de trouver la solution d'un tel programme.

Soit un problème d'optimisation linéaire avec n variables de décision positives et m contraintes. À chaque variable $x_i, \forall i \in \{1, \dots, n\}$, est associé un coefficient c_i dans la fonction objectif. Étant donné $j \in \{1, \dots, m\}$, soit b_j la partie constante de la j-ième contrainte. Enfin, soit $a_{i,j}$ le coefficient de la variable x_i dans la j-ième contrainte. Ce problème peut être formulé par le programme linéaire suivant.

$$(\text{PL}) = \begin{cases} \min \displaystyle\sum_{i=1}^{n} c_i x_i & \\ \text{t.q.} \quad \displaystyle\sum_{i=1}^{n} a_{i,j} x_i = b_j & \forall j = 1, \dots, m, \quad (1.1) \\ x_i \geq 0 & \forall i = 1, \dots, n. \quad (1.2) \end{cases}$$

On parle de programme linéaire en *forme standard* lorsque la formulation n'utilise que des variables positives et des contraintes d'égalité. La formulation (PL) peut être condensée en utilisant l'écriture matricielle suivante, avec deux vecteurs $\boldsymbol{c} \in \mathbb{R}^n$ et $\boldsymbol{b} \in \mathbb{R}^m$, une matrice de coefficients $A \in \mathbb{R}^{m \times n}$ et $\boldsymbol{x} \in \mathbb{R}^n$ un vecteur de n variables continues. Pour une question de simplification et de clarté, nous utilisons la notation $\mathbf{0}$ pour représenter un vecteur dont les composantes sont toutes égales à 0 et dont la dimension est adaptée au contexte.

$$(\text{PL}) = \begin{cases} \min \quad \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{t.q.} \quad A\boldsymbol{x} = \boldsymbol{b}, & (1.3) \\ \boldsymbol{x} \geq \mathbf{0}. & (1.4) \end{cases} \qquad (\text{D}) = \begin{cases} \max \quad \boldsymbol{b}^\top \boldsymbol{\pi} & \\ \text{t.q.} \quad A^\top \boldsymbol{\pi} \leq \boldsymbol{c}, & (1.5) \\ \boldsymbol{\pi} \in \mathbb{R}^m. & (1.6) \end{cases}$$

Le programme linéaire (D) est appelé le *problème dual* de la formulation (PL). Ce problème est défini par un vecteur $\boldsymbol{\pi} \in \mathbb{R}^m$ de m variables de décision réelles et par n contraintes linéaires. D'après le théorème de la dualité forte, s'il existe une solution optimale réalisable pour le problème (PL), alors il existe une solution optimale réalisable de même valeur pour le problème (D).

Afin d'introduire la notion de *dégénerescence*, rappelons brièvement le principe de la méthode du simplexe. Soit (PL) un programme linéaire admettant au moins une solution

réalisable et dont la valeur optimale est bornée. Sans perte de généralité, nous pouvons supposer que la matrice A des contraintes de (PL) est de rang plein et que son nombre de colonnes est supérieur au nombre de lignes $n > m$. Puisque la matrice est de rang plein, il existe au moins un ensemble de m colonnes linéairement indépendantes. Supposons qu'il s'agisse des m premières colonnes de A. Nous pouvons alors réécrire $A = \begin{bmatrix} B & N \end{bmatrix}$ et $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_{\mathcal{B}} \\ \boldsymbol{x}_{\mathcal{N}} \end{pmatrix}$. La matrice carrée B, composée des m premières colonnes de A, est alors appelée une *base* du programme (PL) et les variables $\boldsymbol{x}_{\mathcal{B}}$ (resp. $\boldsymbol{x}_{\mathcal{N}}$) sont dites *variables en base* (resp. *variables hors-base*). Deux bases B_1 et B_2 sont *voisines* si elles ne diffèrent que par une seule colonne/variable. La solution basique correspondant à une base B est la solution unique vérifiant $A\boldsymbol{x} = \boldsymbol{b}$ et $\boldsymbol{x}_{\mathcal{N}} = \boldsymbol{0}$. En conséquence,

$$A\boldsymbol{x} = B\boldsymbol{x}_{\mathcal{B}} + N\boldsymbol{x}_{\mathcal{N}} = B\boldsymbol{x}_{\mathcal{B}} = b \quad \Leftrightarrow \quad \boldsymbol{x}_{\mathcal{B}} = B^{-1}b.$$

Notons que cette solution peut être irréalisable pour le programme (PL), car elle ne vérifie pas forcément les contraintes de positivité $\boldsymbol{x}_{\mathcal{B}} \geq \boldsymbol{0}$.

D'après le théorème fondamental de la programmation linéaire, si un programme linéaire admet une solution réalisable et que la fonction objectif est bornée, alors il existe une base définissant une solution réalisable optimale. Une itération de la méthode du simplexe consiste essentiellement à se déplacer (opération de pivot) d'une base réalisable $B_{(k)}$ à une base réalisable voisine $B_{(k+1)}$ de meilleur coût $\boldsymbol{c}_{\mathcal{B}_{(k+1)}}^{\top} B_{(k+1)}^{-1}\boldsymbol{b} \leq \boldsymbol{c}_{\mathcal{B}_{(k)}}^{\top} B_{(k)}^{-1}\boldsymbol{b}$. L'algorithme se termine lorsque la base courante B^* n'admet aucune base voisine améliorante ; la base B^* est alors optimale.

Une opération de pivot consiste donc à modifier la valeur d'une variable hors-base pour lui donner la plus grande valeur possible (on dit que la variable *entre en base*) tout en vérifiant les contraintes $A\boldsymbol{x} = b$ et $\boldsymbol{x} \geq \boldsymbol{0}$. Les autres variables hors-base gardent une valeur nulle.

Soit j l'indice de la variable entrant en base et A_j sa colonne dans la matrice A. Soit $\boldsymbol{h}_{\mathcal{B}}$ la direction de modification des variables en base lorsque l'on augmente la valeur de la variable x_j d'une unité et γ la longueur du pas effectué. Sachant que $B\boldsymbol{x}_{\mathcal{B}} = \boldsymbol{b}$, nous obtenons l'équation suivante.

$$B(\boldsymbol{x}_{\mathcal{B}} + \gamma\boldsymbol{h}_{\mathcal{B}}) + \gamma A_j = \boldsymbol{b} \quad \Leftrightarrow \quad \gamma B\boldsymbol{h}_{\mathcal{B}} = -\gamma A_j \quad \Leftrightarrow \quad \boldsymbol{h}_{\mathcal{B}} = -B^{-1}A_j.$$

L'opération de pivot modifie la valeur de la fonction objectif par $\gamma(c_j + \boldsymbol{c}_{\mathcal{B}}^{\top}\boldsymbol{h}_{\mathcal{B}}) = \gamma(c_j - \boldsymbol{c}_{\mathcal{B}}^{\top}B^{-1}A_j)$. La valeur $c_j - \boldsymbol{c}_{\mathcal{B}}^{\top}B^{-1}A_j$ est alors appelée le coût réduit de la variable entrante x_j. Notons que le coût réduit d'une variable en base est toujours nul.

Supposons que le coût réduit de toute variable soit positif ou nul. Nous pouvons en déduire

$$\boldsymbol{c}^{\top} - \boldsymbol{c}_{\mathcal{B}}^{\top}B^{-1}A \geq \boldsymbol{0} \quad \Leftrightarrow \quad \boldsymbol{c} - A^{\top}(\boldsymbol{c}_{\mathcal{B}}^{\top}B^{-1})^{\top} \geq \boldsymbol{0} \quad \Leftrightarrow \quad A^{\top}(\boldsymbol{c}_{\mathcal{B}}^{\top}B^{-1})^{\top} \leq \boldsymbol{c}.$$

La solution $\boldsymbol{\pi}_{\mathcal{B}} = (\boldsymbol{c}_{\mathcal{B}}^{\top}B^{-1})^{\top}$ est donc une solution réalisable de (D). De plus,

$$\boldsymbol{b}^{\top}\boldsymbol{\pi}_{\mathcal{B}} = \boldsymbol{b}^{\top}(\boldsymbol{c}_{\mathcal{B}}^{\top}B^{-1})^{\top} = \boldsymbol{c}_{\mathcal{B}}^{\top}B^{-1}\boldsymbol{b} = \boldsymbol{c}_{\mathcal{B}}^{\top}\boldsymbol{x}_{\mathcal{B}}.$$

Les solutions $\boldsymbol{\pi}_{\mathcal{B}}$ et $\boldsymbol{x}_{\mathcal{B}}$ sont donc des solutions basiques réalisables pour les problèmes (D) et (PL), respectivement, et partagent la même valeur. D'après le théorème de la

dualité forte, ce sont des solutions basiques optimales pour leurs problèmes respectifs.

Ainsi, une condition suffisante d'optimalité pour une base donnée B est que toutes les variables aient un coût réduit positif ou nul. En revanche, si le coût réduit $(c_j - \boldsymbol{c}_{\mathcal{B}}^{\top} B^{-1} A_j)$ d'une variable hors-base x_j est négatif, alors il est possible d'améliorer la valeur de la solution courante en effectuant le pivot correspondant. Dans ce cas, la valeur du pas γ est bornée par $\boldsymbol{x}_{\mathcal{B}} + \gamma \boldsymbol{h}_{\mathcal{B}} \geq \mathbf{0}$, car les contraintes $\boldsymbol{x} \geq \mathbf{0}$ doivent rester vérifiées après l'opération de pivot.

Nous parlons d'itération *dégénérée* de l'algorithme du simplexe lorsque l'opération de pivot mène à une nouvelle base correspondant à une solution identique $B_{(k+1)}^{-1}\boldsymbol{b} = B_{(k)}^{-1}\boldsymbol{b}$. Puisque la méthode du simplexe n'utilise que des variables hors-base de coût réduit négatif pour effectuer un pivot, une itération sera dégénérée ssi la direction $\boldsymbol{h}_{\mathcal{B}}$ du pivot amène à réduire la valeur d'une variable basique déjà nulle. En effet, tout pas $\gamma > 0$ impliquerait une valeur strictement négative pour cette variable, violant la contrainte $\boldsymbol{x} \geq \mathbf{0}$. Une base ayant au moins une variable de valeur nulle est dite *dégénérée*. Selon la règle de pivot utilisée, l'algorithme du simplexe peut nécessiter un grand nombre d'itérations avant de quitter un ensemble de bases dégénérées, voire être piégé dans un cycle de pivots dégénérés. De nombreuses techniques ont été mises au point pour réduire les problèmes de dégénérescence, mais nous ne nous intéresserons dans ce document qu'aux techniques spécifiques au contexte de la génération de colonnes, abordées dans une section ultérieure.

La popularité de l'algorithme du simplexe est due à ses performances pratiques, bien que les règles de pivot actuellement connues induisent une complexité exponentielle au pire cas. Les problèmes d'optimisation linéaires appartiennent néanmoins à la classe de complexité P des problèmes pouvant être résolus en temps polynomial [Kha80].

Il existe de très nombreux ouvrages traitant de la programmation linéaire, par exemple [BT97] et [Van14]. Les solveurs modernes de programmation linéaire utilisent essentiellement l'algorithme du simplexe dual, plus efficace en pratique que le simplexe primal, ainsi que des méthodes d'algèbre linéaire spécifiques aux matrices creuses. Ces solveurs disposent également de méthodes de points intérieurs, concurrentes de l'algorithme du simplexe selon le type de problème considéré. En effet, les méthodes de points intérieurs présentent de nombreux avantages, notamment une complexité polynomiale et un plus grand potentiel pour la parallélisation. De plus, ces méthodes sont adaptées pour la résolution de certains problèmes non-linéaires, ce qui en fait des outils de prédilection pour l'optimisation quadratique ou semi-définie positive. Malgré tout, les performances de l'algorithme du simplexe continuent de surpasser celles des méthodes de points intérieurs en pratique pour certaines familles de problèmes. Pour un état de l'art plus détaillé de l'algorithme du simplexe et des méthodes de points intérieurs, voir [Bix12, Gon12].

1.1.2 Méthodes de sous-gradient

Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction convexe non-différentiable à minimiser. Un vecteur $\nabla_{\boldsymbol{x}_0}$ est un sous-gradient de f au point $\boldsymbol{x}_0 \in \mathbb{R}$ si, pour tout $\boldsymbol{x} \in \mathbb{R}$, on a

$$f(\boldsymbol{x}) - f(\boldsymbol{x}_0) \geq \nabla_{\boldsymbol{x}_0} \cdot (\boldsymbol{x} - \boldsymbol{x}_0).$$

Étant donné un point de départ $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$, une méthode de sous-gradient est un algorithme d'optimisation générant une séquence de points $\boldsymbol{x}^{(k)}$, $k = 1, 2, 3, \ldots$ définis par

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \gamma^{(k)} \nabla_{\boldsymbol{x}^{(k)}},$$

avec $\nabla_{\boldsymbol{x}^{(k)}}$ un sous-gradient de f au point $\boldsymbol{x}^{(k)}$ et $\gamma^{(k)}$ la longueur du pas effectué dans la direction $-\nabla_{\boldsymbol{x}^{(k)}}$. Les méthodes de sous-gradient peuvent être vues comme une généralisation de l'algorithme du gradient. Cependant, la capacité et la rapidité de l'algorithme à converger vers une solution optimale $\boldsymbol{x}^*$ minimisant f dépendent essentiellement de la règle utilisée pour déterminer la longueur du pas $\gamma^{(k)}$. En pratique, le pas est généralement calculé en utilisant la règle de Polyak [Pol69]

$$\gamma^{(k)} = \tau^{(k)} \frac{f(\boldsymbol{x}^{(k)}) - f^*}{\|\nabla_{\boldsymbol{x}^{(k)}}\|_2^2}$$

avec f^* la valeur optimale (ou une approximation de la valeur optimale) de la fonction f et $\tau^{(k)}$ un paramètre tel que $0 < \tau^{(k)} \leq 2$.

L'intérêt des méthodes de sous-gradient réside principalement dans leur simplicité d'implémentation et la diversité des fonctions auxquelles elles peuvent s'appliquer. Notons également qu'il est possible d'appliquer une méthode de sous-gradient sur un problème avec contraintes, par projection [Pol67].

Étant donné $\mathcal{P}$ un ensemble convexe, soit le problème d'optimisation suivant.

$$\begin{cases} \min & f(\boldsymbol{x}) \\ \text{t.q.} & \boldsymbol{x} \in \mathcal{P}. \end{cases} \tag{1.7}$$

La règle utilisée pour une méthode de sous-gradient projeté est alors de la forme

$$x^{(k+1)} = \rho\left(x^{(k)} - \gamma^{(k)} \nabla_{x^{(k)}}\right),$$

avec ρ une projection orthogonale sur $\mathcal{P}$.

Malgré leurs aspects pratiques, les méthodes de sous-gradient présentent des limitations parfois importantes. Par exemple, dans le contexte de la résolution d'un problème dual Lagrangien (abordé dans la section suivante), une méthode de sous-gradient convergera vers une solution duale optimale mais ne permet généralement pas de trouver une solution primale optimale. Il existe cependant des variantes permettant de construire ou d'approcher une solution primale optimale à partir d'une combinaison convexe des solutions primales obtenues précédemment. Par exemple, la méthode proposée dans [Sho85] construit une suite ergodique $\left\{\bar{\boldsymbol{y}}^{(k)}\right\}$ de solutions primales à partir de la solution primale $\boldsymbol{y}^{(k)}$ obtenue à chaque itération k de l'algorithme de sous-gradient. Cette suite est donnée par la formule

$$\bar{\boldsymbol{y}}^{(k)} = \sum_{i=1}^{k} \bar{\gamma}_i^k \boldsymbol{y}^{(i)}, \qquad \text{avec} \qquad \bar{\gamma}_i^k = \frac{\gamma_i}{\sum_{j=1}^{k} \gamma_j}, \ \forall i = 1, \ldots, k.$$

Une autre méthode classique est l'algorithme du volume [BA00], qui utilise la formule récursive

$$\bar{\boldsymbol{y}}^{(k)} = \theta^{(k)} \boldsymbol{y}^{(k)} + (1 - \theta^{(k)}) \bar{\boldsymbol{y}}^{(k-1)}, \ k > 1, \qquad \text{avec} \qquad \bar{\boldsymbol{y}}^{(1)} = \boldsymbol{y}^{(1)} \text{ et } \theta^{(k)} \in \,]0, 1], \ \forall k.$$

L'article [AW09] propose une analyse plus détaillée des propriétés de convergence des algorithmes de sous-gradient et des méthodes de construction de solutions primales.

1.1.3 Relaxation Lagrangienne

Étant donné $m_0, m_1, n \in \mathbb{R}$, soit trois vecteurs $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{b}_0 \in \mathbb{R}^{m_0}$ et $\boldsymbol{b}_1 \in \mathbb{R}^{m_1}$, deux matrices de contraintes $D \in \mathbb{R}^{m_0 \times n}$ et $F \in \mathbb{R}^{m_1 \times n}$ et $\boldsymbol{x}$ un vecteur de n variables de décision. Considérons (MIP) le programme linéaire en nombres entiers défini de la manière suivante.

$$(\text{MIP}) = \begin{cases} \min \quad \boldsymbol{c}^\top \boldsymbol{x} & & \\ \text{t.q.} \quad D\boldsymbol{x} = \boldsymbol{b}_0, & & (1.8) \\ \qquad F\boldsymbol{x} = \boldsymbol{b}_1, & & (1.9) \\ \qquad \boldsymbol{x} \geq \boldsymbol{0}, & & (1.10) \\ \qquad x_j \in \mathbb{N} & \forall j \in \mathcal{E}. & (1.11) \end{cases}$$

L'ensemble $\mathcal{E}$ contient les indices des variables de décisions entières. Étant donné un vecteur de multiplicateurs Lagrangiens $\boldsymbol{\mu} \in \mathbb{R}^{m_0}$, nous pouvons définir $(\text{RL}_{\boldsymbol{\mu}})$ la relaxation Lagrangienne de (MIP) par rapport aux contraintes $D\boldsymbol{x} = \boldsymbol{b}_0$ de la manière suivante.

$$(\text{RL}_{\boldsymbol{\mu}}) = \begin{cases} \min \quad L_{\boldsymbol{\mu}}(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\mu}^\top (\boldsymbol{b}_0 - D\boldsymbol{x}) & & \\ \text{t.q.} \quad F\boldsymbol{x} = \boldsymbol{b}_1, & & (1.12) \\ \qquad \boldsymbol{x} \geq \boldsymbol{0}, & & (1.13) \\ \qquad x_j \in \mathbb{N} & \forall j \in \mathcal{E}. & (1.14) \end{cases}$$

Ainsi, les contraintes $D\boldsymbol{x} = \boldsymbol{b}_0$ sont relâchées dans l'objectif de la fonction. Elles ne servent donc plus à définir l'espace des solutions réalisables du problème. La fonction $L_{\boldsymbol{\mu}}(\boldsymbol{x})$ ainsi obtenue est appelée le *Lagrangien*. Ce Lagrangien nous permet ensuite de définir le problème d'optimisation

$$\max_{\boldsymbol{\mu}} \min_{\boldsymbol{x} \geq \boldsymbol{0}} L_{\boldsymbol{\mu}}(\boldsymbol{x}),$$

appelé *problème dual Lagrangien*. Une manière d'interpréter le Lagrangien est de considérer que nous avons relâché les contraintes $D\boldsymbol{x} = \boldsymbol{b}_0$ du problème (MIP) — ce qui donne plus de liberté pour construire une solution $\boldsymbol{x}$ — mais que la violation de ces contraintes donne lieu à une pénalité dans la fonction objectif. Le problème dual Lagrangien consiste alors à trouver les pénalités optimales.

La relaxation Lagrangienne présente deux intérêts majeurs. Premièrement, les contraintes relâchées sont généralement choisies de sorte que le nouveau problème $(\text{RL}_{\boldsymbol{\mu}})$ soit facile à résoudre. Deuxièmement, la borne duale obtenue par résolution du problème dual Lagrangien est au moins aussi forte que la borne donnée par la relaxation continue du problème original (MIP) [Geo74]. En contrepartie, le problème dual Lagrangien est non-différentiable, mais concave et peut donc être résolu par des méthodes d'optimisation adaptées à ce type de problème. Notons également que n'importe quel vecteur de multiplicateurs $\boldsymbol{\mu}$ permet d'obtenir une borne duale Lagrangienne, bien que celle-ci puisse alors être moins forte que la borne issue de la relaxation continue.

La relaxation Lagrangienne est issue de la méthode des multiplicateurs de Lagrange utilisée pour résoudre des problèmes d'optimisation définis par des fonctions dérivables. Elle permet également d'obtenir le problème dual d'un programme linéaire, comme nous allons le voir.

Considérons le programme linéaire générique suivant, avec deux vecteurs $\boldsymbol{c} \in \mathbb{R}^n$ et $\boldsymbol{b} \in \mathbb{R}^m$, une matrice de coefficients $A \in \mathbb{R}^{m \times n}$ et $\boldsymbol{x} \in \mathbb{R}^n$ un vecteur de n variables continues.

$$(\text{PL}) = \begin{cases} \min & \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{t.q.} & A\boldsymbol{x} = \boldsymbol{b}, & (1.15) \\ & \boldsymbol{x} \geq \boldsymbol{0}. & (1.16) \end{cases}$$

Étant donné un vecteur de multiplicateurs Lagrangiens $\boldsymbol{\mu} \in \mathbb{R}^m$, nous pouvons définir le Lagrangien

$$L_{\boldsymbol{\mu}}(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\mu}^\top (\boldsymbol{b} - A\boldsymbol{x}).$$

Notons que le problème dual Lagrangien peut être reformulé comme suit.

$$\begin{aligned} \max_{\boldsymbol{\mu}} \min_{\boldsymbol{x} \geq \boldsymbol{0}} L_{\boldsymbol{\mu}}(\boldsymbol{x}) &= \max_{\boldsymbol{\mu}} \min_{\boldsymbol{x} \geq \boldsymbol{0}} \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\mu}^\top (\boldsymbol{b} - A\boldsymbol{x}) \\ &= \max_{\boldsymbol{\mu}} \left(\boldsymbol{\mu}^\top \boldsymbol{b} + \min_{\boldsymbol{x} \geq \boldsymbol{0}} (\boldsymbol{c}^\top - \boldsymbol{\mu}^\top A)\boldsymbol{x} \right) \end{aligned}$$

Puisque $\boldsymbol{x} \geq \boldsymbol{0}$, nous avons alors

$$\min_{\boldsymbol{x} \geq \boldsymbol{0}} (\boldsymbol{c}^\top - \boldsymbol{\mu}^\top A)\boldsymbol{x} = \begin{cases} 0 & \text{si } (\boldsymbol{c}^\top - \boldsymbol{\mu}^\top A) \geq \boldsymbol{0}, \\ -\infty & \text{sinon.} \end{cases}$$

Nous pouvons ainsi retrouver le problème dual défini dans la section 1.1.1.

$$\max_{\boldsymbol{\mu}} \min_{\boldsymbol{x} \geq \boldsymbol{0}} L_{\boldsymbol{\mu}}(\boldsymbol{x}) \quad \Leftrightarrow \quad (\text{D}) = \begin{cases} \max & \boldsymbol{b}^\top \boldsymbol{\mu} & \\ \text{t.q.} & A^\top \boldsymbol{\mu} \leq \boldsymbol{c}, & (1.17) \\ & \boldsymbol{\mu} \in \mathbb{R}^m & (1.18) \end{cases}$$

La relaxation Lagrangienne est un sujet abordé dans la plupart des livres dédiés à l'optimisation combinatoire. Notons par exemple [Lem01], qui offre une introduction plus détaillée à la relaxation Lagrangienne et à ses extensions.

1.1.4 Décomposition de Dantzig-Wolfe

La décomposition de Dantzig-Wolfe est une méthode générique permettant de construire une formulation étendue pour un programme linéaire donné [DW60]. Cette décomposition s'appuie sur une structure des contraintes par blocs, comme la formulation ci-dessous.

$$(\text{PL}) = \begin{cases} \min & \boldsymbol{c}_1^\top \boldsymbol{x}_1 + \boldsymbol{c}_2^\top \boldsymbol{x}_2 & & \\ \text{t.q.} & D_1\boldsymbol{x}_1 + D_2\boldsymbol{x}_2 = \boldsymbol{b}_0, & & (1.19) \\ & F_1\boldsymbol{x}_1 = \boldsymbol{b}_1, & & (1.20) \\ & F_2\boldsymbol{x}_2 = \boldsymbol{b}_2, & & (1.21) \\ & \boldsymbol{x}_1, \boldsymbol{x}_2 \geq \boldsymbol{0}. & & (1.22) \end{cases}$$

Sans les contraintes (1.19), le problème pourrait être décomposé en deux sous-

problèmes disjoints, car les contraintes (1.20) ne s'appliquent qu'aux variables $\boldsymbol{x}_1$, tandis que les contraintes (1.21) ne s'appliquent qu'aux variables $\boldsymbol{x}_2$. Les contraintes (1.19) sont dites *liantes* car elles sont communes à toutes les variables.

1.1.4.1 Reformulation

La décomposition de Dantzig-Wolfe consiste à séparer le problème original en un *problème maître* (MP) contenant les contraintes liantes, et deux (ou plus, selon le nombre de blocs de contraintes) sous-problèmes (SP_1) et (SP_2) contenant chacun un des blocs de contraintes (1.20) et (1.21). Nous aborderons la définition des sous-problèmes dans la section suivante. Considérons, dans un premier temps, les polyèdres $\mathcal{P}_1$ et $\mathcal{P}_2$ définis de la façon suivante.

$$\mathcal{P}_i = \{\boldsymbol{x}_i \in \mathbb{R}_+ \mid F_i \boldsymbol{x}_i = b_i\}, \qquad i = 1, 2.$$

Ces deux polyèdres permettent de définir une formulation équivalente de (PL).

$$(\mathrm{PL}') = \begin{cases} \min & \boldsymbol{c}_1^\top \boldsymbol{x}_1 + \boldsymbol{c}_2^\top \boldsymbol{x}_2 & & \\ \text{t.q.} & D_1 \boldsymbol{x}_1 + D_2 \boldsymbol{x}_2 = \boldsymbol{b}_0, & & (1.23) \\ & \boldsymbol{x}_1 \in \mathcal{P}_1, & & (1.24) \\ & \boldsymbol{x}_2 \in \mathcal{P}_2. & & (1.25) \end{cases}$$

Le théorème de Minkowski-Weyl nous permet de représenter les polyèdres $\mathcal{P}_1$ et $\mathcal{P}_2$ sous la forme d'une combinaison convexe de leurs points extrêmes et de leurs rayons extrêmes. Nous supposerons que $\mathcal{P}_1$ et $\mathcal{P}_2$ sont réalisables et bornés afin de pouvoir négliger les rayons extrêmes. Soit $\mathcal{G}_i$ l'ensemble des points extrêmes de $\mathcal{P}_i$ et $\boldsymbol{x}_i^g$ la valeur des variables $\boldsymbol{x}_i$ en $g \in \mathcal{G}^i$. Nous pouvons maintenant définir la formulation étendue.

$$(\mathrm{MP}) = \begin{cases} \min & \displaystyle\sum_{i=1,2} \boldsymbol{c}_i^\top \left(\sum_{g \in \mathcal{G}_i} \lambda_i^g \boldsymbol{x}_i^g \right) & & \\ \text{t.q.} & \displaystyle\sum_{i=1,2} D_i \left(\sum_{g \in \mathcal{G}_i} \lambda_i^g \boldsymbol{x}_i^g \right) = \boldsymbol{b}_0, & & (1.26) \\ & \displaystyle\sum_{g \in \mathcal{G}_i} \lambda_i^g = 1, & i = 1, 2, & (1.27) \\ & \boldsymbol{\lambda}_i \geq \boldsymbol{0}, & i = 1, 2. & (1.28) \end{cases}$$

Ce dernier problème est le problème maître de la reformulation. Lorsque les variables originales $\boldsymbol{x}_i$ sont entières ou mixtes, la qualité de la relaxation continue de la formulation (MP) issue de la décomposition de Dantzig-Wolfe est au moins aussi bonne que la relaxation continue de la formulation originale (PL).

1.1.4.2 Génération de colonnes

La formulation (MP) obtenue dans la section précédente décrit les polyèdres $\mathcal{P}_1$ et $\mathcal{P}_2$ en fonction de leurs points extrêmes, au lieu de les décrire par leurs contraintes (1.20)

et (1.21). Cependant, le nombre de points extrêmes d'un polyèdre est exponentiel en son nombre de facettes. Par conséquent, la formulation étendue nécessite, en général, un nombre exponentiel de variables afin d'être complètement définie. Pour palier ce problème, la formulation étendue peut être résolue par génération de colonnes.

En génération de colonnes, seul un sous-ensemble de points extrêmes est utilisé pour définir le problème maître. Comme cette formulation est incomplète, on parle de problème maître restreint (RMP). Étant donné un sous-ensemble de colonnes $\bar{\mathcal{G}}_i \subseteq \mathcal{G}_i$, pour $i = 1, 2$, le problème maître restreint peut être formulé comme suit.

$$
(\text{RMP}) = \begin{cases}
\min & \displaystyle\sum_{i=1,2} \boldsymbol{c}_i^\top \left(\sum_{\bar{\mathcal{G}}_i} \lambda_i^g \boldsymbol{x}_i^g \right) & & & \\
\text{t.q.} & \displaystyle\sum_{i=1,2} D_i \left(\sum_{\bar{\mathcal{G}}_i} \lambda_i^g \boldsymbol{x}_i^g \right) = \boldsymbol{b}_0, & & (\boldsymbol{\pi}) & (1.29) \\
& \displaystyle\sum_{\bar{\mathcal{G}}_i} \lambda_i^g = 1, & i = 1, 2, & (\boldsymbol{\alpha}) & (1.30) \\
& \boldsymbol{\lambda}_i \geq \boldsymbol{0}, & i = 1, 2. & & (1.31)
\end{cases}
$$

L'algorithme de génération de colonnes consiste alors à ajouter des colonnes dans $\bar{\mathcal{G}}_1$ et $\bar{\mathcal{G}}_2$ jusqu'à prouver que la solution optimale courante de (RMP) est également une solution optimale de (MP). Pour ce faire, la méthode la plus répandue s'appuie sur l'algorithme du simplexe et la notion de coûts réduits. Pour rappel, une base B est optimale pour un programme linéaire si elle induit un coût réduit $c_j - \boldsymbol{c}_{\mathcal{B}}^\top B^{-1} A_j$ positif ou nul pour toute variable x_j. Dans le cas de (RMP), le coût réduit d'une colonne $g \in \mathcal{G}_i$ est donné par $\boldsymbol{c}_i^\top \boldsymbol{x}_i^g - \boldsymbol{\pi}_i (D_i \boldsymbol{x}_i^g) - \alpha_i$ avec $\boldsymbol{\pi}_i$ le vecteur de variables duales associées à (1.29) et α_i la variable duale associée à (1.30). Nous pouvons désormais définir le sous-problème (SP_i) consistant à trouver le point extrême du polyèdre $\mathcal{P}_i$ de coût réduit minimum.

$$
(\text{SP}_i) = \begin{cases}
-\alpha + \min\limits_{\boldsymbol{x}_i} & \boldsymbol{c}_i^\top \boldsymbol{x}_i - \left(\boldsymbol{c}_{\mathcal{B}}^\top B^{-1} D_i \right) \boldsymbol{x}_i \\
\text{t.q.} & F_i \boldsymbol{x}_i = \boldsymbol{b}_i, & (1.32) \\
& \boldsymbol{x}_i \geq 0. & (1.33)
\end{cases}
$$

Soit g^* la colonne de coût réduit minimum, tous sous-problèmes confondus. Si g^* a un coût réduit négatif, alors il est possible d'effectuer une opération de pivot de l'algorithme du simplexe en ajoutant g^* au problème restreint (RMP). Sinon, la base courante B est optimale pour le problème maître complet (MP), et l'algorithme de génération de colonnes est terminé. Notons qu'il est souvent possible de calculer une borne duale de (MP) à partir des solutions de (RMP) et (SP_i). Par exemple, soit $\bar{z}$ la valeur d'une solution optimale de (RMP) et soit $\bar{c}^*$ le coût réduit de g^*. Étant donné τ une borne telle que

$$
\tau \geq \sum_{i \in \{1,2\}} \sum_{g \in \mathcal{G}_i} \lambda_i^g,
$$

alors la valeur

$$
\hat{z} = \bar{z} + \tau \bar{c}^*
$$

est une borne duale valide pour le problème maître (MP). Dans le cas présent, les contraintes (1.30) impliquent $\tau \geq 2$.

Lorsque la décomposition de Dantzig-Wolfe est appliquée à un problème linéaire en nombres entiers, la procédure de génération de colonnes est combinée à une méthode de branch-and-bound. Les décisions de branchement sont alors considérées comme des contraintes linéaires, généralement ajoutées au problème maître ou aux sous-problèmes. L'algorithme résultant de cette combinaison appartient à la famille des méthodes de *branch-and-price*. Les méthodes de branch-and-bound seront abordées dans une section ultérieure.

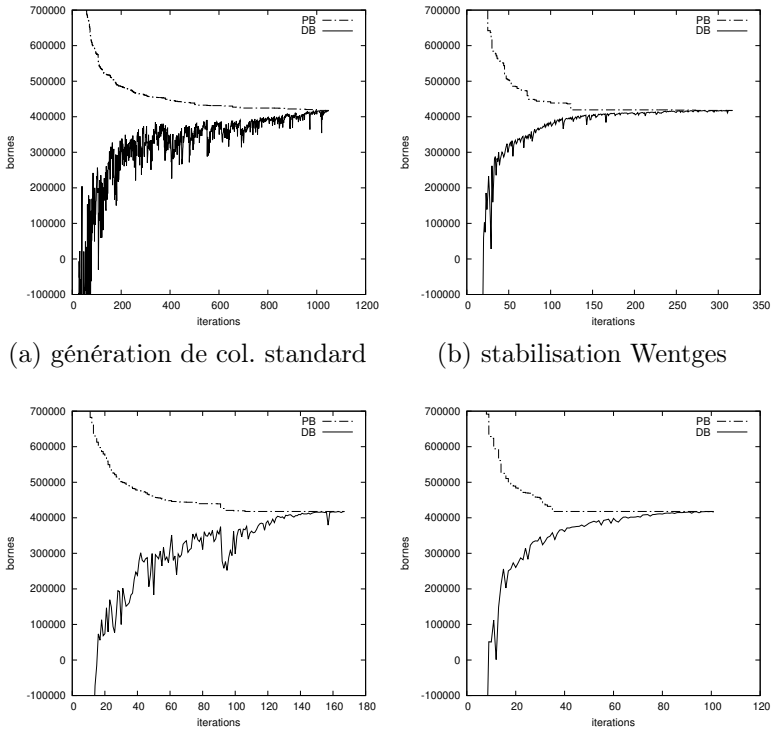
1.1.4.3 Réduction de la dégénérescence

Dans les sections précédentes, nous avons présenté la décomposition de Dantzig-Wolfe et sa résolution par génération de colonnes. Cette génération de colonnes est généralement basée sur l'algorithme du simplexe, ce qui signifie qu'à chaque itération, nous disposons d'une solution primale basique (ou de manière équivalente, d'une solution duale basique) nous permettant de définir la fonction objectif du ou des sous-problèmes. En résolvant ces sous-problèmes, nous obtenons la colonne de coût réduit minimum, qui est ensuite utilisée pour effectuer une opération de pivot de l'algorithme du simplexe.

Les méthodes de génération de colonnes sont connues pour rencontrer des problèmes de convergence (*tailing-off effect*), comme le montre la figure 1.1a. Il existe plusieurs manières d'expliquer cette difficulté à converger en génération de colonnes, et diverses techniques permettant d'y remédier.

Nous parlons d'instabilité duale lorsque les bornes duales (telles que $\hat{z}$ définie dans la section précédente) obtenues à chaque itération ont tendance à osciller violemment. En effet, même si chaque itération ne consiste finalement qu'à échanger une variable en base et une variable hors-base, cela peut avoir des conséquences importantes sur la valeur des coûts réduits des colonnes hors-base, et donc sur la valeur de la borne duale qui en résulte. Une technique classique pour amoindrir cet effet consiste à utiliser une solution duale *lissée*. Ainsi, à chaque itération k, au lieu de résoudre les sous-problèmes (SP_i) en utilisant directement la solution duale $\boldsymbol{\pi}^k = \boldsymbol{c}_{B_{(k)}}^\top B_{(k)}^{-1}$ correspondant à la base optimale du problème maître restreint, nous utilisons la solution lissée $\bar{\boldsymbol{\pi}}^k = \theta\boldsymbol{\pi}^k + (1-\theta)\bar{\boldsymbol{\pi}}^{k-1}$ avec $\theta \in [0;1]$. Dans la suite de ce document, nous ferons généralement référence à cette technique en tant que *stabilisation de Wentges*, du nom de l'auteur de l'article [Wen97]. Les figures 1.1b et 1.1d montrent une diminution nette de l'instabilité duale et du nombre d'itérations lorsque nous utilisons cette technique par rapport aux figures 1.1a et 1.1c. La difficulté principale de cette méthode consiste cependant à déterminer la valeur du paramètre θ (voir la méthode proposée dans [PSUV17], par exemple).

D'un point de vue primal, l'instabilité s'exprime sous la forme de pivots dégénérés, c'est-à-dire des itérations au cours desquelles la valeur optimale du problème maître restreint n'évolue pas. En conséquence, la courbe correspondant à l'évolution de la borne primale prend une forme « d'escalier », dont les marches sont généralement très basses, comme dans les figures 1.1a et 1.1b. De longues marches indiquent un nombre significatif de pivots dégénérés consécutifs. Chaque itération de génération de colonne consiste normalement en une seule opération de pivot de l'algorithme du simplexe. Cependant, lorsque la décomposition de Dantzig-Wolfe génère plusieurs sous-problèmes indépendants, il est



(a) génération de col. standard

(b) stabilisation Wentges

(c) pricing multiple

(d) stabilisation Wentges + pricing multiple

FIGURE 1.1 – Exemple d'instabilité entres les bornes primale (PB) et duale (DB) pendant l'algorithme de génération de colonnes

facile d'ajouter la meilleure colonne de chaque sous-problème dans le problème maître restreint en même temps. Plusieurs pivots peuvent être ensuite nécessaires afin d'obtenir la solution optimale du nouveau problème maître restreint. Ces colonnes offrent généralement moins d'informations individuellement, car elles sont toutes issues de la même solution duale, mais elles permettent d'exploiter davantage le temps investi dans la résolution des sous-problèmes, si celui-ci s'avère être important. Les figures 1.1c et 1.1d montrent que cette technique « adoucit » la courbe correspondant à la borne primale, sans avoir toutefois d'effet important sur l'instabilité duale, contrairement à la stabilisation de Wentges.

Une autre technique permettant de diminuer le nombre d'itérations dégénérées, que nous étudierons plus en détail dans une section ultérieure, consiste à agréger dynamiquement les contraintes du problème maître [EVSD05]. Elle permet de réduire ponctuellement le nombre de contraintes du problème maître, et donc la taille de la base, afin de se rapprocher du nombre de variables ayant une valeur strictement positive dans la solution basique courante. De ce fait, un ensemble de bases dégénérées peut être parcouru en une seule opération de pivot.

Les articles [DL05, Van05, VW10] offrent une présentation plus approfondie de la décomposition de Dantzig-Wolfe et de sa résolution par branch-and-price.

1.1.5 Méthodes de résolution

Bien que les problèmes d'optimisation linéaire appartiennent à la classe de complexité P lorsque les variables de décision sont continues, la plupart des problèmes linéaires en nombres entiers sont NP-complets. Selon la conjecture $\mathsf{P} \neq \mathsf{NP}$, il n'existe pas d'algorithme capable de les résoudre avec une complexité polynomiale au pire cas.

Dans les sections précédentes, nous avons présenté l'algorithme du simplexe primal et les méthodes de sous-gradients qui permettent de résoudre des problèmes de programmation mathématique en variables continues. Dans cette section, nous rappelons les principes de deux méthodes exactes permettant la résolution de programmes linéaires en nombres entier en temps exponentiel ou pseudo-polynomial.

1.1.5.1 Branch-and-bound

Pour résoudre un programme linéaire en nombres entiers, les méthodes de branch-and-bound (séparation et évaluation) utilisent une approche « diviser pour régner ». Au lieu de résoudre directement le problème original, ces méthodes génèrent un ensemble de sous-problèmes plus petits. La solution optimale du problème original est alors donnée par la meilleure solution parmi tous les sous-problèmes.

Étant donné $\mathcal{P} \subseteq \mathbb{R}^n$ un polyèdre, considérons le problème en nombres entiers générique défini par la formulation (MIP) suivante.

$$(\text{MIP}) = \min_{\boldsymbol{x}} \left\{ \boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \mathcal{P} \cap \mathbb{Z}^n \right\}$$

Si nous pouvons diviser l'ensemble de solutions $\mathcal{P}$ en une collection de k sous-ensembles telle que $\mathcal{P}_1 \cup \mathcal{P}_2 \cup \cdots \cup \mathcal{P}_k = \mathcal{P}$ et $\mathcal{P}_i \neq \mathcal{P}\ \forall i$, alors le problème peut être décomposé comme suit (on parle dans ce cas de branchement, ou séparation).

$$(\text{MIP}') = \min_{i=1,\dots,k} \left\{ \min_{\boldsymbol{x}} \left\{ \boldsymbol{c}^\top \boldsymbol{x} : \boldsymbol{x} \in \mathcal{P}_i \cap \mathbb{Z}^n \right\} \right\}$$

Chaque sous-problème associé à un sous-ensemble $\mathcal{P}_i$ est ensuite considéré indépendamment des autres, et peut à son tour être divisé en sous-sous-problèmes de manière récursive. Le but de ces méthodes est de générer des sous-problèmes pouvant être associés à des bornes plus fortes que le problème original (évaluation). Ainsi, un sous-problème peut être élagué, c'est-à-dire retiré, lorsque sa borne indique qu'il ne contient pas la solution optimale de (MIP). Autrement dit, étant donné $\check{z}$ une borne supérieure pour la valeur optimale z^* du problème (MIP), et $\hat{z}_i$ une borne inférieure pour le sous-problème i correspondant à $\mathcal{P}_i$, le sous-problème i peut être élagué si $\hat{z}_i \geq \check{z}$.

L'origine des méthodes de branch-and-bound est attribuée à [LD60]. Les solveurs génériques de programmation linéaire en nombres entiers utilisent généralement un algorithme de branch-and-bound dans lequel chaque sous-problème est évalué par relaxation continue. Les décisions de branchement sont ensuite prises en fonction des variables dont la valeur ainsi obtenue est fractionnaire. Dans ce document, nous utilisons régulièrement des méthodes de branchement basées sur la relaxation Lagrangienne.

1.1.5.2 Programmation dynamique

Tout comme les méthodes de branch-and-bound présentées dans la section précédente, la programmation dynamique aborde la résolution d'un problème en le décomposant en sous-problèmes. La structure des sous-problèmes est ici déterminée par une règle de récurrence. Cette règle de récurrence associe les sous-problèmes à des *états* du programme dynamique, un état étant défini par un ensemble de paramètres. La force d'un programme dynamique repose sur la réutilisation d'un même sous-problème, résolu une seule fois, dans plusieurs états du programme. Cette méthode a été proposée pour la première fois dans [Bel54].

Un exemple classique de programme dynamique est celui permettant la résolution du problème de sac à dos en temps pseudo-polynomial.

Problème 1 (sac à dos 1D (KP))**.** Étant donné deux vecteurs $\boldsymbol{p}, \boldsymbol{w} \in \mathbb{N}^n$ et un entier W, trouver un vecteur $\boldsymbol{x} \in \{0,1\}^n$ tel que $\boldsymbol{w}^\top \boldsymbol{x} \leq W$ et qui maximise $\boldsymbol{p}^\top \boldsymbol{x}$.

Le programme dynamique permettant de résoudre le problème de sac à dos peut être défini comme suit. Étant donné $k \in \{1,\dots,n\}$ et $w \in \{0,1,\dots,W\}$, le label correspondant à l'état $\langle k, w\rangle$ du programme dynamique est calculé via la formule de récurrence suivante.

$$\psi^{\text{KP}}(k,w) = \begin{cases} \max\left\{ p_k + \psi^{\text{KP}}(k-1, w-w_k), \right. & \\ \qquad \left. \psi^{\text{KP}}(k-1,w) \right\} & \text{si } w_k \leq w, \\ \psi^{\text{KP}}(k-1,w) & \text{sinon.} \end{cases} \tag{1.34}$$

Pour plus de rigueur, nous pouvons considérer que $\psi^{\text{KP}}(0,w) = 0$ pour tout $w \in \{0,1,\dots,W\}$.

Ainsi, un état est défini par un couple (k,w) et est associé au sous-ensemble d'objets de profit maximum, ce sous-ensemble étant limité aux k premiers objets et ne devant pas

excéder la capacité w. Chaque label $\psi^{\mathrm{KP}}(k, w)$, calculé une seule fois, est utilisé dans le calcul des deux labels $\psi^{\mathrm{KP}}(k+1, w)$ et $\psi^{\mathrm{KP}}(k+1, w+w_k)$. Enfin, la solution optimale du problème original est associée à $\psi^{\mathrm{KP}}(n, W)$.

Un programme dynamique peut notamment être interprété comme la recherche d'un chemin optimal dans un graphe. Par exemple, le programme dynamique défini ci-dessus équivaut à la recherche d'un plus long chemin dans un graphe acyclique dont les sommets correspondent aux états $\langle k, w\rangle$. Ce graphe est également défini par des arcs $\big(\langle k-1, w\rangle, \langle k, w\rangle\big), \forall k \in \{1, \ldots, n\}, \forall w = \{0, 1, \ldots, W\}$ de coût nul, et des arcs $\big(\langle k-1, w-w_k\rangle, \langle k, w\rangle\big), \forall k \in \{1, \ldots, n\}, \forall w = \{w_k, \ldots, W\}$ de coût p_k. Une solution optimale du problème de sac à dos correspond à un plus long chemin arrivant en $\langle n, W\rangle$.

Les programmes dynamiques offrent souvent une approche directe pour concevoir des algorithmes de complexité pseudo-polynomiale (c'est-à-dire polynomiale en la valeur numérique de l'entrée). Un problème NP-complet pouvant être résolu par un algorithme en temps pseudo-polynomial est dit *faiblement* NP*-complet*. Par opposition, si un problème n'admet pas d'algorithme pseudo-polynomial, il est dit *fortement* NP*-complet*.

Les méthodes de branch-and-bound et de programmation dynamique sont présentées dans la plupart des livres dédiés à la programmation linéaire en nombres entiers (voir par exemple [BT97] et [Wol98]). La programmation dynamique dispose également d'une monographie récente en deux volumes [Ber17, Ber12].

1.2 Agrégation dynamique de contraintes dans le cadre de l'algorithme du simplexe primal

Dans la sous-sous-section 1.1.4.3, nous avons abordé le problème de la convergence dans les méthodes de génération de colonnes et présenté deux techniques génériques permettant d'accélérer cette convergence.

En particulier, la technique de lissage de la solution duale de Wentges appartient à la famille des méthodes de stabilisation. L'objectif de ces méthodes consiste à privilégier les solutions duales situées à proximité d'un « centre de stabilité » (*stability center*), c'est-à-dire proches d'une solution duale que l'on suppose être de bonne qualité. Ainsi, la méthode de Wentges [Wen97, PSUV17] utilise le centre de stabilisation pour lisser la solution duale courante. D'autres approches consistent à altérer explicitement le problème dual pour tenir compte du centre de stabilisation. Cela peut être appliqué en réduisant l'espace des solutions duales réalisables à un hypercube autour du centre (*boxstep method*), ou bien en définissant une *trust region* et en pénalisant la distance entre celle-ci et la solution duale courante, généralement à l'aide d'une fonction convexe linéaire par morceaux, ou bien d'une fonction quadratique (voir [BLM+08]).

Dans cette section, nous nous intéressons à une autre famille de méthodes dont l'objectif est de contrer le phénomène de dégénérescence en diminuant la taille de la base utilisée dans l'algorithme du simplexe. Plus spécifiquement, nous comparons trois de ces approches et présentons leur structure commune.

1.2.1 Structure générale

Étant donné $m, n \in \mathbb{N}$, soit $A \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ et $\boldsymbol{c} \in \mathbb{R}^n$. Soit également $\boldsymbol{x} \in \mathbb{R}^n_+$ un vecteur de variables continues positives ou nulles. Considérons le programme linéaire générique suivant.

$$(\text{PL}) = \begin{cases} \min \quad \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{t.q.} \quad A\boldsymbol{x} = \boldsymbol{b}, & (1.35) \\ \qquad \boldsymbol{x} \geq \boldsymbol{0}. & (1.36) \end{cases}$$

Étant donné une solution basique $\boldsymbol{x}^0$, que l'on suppose réalisable pour (PL), il est possible de partitionner l'ensemble des variables de $\boldsymbol{x}$ en deux. Soit $\mathcal{F}$ l'ensemble des indices des variables dont la valeur dans $\boldsymbol{x}^0$ n'est **pas** égale à zéro. On dit généralement que ce sont des variables *libres*. L'ensemble complémentaire est alors noté $\mathcal{N}$; c'est l'ensemble des indices des variables dont la valeur est égale à zéro. Nous considérons donc ici que $\boldsymbol{x}_\mathcal{F} > \boldsymbol{0}$ et $\boldsymbol{x}_\mathcal{N} = \boldsymbol{0}$, mais nous pourrions généraliser au cas où chaque variable est limitée par une borne inférieure et une borne supérieure arbitraires.

Soit $B \in \mathbb{R}^{m \times m}$ une base de (PL) telle que $\boldsymbol{x}^0 = B^{-1}b$. Si $|\mathcal{F}| = m$, alors B est unique et composée exclusivement des colonnes associées aux variables strictement positives de $\boldsymbol{x}^0$, donc $B = A_\mathcal{F}$. En revanche, si $f = |\mathcal{F}| < m$, alors B est composée des colonnes $A_\mathcal{F}$ ainsi que de $m - f$ colonnes supplémentaires sélectionnées parmi les colonnes de $A_\mathcal{N}$. Il peut alors y avoir jusqu'à $\binom{|\mathcal{N}|}{(m-f)}$ manières de construire B associées à la même solution $\boldsymbol{x}^0$.

Considérons le second cas, lorsque B est une base dégénérée. Soit $\mathcal{N}_1 \subseteq \mathcal{N}$ l'ensemble des indices des $m - f$ colonnes utilisées pour compléter B, et soit $\mathcal{N}_2 = \mathcal{N} \setminus \mathcal{N}_1$. Nous pouvons ré-écrire (PL) de la façon suivante, en fonction des indices $\mathcal{F}, \mathcal{N}_1\mathcal{N}_2$.

$$(\text{PL}) = \begin{cases} \min \quad \boldsymbol{c}_\mathcal{F}^\top \boldsymbol{x}_\mathcal{F} + \boldsymbol{c}_{\mathcal{N}_1}^\top \boldsymbol{x}_{\mathcal{N}_1} + \boldsymbol{c}_{\mathcal{N}_2}^\top \boldsymbol{x}_{\mathcal{N}_2} & \\ \text{t.q.} \quad A_\mathcal{F}\boldsymbol{x}_\mathcal{F} + A_{\mathcal{N}_1}\boldsymbol{x}_{\mathcal{N}_1} + A_{\mathcal{N}_2}\boldsymbol{x}_{\mathcal{N}_2} = \boldsymbol{b}, & (1.37) \\ \qquad \boldsymbol{x}_\mathcal{F} \geq \boldsymbol{0}, & (1.38) \\ \qquad \boldsymbol{x}_{\mathcal{N}_1} \geq \boldsymbol{0}, & (1.39) \\ \qquad \boldsymbol{x}_{\mathcal{N}_2} \geq \boldsymbol{0}. & (1.40) \end{cases}$$

Les colonnes de $A_{\mathcal{N}_1}$ sont généralement considérées comme étant la cause des itérations dégénérées dans l'algorithme du simplexe. Premièrement, nous avons $\boldsymbol{x}_{\mathcal{N}_1} = \boldsymbol{0}$. Cela signifie que pour effectuer un pivot avec une variable entrante x_j, il faut que $\boldsymbol{h}_{\mathcal{N}_1}$ la direction de modification de $\boldsymbol{x}_{\mathcal{N}_1}$ par le pivot soit $\boldsymbol{h}_{\mathcal{N}_1} \geq \boldsymbol{0}$. Sinon, il existe $i \in \mathcal{N}_1$ tel que $h_i < 0$, et le pivot doit utiliser une longueur de pas $\gamma = 0$ pour ne pas violer les contraintes $(\boldsymbol{x} + \gamma\boldsymbol{h}) \geq \boldsymbol{0}$; l'itération ainsi effectuée est dite *dégénérée* (la solution basique est toujours $\boldsymbol{x}^0$). Deuxièmement, la solution duale basique $\boldsymbol{\pi} = \boldsymbol{c}_\mathcal{B}^\top B^{-1}$ est définie par la base B toute entière, y compris les colonnes associées aux variables basiques dégénérées $\boldsymbol{x}_{\mathcal{N}_1}$. Autrement dit, les colonnes composant $A_{\mathcal{N}_1}$, utilisées pour compléter la base B, introduisent un biais dans la valeur de la solution duale basique $\boldsymbol{\pi}$, et donc dans le pricing de la prochaine variable entrante.

En général, l'algorithme du simplexe primal parcourt plusieurs bases dégénérées avant de trouver une nouvelle solution $\boldsymbol{x}^1$ strictement meilleure que $\boldsymbol{x}^0$. Si les opérations de

pivot sont mal choisies, l'algorithme peut également se retrouver coincé dans un cycle de bases dégénérées.

L'idée générale des méthodes proposées dans [Pan98, EVSD05, EMDS11] repose sur la réduction de la base dégénérée B afin qu'elle soit uniquement définie par les variables libres, sans biais induit par l'utilisation de variables basiques dégénérées. Par la suite, nous ferons référence à ces méthodes sous le nom de BDAS (*Basis-Deficiency-Allowing Simplex* [Pan98]), DCA (*Dynamic Constraint Aggregation* [EVSD05]) et IPS (*Improved Primal Simplex* [EMDS11]).

Nous pouvons considérer qu'un point essentiel de ces trois méthodes est qu'une variable x_j induira un pivot non-dégénéré dans l'algorithme du simplexe si et seulement si la colonne correspondante A_j peut être obtenue par une combinaison linéaire des colonnes de $A_\mathcal{F}$. De manière équivalente, A_j doit appartenir au sous-espace $V(A_\mathcal{F})$ engendré par $A_\mathcal{F}$. Dans BDAS, une telle colonne induit une *full iteration*, car elle n'augmente pas la taille de la base courante. Dans les méthodes DCA et IPS, c'est la notion de *compatibilité* qui incarne la capacité d'une colonne à induire un pivot non-dégénéré. Cette notion est fondamentale pour ces deux méthodes.

Par ailleurs, la réduction appliquée à la base est généralement dynamique, c'est-à-dire qu'elle peut être ré-adaptée aux besoins de l'algorithme au cours de la résolution, par exemple pour garantir la convergence vers une solution optimale du problème original. Ainsi, lorsque $A_j \notin V(A_\mathcal{F})$, la méthode BDAS augmente la taille de la base courante de 1 (*rank-increasing iteration*) afin de pouvoir effectuer une opération de pivot. De la même manière, DCA est capable d'intégrer des colonnes *incompatibles* par désagrégation, ce qui revient également à augmenter la taille de la base. Il est aussi possible de réduire davantage la taille de la base, si la base réduite est elle-même dégénérée.

1.2.2 Basis-Deficiency-Allowing Simplex [Pan98]

La méthode BDAS est basée sur une décomposition QR de la matrice $A_\mathcal{F}$ visant à éviter l'utilisation de variables basiques dégénérées par réduction de la taille de la base. Étant donné un programme (PL) et B une base initiale (réalisable) pour ce programme, soit $\mathcal{F}$ l'ensemble des indices des variables basiques ayant une valeur strictement positive dans la solution basique associée à B. Puisque $\boldsymbol{b} \in V(A_\mathcal{F})$, alors il existe $Q \in \mathbb{R}^{m\times m}$ une matrice orthogonale permettant de générer un système de contraintes équivalent à celui définissant (PL) de la manière suivante.

Soit $\bar{A}_\mathcal{F} \in \mathbb{R}^{f\times f}$ une matrice triangulaire supérieure, $\bar{A}^1_\mathcal{N} \in \mathbb{R}^{f\times|\mathcal{N}|}$ et $\bar{A}^2_\mathcal{N} \in \mathbb{R}^{(m-f)\times|\mathcal{N}|}$, et $\bar{b} \in \mathbb{R}^f$ tels que

$$[A_\mathcal{F}\boldsymbol{x}_\mathcal{F} + A_\mathcal{N}\boldsymbol{x}_\mathcal{N} = \boldsymbol{b}] \quad \Leftrightarrow \quad [QA_\mathcal{F}\boldsymbol{x}_\mathcal{F} + QA_\mathcal{N}\boldsymbol{x}_\mathcal{N} = Q\boldsymbol{b}] \quad = \quad \left[\begin{pmatrix}\bar{A}_\mathcal{F}\\ 0\end{pmatrix}\boldsymbol{x}_\mathcal{F} + \begin{pmatrix}\bar{A}^1_\mathcal{N}\\ \bar{A}^2_\mathcal{N}\end{pmatrix}\boldsymbol{x}_\mathcal{N} = \begin{pmatrix}\bar{\boldsymbol{b}}\\ 0\end{pmatrix}\right].$$

De cette manière, nous pouvons construire un programme linéaire (PL$'$) équivalent à (PL).

$$(\text{PL}') = \begin{cases} \min \quad \boldsymbol{c}_{\mathcal{F}}^{\top}\boldsymbol{x}_{\mathcal{F}} + \boldsymbol{c}_{\mathcal{N}}^{\top}\boldsymbol{x}_{\mathcal{N}} & \\ \text{t.q.} \quad \bar{A}_{\mathcal{F}}\boldsymbol{x}_{\mathcal{F}} + \bar{A}_{\mathcal{N}}^{1}\boldsymbol{x}_{\mathcal{N}} = \bar{\boldsymbol{b}}, & (1.41) \\ \qquad\qquad \bar{A}_{\mathcal{N}}^{2}\boldsymbol{x}_{\mathcal{N}} = \boldsymbol{0}, & (1.42) \\ \qquad \boldsymbol{x}_{\mathcal{F}}, \boldsymbol{x}_{\mathcal{N}} \geq \boldsymbol{0}. & (1.43) \end{cases}$$

Les solutions basiques de ce nouveau système sont données par $\bar{\boldsymbol{x}} = \begin{pmatrix} \bar{A}_{\mathcal{F}}^{-1}\bar{\boldsymbol{b}} \\ \boldsymbol{0} \end{pmatrix}$ et $\bar{\boldsymbol{\pi}} = \begin{pmatrix} \bar{A}_{\mathcal{F}}^{-\top}\boldsymbol{c}_{\mathcal{F}} \\ \boldsymbol{0} \end{pmatrix}$. Il est donc possible de ne considérer que la base réduite $\bar{A}_{\mathcal{F}} \in \mathbb{R}^{f\times f}$ au lieu de la base originale $B \in \mathbb{R}^{m\times m}$. Ainsi, plus la base originale B est dégénérée, et plus la base réduite $\bar{A}_{\mathcal{F}}$ est petite. Notons que la solution basique $\bar{\boldsymbol{x}}$ est également la solution basique de (PL) correspondant à la base B, tandis que $\bar{\boldsymbol{\pi}}$ est une solution duale basique de (PL) transformée par la matrice orthogonale Q.

Dans [Pan98], l'auteur propose ainsi plusieurs manières d'adapter les algorithmes du simplexe primal et dual à l'utilisation de bases réduites. Cependant, l'article ne présente pas d'élément théorique permettant de penser que ces nouveaux algorithmes favorisent la proportion de pivots non-dégénérés. Si les résultats expérimentaux présentés montrent une diminution nette du nombre d'itérations dégénérées (diminution de 27% en moyenne), ceux-ci sont compliqués à interpréter de par l'utilisation d'une heuristique pour déterminer une base B initiale. Bien que les algorithmes proposés aient de bonnes performances en pratique, il est difficile de déterminer l'implication de la méthode BDAS dans la diminution observée de la dégénérescence.

Plusieurs méthodes, qui ne font pas nécessairement appel à la décomposition QR, ont été développées autour de cette idée de base réduite. Pour un aperçu plus détaillés de ces méthodes et de leurs implémentations, voir [Pan14], chapitre 20.

Nous allons à présent nous intéresser aux méthodes DCA et IPS, qui, contrairement à la méthode BDAS, cherchent explicitement à favoriser le pricing de variables compatibles (*i.e.* appartenant à $V(A_{\mathcal{F}})$). Cela se fait par le biais de l'opération de *désagrégation des variables duales* dans DCA, et la résolution du *problème complémentaire* dans IPS.

1.2.3 Dynamic Constraint Aggregation [EVSD05]

L'agrégation dynamique de contraintes (DCA) est une méthode spécialisée dans la résolution de problèmes de *set-partitioning* dans un contexte de génération de colonnes.

Problème 2 (set-partitioning)**.** Étant donné V un ensemble d'éléments à couvrir, $\mathcal{S} \subseteq \{V_i : V_i \subseteq V\}$ une collection de parties et un coût c_i pour tout $i : V_i \in \mathcal{S}$, trouver une partition $P \subseteq \mathcal{S} : \bigcup(V_i \in P) = V$ telle que la somme $\sum_{i:V_i\in P} c_i$ est minimisée.

Étant donné $V_j \in \mathcal{S}$, soit $a_{i,j} = 1$ si $i \in V_j$, et $a_{i,j} = 0$ sinon, et soit $\boldsymbol{x} \in \{0,1\}^{|\mathcal{S}|}$ un vecteur de variables de décision binaires. Il est possible de formuler le problème de *set-partitioning* par le programme maître suivant.

$$(\text{MP}) = \begin{cases} \min \displaystyle\sum_{i:V_i \in \mathcal{S}} c_i x_i & & & \\ \text{t.q.} \displaystyle\sum_{j:V_j \in \mathcal{S}} a_{i,j} x_j = 1 & \forall i \in V, & (\boldsymbol{\phi}) & (1.44) \\ x_i \in \{0,1\} & \forall i : V_i \in \mathcal{S}. & & (1.45) \end{cases}$$

Dans la formulation ci-dessus, la variable binaire x_i vaut 1 si la solution utilise V_i, et $x_i = 0$ sinon. Le vecteur de variables duales $\boldsymbol{\phi}$ est associé aux contraintes de partitionnement. Cette formulation peut être utilisée comme problème maître dans un algorithme de génération de colonnes.

Étant donné une partition initiale $\mathcal{A}$ des sommets du graphe, DCA défini une colonne compatible V_j telle que, pour toute colonne V_i de la solution $\mathcal{A}$, soit V_j couvre tous les sommets couverts par V_i, soit elle n'en couvre aucun. On peut donc définir l'ensemble des colonnes compatibles $\mathcal{C}$ de la manière suivante.

$$\mathcal{C} = \{V_j : V_j \in \mathcal{S}, (V_j \cap V_i) \in \{\emptyset, V_i\} \quad \forall V_i \in \mathcal{A}\}$$

De cette manière, on regroupe plusieurs sommets de V en *clusters*, ou *agrégats*. À présent, nous pouvons définir le problème maître agrégé (AMP) comme étant équivalent au problème maître (MP) limité à l'ensemble des colonnes compatibles. Nous pouvons remarquer que, pour un cluster $V_i \in \mathcal{A}$ donné, les contraintes de partitionnement (1.44) associées à tout $j \in V_i$ sont identiques dans le problème agrégé. Ainsi, (AMP) peut être complètement défini en utilisant une seule contrainte de partitionnement par cluster. Étant donné $V_i \in \mathcal{A}$ et $V_j \in \mathcal{S}$, soit $\bar{a}_{i,j} = 1$ si $V_i \subseteq V_j$, et $\bar{a}_{i,j} = 0$ sinon. Nous pouvons définir (AMP) par le programme mathématique suivant.

$$(\text{AMP}) = \begin{cases} \min \displaystyle\sum_{i:V_i \in \mathcal{C}} c_i x_i & & & \\ \text{t.q.} \displaystyle\sum_{j:V_j \in \mathcal{C}} \bar{a}_{i,j} x_j = 1 & \forall i : V_i \in \mathcal{A}, & (\boldsymbol{\Phi}) & (1.46) \\ x_i \in \{0,1\} & \forall i : V_i \in \mathcal{C}. & & (1.47) \end{cases}$$

Ce nouveau programme possède $|\mathcal{A}|$ contraintes (1.46) seulement, et a donc généralement une base plus petite que (MP). Cela a un impact direct sur la solution duale, qui est désormais définie par le vecteur de variables agrégées $\boldsymbol{\Phi}$.

Un point important de la méthode DCA est que l'agrégation $\mathcal{A}$ des contraintes qui mène à la formulation (AMP) n'altère pas le sous-problème de pricing. À l'itération courante t, l'algorithme doit ainsi construire une solution duale désagrégée $\boldsymbol{\phi}^{(t)}$ à partir de la solution agrégée $\boldsymbol{\Phi}^{(t)}$ donnée par (AMP). Cette solution désagrégée $\boldsymbol{\phi}^{(t)}$ — qui sera ensuite utilisée dans le sous-problème de pricing — doit garantir que 1) les colonnes actuellement en base ont un coût réduit nul ; et 2) le coût réduit des colonnes incompatibles est maximisé. En effet, puisque les colonnes incompatibles ne peuvent pas être ajoutées à (AMP), il est préférable d'éviter qu'elles ne soient générées, car elles sont plus susceptibles d'induire une opération de pivot dégénérée.

Soit $\mathcal{I} = \mathcal{S} \setminus \mathcal{C}$ l'ensemble des colonnes incompatibles. Désagréger la solution duale $\boldsymbol{\Phi}^{(t)}$ revient à résoudre le problème dual suivant.

$$
(\text{DDP}) = \begin{cases} \max \quad \mu & \\ \text{t.q.} \quad \mu \leq c_j - \sum_{i \in V_j} a_{i,j} \phi_i^{(t)} & \forall j : V_j \in \mathcal{I}, \quad (1.48) \\ \quad \sum_{j \in V_i} \phi_j^{(t)} = \Phi_i^{(t)} & \forall V_i \in \mathcal{A}. \quad (1.49) \\ & (1.50) \end{cases}
$$

En utilisant la solution désagrégée $\boldsymbol{\phi}^{(t)}$, l'algorithme favorise le pricing des colonnes compatibles, et donc de pivots non-dégénérés. On parle de *désagrégation duale* car la valeur des variables agrégées $\boldsymbol{\Phi}$ est « répartie » entre les variables originales $\boldsymbol{\phi}$.

Au cours de l'exécution de l'algorithme, l'agrégation peut être amenée à changer. Par exemple, si les seules colonnes de coût réduit négatif disponibles sont incompatibles, alors l'algorithme doit modifier l'agrégation (généralement en divisant des agrégats existants) afin de pouvoir faire entrer ces colonnes en base. De même, si la base courante est fortement dégénérée, l'algorithme peut être amené à fusionner certains agrégats pour générer une base plus petite.

L'efficacité de la méthode DCA repose également sur le fait que, pour le problème de *crew-scheduling* considéré dans [EVSD05], les colonnes ont une structure très particulière, et peuvent être représentées par des chemins dans le problème de pricing. De ce fait, les auteurs n'ont pas besoin de résoudre (DDP) de manière exacte, car ils disposent d'une heuristique rapide générant des solutions désagrégées de bonne qualité.

Par la suite, la méthode DCA a été déclinée en plusieurs variantes, notamment l'algorithme de *multi-phase dynamic constraint aggregation* [EMSD10]. Cependant, DCA est aujourd'hui plus souvent considérée comme un cas particulier de la méthode IPS [EMDS11, GDL16], comme nous allons le voir dans la section suivante.

1.2.4 Improved Primal Simplex [EMDS11]

La méthode IPS, tout comme la méthode BDAS décrite précédemment, est basée sur une notion de base réduite. Elle est également générique et peut donc être appliquée à n'importe quel type de programme linéaire, contrairement à la méthode DCA qui est spécialisée aux problèmes de *set-partitioning*.

Étant donné $m, n \in \mathbb{N}$, soit $A \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ et $\boldsymbol{c} \in \mathbb{R}^n$. Considérons le programme linéaire générique suivant.

$$
(\text{PL}) = \begin{cases} \min \quad \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{t.q.} \quad A\boldsymbol{x} = \boldsymbol{b}, & (1.51) \\ \quad \boldsymbol{x} \geq \boldsymbol{0}. & (1.52) \end{cases}
$$

Étant donné une solution basique $\boldsymbol{x}^0$ et la sous-matrice $A_{\mathcal{F}}$ correspondant aux colonnes associée aux variables libres dans $\boldsymbol{x}^0$, la méthode IPS génère une base B de (PL) en complétant $A_{\mathcal{F}}$ avec des variables artificielles, correspondant généralement à des variables d'écart. De plus, les contraintes sont partitionnées en deux ensembles $\mathcal{P}$ et $\mathcal{Z}$ afin d'effectuer une division supplémentaire de $A_{\mathcal{F}}$ et obtenir ainsi une base réduite $A_{\mathcal{P}\mathcal{F}} \in \mathcal{R}^{f \times f}$

avec $f = |\mathcal{F}|$ le nombre de variables libres.

$$[A\boldsymbol{x} = \boldsymbol{b}] \quad = \quad [A_{\mathcal{F}}\boldsymbol{x}_{\mathcal{F}} + A_{\mathcal{N}}\boldsymbol{x}_{\mathcal{N}} = \boldsymbol{b}] \quad = \quad \left[\begin{pmatrix} A_{\mathcal{P}\mathcal{F}} \\ A_{\mathcal{Z}\mathcal{F}} \end{pmatrix} \boldsymbol{x}_{\mathcal{F}} + \begin{pmatrix} A_{\mathcal{P}\mathcal{N}} \\ A_{\mathcal{Z}\mathcal{N}} \end{pmatrix} \boldsymbol{x}_{\mathcal{N}} = \begin{pmatrix} \boldsymbol{b}_{\mathcal{P}} \\ \boldsymbol{b}_{\mathcal{Z}} \end{pmatrix}\right].$$

La base B est ensuite générée de sorte à pouvoir être facilement inversée, en sélectionnant les bonnes variables d'écart pour compléter la base.

$$B = \begin{pmatrix} A_{\mathcal{P}\mathcal{F}} & 0 \\ A_{\mathcal{Z}\mathcal{F}} & I \end{pmatrix}, \qquad B^{-1} = \begin{pmatrix} A_{\mathcal{P}\mathcal{F}}^{-1} & 0 \\ -A_{\mathcal{Z}\mathcal{F}}A_{\mathcal{P}\mathcal{F}}^{-1} & I \end{pmatrix}.$$

Soit $\bar{A}_{\mathcal{P}\mathcal{N}} = A_{\mathcal{P}\mathcal{F}}^{-1}A_{\mathcal{P}\mathcal{N}}$ et $\bar{A}_{\mathcal{Z}\mathcal{N}} = A_{\mathcal{Z}\mathcal{N}} - A_{\mathcal{Z}\mathcal{F}}A_{\mathcal{P}\mathcal{F}}^{-1}A_{\mathcal{P}\mathcal{N}}$. En multipliant le système de contraintes de (PL) par la matrice B^{-1}, nous arrivons à un problème (PL′) équivalent, défini comme suit.

$$(\text{PL}') = \begin{cases} \min \quad \boldsymbol{c}^\top \boldsymbol{x} & & \\ \text{t.q.} \quad \boldsymbol{x}_{\mathcal{F}} + \bar{A}_{\mathcal{P}\mathcal{N}}\boldsymbol{x}_{\mathcal{N}} = \boldsymbol{x}^0_{\mathcal{F}}, & (\boldsymbol{\pi}_{\mathcal{P}}) & (1.53) \\ \qquad \bar{A}_{\mathcal{Z}\mathcal{N}}\boldsymbol{x}_{\mathcal{N}} = \boldsymbol{0}, & (\boldsymbol{\pi}_{\mathcal{Z}}) & (1.54) \\ \qquad \boldsymbol{x} \geq \boldsymbol{0}. & & (1.55) \end{cases}$$

Ainsi, une colonne A_j est dite compatible si $\bar{A}_{\mathcal{Z},j} = \boldsymbol{0}$, ce qui est équivalent à $A_j \in V(A_{\mathcal{F}})$. Remarquons que tout pivot basé sur une colonne compatible est non-dégénéré. Ensuite, l'algorithme IPS résout un *problème complémentaire*, similaire au problème de désagrégation duale de DCA. En effet, il s'agit ici de maximiser le coût réduit des colonnes hors-base afin de prouver l'optimalité de la solution courante.

$$(\text{CP}) = \begin{cases} \max \quad (\boldsymbol{c}_{\mathcal{N}}^\top - \boldsymbol{c}_{\mathcal{F}}^\top \bar{A}_{\mathcal{P}\mathcal{N}})\boldsymbol{x}_{\mathcal{N}} & \\ \text{t.q.} \quad \boldsymbol{1}^\top \boldsymbol{x}_{\mathcal{N}} = 1, & (1.56) \\ \qquad \bar{A}_{\mathcal{Z}\mathcal{N}}\boldsymbol{x}_{\mathcal{N}} = \boldsymbol{0}, & (1.57) \\ \qquad \boldsymbol{x}_{\mathcal{N}} \geq \boldsymbol{0}. & (1.58) \\ & (1.59) \end{cases}$$

Résoudre ce sous-problème permet alors de trouver une colonne compatible de coût réduit partiel (dépendant uniquement de $A_{\mathcal{P}}$) minimum, ou bien de de trouver une combinaison de colonnes incompatibles telle que la combinaison est elle-même compatible (le coût réduit de chaque colonne est combiné lui aussi). Puisque la colonne (ou l'ensemble de colonnes) ainsi trouvé est compatible, elle permet de trouver une solution $\boldsymbol{x}^1$ strictement meilleure que $\boldsymbol{x}^0$ en une opération de pivot (ou en $\mathcal{O}(m-f)$ pivots), ou bien de prouver l'optimalité de $\boldsymbol{x}^0$ le cas échéant.

Pour une revue de la littérature autour de la méthode IPS et de ses nombreuses variantes, voir [GDL16]. Cet article montre également que, lorsque les clusters correspondent à une solution réalisable entière, l'algorithme DCA est une application directe de la méthode IPS.

1.3 Problèmes de sectorisation

Dans ce document, nous nous intéressons à la résolution exacte de deux problèmes de sectorisation. Afin de pouvoir établir un lien avec d'autres problèmes étudiés dans la littérature scientifique, nous consacrons cette section à la présentation générale des problèmes de sectorisation. En particulier, nous nous concentrons sur deux problèmes classiques de sectorisation : le problème du p-median et celui du p-centre, dont les formulations mathématiques sont proches des modèles que nous utilisons par la suite.

1.3.1 Revue de la littérature

On parle de sectorisation lorsque l'on cherche à regrouper des zones géographiques de petites tailles, appelées *unités basiques*, en zones de plus grande taille, appelées *secteurs*.

Depuis plus de 50 ans, ces problèmes sont étudiés pour effectuer des découpages politiques [HWS+65, MJN98], dessiner des territoires de ventes [ZS83, RMF09, EARMD14] ou encore optimiser l'organisation de services publics [HFV99, MCVO03] (voir par exemple la revue de littérature récente de Kalcsics [Kal15]). Dans la plupart des problèmes de sectorisation, on cherche à construire des secteurs « **connexes** », « **compacts** » et « **équilibrés** ». Une partie importante du travail dans ce champ de recherche est de définir de manière formelle ces trois concepts, dont l'acception dépend très fortement du secteur d'application.

La contrainte de connexité implique que toute paire de sommets appartenant à un même secteur puisse être reliée sans sortir de ce dernier. Elle permet d'éviter d'avoir des secteurs qui s'intersectent, ou peut servir à représenter des obstacles géographiques. Dans [RMF09], les auteurs utilisent la contrainte de connexité originale, qui amène à résoudre des modèles de programation linéaire avec un nombre exponentiel de contraintes. Dans [ZS83] et [MJN98], les auteurs ont proposé une version plus restrictive de la connexité, qui utilise la notion de *centre*. Pour chaque secteur, on doit déterminer un centre j ; si le problème est défini dans un graphe, un centre correspond à un sommet. La contrainte de connexité impose que chaque secteur doit être une sous-arborescence de l'arborescence des plus courts chemins partants de j.

Parmi les trois critères, la notion de compacité est celle qui est le plus sujette à interpretation. Elle est en général prise en compte dans l'objectif de l'optimisation. Elle peut être modélisée à la manière d'un problème de p-median [ZS83, MJN98], où l'on minimise la somme des distances de chaque sommets au centre de son secteur ou bien comme un problème de p-centre [RMF09, EARMD14] où l'on ne considére que la plus grande distance employée, tous secteurs confondus. Dans ce document, la plus grande distance au sein d'un secteur sera appelée le rayon du secteur. Il existe également des modèles considérant la somme des rayons d'une solution. Par exemple, l'un des critères de [MCVO03] est la somme des rayons *relatifs* de chaque secteur généré, tandis que [SdAFU14] utilise la somme des diamètres, c'est-à-dire de la distance maximale entre toute paire de sommets du même secteur.

La contrainte d'équilibre implique que la différence de volumétrie entre toute paire de secteurs doit être faible, par exemple en terme de charge de travail, de temps de traitement ou de taille de population. La plupart du temps, cette contrainte impose, pour chaque attribut considéré, que le volume de tout secteur soit proche de la moyenne,

avec une marge d'erreur donnée. Il arrive aussi que cette contrainte soit représentée dans la fonction objectif [SdAFU14], le but étant alors de minimiser l'écart de chaque secteur avec la moyenne.

Plusieurs travaux relatifs aux problèmes de sectorisation font usage d'heuristiques basées sur la recherche locale [HFV99, RMF09, SdAFU14]. On trouve aussi des travaux utilisant la programmation mathématique, et notamment la décomposition de Dantzig et Wolfe (voir par exemple [MJN98]). Les méthodes de génération de colonnes se prêtent bien à ce type de problème, qui se décompose naturellement en un problème de partitionnement/couverture et à un sous-problème pour chaque secteur ; par exemple, les sous-problèmes de [MJN98] sont des problèmes de sac à dos avec une consommation minimale et maximale, résolus par un algorithme de branch-and-bound.

Dans cette section, nous présentons deux problèmes de sectorisation classiques : le problème du p-median et le problème du p-centre, ainsi que leurs généralisations à des contraintes de capacité. Pour chaque problème, nous rapportons plusieurs modèles mathématiques issus de la littérature.

1.3.2 Problème du p-median

L'un des deux problèmes fondamentaux introduits par Hakimi [Hak64] consiste à trouver la médiane absolue d'un graphe. Considérons l'exemple d'un réseau de télécommunication. Lorsqu'une communication est établie entre deux nœuds du réseau, l'information doit transiter par un *switching center*. Si nous représentons le réseau de télécommunication par un graphe, ce *switching center* doit être localisé soit sur un nœud, soit sur un arc. Le problème posé par Hakimi consiste alors à déterminer la position optimale du centre, appelée médiane absolue, telle que la somme des distances entre tout nœud du réseau et le centre soit minimisée. Dans ce premier article, l'auteur montre qu'il existe toujours une position optimale correspondant à l'un des nœuds du réseau. Pour cette raison, les modèles mathématiques présentés dans cette section considèrent toujours qu'un centre est placé sur un nœud du réseau (et non sur un arc, par exemple). Dans un second article, Hakimi pose le problème du p-median [Hak65], consistant à couvrir un réseau en utilisant exactement p centres, où $p \in \mathbb{N}$ est une donnée de l'instance. Dans le cas général, le problème du p-median est NP-complet [KH79b].

Problème 3 (p-median)**.** Étant donné un graphe $G = (V, A)$, une longueur $c_{i,j} \in \mathbb{R}_+$ pour tout $(i, j) \in A$, et un entier p, trouver un sous-ensemble $P \subseteq V$ de p sommets tel que, si $d_i(P)$ est la longueur du plus court chemin de $i \in V$ au sommet le plus proche dans P, alors la somme $\sum_{i \in V} d_i(P)$ est minimisée.

Étant donné $G = (V, A)$ un réseau composé de $|V|$ nœuds, soit $d_{i,j}$ la longueur du plus court chemin entre les nœuds i et j de V, et soit $\boldsymbol{x} \in \{0, 1\}^{|V| \times |V|}$ et $\boldsymbol{y} \in \{0, 1\}^{|V|}$ deux vecteurs de variables de décision binaires. Étant donné $p \in \mathbb{N}$, le problème du p-median peut être modélisé par le programme mathématique suivant [RS70].

$$
(p\text{-M}^{RS}) = \begin{cases}
\min \displaystyle\sum_{i\in V}\sum_{j\in V} d_{i,j}x_{i,j} & & & \\
\text{t.q.} \quad \displaystyle\sum_{j\in V} x_{i,j} = 1 & \forall i \in V, & & (1.60)\\
y_j \geq x_{i,j} & \forall i \in V, & \forall j \in V, & (1.61)\\
\displaystyle\sum_{j\in V} y_j = p, & & & (1.62)\\
x_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V, & (1.63)\\
y_j \in \{0,1\} & & \forall j \in V. & (1.64)
\end{cases}
$$

La variable y_j vaut 1 si le nœud j est utilisé comme centre dans la solution, et 0 sinon, tandis que la variable de décision $x_{i,j}$ vaut 1 si le nœud i est couvert par le centre j, et 0 sinon. Les contraintes (1.60) permettent de garantir que chaque nœud du réseau soit couvert par exactement un centre, tandis que les contraintes (1.61) impliquent que si le nœud i est couvert par j, alors j est sélectionné comme centre. Enfin, la solution doit contenir exactement p centres, d'après la contrainte (1.62).

Il existe de nombreuses variantes du problème du p-median. Dans le problème du *capacitated p-median*, chaque centre est limité par une contrainte de capacité pour couvrir les nœuds du réseau, à la manière d'un problème de sac à dos. Ce problème a été introduit dans [MB84] sous le nom de *capacitated clustering problem*.

Pour un réseau $G = (V, A)$ donné, soit w_i la consommation de ressource associée au nœud $i \in V$ et soit W_j la capacité du centre j. Nous pouvons étendre le modèle précédent afin de prendre en compte la capacité des centres.

$$
(p\text{-M}^{MB}) = \begin{cases}
\min \displaystyle\sum_{i\in V}\sum_{j\in V} d_{i,j}x_{i,j} & & & \\
\text{t.q.} \quad \displaystyle\sum_{j\in V} x_{i,j} = 1 & \forall i \in V, & & \\
y_j \geq x_{i,j} & \forall i \in V, & \forall j \in V, & \\
\displaystyle\sum_{j\in V} y_j = p, & & & \\
\displaystyle\sum_{i\in V} w_i x_{i,j} \leq W_j, & & \forall j \in V, & (1.65)\\
x_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V, & \\
y_j \in \{0,1\} & & \forall j \in V. &
\end{cases}
$$

La nouvelle contrainte de capacité est modélisée par (1.65). En général, les centres ont une capacité identique $W_j = W$, $\forall j \in V$.

Bien que cette variante ne soit pas la plus répandue, plusieurs algorithmes ont été proposés dans la littérature, notamment des méthodes heuristiques [MB84] ou métaheuristiques [OC94, FH08]. Notons également [LS04], qui utilise une approche par génération de colonnes couplée à une relaxation Lagrangienne/surrogate. Dans ce dernier article, les auteurs considèrent le couple problème maître/sous-problèmes suivant.

Soit $G = (V, A)$, $d_{i,j}$, p, w_i et W définis comme précédemment. L'ensemble $\mathcal{G}^j$ est un ensemble de colonnes associé au sous-problème (P^j) — correspondant aux secteurs de centre j — et λ_j^g est la variable de décision indiquant l'utilisation de la colonne g, pour tout $g \in \mathcal{G}^j$. Étant donné $j \in V$ et $g \in \mathcal{G}^j$, les valeurs dans g des variables y_j et $x_{i,j}$ $\forall i \in V$ sont notées $x_{i,j}^g$ et y_j^g, respectivement. Le problème maître (P^M) peut alors être défini de la façon suivante.

$$(\mathrm{P}^M) = \begin{cases} \min \displaystyle\sum_{j \in V} \sum_{i \in V} d_{i,j} \left(\sum_{g \in \mathcal{G}^j} x_{i,j}^g \, \lambda_j^g \right) & & & \\ \text{t.q.} \quad \displaystyle\sum_{j \in V} \left(\sum_{g \in \mathcal{G}^j} y_j^g \, \lambda_j^g \right) = p, & & (\kappa) & (1.66) \\ \displaystyle\sum_{j \in V} \left(\sum_{g \in \mathcal{G}^j} x_{i,j}^g \, \lambda_j^g \right) \geq 1 & \forall i \in V, & (\phi_i) & (1.67) \\ \lambda_j^g \in [0, 1] & \forall g \in \mathcal{G}^j, \ \forall j \in V. & & (1.68) \end{cases}$$

Chaque nœud $j \in V$ est associé au sous-problème de pricing (P^j) suivant.

$$(\mathrm{P}^j) = \begin{cases} -\kappa \min \displaystyle\sum_{i \in V} (d_{i,j} - \tau \phi_i) \, x_{i,j} + \tau \sum_{i \in V} \phi_i & & \\ \text{t.q.} \displaystyle\sum_{i \in V} w_i \, x_{i,j} \leq W, & & (1.69) \\ x_{i,j} \in \{0, 1\} & \forall i \in V. & (1.70) \end{cases}$$

La fonction objectif des sous-problèmes est définie, entre autres, par κ la variable duale de la contrainte sur le nombre de centres (1.66) et ϕ_i la variable duale correspondant à la contrainte de couverture (1.67) du nœud i, pour tout $i \in V$. Dans l'algorithme, les variables duales $\boldsymbol{\phi}$ sont considérées comme des multiplicateurs Lagrangiens, et les auteurs utilisent un algorithme de sous-gradient pour résoudre le problème dual Lagrangien. Notons également la valeur τ, qui est ici un paramètre de la relaxation Lagrangienne/surrogate. Ce paramètre est optimisé par recherche dichotomique pendant la résolution du problème dual Lagrangien, et permet d'accélérer la convergence de l'algorithme de sous-gradient.

Cette formulation présente plusieurs autres points notables. Premièrement, la contrainte de partition (1.60) a été relâchée en contrainte de couverture (1.67), ce qui étend l'espace des solutions réalisables. Deuxièmement, le problème maître n'est pas sujet à la contrainte de convexité que l'on obtiendrait par décomposition de Dantzig-Wolfe. Troisièmement, les auteurs se focalisent sur la relaxation continue du problème, avec des variables λ_j^g réelles.

Dans [BHMM02], les auteurs utilisent une méthode exacte basée également sur la formulation étendue du problème en set-covering, mais sans génération de colonnes. À la place, la méthode proposée consiste à déterminer un sous-ensemble de l'ensemble des colonnes réalisables tel que ce sous-ensemble suffit à construire une solution optimale du problème. Cela permet de définir une formulation réduite, qui est ensuite transmise à un solveur générique. Si l'ensemble de colonnes généré est trop grand, alors les auteurs re-

tirent davantage de colonnes mais, dans ce cas, il n'y a plus de garantie que la formulation obtenue soit associée à une solution optimale du problème original.

1.3.3 Problème du p-centre

Le second problème fondamental introduit par Hakimi [Hak64] consiste à trouver le centre absolu d'un graphe. L'auteur propose cette fois-ci de considérer l'emplacement d'un hôpital ou d'un commissariat. Dans ce contexte, le critère dominant n'est plus la distance moyenne entre les points du réseau et le centre, mais la distance maximale entre un point du réseau et le centre. Le problème du centre absolu consiste donc à trouver le point situé sur un nœud ou une arête du réseau qui minimise la distance maximale avec les autres nœuds du réseau. Ce problème peut être généralisé au problème du p-centre [Hak65] consistant à couvrir un réseau avec au plus p centres, où $p \in \mathbb{N}$ est une donnée de l'instance, de sorte à minimiser la distance maximale entre un nœud du réseau et son centre le plus proche. La plupart du temps, l'emplacement des centres est limité aux nœuds du graphe, sans tenir compte des arêtes. Dans le cas général, le problème du p-centre est NP-complet [KH79a].

Problème 4 (p-centre)**.** Étant donné un graphe $G = (V, A)$, une longueur $c_{i,j} \in \mathbb{R}_+$ pour tout $(i, j) \in A$, et un entier p, trouver un sous-ensemble $P \subseteq V$ de p sommets tel que, si d_i est la longueur du plus court chemin de $i \in V$ au sommet le plus proche dans P, alors la valeur $\max_{i \in V} d_i$ est minimisée.

Plusieurs modèles ont été proposées pour représenter le problème du p-centre sous la forme d'un programme mathématique. Étant donné $G = (V, A)$ un réseau composé de $|V|$ nœuds et $p \in \mathbb{N}$, soit $d_{i,j}$ la distance entre les nœuds i et j de V, $\boldsymbol{x} \in \{0,1\}^{|V|\times|V|}$ et $\boldsymbol{y} \in \{0,1\}^{|V|}$ deux vecteurs de variables de décision binaires, et $r \in \mathbb{R}_+$ une variable continue. La variable de décision $x_{i,j}$ vaut 1 si le nœud i est couvert par le centre j, et 0 sinon, tandis que la variable y_j indique si le nœud j est utilisé comme centre ou non. Enfin, la variable r a pour valeur la distance maximale entre un nœud du réseau et son centre le plus proche. Étant donné $p \in \mathbb{N}$, le problème du p-centre peut être modélisé par le programme mathématique suivant [Das95].

$$
(p\text{-C}^D) = \begin{cases}
\min \quad r & & & \\
\text{t.q.} \quad \displaystyle\sum_{j \in V} x_{i,j} = 1 & \forall i \in V, & & (1.71)\\
\displaystyle\sum_{j \in V} y_j \leq p, & & & (1.72)\\
x_{i,j} \leq y_j & \forall i \in V, & \forall j \in V, & (1.73)\\
\displaystyle\sum_{j \in V} d_{i,j} x_{i,j} \leq r & \forall i \in V, & & (1.74)\\
x_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V, & (1.75)\\
y_j \geq 0 & & \forall j \in V. & (1.76)
\end{cases}
$$

Les contraintes (1.71) permettent de garantir que chaque nœud du réseau soit couvert par exactement un centre, tandis que les contraintes (1.73) impliquent que si le nœud i

est couvert par j, alors j est un centre. De plus, la solution doit contenir exactement p centres, d'après la contrainte (1.72). Enfin, la contrainte (1.74) implique que, pour tout nœud $i \in V$, la variable r est supérieure à la distance entre i et le centre $j \in V$ qui lui est associé.

Par la suite, une formulation plus forte a été proposée [ELP04]. Soit $\{R^0, R^1, \ldots, R^K\}$ l'ensemble des distances $d_{i,j}$ dans l'ordre croissant. Soit également $\boldsymbol{z} \in \{0,1\}^K$ un vecteur de variables de décision, et soit $\boldsymbol{y}$ défini comme dans le modèle précédent. Le problème du p-centre peut être reformulé comme suit.

$$
(p\text{-C}^{ELP}) = \begin{cases}
\min \quad R^0 + \displaystyle\sum_{k=1}^{K} \left(R^k - R^{k-1}\right) z^k & & & \\
\text{t.q.} \quad \displaystyle\sum_{j \in V} y_j \leq p, & & & \\
\qquad \displaystyle\sum_{j : d_{i,j} < R^k} y_j + z^k \geq 1 & \forall i \in V, & \forall k = 1, \ldots, K, & (1.77) \\
\qquad y_j \in \{0,1\} & \forall j \in V, & & (1.78) \\
\qquad z^k \in \{0,1\} & \forall k = 1, \ldots, K. & & (1.79)
\end{cases}
$$

Dans cette formulation, les variables $\boldsymbol{y}$ sont définies comme dans le modèle précédent. En revanche, la variable z^k indique que la distance maximale entre un nœud du réseau et son centre le plus proche est supérieure ou égale à R^k. Ainsi, la contrainte (1.77) indique que, pour tout nœud $j \in V$, soit la solution contient un centre dont la distance à i est strictement inférieure à R^k, soit la solution est de coût supérieur ou égal à R^k. La relaxation continue du nouveau modèle (p-C^{ELP}) est plus forte que celle du modèle (p-C^D). Dans [AE18], les auteurs montrent qu'en ajoutant les contraintes valides $z^k \geq z^{k+1}$, $\forall k = 1, \ldots, K-1$, il est possible de retirer toutes les contraintes (1.77) pour i et k tels que R^k n'est pas un rayon déterminant pour le client i, c'est-à-dire $\nexists j : d_{i,j} = R^k$.

Inspirés par le modèle (p-C^{ELP}), les auteurs de [CT13] ont proposé un troisième modèle mathématique du problème de p-centre, utilisant le vecteur de variables binaires $\boldsymbol{u} \in \{0,1\}^K$.

$$(p\text{-C}^{CT}) = \begin{cases} \min \sum_{k=1}^{K} R^k u^k & & & \\ \text{t.q.} \quad \sum_{j \in V} y_j \leq p, & & & \\ \quad \sum_{j:d_{i,j} \leq R^k} y_j \geq \sum_{q=1}^{k} u_q & \forall i \in V, & \forall k = 1, \ldots, K, & (1.80) \\ \quad \sum_{k=1}^{K} u_k = 1, & & & (1.81) \\ \quad y_j \in \{0,1\} & \forall j \in V, & & (1.82) \\ \quad u_k \in \{0,1\} & & \forall k = 1, \ldots, K. & (1.83) \end{cases}$$

Ici, le modèle utilise directement la distance R^k associée à la variable binaire u_k, au lieu d'utiliser la différence $(R^k - R^{k-1})$ entre deux distances consécutives. Ainsi, la contrainte (1.80) indique que si la valeur de la solution est inférieure ou égale à R^k, alors tout nœud $i \in V$ doit avoir à disposition un centre j à une distance $d_{i,j}$ inférieure ou égale à R^k. Les auteurs ont également démontré l'existence d'une bijection entre l'ensemble des solutions réalisables de leur modèle et du modèle $(p\text{-C}^{ELP})$, y compris dans le cas continu, ce qui signifie que ces deux modèles sont équivalents en termes d'espace des solutions réalisables, et donc de qualité des bornes issues de la relaxation continue.

Enfin, un dernier modèle a été proposé dans [AE18], utilisant les mêmes variables binaires $\boldsymbol{y}$ que les modèles précédents, ainsi qu'une variable entière $\tilde{k} \in \mathbb{N}_+$.

$$(p\text{-C}^{AE}) = \begin{cases} \min \quad \tilde{k} & & & \\ \text{t.q.} \quad \tilde{k} + k \sum_{j:d_{i,j} < R^k} y_j \geq k & \forall i \in V, & \forall k \in S_i, & (1.84) \\ \quad y_j \in \{0,1\} & \forall j \in V, & & (1.85) \\ \quad \tilde{k} \in \{0,1,\ldots,K\}. & & & (1.86) \end{cases}$$

Ce modèle repose sur le fait que l'on peut toujours supposer, sans perte de généralité, que $R^0 = 0$ et $R^k - R^{k-1} = 1$, $\forall k = 1, \ldots, K-1$. En conséquence, l'information pertinente dans le problème du p-centre est davantage de connaître l'indice $\tilde{k}$ du rayon utilisé que la valeur réelle $R^{\tilde{k}}$ de ce rayon. Par ailleurs, ce modèle est le plus compact présenté jusqu'ici, avec $\mathcal{O}(|V|)$ variables et $\mathcal{O}(\min\{|V|^2, |V| \cdot K\})$ contraintes. Cependant, la borne de la relaxation continue peut être moins forte que celle obtenue par la formulation $(p\text{-C}^{ELP})$ — au mieux, les bornes sont égales —, et l'utilisation de la variable $\tilde{k}$ non-binaire peut rendre le problème plus long à résoudre par un solveur générique.

Tout comme dans le cas du problème du p-median, le problème du p-centre peut être étendu pour inclure une contrainte de capacité servant à équilibrer la charge assignée à chaque centre [BKP93]. Pour un réseau $G = (V, A)$ donné, soit w_i la consommation de ressource associée au nœud $i \in V$ et soit W_j la capacité du centre j. Le problème du p-centre sous contraintes de capacités peut être modélisé en ajoutant directement les

contraintes de capacité au modèle (p-C^D) [SPS04].

$$
(\text{C}p\text{-C}^{SPS}) = \begin{cases}
\min \quad r & & & & \\
\text{t.q.} \quad \displaystyle\sum_{j\in V} x_{i,j} = 1 & \forall i \in V, & & & \\
\displaystyle\sum_{j\in V} y_j \leq p, & & & & \\
x_{i,j} \leq y_j & \forall i \in V, & \forall j \in V, & & \\
\displaystyle\sum_{j\in V} d_{i,j} x_{i,j} \leq r & \forall i \in V, & & & \\
\displaystyle\sum_{i\in V} w_i x_{i,j} \leq W_j y_j & & \forall j \in V, & & (1.87) \\
x_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V, & & (1.88) \\
y_j \geq 0 & & \forall j \in V. & & (1.89)
\end{cases}
$$

La nouvelle contrainte (1.87) permet de garantir qu'aucun centre n'excède sa capacité à couvrir les nœuds du réseau. Notons que les modèles (p-C^{ELP}), (p-C^{CT}) et (p-C^{AE}) n'assignent pas explicitement un nœud à un centre ; chaque nœud $i \in V$ est implicitement associé au centre j le plus proche. L'extension de ces modèles au problème du p-centre sous contraintes de capacité est donc moins immédiate, mais peut tout de même être effectuée en ré-introduisant les variables $x_{i,j}$ [ASDF10].

$$
(\text{C}p\text{-C}^{ADF}) = \begin{cases}
\min \quad R^0 + \displaystyle\sum_{k=1}^{K} \left(R^k - R^{k-1}\right) z^k & & & & \\
\text{t.q.} \quad \displaystyle\sum_{j\in V} x_{i,j} = 1 & \forall i \in V, & & & \\
\displaystyle\sum_{j\in V} y_j \leq p, & & & & \\
x_{i,j} \leq y_j & \forall i \in V, & \forall j \in V & & \\
x_{i,j} \leq z^k & \forall i \in V, & \forall j \in V, \ \forall k : d_{i,j} \geq R^k, & & (1.90) \\
\displaystyle\sum_{i\in V} w_i x_{i,j} \leq W_j y_j & & \forall j \in V, & & (1.91) \\
x_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V, & & (1.92) \\
y_j \geq 0 & & \forall j \in V, & & (1.93) \\
z^k \in \{0,1\} & & & \forall k = 1, \dots K. & (1.94)
\end{cases}
$$

Pour une revue de la littérature plus détaillée du problème de p-centre et ses variantes, voir [Cal13].

1.4 Problème de sac à dos sous contraintes de précédence en arborescence

Dans cette section, nous nous intéressons à une variante du problème de sac à dos appelée *problème de sac à dos sous contraintes de précédence en arborescence* (que nous appelerons aussi *Tree Knapsack Problem*, ou TKP). En effet, les sous-problèmes de génération de colonnes que nous considérons dans les chapitres suivants sont principalement traités comme des généralisations de ce problème. Ainsi, les approches de résolution étudiées sont inspirées d'algorithmes issus de la littérature du TKP et que nous présentons ici.

1.4.1 Définition formelle

Le problème de sac à dos sous contraintes de précédence (*Precedence constraint knapsack problem*) est une généralisation du problème de sac à dos prenant en compte un ordre partiel des objets. Cela signifie que certains objets ne peuvent être ajoutés à la solution, à moins que leurs prédécesseurs n'en fassent également partie. Les travaux de recherche se concentrent souvent sur le cas particulier du problème de sac à dos sous contraintes de précédence en arborescence (*Tree Knapsack Problem*), c'est-à-dire que le graphe de précédence forme un arbre. En effet, si le cas général est NP-difficile au sens fort [GJ79, JN83] le cas particulier du *Tree Knapsack Problem* peut être résolu en temps pseudo-polynomial [Luk74, JN83, CS97].

Formellement, le problème peut être défini comme suit.

Problème 5 (sac à dos avec contraintes de précédence en arborescence (TKP))**.** Étant donné une arborescence $T = (V, A)$ enracinée en j, un vecteur $\boldsymbol{p}$ de $\mathbb{R}^{|V|}$, un vecteur $\boldsymbol{w}$ de $\mathbb{N}^{|V|}$, ainsi qu'un entier $W \in \mathbb{N}$, trouver un sous-ensemble de sommets $U \subseteq V$ tel que U induit un sous-arbre de T enraciné en j, la capacité du sac n'est pas dépassée $\sum_{i \in U} w_i \leq W$ et qui maximise $\sum_{i \in U} p_i$.

Ce problème est notamment lié à des applications dans le design de réseaux de télécommunication [CS97, CS98], et plus récemment à des problèmes d'allocation de ressources [PS18].

Notons $\pi(k)$ le prédécesseur du sommet $k \in V$ dans l'arborescence T. Le TKP peut être formulé par le programme mathématique suivant [CS97].

$$
(\text{TKP}) = \begin{cases}
\max \displaystyle\sum_{i \in V} p_i \, x_i & & \\
\text{t.q.} \quad \displaystyle\sum_{i \in V} w_i \, x_i \leq W, & & (1.95) \\
x_i \leq x_{\pi(i)} & \forall i \in V, & (1.96) \\
x_i \in \{0, 1\} & \forall i \in V. & (1.97)
\end{cases}
$$

Dans le programme (TKP), les variables binaires $x_i, \forall i \in V$ sont telles que $x_i = 1$ si l'objet i fait partie de la solution, et $x_i = 0$ sinon. La contrainte (1.95) garantie que la

solution ne dépassera pas la capacité du sac, tandis que les contraintes (1.96) impliquent que si $i \in V$ est couvert par la solution, alors son prédécesseur $\pi(i)$ l'est également.

1.4.2 Résolution par programmation dynamique

Dans [CS97], les auteurs proposent de résoudre le TKP en temps pseudo-polynomial à l'aide d'un algorithme de programmation dynamique inspiré de [JN83].

Supposons, sans perte de généralité, que les sommet de $V = \{0, 1, 2, \ldots, |V|-1\}$ sont ordonnés selon un ordre en profondeur d'abord de l'arborescence T. Ainsi, le sommet 0 est la racine de T. Étant donné $k \in V$, soit $L_k = \{0, 1, \ldots, k\} \subseteq V$, et soit $T(k) \subseteq T$ la sous-arborescence de T contenant à la fois k et tous ses descendants (successeurs directs ou indirects). Enfin, soit $L_k^+ = L_k \cup T(k)$, c'est-à-dire que L_k^+ contient les $k+1$ premiers sommets de l'ordre en profondeur d'abord, ainsi que tous les descendants du sommet k. En conséquence, si k' est une feuille de T, alors $L_{k'}^+ = L_{k'}$, et si k'' est le dernier descendant de k dans l'ordre en profondeur d'abord, alors $L_k^+ = L_{k''}$. Chaque état du programme dynamique est donc défini par le couple (k, w) De plus, le label $\psi_L^{\mathrm{CS}}(k, w)$ correspond au profit maximal d'une solution limitée aux sommets de L, contenant le sommet k, et n'excédant pas la capacité w. Le programme dynamique peut être défini par les règles suivantes.

$$\psi_{L_0}^{\mathrm{CS}}(0, w) = \begin{cases} -\infty & \forall w = 0, \ldots, w_0 - 1 \\ p_0 & \forall w = w_0, \ldots, W \end{cases} \tag{1.98}$$

$$\psi_{L_k}^{\mathrm{CS}}(k, w) = \begin{cases} -\infty & \forall k \neq 0,\ \forall w = 0, \ldots, w_k - 1 \\ \psi_{L_{k-1}}^{\mathrm{CS}}(\pi(k), w - w_k) + p_k & \forall k \neq 0,\ \forall w = w_k, \ldots, W \end{cases} \tag{1.99}$$

$$\psi_{L_{k-1}^+}^{\mathrm{CS}}(\pi(k), w) = \max\left\{\psi_{L_{k-1}}^{\mathrm{CS}}(\pi(k), w),\ \psi_{L_{k-1}^+}^{\mathrm{CS}}(k, w)\right\} \quad \forall k \neq 0,\ \forall w = 0, \ldots, W \tag{1.100}$$

L'idée derrière ce programme dynamique est la suivante. À chaque itération, l'algorithme génère les labels liés au sommet k à partir des labels de son prédécesseur $k-1$ dans l'ordre en profondeur d'abord (phase « avant »). Toutes les solutions associées aux labels ainsi générées couvrent le sommet k sans dépasser la contrainte de capacité. Notons que $k-1$ n'est pas nécessairement un prédécesseur de k dans l'arborescence T. Lorsqu'un sous-arbre $T(k)$ a été complètement résolu (autrement dit, on a considéré k et tous ses descendants), l'algorithme retourne les labels ainsi générés au sommet $\pi(k)$, parent dans l'arbre de précédence (phase « arrière »).

La règle (1.98) sert à initialiser les labels du programme dynamique correspondant au sommet racine 0. Ensuite, la règle (1.99) correspond aux mouvements « avant », générant les labels initiaux du sommet k à partir du sommet précédent $k-1$, tandis que la règle (1.100) correspond aux mouvements « vers l'arrière », retournant les labels du sommet k à son prédécesseur $\pi(k)$. Ce programme dynamique permet ainsi de résoudre le TKP en $\mathcal{O}(|V| \cdot W)$.

1.4.3 Résolution par branch-and-bound

Les auteurs du programme dynamique présenté dans la section précédente ont également proposé un algorithme de type branch-and-bound pour résoudre le TKP [CS98]. Bien que ce second algorithme ait une complexité exponentielle, et non pseudo-polynomiale, ses performances sont significativement meilleures en pratique que celles de l'algorithme de programmation dynamique.

L'algorithme de branch-and-bound est basé sur une relaxation du TKP appelée problème de sous-arborescence de poids maximum, dans laquelle la contrainte de capacité est relâchée.

Problème 6 (sous-arborescence de poids maximum)**.** Étant donné une arborescence $T = (V, A)$ enracinée en j et un vecteur $\mathbf{p} \in \mathbb{R}^{|V|}$, trouver une sous-arborescence $T[U]$ de T qui maximise $\sum_{i \in U} p_i$.

Étant donné $\mu \in \mathbb{R}^+$ un multiplicateur Lagrangien, nous pouvons définir la relaxation Lagrangienne de (TKP) par rapport à la contrainte de capacité de la façon suivante.

$$\big(\mathrm{RL}_\mu(\mathrm{TKP})\big) = \begin{cases} \max \displaystyle\sum_{i \in V} (p_i - \mu w_i)\, x_i + \mu W & \\ \text{t.q.} \quad x_i \leq x_{\pi(i)} & \forall i \in V, \qquad (1.101) \\ \quad\; x_i \in \{0,1\} & \forall i \in V. \qquad (1.102) \end{cases}$$

La relaxation ainsi obtenue est un problème de sous-arborescence maximum dans lequel chaque objet i est associé à un profit $\tilde{p}_i = p_i - \mu w_i$. Les auteurs [CS98] ont montré que 1) le problème de sous-arborescence de poids maximum peut être résolu en temps linéaire $\mathcal{O}(|V|)$ grâce à un parcours en profondeur inverse de l'arbre T ; 2) en conséquence, le problème Lagrangien associé à la relaxation $(\mathrm{RL}_\mu(\mathrm{TKP}))$ peut être résolu en $\mathcal{O}(|V|^2)$. En effet, le Lagrangien est une fonction convexe linéaire par morceaux, et chaque sommet $i \in V$ est associé à une intersection de deux segments de la fonction. Il y a donc au maximum $|V|$ valeurs pertinentes pour le multiplicateur μ, ces valeurs pouvant être énumérées par une recherche linéaire.

Dans l'algorithme proposé par Cho et Shaw, chaque nœud de branchement est défini par un ensemble $\mathcal{U}$ d'objets qui sont obligatoirement couverts par la solution, $\mathcal{L}$ un autre ensemble d'objets qui, eux, sont interdits, et une valeur $\bar{\mu}$. Notons $\mathcal{F} = V \setminus (\mathcal{U} \cup \mathcal{L})$ et soit T^* un sous-arbre de T optimal pour $(\mathrm{RL}_{\bar{\mu}}(\mathrm{TKP}))$. Dans l'algorithme, la valeur $\bar{\mu}$ est telle que T^* est induit par les sommets de $\mathcal{U} \cup \mathcal{F}$.

La décision prise à chaque nœud de branchement consiste à sélectionner un objet et soit l'ajouter dans la solution partielle courante, soit l'en exclure. Notons qu'à cause des contraintes de précédence, une décision de branchement sur l'objet i implique également l'ajout de tous ses prédécesseurs à la solution, ou l'exclusion de tous ses descendants, selon la décision prise sur i. L'évaluation de ce nœud de branchement se fait en calculant la valeur optimale de $(\mathrm{RL}_{\bar{\mu}}(\mathrm{TKP}))$, sachant que la solution optimale doit respecter les décisions de branchement prises précédemment.

Notons T_i le sous-arbre de T^* enraciné en i et composé de tous les descendants de i dans T^*, et notons également $p(T_i)$ (resp. $w(T_i)$) la somme des profits (resp. poids) des objets appartenant à T_i. Si le sous-arbre T^* vérifie la contrainte de sac à dos, alors le

nœud courant est une feuille de l'arbre de branchement. Sinon, soit $k \in \mathcal{F}$ tel que T_k minimise le ratio profit/poids $\bar{\mu}' = \frac{p(T_k)}{w(T_k)}$. En conséquence, le nouveau profit Lagrangien de T_k est $p(T_k) - \bar{\mu}' w(T_k) = 0$, et nous avons également $\bar{\mu}' \geq \bar{\mu}$ car $p(T_k) - \bar{\mu} w(T_k) \geq 0$. La prochaine décision de branchement consiste alors à ajouter k soit dans $\mathcal{U}$, soit dans $\mathcal{L}$. Considérons le nœud de branchement dans lequel k est ajouté à $\mathcal{L}$ l'ensemble des objets interdits. Nous pouvons associer ce nœud de branchement à la valeur $\bar{\mu}'$ du multiplicateur lagrangien, ce qui nous permet de calculer immédiatement une borne Lagrangienne plus précise que celle du nœud de branchement parent, car $\bar{\mu}' \geq \bar{\mu}$ et $T^* \setminus T_k$ est une solution optimale pour $\big(\mathrm{RL}_{\bar{\mu}'}(\mathrm{TKP})\big)$. Si nous considérons maintenant le nœud de branchement dans lequel k est ajouté à $\mathcal{U}$ l'ensemble des objets appartenant à la solution partielle, alors nous obtenons également une borne Lagrangienne plus précise grâce à la solution T^* qui est optimale pour $\big(\mathrm{RL}_{\bar{\mu}'}(\mathrm{TKP})\big)$, en tenant compte de la nouvelle contrainte de branchement.

Notons que les nœuds de branchement ajoutant un sommet dans $\mathcal{L}$ correspondent précisément à la séquence de solutions obtenues pendant l'optimisation du problème dual Lagrangien. De ce fait, la nouvelle borne Lagrangienne calculée à partir de $\bar{\mu}'$ est également valide pour le nœud de branchement parent.

L'algorithme de Cho et Shaw bénéficie donc d'un schéma de branchement basé sur une borne Lagrangienne forte pour un faible coût en calcul, car chaque calcul effectué à un nœud de branchement est ré-utilisé dans les nœuds de branchements qui en descendent.

Chapitre 2

Minimisation de la somme des distances de plus courts chemins

Dans ce chapitre, nous présentons un problème de sectorisation que l'on modélise comme la recherche d'une couverture des sommets d'un graphe par des arborescences. Le coût des secteurs est basé sur une somme de distances, à la manière du problème de p-median. Nous présentons deux modèles de programmation linéaire en nombres entiers, un compact et un basé sur une reformulation de Dantzig et Wolfe. Pour ce dernier, nous proposons plusieurs algorithmes permettant de résoudre le sous-problème de génération de colonnes. Nous proposons également une méthode d'agrégation des contraintes afin de diminuer le nombre d'itérations dégénérées, et donc d'accélérer la convergence du problème maître pendant la résolution de la formulation étendue. Des résultats numériques sont rapportés sur des instances de la littérature, et des instances réelles issues d'une application industrielle. Les expérimentations montrent que, selon le type d'instance considéré, les deux formulations présentent des avantages, et permettent chacune de résoudre un nombre significatif d'instances. En revanche, notre méthode d'agrégation ne permet pas de réduire suffisamment le phénomène de dégénérescence et d'accélérer l'algorithme de branch-and-price.

2.1 Introduction

Dans ce chapitre, nous nous intéressons à un problème de sectorisation dont la fonction objectif est proche du problème classique du p-median. Plus spécifiquement, nous considérons le problème rencontré par une entreprise de collecte de déchets consistant à couvrir une zone géographique, représentée par un graphe, en utilisant des secteurs ; chaque secteur doit être *connexe*, *compact* et *équilibré* (cf. la section 1.3), et est ensuite affecté à un véhicule de la flotte de collecte.

Une manière de formuler la connexité consiste à considérer qu'un secteur est connexe si, pour toute paire de sommets appartenant à ce secteur, il existe un chemin entre ces derniers qui ne traverse que des sommets appartenant, eux-aussi, au secteur. Cependant, cette définition de la connexité est relativement complexe à traiter directement, car elle implique des formulations ayant un nombre conséquent de contraintes et des relaxations continues potentiellement faibles. Afin de rendre le problème plus abordable,

nous utilisons une définition restreinte de la connexité, utilisée par exemple dans [ZS83] et [MJN98]. Ainsi, parmi l'ensemble des sommets couverts par le secteur, nous sélectionnons un sommet comme étant le centre du secteur. La contrainte de connexité impose alors que tout secteur doit être une sous-arborescence de l'arbre des plus courts chemins partant du centre. S'il existe plusieurs arbres des plus courts chemins, nous en sélectionnons un (arbitrairement) pour définir la contrainte de connexité.

Afin de favoriser l'utilisation de secteurs compacts, nous définissons le coût d'un secteur comme étant la somme des distances du centre vers tout sommet couvert par le secteur. Notre problème consiste alors à trouver une couverture des sommets du réseau minimisant la somme des coûts des secteurs utilisés. Cette approche de la compacité se retrouve, par exemple, dans les travaux de [ZS83] et [MJN98]. Dans ce document, la distance d'un centre vers un sommet est définie comme étant la longueur du plus court chemin dans le réseau allant du centre au sommet. Notre modèle a pour objectif de répondre à une problématique industrielle encourageant les secteurs *convexes*. Bien que la notion de convexité soit marginale dans la littérature scientifique sur les problèmes de sectorisation, elle a orienté notre choix concernant la fonction objectif.

Enfin, nous souhaitons équilibrer la charge de travail associée à chaque secteur. Pour cela, nous associons tout sommet à deux consommations de ressource et imposons une consommation maximale à chaque secteur, à la manière d'un problème de sac à dos en deux dimensions. Ces ressources correspondent respectivement au temps nécessaire pour traiter un sommet et au volume de déchet qui y est collecté. Notons que nous n'imposons pas de charge de travail minimale par secteur, contrairement aux approches classiques que l'on peut retrouver dans la littérature scientifique. En effet, l'application industrielle sur laquelle nous nous basons nécessite uniquement la garantie de ne pas avoir de secteur surchargé. L'absence de charge minimale nous permet ainsi plus de flexibilité dans la construction d'une solution, tout en préservant cette garantie.

Une instance de notre problème définit également un nombre maximal de secteurs pour couvrir l'ensemble du réseau, correspondant à la taille de la flotte de véhicules disponibles. De plus, un sommet ne peut pas être le centre de plusieurs secteurs simultanément.

Notre travail concerne des méthodes de résolution exactes basées sur la programmation mathématique pour résoudre notre variante du problème. On s'intéresse en particulier à la méthode de branch-and-price. Cette méthode requiert, entre autres, la mise au point d'un oracle efficace pour le sous-problème associé à un centre, qui cherche à calculer le sous-arbre enraciné en ce centre minimisant le coût réduit. Ce sous-problème peut également être vu comme une extension du problème de sac à dos prenant en compte à la fois des contraintes de précédence, deux contraintes de capacités, et la somme des distances au centre. Nous proposons plusieurs méthodes, basées sur des approches par branch-and-bound et programmation dynamique, pour résoudre ce problème. Nous étudions également l'application d'une technique d'agrégation de contraintes [EVSD05, EMDS11] visant à limiter le nombre d'itérations dégénérées pendant la génération de colonnes.

Dans la section 2.2, nous présentons formellement le problème, ainsi que les différentes formulations étudiées dans ce chapitre. La section 2.3 est dédiée à la formulation étendue, obtenue par décomposition de Dantzig-Wolfe, et au sous-problème de génération de colonnes. Nous étudions plusieurs applications possibles de méthodes d'agrégation pour notre problème dans la section 2.4. La section 2.5 décrit les techniques utilisées dans le branch-and-price. Nous présentons nos résultats numériques dans la section 2.6 avant de

conclure.

2.2 Description du problème et formulation mathématique

Dans cette section, nous présentons formellement le problème étudié. Pour ce faire, nous nous abstrayons de l'application et définissons le problème comme la recherche d'une couverture d'un graphe par des arborescences.

2.2.1 Description formelle du problème

Nous représentons la zone géographique à traiter par un graphe orienté et valué $G = (V, A, c)$ dans lequel chaque sommet $i \in V$ représente un point de collecte et chaque arc $(i, j) \in A$ représente une connexion de i à j de distance $c_{i,j}$. On pose $n = |V|$. À chaque sommet $i \in V$, nous associons deux valeurs w_i^1 et w_i^2 qui correspondent respectivement au temps de traitement et à la quantité de déchets à traiter. La flotte dispose de K véhicules, tous limités par un temps de travail effectif maximal W^1 et une capacité W^2 de déchets pouvant être collectés. Le temps de travail W^1 ne comprend pas les temps de trajet.

Chaque *secteur* que nous cherchons à construire est défini par une paire (U, j), où $U \subseteq V$ est un ensemble de sommets, et $j \in U$ un de ses sommets appelé *centre*. Étant donné $j \in V$ un centre, on note $d_{i,j}$ la longueur du plus court chemin dans G allant de j à un sommet i et $\pi_j(i)$ le sommet précédant i sur ce chemin. S'il n'existe pas de chemin allant de j à i, alors $d_{i,j} = +\infty$ et par convention, on considère dans ce cas que $\pi_j(i) = j$. Ces plus courts chemins définissent ainsi une arborescence $T^j = (V, A^j)$ enracinée en j avec $A^j = \{(\pi_j(i), i), i \in V \setminus \{j\}\}$. Le coût d'un secteur (U, j) est défini par $\sum_{i \in U} d_{i,j}$. D'autre part, pour un secteur (U, j), on a $i \in U \implies \pi_j(i) \in U$.

Dans la suite, si $T^j = (V, A^j)$ est une arborescence enracinée en $j \in V$ et $\forall i \in V \setminus \{j\}$, on notera par $T^j(i)$ la sous-arborescence de T^j enracinée en i et contenant tous les descendants de i. De plus, pour $U \subseteq V$ un sous-ensemble de sommets, on notera $T^j[U] = (U, A^j \cap (U \times U))$ la sous-arborescence de T^j enracinée en j et contenant les sommets de U. On dira que l'ensemble U induit une sous-arborescence *valide* si $j \in U$ et $T^j[U]$ forme une composante connexe dans T^j.

Définition 1 (secteur valide)**.** Étant donné une arborescence $T^j = (V, A^j)$ enracinée en j, un vecteur $\boldsymbol{d}_{\cdot,j}$ de $\mathbb{R}_+^n$, deux vecteurs $\boldsymbol{w}^1$, $\boldsymbol{w}^2$ de $\mathbb{N}_+^n$, et deux entiers W^1 et W^2, un *secteur valide de centre* j est une paire (U, j), telle que $T^j[U]$ est une sous-arborescence valide et telle que $\sum_{i \in U} w_i^\ell \leq W^\ell$ pour $\ell = 1, 2$. Le coût d'un secteur valide (U, j) est défini comme $c(U, j) = \sum_{i \in U} d_{i,j}$.

Problème 7 (couverture par secteurs valides de coût minimum)**.** Étant donné un ensemble V de sommets, deux vecteurs $\boldsymbol{w^1}$ et $\boldsymbol{w^2}$ de $\mathbb{N}_+^n$, deux entiers W^1 et W^2, un entier K, et pour chaque sommet $j \in V$, une arborescence T^j des plus courts chemins de j vers les autres sommets de V, et un vecteur $\boldsymbol{d}_{\cdot,j} \in \mathbb{R}^n$ de distances de j vers les sommets de V dans cette arborescence, trouver une couverture de V en K secteurs valides de centres différents qui minimise la somme des coûts des secteurs utilisés.

Le problème de couverture par secteurs valides est NP-difficile, car il est équivalent au problème de bin packing si on considère des arborescences de profondeur 1, des distances nulles et la capacité W^2 égale à $+\infty$.

2.2.2 Formulation compacte

Le problème de couverture par secteurs valides de coût minimum peut se modéliser à l'aide de la programmation linéaire en nombres entiers. Notre modèle repose sur les variables naturelles du problème (choix des centres et affectation). Soit $\boldsymbol{x} \in \{0,1\}^{n\times n}$ une matrice de variables de décision. Pour $i \neq j$, $x_{i,j}$ est égale à 1 si i est couvert par un secteur de centre j, 0 sinon. Par convention, $x_{j,j} = 1$ indique que j est utilisé comme centre d'un secteur. Le problème peut alors être modélisé de la manière suivante :

$$
(\mathrm{P}) = \begin{cases}
\min \displaystyle\sum_{j\in V} \sum_{i\in V\setminus\{j\}} d_{i,j}\, x_{i,j} & & & \\
\text{s.t.} \displaystyle\sum_{j\in V} x_{j,j} \leq K, & & & (2.1)\\
\displaystyle\sum_{j\in V} x_{i,j} \geq 1 & \forall i \in V, & & (2.2)\\
\displaystyle\sum_{i\in V} w_i^\ell\, x_{i,j} \leq W^\ell & \ell = 1,2, & \forall j \in V, & (2.3)\\
x_{i,j} \leq x_{\pi_j(i),j} & \forall i \in V, i \neq j, & \forall j \in V, & (2.4)\\
x_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V. & (2.5)
\end{cases}
$$

La fonction objectif est égale à la somme des distances de plus courts chemins. La contrainte (2.1) borne supérieurement le nombre de secteurs créés ; les contraintes (2.2) imposent la couverture de chaque sommet par au moins un secteur ; et les contraintes (2.3) assurent que les secteurs ne dépassent pas les capacités maximum. Les contraintes de précédence sont exprimées par (2.4).

2.3 Formulation étendue

Dans cette section, nous montrons comment construire une formulation étendue de (P) en utilisant la décomposition de Dantzig-Wolfe.

2.3.1 Décomposition du problème

Pour $j \in V$, on note $\mathcal{G}^j$ l'ensemble des points extrêmes de l'enveloppe convexe des solutions entières au système (2.3)–(2.5) associé à j. Nous pouvons réécrire le modèle compact (P) en exprimant la valeur du vecteur $\boldsymbol{x}$ comme une combinaison entière des éléments de $\mathcal{G}^j$. Étant donné $g \in \mathcal{G}^j$, soit $x^g_{i,j}$ la valeur de $x_{i,j}$ dans g. Soit $\boldsymbol{\lambda}_j \in \{0,1\}^{|\mathcal{G}^j|}$ un vecteur de variables binaires tel que λ^g_j est égal à 1 si le point extrême g est utilisé dans la solution, 0 sinon. Le nouveau modèle peut être défini comme suit.

$$
(\mathrm{P}^M) = \begin{cases}
\min \displaystyle\sum_{j\in V}\sum_{i\in V} d_{i,j} \left(\sum_{g\in\mathcal{G}^j} x^g_{i,j}\,\lambda^g_j\right) & & & & \\
\text{t.q.} \quad \displaystyle\sum_{j\in V}\left(\sum_{g\in\mathcal{G}^j} x^g_{j,j}\,\lambda^g_j\right) \le K, & & & (\kappa) & (2.6)\\
\displaystyle\sum_{j\in V}\left(\sum_{g\in\mathcal{G}^j} x^g_{i,j}\,\lambda^g_j\right) \ge 1 & \forall i \in V, & & (\boldsymbol{\phi}) & (2.7)\\
\displaystyle\sum_{g\in\mathcal{G}^j} \lambda^g_j = 1 & \forall j \in V, & & (\boldsymbol{\alpha}) & (2.8)\\
\lambda^g_j \in \{0,1\} & \forall g \in \mathcal{G}^j, & \forall j \in V. & & (2.9)
\end{cases}
$$

On appelle ce problème le problème maître. Le modèle obtenu possède un nombre polynomial de contraintes mais un nombre exponentiel de variables. Pour le résoudre efficacement, on a recours à une méthode de génération de colonnes. Pour résoudre la relaxation linéaire de P^M, on résout initialement une restriction de ce modèle à un sous-ensemble de variables (on parle de problème maître restreint). On doit ensuite résoudre un sous-problème pour déterminer la variable non générée avec le coût réduit le plus négatif. Ce sous-problème peut se décomposer en n sous-problèmes (un par centre possible). En associant les variables duales κ, ϕ_i pour tout $i \in V$ et α_j pour tout $j \in V$ respectivement aux contraintes (2.6), (2.7) et (2.8), nous obtenons pour chaque centre le problème de pricing suivant.

$$
(\mathrm{P}^j) = \begin{cases}
\min \displaystyle\sum_{i\in V} d_{i,j}\,x_{i,j} - \kappa\,x_{j,j} - \sum_{i\in V}\phi_i\,x_{i,j} - \alpha_j & & & \\
\Leftrightarrow \quad -\alpha_j - \max \quad \kappa\,x_{j,j} + \displaystyle\sum_{i\in V}(\phi_i - d_{i,j})x_{i,j} & & & \\
\text{t.q.} \quad \displaystyle\sum_{i\in V} w^\ell_i\,x_{i,j} \le W^\ell & \ell = 1,2 & & (2.10)\\
x_{i,j} \le x_{\pi_j(i),j} & \forall i \in V, i \ne j, & & (2.11)\\
x_{i,j} \in \{0,1\} & \forall i \in V. & & (2.12)
\end{cases}
$$

En considérant $(\phi_i - d_{i,j})$ comme étant un profit sur la couverture d'un sommet par un secteur, ce problème revient à chercher un secteur valide qui maximise le profit des sommets couverts. Notons qu'une colonne associée au centre j peut correspondre à la solution vide, consistant donc à ne pas utiliser j comme centre.

Pour obtenir une solution entière, on utilise une méthode de branch-and-price. Des éléments de cette méthode sont décrits ultérieurement. Dans la section suivante, nous nous concentrons sur la résolution de la relaxation linéaire du modèle (P^M). Le problème maître restreint est résolu à l'aide d'un solveur de programmation linéaire. La difficulté du problème réside principalement dans la résolution des sous-problèmes de génération de colonnes, nous proposons donc plusieurs algorithmes pour les résoudre efficacement.

Pour un centre $j \in V$ donné, le sous-problème de pricing (P^j) peut être vu comme une extension à deux dimensions du *Tree Knapsack Problem* [Luk74, CS97, CS98], lui-même étant une extension du problème classique de sac à dos.

Problème 8 (sac à dos avec contraintes de précédence en arborescence (TKP))**.** Étant donné une arborescence $T = (V, A)$ enracinée en j, un vecteur $\boldsymbol{p}$ de $\mathbb{R}^{|V|}$, un vecteur $\boldsymbol{w}$ de $\mathbb{N}^{|V|}$, ainsi qu'un entier $W \in \mathbb{N}$, trouver un sous-ensemble de sommets $U \subseteq V$ tel que U induit un sous-arbre de T enraciné en j, la capacité du sac n'est pas dépassée $\sum_{i \in U} w_i \leq W$ et qui maximise $\sum_{i \in U} p_i$.

Deux méthodes, basées respectivement sur une approche par branch-and-bound [CS98] et par programmation dynamique [CS97], on été proposées pour résoudre ce problème. Dans les sections suivantes, nous montrons comment adapter ces algorithmes à deux dimensions de sac à dos.

2.3.2 Résolution du problème de pricing par algorithme de branch-and-bound

L'algorithme de branch-and-bound proposé dans [CS98] repose sur un schéma de branchement classique consistant, pour un nœud de branchement donné, à sélectionner un sommet libre (qui n'est pas sujet à une décision de branchement antérieure) et à prendre la décision de l'ajouter ou de l'exclure de la solution partielle courante.

La spécificité de l'algorithme se trouve dans la méthode d'évaluation des nœuds de branchement, qui permet à la fois de calculer une borne duale valide pour la solution partielle et de sélectionner le prochain sommet sur lequel brancher (voir la sous-section 1.4.3).

D'un autre côté, le sous-problème (P^j) issu de la décomposition de Dantzig-Wolfe est un problème de sac à dos en deux dimensions avec contraintes de précédence en arborescence (TKP-2D). Soit $\boldsymbol{x} \in \{0,1\}^n$ un vecteur de variables de décision binaires tel que la variable x_i pour tout $i \in V$ est égale à 1 si le sommet i fait partie de la solution, et 0 sinon. Considérons $\boldsymbol{\mu} \in \mathbb{R}^2$ le vecteur de multiplicateurs Lagrangiens associés aux deux contraintes de sac à dos. La relaxation Lagrangienne du TKP-2D en fonction de $\boldsymbol{\mu}$ peut être modélisée comme suit.

$$\big(\mathrm{RL}_\mu(\text{TKP-2D})\big) = \begin{cases} \max \displaystyle\sum_{i \in V} (p_i - \mu_1 w_i^1 - \mu_2 w_i^2)\, x_i + \mu_1 W^1 + \mu_2 W^2 & \\ \text{t.q.} \quad x_i \leq x_{\pi(i)} & \forall i \in V, \qquad (2.13) \\ \qquad x_i \in \{0,1\} & \forall i \in V. \qquad (2.14) \end{cases}$$

La difficulté ici est que pour déterminer la prochaine solution $\boldsymbol{\mu}$ induisant un changement dans la solution optimale $\boldsymbol{x}^*$ du Lagrangien, il faut d'abord déterminer une direction dans laquelle faire évoluer le vecteur $\boldsymbol{\mu}$.

De manière générale, étant donné un vecteur de multiplicateurs Lagrangiens $\bar{\boldsymbol{\mu}} = \big(\bar{\mu}_1, \bar{\mu}_2\big)^\top$ et $\bar{\boldsymbol{x}}^*$ une solution optimale du Lagrangien de profit $\bar{p}$ et de poids $(\bar{w}^1, \bar{w}^2)$, la fonction Lagrangienne admet un sous-gradient ∇ en $\bar{\boldsymbol{\mu}}$ de valeur

$$\nabla_{\bar{\boldsymbol{\mu}}} = \big(\bar{w}^1 - W^2, \bar{w}^2 - W^2\big)^\top.$$

Si le Lagrangien admet une solution optimale unique, alors ce sous-gradient est également unique ; autrement dit, c'est un gradient du Lagrangien en $\bar{\boldsymbol{\mu}}$. En revanche, puisque tout nœud de branchement est associé à un vecteur $\bar{\boldsymbol{\mu}}$ tel qu'au moins un sous-arbre a un profit Lagrangien nul, il existe nécessairement plusieurs solutions optimales entières associées à $\bar{\boldsymbol{\mu}}$. Chaque solution optimale est associée à un sous-gradient, et toute combinaison convexe de ces sous-gradients est également un sous-gradient du Lagrangien en $\bar{\boldsymbol{\mu}}$.

Ainsi, nous pouvons adapter l'algorithme de branch-and-bound au cas avec deux dimensions de sac à dos en faisant évoluer le vecteur de multiplicateurs lagrangiens $\boldsymbol{\mu}$ en suivant la direction du sous-gradient associé à $\bar{\boldsymbol{x}}^*$ la solution optimale courante du Lagrangien.

Cette direction implique cependant une contre-partie. Rappelons que, dans le cas à une dimension, la valeur suivante $\bar{\mu}'$ du multiplicateur Lagrangien est obtenue en augmentant la valeur courante $\bar{\mu}$ de sorte que, étant donné T^* une solution optimale de $((\mathrm{RL}_{\bar{\mu}}(\mathrm{TKP}))$, il y ait au moins un sous-arbre de T^* de profit Lagrangien nul selon $\bar{\mu}'$, tandis que les autres sous-arbres de T^* ont un profit Lagrangien positif. Ainsi, prendre la décision d'exclure la racine de ce sous-arbre de la solution partielle revient à suivre l'évolution de la solution optimale du Lagrangien lorsque l'on passe de $\bar{\mu}$ à $\bar{\mu}'$. L'intérêt de cette décision est que la borne Lagrangienne calculée à partir du nouveau nœud de branchement est également valide pour le nœud de branchement parent. Cependant, dans le cas à deux dimensions de sac à dos, si la solution courante vérifie strictement l'une des deux contraintes de sac à dos, alors la direction de sous-gradient est négative sur la contrainte vérifiée. Autrement dit, cette direction consiste à diminuer la pénalité associée à cette contrainte. Soit $\bar{\boldsymbol{\mu}}'$ le nouveau vecteur de multiplicateurs lagrangiens, obtenu en modifiant $\bar{\boldsymbol{\mu}}$ dans la direction de sous-gradient $\nabla_{\bar{\boldsymbol{\mu}}}$. Supposons qu'il existe un sous-arbre T_i, dont le coût Lagrangien induit par $\bar{\boldsymbol{\mu}}$ est négatif, devienne positif avec $\bar{\boldsymbol{\mu}}'$ car l'une des pénalités a diminué. De fait, T_i n'est pas un sous-arbre de T^* le sous-arbre de coût Lagrangien maximal induit par $\bar{\boldsymbol{\mu}}$. Ainsi, aucune solution construite en retirant des objets de T^* ne peut être une solution optimale du Lagrangien, et la borne Lagrangienne calculée dans le nœud courant avec les valeurs $\bar{\boldsymbol{\mu}}'$ n'est pas valide pour le nœud de branchement parent.

Afin d'éviter ce cas de figure, nous projetons la direction du sous-gradient utilisé sur $\mathbb{R}^2_+$. Notons que si nous appliquons cette projection, la direction qui en résulte ne correspond plus à un sous-gradient du Lagrangien.

Une autre direction intéressante pour faire évoluer la valeur des multiplicateurs Lagrangiens consiste à ne considérer que la contrainte de sac à dos la plus violée, et à laisser l'autre pénalité inchangée. En général, cette direction ne correspond pas à un sous-gradient de la fonction Lagrangienne. En effet, soit $L(\boldsymbol{\mu})$ la valeur du Lagrangien en $\boldsymbol{\mu}$. Notons que dans notre cas, $L(\cdot)$ est une fonction convexe linéaire par partie. Une direction $\nabla \in \mathbb{R}^2$ est un sous-gradient de $L(\cdot)$ en $\boldsymbol{\mu}$ ssi pour tout $\boldsymbol{h} \in \mathbb{R}^2$ il existe $\epsilon > 0$ tel que

$$L(\boldsymbol{\mu} \pm \epsilon \boldsymbol{h}) - L(\boldsymbol{\mu}) \geq \pm \epsilon \nabla^\top \boldsymbol{h} \quad \Leftrightarrow \quad L(\boldsymbol{\mu} \pm \epsilon \boldsymbol{h}) \geq L(\boldsymbol{\mu}) \pm \epsilon \nabla^\top \boldsymbol{h}.$$

Supposons que $L(\boldsymbol{\mu})$ soit associée à exactement deux solutions optimales de poids respectivement $(4, 2)$ et $(3, 2)$ pour une capacité $(1, 1)$. La direction que nous considérons est alors $\nabla = \big(x, 0\big)^\top$, avec $x \in \{3, 2\}$ selon la solution utilisée pour calculer la direction, car la seconde contrainte de sac à dos est moins violée que la première. Or, pour $\boldsymbol{h} = \big(0, 1\big)^\top$,

il n'existe aucun $\epsilon > 0$ tel que

$$L(\boldsymbol{\mu} + \epsilon \boldsymbol{h}) = L(\boldsymbol{\mu}) - \epsilon \geq L(\boldsymbol{\mu}) + \epsilon \nabla^\top \boldsymbol{h} = L(\boldsymbol{\mu}).$$

Bien que cette seconde direction proposée ne corresponde généralement pas à un sous-gradient, elle offre de bonnes performances en pratique pour résoudre le problème de pricing dans l'algorithme de branch-and-price.

2.3.3 Résolution du problème de pricing par algorithme de programmation dynamique

Dans [CS97], les auteurs proposent de résoudre le *Tree Knapsack Problem* en temps pseudo-polynomial à l'aide d'un algorithme de programmation dynamique inspiré de [JN83]. Étant donné un arbre de précédence T, cet algorithme se base sur un ordre en profondeur d'abord (*depth-first search*, ou DFS) des sommets de T. Ainsi, l'algorithme intègre nativement les contraintes de précédence, impliquant qu'un sommet ne peut pas être couvert par une solution si la chaîne de ses prédécesseurs dans T n'est pas couverte également.

Pour résoudre le problème du TKP en deux dimensions, nous avons simplement ajouté une dimension supplémentaire à l'espace des états du programme dynamique de [CS97]. Chaque état du programme est donc défini par le couple $(k, \boldsymbol{w}) = \big(k, (w^1, w^2)\big)$. De la même manière, le label $\psi_L^{\text{2D}}(k, \boldsymbol{w})$ correspond au profit maximal d'une solution limitée aux sommets de L, contenant le sommet k, et n'excédant pas les capacités w^1 et w^2.

De plus, nous utilisons des bornes duales afin de filtrer les labels sous-optimaux. À un label $\psi_L^{\text{2D}}(k, \boldsymbol{w})$ nous associons $CB_L(k, \boldsymbol{w})$ une borne supérieure complémentaire sur le profit maximum pouvant être atteint à l'aide des sommets restants dans $V \setminus L$ et tenant compte de la capacité résiduelle $\boldsymbol{W} - \boldsymbol{w}$. Notons que puisque nous utilisons un ordre DFS et que l'état $(k, \boldsymbol{w})$ suppose la présence du sommet k dans la solution, chacun des sommets de $V \setminus L$ est éligible pour être ajouté à la solution, quelle que soit la composition de celle-ci. Étant donné LB une borne inférieure sur la solution optimale de l'instance TKP-2D, nous pouvons filtrer le label $\psi_L^{\text{2D}}(k, \boldsymbol{w})$ si nous disposons d'une borne $CB_L(k, \boldsymbol{w})$ telle que

$$\psi_L^{\text{2D}}(k, \boldsymbol{w}) + CB_L(k, \boldsymbol{w}) \leq LB.$$

Dans notre implémentation, les bornes de complétion $CB_L(k, \boldsymbol{w})$ sont obtenues en utilisant deux relaxations différentes. La première est la relaxation continue du sac à dos en une dimension. Dans cette relaxation, nous autorisons une couverture fractionnaire des sommets et relâchons les contraintes de précédence ainsi qu'une contrainte de capacité. Le calcul de cette relaxation peut être fait en $\mathcal{O}(|V|)$ pour pré-calculer la séquence des sommets critiques pour chaque dimension, puis en $\mathcal{O}(\log_2 |V|)$ par label pour trouver le sommet critique correspondant à la capacité résiduelle $W^\ell - w^\ell$ du label. Nous utilisons également une relaxation lagrangienne de la contrainte de sac à dos du TKP pour compléter les labels. Dans un premier temps, nous relâchons complètement une des contraintes de capacité pour revenir au cas à une seule dimension ℓ, comme dans la relaxation précédente. Le problème obtenu est identique à $(\text{RL}_\mu(\text{TKP}))$ limité aux sommets de $V \setminus L$ et à une capacité résiduelle $W^\ell - w^\ell$. Ensuite, nous calculons une sous-forêt de $V \setminus L$ telle que le profit Lagrangien est maximisé en fonction de la capacité résiduelle du label. Cette nouvelle borne demande un temps de pré-calcul en $\mathcal{O}(|V|^2)$, puis, à nouveau, une

recherche en $\mathcal{O}(\log_2 |V|)$ pour trouver la sous-forêt correspondant à la capacité résiduelle.

2.4 Application de méthodes d'agrégation de contraintes au problème de sectorisation

Dans la section précédente, nous avons présenté une formulation étendue (P^M) ainsi que deux oracles pour résoudre les problèmes de pricing (P^j). Des tests préliminaires ont montré que, pendant l'exécution de l'algorithme de branch-and-price, il y avait un nombre conséquent d'itérations de génération de colonnes dégénérées, c'est-à-dire des itérations pendant lesquelles la valeur optimale du problème maître restreint n'évolue pas, malgré les nouvelles colonnes.

Nous avons présenté dans la section 1.1 et la section 1.2 plusieurs méthodes issues de la littérature scientifique visant à réduire le phénomène de dégénérescence, notamment dans le contexte de la génération de colonnes. En particulier, la section 1.2 est dédiée à des méthodes basées sur la notion de réduction de base et d'agrégation de contraintes. L'agrégation de contraintes est une méthode cherchant à pallier les problèmes de dégénérescence dans les programmes linéaires. Celle-ci a été adaptée au contexte de la résolution de programmes linéaires avec la méthode de Pan [Pan98] et l'algorithme *Improved Primal Simplex* (IPS [EMDS11]). Elle a également été appliquée aux méthodes de génération de colonnes pour des problèmes de partitionnement (*set partitionning problems*) avec l'algorithme de *Dynamic Constraint Aggregation* (DCA [EVSD05, EMSD10, BDD12]).

Dans cette section, nous montrons comment adapter le schéma d'agrégation de DCA à notre problème. Nous présentons également les différences entre ce schéma d'agrégation et un schéma alternatif inspiré de l'algorithme IPS.

2.4.1 Réduction de la dégénérescence par contraintes d'agrégation

Nous définissons une *agrégation* $\mathcal{A} = \{V_1, V_2, \ldots, V_k\}$ comme étant un ensemble de k *agrégats* disjoints formant une partition de V, c'est-à-dire $V_i \cap V_j = \emptyset \quad \forall V_i, V_j \in \mathcal{A}$ et $\bigcup(V_i \in \mathcal{A}) = V$.

Le schéma d'agrégation que nous proposons est une variante de celui utilisé par l'algorithme DCA, comme nous allons le voir. En effet, dans DCA, les agrégats sont équivalents aux secteurs d'une solution réalisable entière du problème de *set-partitioning*. Nous utilisons une approche similaire, puisque nous construisons l'agrégation à partir d'un sous-ensemble des colonnes de (P^M). Plus spécifiquement, étant donné B la base courante du problème maître restreint obtenue après la dernière itération de génération de colonnes, nous considérons les couvertures correspondant aux variables dont la valeur est strictement positive dans la solution basique $\boldsymbol{x}_{\mathcal{B}}$. Cependant, cela pourrait impliquer la présence d'un même sommet dans plusieurs agrégats (soit parce que la solution basique est fractionnaire, soit parce que certains sommets sont sur-couverts). Afin d'éviter cela, nous nous autorisons à décomposer les agrégats afin d'obtenir des ensembles de sommets disjoints, tout en cherchant à générer le moins d'agrégats possibles. Autrement dit, nous cherchons à orthogonaliser la base B. Étant donné deux agrégats V_i et V_j, si $V_i \cap V_j \neq \emptyset$,

alors nous les décomposons en trois agrégats V'_i, V'_j et V_k tels que $V'_i = V_i \setminus V_j$, $V'_j = V_j \setminus V_i$ et $V_k = V_i \cap V_j$. Cette opération est effectuée itérativement jusqu'à ce que tous les agrégats soient disjoints.

Étant donné une agrégation $\mathcal{A}$ des sommets de V, nous disons qu'un secteur (U, j) est *compatible* avec $\mathcal{A}$ si, pour tout agrégat $V_i \in \mathcal{A}$, U couvre tous les sommets composant V_i, ou bien U ne couvre aucun des sommets de V_i. Formellement, étant donné $\mathcal{S}$ l'ensemble des secteurs valides — qui respectent les contraintes des sous-problèmes —, nous définissons $\mathcal{C}$ l'ensemble des secteurs valides compatibles avec $\mathcal{A}$ comme étant un sous-ensemble de $\mathcal{S}$ de la manière suivante.

$$\mathcal{C} = \big\{(U, j) \in \mathcal{S} \text{ et } (U \cap V_i) \in \{\emptyset, V_i\} \ \forall V_i \in \mathcal{A}\big\}.$$

L'ensemble complémentaire de $\mathcal{C}$ est noté $\mathcal{I}$ et contient tous les secteurs valides incompatibles avec l'agrégation $\mathcal{A}$. Par extension, nous disons également qu'une colonne de (P^M) est compatible (resp. incompatible) avec $\mathcal{A}$ si elle correspond à un secteur compatible (resp. incompatible) avec $\mathcal{A}$.

Soit (A^M) le problème maître *agrégé* équivalent au problème maître original (P^M) restreint aux colonnes compatibles. Remarquons que toutes les contraintes de couverture (2.7) associées aux sommets d'un même agrégat V_i sont désormais identiques ; elles induisent donc un problème de désagrégation duale similaire à DCA. Puisque les contraintes de couverture associées à V_i sont identiques, il est possible de ne conserver qu'une seule de ces contraintes pour définir (A^M). Notons $\bar{v}_i$ le sommet représentatif de l'agrégat V_i, c'est-à-dire le sommet associé à la seule contrainte de couverture conservée dans (A^M). Notons également $\mathcal{G}^j_{\mathcal{C}} \subseteq \mathcal{G}^j$ l'ensemble des points de $\mathcal{G}^j$ correspondant à un secteur compatible avec $\mathcal{A}$. Nous pouvons alors définir (A^M) de la manière suivante.

$$(\mathrm{A}^M) = \begin{cases} \min \displaystyle\sum_{j \in V} \sum_{i \in V} d_{i,j} \left(\sum_{g \in \mathcal{G}^j_{\mathcal{C}}} x^g_{i,j}\, \lambda^g_j \right) & & & \\ \text{t.q.} \quad \displaystyle\sum_{j \in V} \left(\sum_{g \in \mathcal{G}^j_{\mathcal{C}}} x^g_{j,j}\, \lambda^g_j \right) \leq K, & & & (\kappa) \\ \displaystyle\sum_{j \in V} \left(\sum_{g \in \mathcal{G}^j_{\mathcal{C}}} x^g_{\bar{v}_i,j}\, \lambda^g_j \right) \geq 1 & \forall V_i \in \mathcal{A}, & & (\boldsymbol{\Phi}) \\ \displaystyle\sum_{g \in \mathcal{G}^j_{\mathcal{C}}} \lambda^g_j = 1 & \forall j \in V, & & (\boldsymbol{\alpha}) \\ \lambda^g_j \in \{0, 1\} & \forall g \in \mathcal{G}^j_{\mathcal{C}}, & \forall j \in V. & \end{cases} \tag{2.15}$$

Le problème agrégé (A^M) étant défini par moins de contraintes que le problème original (P^M), les bases de (A^M) sont de taille inférieure à celles de (P^M), ce qui diminue la probabilité d'avoir une base dégénérée. Notons également que toute solution réalisable pour (A^M) est également réalisable pour (P^M), alors qu'une solution de (P^M) utilisant des colonnes incompatibles ne sera pas applicable à (A^M).

Soit $\boldsymbol{\lambda}^{(t)}$ et $(\kappa^{(t)}, \boldsymbol{\Phi}^{(t)}, \boldsymbol{\alpha}^{(t)})$ les solutions basiques primale et duale de (A^M) obtenues à l'itération t de l'algorithme du simplexe primal. Soit $\bar{\mathcal{G}}^j$ l'ensemble des colonnes associées au centre $j \in V$ qui n'ont pas encore générées et ne font donc pas partie du problème maître restreint. Le problème de désagrégation duale (DPP) consiste à trouver une solution duale désagrégée $(\kappa, \boldsymbol{\phi}, \boldsymbol{\alpha})$ de (P^M) telle que les colonnes basiques de $\boldsymbol{\lambda}^{(t)}$ conservent leur coût réduit nul tandis que le plus petit coût réduit associé à $\bar{\mathcal{G}}$ est maximisé. Nous modélisons ce problème par le programme linéaire suivant.

$$
(\mathrm{DDP}) = \begin{cases}
\max\limits_{\mu,\boldsymbol{\phi}} \quad \mu & & \\
\text{t.q.} \quad \mu \leq \sum\limits_{i \in V} d_{i,j}\, x^g_{i,j} - \kappa^{(t)}\, x^g_{j,j} & & (2.16) \\
\qquad\quad - \sum\limits_{i \in V} \phi_i\, x^g_{i,j} - \alpha^{(t)}_j & \forall g \in \bar{\mathcal{G}}^j, \forall j \in V, & (2.17) \\
\sum\limits_{i \in V_j} \phi_i = \Phi^{(t)}_j & \forall V_j \in \mathcal{A}, & (2.18) \\
\phi_i \geq 0 & \forall i \in V. & (2.19)
\end{cases}
$$

(DDP) est donc équivalent au problème dual de (P^M) projeté sur les contraintes d'agrégation (2.18). Notons que, pour simplifier le problème, nous avons fait le choix de ne pas désagréger les variables duales κ et $\boldsymbol{\alpha}$, qui conservent donc la valeur qu'elles avaient dans la solution duale agrégée.

Une autre manière de concevoir un schéma d'agrégation consiste à appliquer la méthode IPS (*Improved Primal Simplex*, [EMDS11]). Il peut être intéressant de noter que certaines propriétés, intrinsèques au framework IPS, ne sont pas vérifiées par notre schéma d'agrégation. Considérons $(\mu^*, \boldsymbol{\phi}^*)$ une solution optimale de (DDP) de valeur μ^*, et soit $\bar{\mathcal{G}}^*$ l'ensemble des colonnes dont le coût réduit induit par $\boldsymbol{\phi}^*$ est égal à μ^*. Dans IPS, si la valeur μ^* est strictement négative, alors cet ensemble $\bar{\mathcal{G}}^*$ doit permettre de construire une nouvelle solution réalisable de (P^M) strictement meilleure que $\boldsymbol{\lambda}^{(t)}$ en effectuant une (ou plusieurs) opération de pivot. En revanche, si $\mu^* \geq 0$, cela suffit à prouver que la solution $\boldsymbol{\lambda}^{(t)}$ obtenue en résolvant (A^M) est également optimale pour (P^M). Le schéma d'agrégation que nous appliquons ici n'offre pas de telles garanties, car nous ne modifions pas la valeur des variables duales κ et $\boldsymbol{\alpha}$ liées aux contraintes additionnelles du problème maître. Il est ainsi possible que la solution optimale de (DDP) soit de valeur $\mu^* < 0$, même si la solution primale $\boldsymbol{\lambda}^{(t)}$ est optimale pour (P^M). En d'autres termes, les contraintes d'agrégation (2.18) ne sont pas valides pour le dual de (P^M). De la même manière, il est possible de trouver un ensemble de colonnes $\mathcal{G}^*$ ayant un coût réduit négatif, quelle que soit la désagrégation des variables duales $\boldsymbol{\Phi}$, sans que cela n'induise de pivot non-dégénéré dans le problème maître original (P^M). Nous proposons une analyse plus détaillée de ces particularités dans la section suivante.

2.4.2 Analyse des différences entre les méthodes DCA et IPS appliquées à notre problème

Dans la section précédente, nous avons étudié une application de la méthode d'agrégation de l'algorithme DCA sur un problème de sectorisation. Cependant, dans [GDL16], les auteurs proposent de considérer l'algorithme DCA comme un cas particulier de l'algorithme IPS [EMDS11] appliqué à un problème de *set-partitioning*. Dans la même perspective, nous consacrons cette section aux différences entre le schéma d'agrégation de DCA et celui d'IPS appliqués à notre problème, qui est un problème de couverture sujet à des contraintes additionnelles.

2.4.2.1 Problème de partitionnement et problème de couverture

L'algorithme DCA a été initialement proposé pour un problème de *set-partitioning* [EVSD05]. Étant donné $n \in \mathbb{N}$, soit $V = \{v_1, v_2, \dots, v_n\}$ un ensemble de n éléments à couvrir. Pour couvrir V, nous disposons d'un ensemble $\mathcal{S} = \{V_1, V_2, \dots, V_m\}$ de m couvertures, avec $m \in \mathbb{N}$ et $V_i \subseteq V$ pour tout $i = 1, \dots, m$. Soit $A \in \{0,1\}^{m \times n}$ la matrice binaire telle que $a_{ij} = 1$ ssi $v_i \in V_j$, et soit $\boldsymbol{x} \in \{0,1\}^m$ un vecteur de variables binaires tel que x_i est égale à 1 si la solution utilise la couverture V_i, et $x_i = 0$ sinon. Nous utilisons la notation $\mathbf{0}_d$ (resp. $\mathbf{1}_d$) pour représenter le vecteur dont les $d \in \mathbb{N}$ composants sont tous égaux à 0 (resp. 1), ainsi que la notation analogue $\mathbf{0}_{d_1 \times d_2}$ pour les matrices. Nous pouvons alors formuler le problème de *set-partitioning* de la manière suivante.

$$\text{(set-PP)} = \begin{cases} \min \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{t.q.} \quad A\boldsymbol{x} = \mathbf{1}_n, & (2.20) \\ \qquad \boldsymbol{x} \in \{0,1\}^m. & (2.21) \end{cases}$$

Une solution $\boldsymbol{x}$ de (set-PP) est réalisable ssi chaque élément $v_i \in V$ est couvert par une et une seule couverture V_j. Le problème de *set-covering* est une relaxation du problème de *set-partitioning* dans lequel chaque élément $v_i \in V$ doit être couvert par *au moins* une couverture. Cela revient alors à relâcher le système d'égalités $A\boldsymbol{x} = 1$ en système d'inégalités $A\boldsymbol{x} \geq 1$. Du point de vue de l'algorithme du simplexe, ce changement se traduit par l'ajout de variables d'écart aux contraintes de partitionnement. Notons I_d la matrice identité de taille $d \times d$ et soit $\boldsymbol{s} \in \mathbb{R}^n_+$ un vecteur de variables d'écart. Le problème *set-covering* peut être formulé par le MIP suivant.

$$\text{(set-CP)} = \begin{cases} \min \boldsymbol{c}^\top \boldsymbol{x} & \\ \text{t.q.} \quad A\boldsymbol{x} - I_n \boldsymbol{s} = 1, & (2.22) \\ \qquad \boldsymbol{x} \in \{0,1\}^m, & (2.23) \\ \qquad \boldsymbol{s} \geq \mathbf{0}_n. & (2.24) \end{cases}$$

Nous allons maintenant rappeler brièvement comment agréger le problème (set-PP) à l'aide de l'algorithme IPS, puis présenter la différence obtenue lorsque la même méthode est appliquée à (set-CP). Soit $A_{\mathcal{F}}$ une sous-matrice de A composée de $f \leq n$ colonnes associées à une partition de V. Formellement, nous avons $A_{\mathcal{F}} \cdot \mathbf{1}_f = \mathbf{1}_n$. Nous considérons généralement que $A_{\mathcal{F}}$ est composée des colonnes associées aux variables égales à 1 dans

une base réalisable entière de (set-PP) donnée. Nous pouvons également considérer, sans perte de généralité, que les contraintes sont ordonnées de sorte que les f premières lignes de $A_{\mathcal{F}}$ forment la sous-matrice $A_{\mathcal{P}\mathcal{F}} = I_f$. Notons $A_{\mathcal{Z}\mathcal{F}}$ la sous-matrice de $A_{\mathcal{F}}$ composée des $(n-f)$ lignes restantes. Nous pouvons également construire une base B de (set-PP) en complétant $A_{\mathcal{F}}$ à l'aide de colonnes artificielles de la manière suivante.

$$A_{\mathcal{F}} = \begin{pmatrix} A_{\mathcal{P}\mathcal{F}} \\ A_{\mathcal{Z}\mathcal{F}} \end{pmatrix} \qquad B = \begin{pmatrix} A_{\mathcal{P}\mathcal{F}} & \mathbf{0}_{f\times(n-f)} \\ A_{\mathcal{Z}\mathcal{F}} & I_{n-f} \end{pmatrix} = \begin{pmatrix} I_f & \mathbf{0}_{f\times(n-f)} \\ A_{\mathcal{Z}\mathcal{F}} & I_{n-f} \end{pmatrix}$$

Notons que la matrice $A_{\mathcal{Z}\mathcal{F}} \in \{0,1\}^{(n-f)\times f}$ contient exactement un élément à 1 par ligne. L'étape suivante consiste à transformer (set-PP) en un problème équivalent défini par le système de contraintes $B^{-1}A\boldsymbol{x} = B^{-1}b$. Cette transformation permet notamment de construire un sous-problème, originellement appelé *problème complémentaire* [EMDS11], dans l'algorithme IPS.

Nous allons maintenant nous intéresser à un problème intermédiaire dans la construction du problème complémentaire. Ce problème intermédiaire nous permet d'établir un lien simple avec l'algorithme DCA. Notons $\boldsymbol{\phi} = \begin{pmatrix} \boldsymbol{\phi}_{\mathcal{P}} & \boldsymbol{\phi}_{\mathcal{Z}} \end{pmatrix}^\top$ le vecteur de variables duales associées aux contraintes $A\boldsymbol{x} = b$ de (set-PP), et $\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\Phi}_{\mathcal{P}} & \boldsymbol{\Phi}_{\mathcal{Z}} \end{pmatrix}^\top$ le vecteur de variables duales associées aux contraintes transformées $B^{-1}A\boldsymbol{x} = B^{-1}b$. Les vecteurs $\boldsymbol{\phi}$ et $\boldsymbol{\Phi}$ sont de même taille et l'algorithme IPS construit une solution $\boldsymbol{\phi}$ à partir de $\boldsymbol{\Phi}$ de la manière suivante.

$$\begin{aligned} \boldsymbol{\phi}_{\mathcal{P}}^\top &= \boldsymbol{\Phi}_{\mathcal{P}}^\top A_{\mathcal{P}\mathcal{F}}^{-1} - \boldsymbol{\Phi}_{\mathcal{Z}}^\top A_{\mathcal{Z}\mathcal{F}} A_{\mathcal{P}\mathcal{F}}^{-1}, \\ \boldsymbol{\phi}_{\mathcal{Z}}^\top &= \boldsymbol{\Phi}_{\mathcal{Z}}^\top. \end{aligned}$$

Dans le cas du problème de *set-partitioning*, puisque $A_{\mathcal{P}\mathcal{F}} = I_f$, cela donne le système suivant.

$$\boldsymbol{\phi}_{\mathcal{P}}^\top = \boldsymbol{\Phi}_{\mathcal{P}}^\top - \boldsymbol{\Phi}_{\mathcal{Z}}^\top A_{\mathcal{Z}\mathcal{F}}, \tag{2.25}$$

$$\boldsymbol{\phi}_{\mathcal{Z}}^\top = \boldsymbol{\Phi}_{\mathcal{Z}}^\top. \tag{2.26}$$

Du point de vue de DCA, les colonnes $A_{\mathcal{F}}$ forment une solution entière — potentiellement sous-optimale — de (set-PP) et sont utilisées pour former une agrégation des éléments de V. Ainsi, chaque colonne A_j de $A_{\mathcal{F}}$ forme un agrégat d'éléments. La partition $\{\mathcal{P}, \mathcal{Z}\}$ des contraintes revient à choisir un élément « représentatif » par agrégat. Nous pouvons ainsi retrouver le problème de désagrégation duale associé à l'algorithme DCA, en considérant que la solution duale agrégée est représentée par le vecteur $\boldsymbol{\Phi}_{\mathcal{P}}$, tandis que les équations (2.25) représentent les contraintes d'agrégation assurant la cohérence des variables désagrégées $\boldsymbol{\phi}$.

Nous allons maintenant tenter d'appliquer le même procédé pour le problème de *set-covering*. Dans ce cas, une solution réalisable peut couvrir tout élément de V plusieurs fois, à condition que chaque élément soit couvert au moins une fois. Comme précédemment, considérons $A_{\mathcal{F}}$ la sous-matrice de A correspondant aux variables associées à une valeur strictement positive selon une base entière de (set-CC). Si aucun élément de V n'est sur-couvert par $A_{\mathcal{F}}$, alors nous pouvons procéder comme si nous étions dans un problème de *set-partitioning*. Sinon, il existe au moins un élément v_i couvert par plusieurs colonnes de $A_{\mathcal{F}}$. Notons que, le cas échéant, $A_{\mathcal{F}}$ contient la colonne correspondant à la variable d'écart s_i associée à v_i. De nouveau, nous pouvons compléter $A_{\mathcal{F}}$ à l'aide de colonnes artificielles.

$$B = \begin{pmatrix} A_{\mathcal{P}\mathcal{F}} & \mathbf{0}_{f\times(n-f)} \\ A_{\mathcal{Z}\mathcal{F}} & I_{n-f} \end{pmatrix} \quad B^{-1} = \begin{pmatrix} A_{\mathcal{P}\mathcal{F}} & \mathbf{0}_{f\times(n-f)} \\ -A_{\mathcal{Z}\mathcal{F}} & I_{n-f} \end{pmatrix} \quad \bar{B} = \left(\begin{array}{cccc|cc} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ 1 & & 1 & -1 & & \\ \hline & 1 & & & 1 & \\ & 1 & & & & 1 \\ & & 1 & & & & 1 \end{array}\right)$$

La matrice $\bar{B}$ est un exemple de base pouvant être obtenue de cette façon. Notons que la sous-matrice $A_{\mathcal{P}\mathcal{F}}$ est orthogonale, comme dans le cas du *set-partitioning*. Si nous reprenons la formule de désagrégation de $\mathbf{\Phi}$ précédente, nous obtenons les égalités suivantes.

$$\boldsymbol{\phi}_{\mathcal{P}}^{\top} = \mathbf{\Phi}_{\mathcal{P}}^{\top} A_{\mathcal{P}\mathcal{F}}^{-1} - \mathbf{\Phi}_{\mathcal{Z}}^{\top} A_{\mathcal{Z}\mathcal{F}} A_{\mathcal{P}\mathcal{F}}^{-1} = \left(\mathbf{\Phi}_{\mathcal{P}}^{\top} - \mathbf{\Phi}_{\mathcal{Z}}^{\top} A_{\mathcal{Z}\mathcal{F}}\right) A_{\mathcal{P}\mathcal{F}}^{-1} = \left(\mathbf{\Phi}_{\mathcal{P}}^{\top} - \mathbf{\Phi}_{\mathcal{Z}}^{\top} A_{\mathcal{Z}\mathcal{F}}\right) A_{\mathcal{P}\mathcal{F}}. \quad (2.27)$$

Étant donné l'agrégat V_j correspondant à la colonne A_j, notons $\bar{v}_j$ l'élément « représentatif » de V_j et ξ_j l'ensemble des éléments sur-couverts qui font partie de V_j. En développant l'équation (2.27), la désagrégation duale prend la forme

$$\phi_{\bar{v}_j} = \Phi_{\bar{v}_j} - \mathbf{\Phi}_{\mathcal{Z}}^{\top} (A_{\mathcal{Z}\mathcal{F}})_{\cdot,j} + \sum_{i\in\xi_j} \Phi_i.$$

Autrement dit, la variable duale associée à un élément sur-couvert contribue dans la désagrégation de chaque agrégat auquel cet élément appartient. De plus, étant donné un élément sur-couvert v_j, nous avons

$$\phi_j = -\Phi_j.$$

De ce fait, la valeur désagrégée correspondant à l'élément sur-couvert v_j est égale à l'opposé de sa valeur agrégée.

Dans notre contexte de génération de colonnes, le théorème des écarts complémentaires implique que $\Phi_j = 0$ pour tout élément v_j sur-couvert, et l'équation $\phi_j = -\Phi_j$ ne viole donc pas les contraintes de positivité sur les variables duales. Cela signifie donc que le problème de désagrégation duale n'est pas altéré par l'utilisation des contraintes de couvertures. Notons cependant que notre méthode d'agrégation diffère un peu de l'application que nous venons de présenter, dans le sens où chaque élément sur-couvert devrait normalement faire partie d'un agrégat indépendant, dans le framework IPS, cet agrégat correspondant à la variable d'écart associée à l'élément sur-couvert. Dans nos travaux, nous ne tenons pas compte des variables d'écart, ce qui peut mener au regroupement de plusieurs sommets sur-couverts dans un même agrégat, à condition que ces sommets soient tous couverts par le même ensemble de colonnes dans la solution basique courante.

L'utilisation de contraintes de couverture a tout de même un effet sur le problème complémentaire d'IPS. En effet, une propriété fondamentale de l'algorithme est que toute désagrégation duale optimale permet de trouver un ensemble compatible de colonnes permettant d'effectuer une opération de pivot non-dégénéré. Dans le contexte du problème de *set-partitioning*, une couverture V_i est compatible avec une agrégation $\mathcal{A}$ si, pour tout agrégat $V_j \in \mathcal{A}$, V_i couvre tous les éléments de V_j, ou bien n'en couvre aucun. Réciproquement, s'il existe un agrégat $V_j \in \mathcal{A}$ couvert partiellement par V_i, alors V_i est incompatible avec l'agrégation $\mathcal{A}$. Ainsi, la résolution du problème de désagrégation dual permet soit de trouver une colonne compatible de coût réduit négatif, soit de trouver un ensemble de colonnes tel que chaque colonne est individuellement incompatible,

mais leur combinaison est compatible avec l'agrégation $\mathcal{A}$. Cet ensemble est constitué des colonnes incompatibles de coût réduit minimum, selon la solution duale désagrégée $\boldsymbol{\phi}$. Une manière de le voir consiste à considérer, par contradiction, que l'ensemble des colonnes de coût réduit minimum couvre un élément $v_i \in V_j$ mais qu'il existe un autre élément $v_{i'} \in V_j$ qui n'est pas couvert. Dans le problème de *set-partitioning*, les variables duales $\boldsymbol{\phi}$ sont définies dans $\mathbb{R}$, il est alors toujours possible de transférer une quantité ϵ de la valeur duale associée à v_i vers $v_{i'}$ et ainsi augmenter le coût réduit d'au moins une des colonnes de l'ensemble. Si nous pouvons faire cela pour chaque colonne de l'ensemble, alors nous avons pu faire augmenter le coût réduit global, et la solution duale désagrégée $\boldsymbol{\phi}$ n'était pas optimale. Cette opération n'est pas forcément applicable dans un problème de *set-covering*, car les variables duales $\boldsymbol{\phi}$ doivent être positives. Il est alors possible d'avoir une colonne incompatible (au sens de DCA) avec un coût réduit négatif dans une solution duale désagrégée optimale. Cela signifie que la notion de compatibilité est altérée par l'utilisation de contraintes de couverture. En effet, puisqu'il est possible de sur-couvrir un élément, il est tout à fait possible d'effectuer une opération de pivot améliorant strictement la valeur de la solution courante en sur-couvrant une partie d'un agrégat.

2.4.2.2 Contraintes additionnelles dans le maître

Une autre différence entre le problème de *set-partitioning* et notre problème est que nous devons vérifier d'autres familles de contraintes, en plus des contraintes de couverture. Plus spécifiquement, notre problème est sujet à une contrainte de cardinalité, qui implique que nous ne pouvons pas utiliser plus de K secteurs, et des contraintes de convexité, issues de la décomposition de Dantzig-Wolfe.

La variable duale associée à la contrainte de cardinalité est notée κ. Soit $\bar{\kappa}$ la valeur de cette variable après désagrégation. En ajoutant la contrainte de cardinalité, nous avons deux cas de figure.

En premier, considérons que $f < K$, ce qui signifie que la variable d'écart associée à la contrainte de cardinalité a une valeur strictement positive dans la solution basique, et supposons que cette variable d'écart correspond à la dernière colonne de $A_{\mathcal{F}}$. Dans ce cas, l'indice de la contrainte de cardinalité appartient à $\mathcal{P}$ dans la partition $(\mathcal{P}, \mathcal{Z})$ des contraintes. Notons $\mathcal{P}'$ l'ensemble des indices des contraintes de partitionnement de $\mathcal{P}$, c'est-à-dire l'ensemble des indices de $\mathcal{P}$ excepté celui de la contrainte de cardinalité. La matrice $A^{-1}_{\mathcal{P}\mathcal{F}}$ vérifie donc la structure suivante.

$$A_{\mathcal{P}\mathcal{F}} = \begin{pmatrix} & & 0 \\ & I_{f-1} & \vdots \\ & & 0 \\ 1 & \cdots 1 & 1 \end{pmatrix} \quad \Rightarrow \quad A^{-1}_{\mathcal{P}\mathcal{F}} = \begin{pmatrix} & & 0 \\ & I_{f-1} & \vdots \\ & & 0 \\ -1 & \cdots -1 & 1 \end{pmatrix}$$

Dans les matrices ci-dessus, la dernière ligne correspond à la contrainte de cardinalité. De nouveau, nous pouvons développer (2.25) pour produire les équations de désagrégation suivante.

$$\begin{pmatrix} \boldsymbol{\phi}_{\mathcal{P}'} \\ \bar{\kappa} \end{pmatrix}^\top = \left(\begin{pmatrix} \boldsymbol{\Phi}_{\mathcal{P}'} \\ \kappa \end{pmatrix}^\top - \boldsymbol{\Phi}_{\mathcal{Z}}^\top A_{\mathcal{Z}\mathcal{F}} \right) A_{\mathcal{P}\mathcal{F}}^{-1} = \left(\left(\boldsymbol{\Phi}_{\mathcal{P}'}^\top - \boldsymbol{\Phi}_{\mathcal{Z}}^\top A_{\mathcal{Z}\mathcal{F}} - \kappa \mathbf{1}_f^\top \right) \quad \kappa \right) \tag{2.28}$$

Étant donné l'agrégat V_j correspondant à la colonne A_j, notons $\bar{v}_j$ l'élément « représentatif » de V_j. Nous pouvons reformuler les équations de désagrégation (2.28) sous une forme plus claire et observer que la variable κ intervient dans chaque contrainte de désagrégation.

$$\phi_{\bar{v}_j} = \Phi_{\bar{v}_j} - \boldsymbol{\Phi}_{\mathcal{Z}}^\top (A_{\mathcal{Z}\mathcal{F}})_{\cdot,j} - \kappa, \tag{2.29}$$

$$\bar{\kappa} = \kappa. \tag{2.30}$$

Considérons désormais que $f = K$ et que l'ensemble $\mathcal{Z}$ contient la contrainte de cardinalité. Nous allons montrer que les équations de désagrégation sont les mêmes dans ce cas de figure. Notons $\mathcal{Z}'$ l'ensemble des contraintes de partitionnement dans $\mathcal{Z}$. Nous pouvons supposer que la contrainte de cardinalité est représentée par la dernière ligne de la matrice $A_{\mathcal{Z}\mathcal{F}}$. Développer les contraintes d'agrégation (2.25) induit alors le système d'équations suivant.

$$\boldsymbol{\phi}_{\mathcal{P}}^\top = \left(\boldsymbol{\Phi}_{\mathcal{P}}^\top - \begin{pmatrix} \boldsymbol{\Phi}_{\mathcal{Z}'} \\ \kappa \end{pmatrix}^\top (A_{\mathcal{Z}\mathcal{F}}) \right) A_{\mathcal{P}\mathcal{F}}^{-1} \tag{2.31}$$

$$= \left(\boldsymbol{\Phi}_{\mathcal{P}}^\top - \begin{pmatrix} \boldsymbol{\Phi}_{\mathcal{Z}'} \\ \kappa \end{pmatrix}^\top \begin{pmatrix} A_{\mathcal{Z}'\mathcal{F}} \\ 1 \quad \cdots \quad 1 \end{pmatrix} \right) I_f \tag{2.32}$$

$$= \boldsymbol{\Phi}_{\mathcal{P}}^\top - \boldsymbol{\Phi}_{\mathcal{Z}'}^\top A_{\mathcal{Z}'\mathcal{F}} - \kappa \mathbf{1}_f^\top \tag{2.33}$$

Ainsi, dans le problème de désagrégation duale, la variable κ peut ainsi être considérée comme une variable commune à chaque agrégat.

Nous allons maintenant nous intéresser brièvement aux contraintes de convexité. Dans un premier temps, notons que chaque sous-problème P^j admet une solution $\boldsymbol{x}$ ne couvrant aucun sommet, pas même le centre j. Nous pouvons considérer que la variable de (P^M) associée à cette solution est une variable d'écart pour les contraintes de convexité (ce qui les transforme en inégalités), car elles n'apparaissent dans aucune autre contrainte et ont un coût nul.

Ainsi, toute base du problème maître contient les variables d'écart correspondant aux sous-problèmes dont nous utilisons moins d'une colonne dans la solution basique (dans une solution entière de notre problème, cela correspond aux sommets qui ne sont pas centres d'un secteur). De ce fait, l'ajout de ces variables d'écart revient à étendre la sous-matrice A_{PF} en utilisant une matrice identité.

Ainsi, les contraintes additionnelles présentes dans notre problème ne modifient pas le problème de désagrégation duale de manière radicale. Cependant, elle devraient tout de même être prises en compte : la variable duale de cardinalité intervient dans la désagrégation de chaque agrégat, et les variables duales de convexité interviennent dans la désagrégation de leurs agrégats respectifs (potentiellement plusieurs agrégats simultanément, si la solution basique est fractionnaire). Dans notre application, nous avons choisi de ne pas désagréger les variables duales correspondant aux contraintes additionnelles afin de rester le plus proche possible de l'algorithme DCA. Ainsi, la variable duale désa-

grégée d'une contrainte additionnelle aura la même valeur que la variable duale agrégée, ce qui restreint les possibilités de désagrégation. De fait, il est alors possible de ne pas trouver de solution duale désagrégée réalisable, même en partant d'une solution primale optimale.

2.4.2.3 Contre-exemple des contraintes duales d'agrégation

Considérons le problème suivant. Soit $V = \{v_1, v_2\}$ l'ensemble de sommets à couvrir en utilisant au plus $K = 1$ secteur. L'ensemble des secteurs valides est donné par $V_2^1 = \{1, 1\}$ de coût 1 et de centre v_1, et les secteurs $V_0^1 = \{0, 0\}, V_0^2 = \{0, 0\}, V_1^1 = \{1, 0\}, V_1^2 = \{0, 1\}$ de coûts nuls.

Soit $\lambda_i^j \in \{0, 1\}$ la variable binaire telle que $\lambda_i^j = 1$ si la solution utilise V_i^j le i-ième secteur de centre j, et 0 sinon. L'agrégation courante correspond à la solution (entière) optimale $\lambda_1^1 = 1$ et combine donc les sommets v_1 et v_2.

Le problème maître agrégé (A^M) peut être formulé de la manière suivante.

$$(\mathrm{A}^M) = \begin{cases} \min \lambda_2^1 & & \\ \quad \text{s.t. } \lambda_2^1 \leq 1, & (\kappa) & (2.34) \\ \quad\quad \lambda_2^1 \geq 1, & (\Phi_{1,2}) & (2.35) \\ \quad\quad \lambda_0^1 + \lambda_2^1 = 1, & (\alpha_1) & (2.36) \\ \quad\quad \lambda_0^2 = 1, & (\alpha_2) & (2.37) \\ \quad\quad \lambda_0^1, \lambda_0^2, \lambda_1^1 \geq 0. & & \end{cases}$$

Une base optimale pour ce programme est définie par les variables $\{\lambda_1^1, s_1, \lambda_0^1, \lambda_0^2\}$ avec s_1 la variable d'écart pour la contrainte de cardinalité (2.34). Cette base correspond à la matrice carrée B suivante.

$$B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La solution duale agrégée, calculée par $c_{\mathcal{B}}^\top B^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \cdot B^{-1}$ est égale à $\kappa = 0, \Phi_{1,2} = 1, \alpha_1 = 0$ et $\alpha_2 = 0$. Notons que les valeurs de $\Phi_{1,2}$ et κ auraient été les mêmes si nous avions retiré les contraintes de convexité.

Considérons maintenant les contraintes duales associées aux coûts réduits des colonnes incompatibles λ_1^1 et λ_1^2.

$$\phi_1 + \kappa + \alpha_1 \leq 0$$
$$\phi_2 + \kappa + \alpha_2 \leq 0$$

Puisque $\phi_1 + \phi_2 = \Phi_{1,2} = 1$ et $\kappa = \alpha_1 = \alpha_2 = 0$, ces deux contraintes duales ne peuvent pas être vérifiées uniquement en répartissant la valeur de $\Phi_{1,2}$ dans les variables désagrégées ϕ_1 et ϕ_2. La solution duale désagrégée $\kappa = -1, \phi_1 = 1, \phi_2 = 1, \alpha_1 = 0, \alpha_2 = 0$ est réalisable mais ne peut être obtenue par désagrégation sans implémenter les changements étudiés dans les sections précédentes.

2.4.3 Résolution du problème de désagrégation duale par algorithme de sous-gradient

Le problème de désagrégation duale (DDP) peut être résolu par un algorithme de plans sécants. Il suffit pour cela de considérer (DDP) comme un problème maître restreint à un sous-ensemble des contraintes (2.17), tandis que toutes les contraintes (2.18) et (2.19) sont conservées. Ensuite, nous pouvons générer de nouvelles coupes en résolvant les sous-problèmes (P^j) définis par la valeur courante des variables duales désagrégées. Cependant, tout comme les méthodes de génération de colonne, les méthodes de plans sécants peuvent mettre du temps à converger, même si l'on interrompt l'algorithme à la première solution duale induisant une combinaison de colonnes de coût réduit négatif, sans chercher à atteindre la désagrégation optimale $\boldsymbol{\phi}^*$.

À la place, il est possible d'approcher la solution optimale de (DDP) en utilisant un algorithme de sous-gradient. Dans cette section, nous présentons un algorithme de sous-gradient proche de l'algorithme du volume [BA00] pour résoudre le problème de désagrégation duale.

2.4.3.1 Description des éléments de la méthode

Étant donné une solution désagrégée $\boldsymbol{\phi}^{(t)}$, soit $g^* \in \mathcal{G}_J^j$ la colonne de plus petit coût réduit, obtenue en résolvant les sous-problèmes (P^j) pour tout $j \in V$, et soit $\boldsymbol{x}^{g^*}$ la valeur des variables originales $\boldsymbol{x}$ en g^*. Un sous-gradient $\nabla_{\boldsymbol{\phi}^{(t)}}$ peut alors être calculé à partir de la violation des contraintes de couverture.

$$\nabla_{\boldsymbol{\phi}^{(t)}} = \left(\boldsymbol{x}^{g^*} - \mathbf{1}\right) \tag{2.38}$$

La solution désagrégée suivante est donnée par $\boldsymbol{\phi}^{(t+1)} = \boldsymbol{\phi}^{(t)} - \gamma \nabla_{\boldsymbol{\phi}^{(t)}}$ où $\gamma \in \mathbb{R}_+$ est la longueur du pas effectué. Notons que les variables duales $\boldsymbol{\phi}$ doivent vérifier deux familles de contraintes : les contraintes de non-négativité $\boldsymbol{\phi} \geq \mathbf{0}$ et les contraintes d'agrégation $\sum_{i \in V_j} \phi_i = \Phi_j$, pour tout $V_j \in \mathcal{A}$. Il est donc nécessaire d'effectuer une projection orthogonale de la nouvelle solution $\boldsymbol{\phi}^{(t+1)}$ sur l'espace réalisable de ces contraintes. Dans la section suivante, nous décrivons cette projection en détail.

Tout comme l'algorithme du volume, l'algorithme que nous proposons utilise un sous-gradient lissé $\bar{\nabla}$. Celui-ci est normalement calculé à partir d'une combinaison des solutions primales (ici, des colonnes) générées pendant les itérations précédentes de l'algorithme. Puisque nous ne nous intéressons ici qu'à la solution duale désagrégée et aux colonnes qui lui sont associées, nous pouvons, de manière équivalente, calculer directement un sous-gradient lissé obtenu par combinaison linéaire des sous-gradients des itérations précédentes. Étant donné $\bar{\nabla}^{(t)}$ un sous-gradient lissé utilisé pour générer la colonne g^*, et $\nabla^{(t+1)}$ le sous-gradient en g^* obtenu par l'équation (2.38), nous calculons le sous-gradient lissé de la façon suivante.

$$\bar{\nabla}^{(t+1)} = \theta \nabla^{(t+1)} + (1-\theta)\bar{\nabla}^{(t)}, \tag{2.39}$$

où $\theta \in]0;1]$ est un coefficient minimisant $\|\theta \nabla^{(t+1)} + (1-\theta)\bar{\nabla}^{(t)}\|$, ce qui signifie que sa valeur est calculée en minimisant un polynôme de degré 2.

Par ailleurs, la longueur γ du pas effectué dans la direction $-\bar{\nabla}^{(t+1)}$ est basée sur la

règle de Polyak [Pol69].

$$\gamma = \tau \frac{UB - LB}{\|\bar{\nabla}^{(t+1)}\|}, \tag{2.40}$$

avec UB (resp. LB) une borne supérieure (resp. inférieure) sur la valeur μ^* optimale de (DDP) et τ un paramètre. En général, nous remplaçons la valeur $(UB - LB)$ par une approximation de l'écart d'optimalité, estimé à 10% de $\bar{\mu}$ la meilleure valeur trouvée pendant les itérations précédentes de l'algorithme du sous-gradient. Enfin, nous calculons la valeur de τ selon le concept des itérations rouges, jaunes et vertes de l'algorithme du volume [BA00]. Si la nouvelle valeur $\mu^{(t+1)}$ est inférieure à la borne $\bar{\mu}$ (itération rouge), nous diminuons la valeur de τ ; en général $\tau^{(t+1)} = 0.98\tau^{(t)}$. Sinon, nous calculons le produit scalaire $q = \nabla^{(t)} \cdot \nabla^{(t+1)}$. Si $q < 0$ (itération jaune), nous ne modifions pas la valeur de τ. Sinon, la valeur de τ est augmentée (itération verte) ; en général $\tau^{(t+1)} = 1.4\tau^{(t)}$.

L'algorithme est interrompu dès que l'une des conditions suivantes est vérifiées : 1) nous trouvons une solution désagrégée réalisable (c'est-à-dire $\mu = 0$), ou 2) la norme $\|\nabla\|$ du sous-gradient est trop petite, ou bien 3) l'algorithme a effectué plusieurs itérations non-améliorantes consécutives.

2.4.3.2 Projection orthogonale des variables duales

Le problème sur lequel nous appliquons l'algorithme de sous-gradient est doublement contraint : les variables duales doivent vérifier les contraintes de positivité $\boldsymbol{\phi} \geq \mathbf{0}$ et les contraintes d'agrégation $\sum_{i \in V_j} \phi_i = \Phi_j \ \forall V_j \in \mathcal{A}$. Il existe une manière classique d'étendre l'algorithme du sous-gradient afin de résoudre des problèmes d'optimisation sous contraintes. À chaque itération de l'algorithme, nous calculons la solution suivante en effectuant un pas dans la direction $-\nabla$, puis en projetant le point obtenu sur l'espace réalisable des contraintes du problème. Formellement, nous calculons $\boldsymbol{\phi}^{(t+1)} = \rho(\boldsymbol{\phi}^{(t)} - \gamma \nabla_{\boldsymbol{\phi}^{(t)}})$, avec $\rho(\cdot)$ une projection orthogonale sur le demi-plan défini par (2.18) et (2.19).

Dans un premier temps, nous allons décrire les projections applicables pour les contraintes (2.18) et (2.19) indépendamment. Ensuite, nous montrerons qu'il suffit d'appliquer itérativement ces projections pour satisfaire les deux familles de contraintes simultanément.

Les contraintes de positivité $\boldsymbol{\phi} \geq \mathbf{0}$ sont les plus faciles à projeter. Il suffit de mettre à 0 les variables duales négatives. Notons $\rho_+(\boldsymbol{\phi})$ la projection orthogonale d'une solution duale $\boldsymbol{\phi}$ sur le cône défini par les contraintes de positivité.

$$\rho_+(\boldsymbol{\phi}) = \big(\max\{0, \phi_i\} \ \forall i = 1, \ldots, n\big)$$

Considérons maintenant les contraintes d'agrégation. Soit $\mathcal{A} = \{V_1, V_2, \ldots, V_k\}$ un agrégation de V ; pour rappel, notre schéma d'agrégation génère des agrégats **disjoints**. Dans ce cas de figure, les contraintes d'agrégation $\sum_{i \in V_j} \phi_i = \Phi_j \ \forall V_j \in \mathcal{A}$ nécessitent une translation pour chaque agrégat V_j. Étant donné V_j un agrégat, notons $\tilde{\boldsymbol{v}}_j \in \{0,1\}^n$ le vecteur tel que

$$(\tilde{v}_j)_i = \begin{cases} 1 & \text{si } i \in V_j, \\ 0 & \text{sinon.} \end{cases}$$

Ainsi, nous pouvons reformuler les contraintes d'agrégation en utilisant l'écriture matricielle suivante.

$$\tilde{\boldsymbol{v}}_j^\top \boldsymbol{\phi} = \Phi_j \quad \forall V_j \in \mathcal{A} \tag{2.41}$$

Chaque agrégat $V_j \in \mathcal{A}$ est donc associé à un hyperplan de dimension $n-1$. Supposons que $V_j = \{1, 2, \ldots, k\}$. Nous pouvons définir $B \in \mathbb{R}^{n\times(n-1)}$ une base de cet hyperplan comme suit.

$$B = \begin{pmatrix} -\mathbf{1}_{k-1}^\top & \mathbf{0}_{n-k}^\top \\ I_{n-1} \end{pmatrix}$$

Intuitivement, cette base signifie que si nous modifions la valeur d'une variable $\phi_i,\ i \in \{2, \ldots, k\}$ par un facteur τ, alors nous devons retrancher τ dans ϕ_1 ; en revanche, les variables $\phi_j,\ j \in \{k+1, \ldots, n\}$ peuvent être modifiées arbitrairement. Désormais, nous pouvons définir $\rho_j(\boldsymbol{\phi})$ la projection orthogonale de $\boldsymbol{\phi}$ sur la contrainte d'agrégation associée à V_j comme étant la solution du système suivant.

$$\begin{cases} B^\top(\boldsymbol{\phi} - \rho_j(\boldsymbol{\phi})) & = \mathbf{0} \\ \tilde{\boldsymbol{v}}_{\boldsymbol{j}}^\top \rho_j(\boldsymbol{\phi}) - \Phi_j & = 0 \end{cases} \Leftrightarrow \quad \rho_j(\boldsymbol{\phi}) = \boldsymbol{\phi} - \boldsymbol{q}_j \quad \text{avec} \quad (\boldsymbol{q}_j)_i = \begin{cases} \frac{\mathbf{1}^\top \boldsymbol{\phi} - \Phi_j}{|V_j|} & \text{si } i \in V_j, \\ 0 & \text{sinon.} \end{cases}$$

Enfin, nous pouvons assembler les combinaisons $\rho_j(\cdot)$ en $\rho_{\mathcal{A}}(\cdot)$ la projection orthogonale sur l'ensemble des contraintes d'agrégation.

$$\rho_{\mathcal{A}}(\boldsymbol{\phi}) = \boldsymbol{\phi} - \sum_{j:V_j\in\mathcal{A}} \boldsymbol{q}_j$$

Pour terminer, il ne reste plus qu'à combiner ces deux projections. Soit $\boldsymbol{\delta} \in \mathbb{R}^n$ un vecteur de variables correspondant à la modification de $\boldsymbol{\phi}$. Trouver une projection orthogonale sur le demi-plan défini par les deux familles de contrainte correspond à calculer une solution optimale pour le problème quadratique suivant.

$$(\text{OPP}) = \begin{cases} \min\limits_{\boldsymbol{\delta}} \sum\limits_{i\in V} (\delta_i)^2 & & & \\ \text{t.q.} \quad \delta_i - \phi_i \le 0 & \forall i \in V, & (\boldsymbol{\pi}_+) & (2.42) \\ \quad \sum\limits_{i\in V_j} (\phi_i - \delta_i) = \Phi_j & \forall V_j \in \mathcal{A}, & (\boldsymbol{\pi}_{\mathcal{A}}) & (2.43) \\ \quad \boldsymbol{\delta} \in \mathbb{R}^n. & & & \end{cases}$$

Nous allons montrer que ce problème peut être résolu en appliquant $\rho_+(\cdot)$ et $\rho_{\mathcal{A}}(\cdot)$ itérativement. Puisque les agrégats de $\mathcal{A}$ sont disjoints, nous pouvons décomposer (OPP) en $|\mathcal{A}|$ sous-problèmes. Notons $\boldsymbol{\delta}^{j,*}$ une solution optimale du sous-problème (OPP^j) associé à l'agrégat V_j. Remarquons que $\boldsymbol{\delta}^* = \sum_{j:V_j\in\mathcal{A}} \boldsymbol{\delta}^{j,*}$ est une solution optimale pour (OPP).

Pour trouver une solution optimale $\boldsymbol{\delta}^{j,*}$ de (OPP^j), considérons $\boldsymbol{\delta}^{j,1} = \boldsymbol{q}_j$. Si la solution $\boldsymbol{\delta}^{j,1}$ vérifie les contraintes de positivité (2.42), alors $\boldsymbol{\delta}^{j,*} = \boldsymbol{\delta}^{j,1}$ est une solution optimale de (OPP^j). Sinon, soit

$$\boldsymbol{\delta}^{j,1+} = \left(\min\left\{\phi_i, \boldsymbol{\delta}_i^{j,1}\right\}\ i = 1, \ldots, n\right).$$

Cette nouvelle solution vérifie désormais les contraintes de positivité (2.42), mais ne vérifie

plus les contraintes d'agrégation (2.43). Cependant, il existe par définition au moins un sommet $i \in V_j$ tel que $\phi_i - \boldsymbol{\delta}_i^{j,1+} = 0$. Notons n_1 le nombre total de ces sommets, et définissons la solution $\boldsymbol{\delta}^{j,2}$ de la façon suivante.

$$\boldsymbol{\delta}^{j,2} = \begin{cases} \frac{\mathbf{1}^\top \left(\boldsymbol{\phi} - \boldsymbol{\delta}^{j,1+}\right) - \Phi_j}{|V_j| - n_1} & \text{si } i \in V_j \text{ et } \phi_i - \delta_i^{j,1+} > 0, \\ 0 & \text{sinon.} \end{cases}$$

Remarquons que $\mathbf{1}^\top \left(\boldsymbol{\phi} - \boldsymbol{\delta}^{j,1+}\right) - \Phi_j > 0$, sous l'hypothèse que $\boldsymbol{\delta}^{j,1}$ ne vérifie pas les contraintes de positivité. En calculant la suite de solutions $\boldsymbol{\delta}^{j,k}$, avec $k < |V_k|$, nous obtenons une solution optimale $\boldsymbol{\delta}^{j,*} = \sum_k \boldsymbol{\delta}^{j,k}$ de (OPP^j).

Preuve d'optimalité. Considérons la solution duale $(\boldsymbol{\pi}_+^*, \boldsymbol{\pi}_{\mathcal{A}}^*)$ calculée à partir de $\boldsymbol{\delta}^*$ de la manière suivante.

$$\left(\boldsymbol{\pi}_+^*\right)_i = 2\left(\max_{j \in V_j, V_j \ni i} \delta_j^*\right) - 2\delta_i^* \qquad \forall i \in V$$

$$\left(\boldsymbol{\pi}_{\mathcal{A}}^*\right)_j = -2\left(\max_{j \in V_j} \delta_j^*\right) \qquad \forall V_j \in \mathcal{A}$$

Dans la suite, nous utilisons les notations suivantes : la fonction $f(\boldsymbol{\delta}) = \sum_{i \in V} (\delta_i)^2$ est la fonction objectif de (OPP) à minimiser ; le gradient de $f(\cdot)$ au point $\bar{\boldsymbol{\delta}}$ est noté $\nabla_{\bar{\boldsymbol{\delta}}}^f = \left(2\bar{\delta}_i,\ \forall i \in V\right)$; la fonction $g_i(\boldsymbol{\delta}) = \delta_i - \phi_i$ représente la partie gauche de la contrainte de positivité (2.42) associée à $i \in V$; le gradient de $g_i(\cdot)$ au point $\bar{\boldsymbol{\delta}}$ est noté $\nabla_{\bar{\boldsymbol{\delta}}}^{g_i} = \boldsymbol{e}_i$, avec $\boldsymbol{e}_i$ le vecteur $(0, \cdots, 0, 1, 0, \cdots, 0)$ dont le seul composant égal à 1 est situé en i-ième position ; la fonction $h_j(\boldsymbol{\delta}) = \Phi_j - \sum_{i \in V_j} (\phi_i - \delta_i)$ représente la partie gauche de la contrainte d'agrégation (2.43) associées à $V_j \in \mathcal{A}$; enfin, le gradient de $h_j(\cdot)$ au point $\bar{\boldsymbol{\delta}}$ est noté $\nabla_{\bar{\boldsymbol{\delta}}}^{h_j} = \tilde{\boldsymbol{v}}_j$.

Nous allons maintenant montrer que les solutions $\boldsymbol{\delta}^*$ et $(\boldsymbol{\pi}_+^*, \boldsymbol{\pi}_{\mathcal{A}}^*)$ vérifient les conditions de Karush-Kuhn-Tucker.

— **Point stationnaire :** la solution vérifie l'égalité suivante.

$$-\nabla_{\boldsymbol{\delta}^*}^f = \sum_{i \in V} (\boldsymbol{\pi}_+)_i \nabla_{\boldsymbol{\delta}^*}^{g_i} + \sum_{V_j \in \mathcal{A}} \left(\boldsymbol{\pi}_{\mathcal{A}}^*\right)_j \nabla_{\boldsymbol{\delta}}^{h_j}$$

— **Réalisabilité primale :** la solution primale $\boldsymbol{\delta}^*$ est réalisable par construction.

— **Réalisabilité duale :** la solution duale $(\boldsymbol{\pi}_+^*, \boldsymbol{\pi}_{\mathcal{A}}^*)$ vérifie la contrainte de positivité $\boldsymbol{\pi}_+ \geq \mathbf{0}$.

— **Écarts complémentaires :** si la contrainte de positivité (2.42) est vérifiée strictement, c'est-à-dire qu'il existe $i \in V$ tel que $g_i(\boldsymbol{\delta}^*) > 0$, alors pour j tel que $V_j \ni i$, nous avons

$$\delta_i^* = \max_{k \in V_j} \left\{\delta_j^*\right\} \quad \Rightarrow \quad \left(\boldsymbol{\pi}_+^*\right)_i = 0 \quad \Rightarrow \quad g_i(\boldsymbol{\delta}^*) \cdot \left(\boldsymbol{\pi}_+^*\right)_i = 0\ \forall i \in V.$$

□

2.5 Accélération de la convergence de la génération de colonnes et branch-and-price

Dans cette section, nous décrivons les éléments composant notre méthode de branch-and-price. En particulier, nous générons un ensemble de colonnes initiales aléatoires afin d'aider l'algorithme à démarrer, et nous employons une méthode générique de stabilisation duale pour améliorer la convergence de l'algorithme.

2.5.1 Génération de colonnes initiales

L'algorithme de génération de colonnes repose fortement sur l'algorithme du simplex primal. Une des difficultés de cet algorithme consiste à trouver une base réalisable initiale. La méthode la plus utilisée consiste à générer une base réalisable artificielle, composée de colonnes qui n'existent pas dans le programme linéaire mais dont le coût est si élevé qu'il est impossible qu'elles fassent partie d'une base optimale avec un coefficient non-nul. Cette phase peut demander un grand nombre d'itérations de génération de colonnes pour retirer toutes les colonnes artificielles de la base. Ainsi, générer un ensemble de colonnes aléatoires sans tenir compte de la valeur des coûts réduits, puisque ces derniers sont biaisés par le coût des colonnes artificielles en base, permet de réduire la durée de la phase 1 du simplexe.

Pour générer de telles colonnes, nous disposons de l'algorithme glouton suivant, appliqué à chaque sous-problème : Nous démarrons à partir de la colonne vide (ne couvrant aucun sommet). Si la solution est maximale, c'est-à-dire qu'il n'est pas possible de couvrir davantage de sommets sans violer les contraintes du sous-problème, retourner la colonne générée. Sinon, ajouter le sommet non-couvert le plus proche (en terme de distance au centre) n'entraînant pas une violation des contraintes du sous-problème. Les colonnes générées par cet algorithme correspondront à des secteurs compacts, c'est-à-dire que les sommets couverts seront très proches du centre. Il est possible de générer des colonnes plus « éparses » en choisissant d'ajouter le sommet non-couvert le plus éloigné du centre pendant la génération.

2.5.2 Branch-and-price

Afin d'obtenir une solution entière au problème maître (P^M), nous utilisons un algorithme de branchement. Chaque nœud de l'arbre de branchement consiste à résoudre la relaxation continue du problème (P^M) auquel sont ajoutées des contraintes représentant les décisions de branchement.

Les décisions de branchement sont prises sur les variables $x_{i,j}$ du modèle compact (P). Pour deux sommets i et j donnés, ajouter la contrainte $x_{i,j} \geq 1$ (resp. $x_{i,j} \leq 0$) revient à ajouter la contrainte $\sum_{g \in \mathcal{G}^j} x^g_{i,j} \lambda^g_j \geq 1$ (resp. $\sum_{g \in \mathcal{G}^j} x^g_{i,j} \lambda^g_j \leq 0$) au problème ($\mathrm{P}^M$). Cette contrainte ajoute une nouvelle variable duale qui intervient dans la fonction objectif du sous-problème (P^j). Notons que le sous-problème (P^j) peut continuer à générer des colonnes de vérifiant pas cette contrainte de branchement. La variable $x_{i,j}$ sur laquelle l'algorithme branche est celle dont la valeur est la plus fractionnaire, c'est-à-dire la plus proche de 0.5. Par ailleurs, les nœuds de branchement sont évalués dans un ordre *Best-first*.

Afin de compenser les problèmes de convergence liés à la génération de colonne (*tailing-off effect*), nous utilisons la technique de stabilisation de Wentges [Wen97] qui consiste à lisser la solution duale du problème (P^M) de la manière suivante. Soit $(\kappa, \boldsymbol{\phi}, \boldsymbol{\alpha})^t$ la solution duale obtenue à l'itération t et soit $(\hat{\kappa}, \hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\alpha}})$ la solution duale ayant donné la meilleure borne duale pour (P^M). La solution lissée utilisée dans les sous-problème est donnée par la combinaison convexe

$$(\tilde{\kappa}, \tilde{\boldsymbol{\phi}}, \tilde{\boldsymbol{\alpha}})^t = \gamma(\hat{\kappa}, \hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\alpha}}) + (1-\gamma)(\kappa, \boldsymbol{\phi}, \boldsymbol{\alpha})^t.$$

Nous utilisons l'implémentation décrite dans [PSUV17].

2.6 Résultats numériques

Nos expérimentations ont plusieurs objectifs. Dans un premier temps, nous déterminons empiriquement quel est le meilleur oracle pour la génération de colonnes. Ensuite, nous comparons les performances obtenues lorsque le problème est traité à travers la formulation étendue ou la formulation compacte originale. Enfin, nous montrons les effets de plusieurs techniques de stabilisation sur la proportion d'itérations de génération de colonnes dégénérées pendant la résolution de la formulation étendue.

2.6.1 Instances utilisées et configuration informatique

Deux ensembles d'instances ont été utilisés pour évaluer la performance des différentes méthodes proposées. Les instances `exeo` correspondent aux 13 instances issues de données réelles fournies par la société eXEO Solutions. La taille de ces instances varie entre 31 points de collecte et 245 points de collecte. Les instances `cvrp` sont issues de la littérature scientifique. Nous avons transformé 16 instances du CVRP (*capacitated vehicle routing problem*) issues de la TSPLIB en mélangeant les consommations et en bruitant les distances. Chaque instance originale a donné 25 instances bruitées, pour un total de 400 instances dont la taille varie entre 7 et 262 points de collecte.

Afin de générer plusieurs instances à partir d'une seule instance de la TSPLIB, nous avons ajouté du bruit aléatoire à la longueur des arcs de la manière suivante. Pour (i, j) un arc donné de longueur $c_{i,j}$, nous générons la longueur $c'_{i,j} = c_{i,j} + r$ avec r un nombre réel aléatoire généré selon une distribution uniforme dans $[0, (c_{i,j})^{1.5}]$. De plus, nous générons une seconde contrainte de sac-à-dos en gardant la même capacité et en permutant les consommations des sommets de façon aléatoire. De manière générale, les consommations de ressources peuvent monter jusqu'à 4 100 pour une capacité par secteur allant de 3 à 14 400 avec une moyenne de 1 800.

Les tests ont été effectués sur PlaFRIM (Plateforme Fédérative pour la Recherche en Informatique et Mathématiques), sur des machines équipées de processeurs *Dodeca-core Haswell Intel® Xeon® E5-2680 v3 @ 2,5 GHz* et de 128Go de RAM.

Les programmes linéaires et les programmes linéaires et les formulations MIP compactes sont résolues en utilisant IBM ILOG CPLEX Optimization Studio 12.6.0 [CPL]. Le branch-and-price est implémenté dans le framework BaPCod 2.0.0 [Van], en utilisant en particulier les paramètres par défaut de stabilisation et la gestion de l'arbre de recherche dans le branch-and-price.

2.6.2 Comparaison des oracles de pricing

Les premiers tests expérimentaux servent à déterminer le meilleur algorithme permettant de résoudre le problème de pricing issu de la décomposition de Dantzig-Wolfe. Nous appliquons ainsi quatre oracles différents et déterminons celui présentant les meilleures performances au sein de l'algorithme de branch-and-price pour résoudre les instances `exeo` et `cvrp` à l'optimum.

Parmi les oracles étudiés, il y a trois variantes de l'algorithme de branch-and-bound et une implémentation du programme dynamique. Les colonnes `PROP` correspondent à l'algorithme de branch-and-bound dans lequel la direction de descente est proportionnelle à la violation des contraintes de sac à dos par la solution optimal du lagrangien (projetée sur $\mathbb{R}^2_+$), tandis que les colonnes `STEEP` correspondent à l'algorithme de branch-and-bound dans lequel seul le multiplicateur lagrangien associé à la contrainte de sac à dos la plus violée est augmenté. Cette seconde configuration est ensuite modifiée en changeant l'ordre de résolution des problèmes de pricing : dans la configuration `STEEP+BC` l'algorithme donne la prorité aux sous-problème ayant le plus souvent généré la colonne de coût réduit minimum lors des itérations de pricing précédentes. De plus, au cours du pricing, le coût réduit de la meilleure colonne obtenue lors de la résolution des t premiers sous-problèmes est utilisée comme borne primale pour les sous-problèmes restants, ce qui peut permettre d'interrompre la résolution d'un sous-problème avant d'avoir trouvé la colonne de coût réduit minimum qui lui est associée. Enfin, les colonnes `DP` correspondent à une implémentation du programme dynamique utilisant notamment des bornes duales sur les états, calculées à partir de la relaxation continue du problème de pricing et d'une relaxation lagrangienne des contraintes de sac à dos.

Les résultats obtenus sont présentés dans le Tableau 2.1. Pour chaque oracle, la colonne `time` correspond au temps total en secondes utilisé pour résoudre une instance à l'optimum entier (avec une limite de temps de 2h). La colonne `nodes/cg` indique le nombre de nœuds de branchement (générés par l'oracle de branch-and-bound) en moyenne par itération de génération de colonne. Lorsque l'oracle est basé sur la programmation dynamique, nous reportons la valeur `labels/cg` correspondant au nombre moyen de labels générés à la place du nombre de nœuds de branchement. Enfin, la colonne `solved` correspond au nombre d'instances résolues à l'optimum en moins de 2h dans la classe d'instance considérée.

Il ressort des experimentations que l'oracle `STEEP+BC` domine les autres approches, même si la différence est parfois marginale. Sauf indication contraire, c'est cet oracle qui est utilisé dans la suite des expérimentations.

Lorsque l'on compare les directions de descente pour l'oracle de branch-and-bound, on observe que la direction `STEEP` induit systèmatiquement un nombre inférieur ou égale de nœuds de branchement par itération de génération de colonnes que la direction `PROP`, ce qui se traduit généralement par une résolution plus rapide du problème maître. Les différences les plus nettes apparaissent lors de la résolution des instances `exeo/174`, `exeo/245` et `cvrp/eilA101`. Notons également les instances `cvrp/eilB101`, pour lesquelles la direction `PROP` semble mieux adaptée en terme de nombre d'instances résolues à l'optimum, bien que le nombre de nœuds de branchement moyen soit plus élevé. Dans la plupart des itérations de génération de colonnes, il y a probablement plusieurs colonnes optimales possibles pour un sous-problème donné. Ainsi, le hasard des colonnes générées via la di-

	PROP			STEEP			STEEP+BC			DP		
Instance	**time**	**nodes/cg**	**solved**	**time**	**nodes/cg**	**solved**	**time**	**nodes/cg**	**solved**	**time**	**labels/cg**	**solved**
exeo/031	> 0	35	1/1	> 0	21	1/1	> 0	1	1/1	> 0	90	1/1
exeo/032	> 0	1	1/1	> 0	1	1/1	> 0	1	1/1	> 0	120	1/1
exeo/054	> 0	1	1/1	> 0	1	1/1	2	1	1/1	> 0	351	1/1
exeo/075	1	1	1/1	1	1	1/1	> 0	1	1/1	1	439	1/1
exeo/087	2	9	1/1	2	9	1/1	1	3	1/1	2	1 201	1/1
exeo/102	2	1	1/1	2	1	1/1	2	1	1/1	3	876	1/1
exeo/109	5	125	1/1	5	125	1/1	5	44	1/1	8	1 692	1/1
exeo/125	12	35	1/1	13	35	1/1	12	13	1/1	16	929	1/1
exeo/162	6	36	1/1	6	36	1/1	7	11	1/1	9	1 390	1/1
exeo/163	72	1	1/1	71	1	1/1	60	1	1/1	100	1 780	1/1
exeo/171	695	54	1/1	693	54	1/1	483	17	1/1	1 093	2 068	1/1
exeo/174	799	34 694	1/1	31	273	1/1	22	32	1/1	81	7 018	1/1
exeo/245	8 557	2 725 771	0/1	501	34	1/1	184	7	1/1	2 832	26 705	1/1
	time	**nodes/cg**	**solved**	**time**	**nodes/cg**	**solved**	**time**	**nodes/cg**	**solved**	**time**	**labels/cg**	**solved**
cvrp/att48	1	12	25/25	1	4	25/25	1	2	25/25	1	278	25/25
cvrp/eil7	> 0	2	25/25	> 0	2	25/25	> 0	2	25/25	> 0	6	25/25
cvrp/eil13	> 0	8	25/25	> 0	7	25/25	> 0	5	25/25	> 0	19	25/25
cvrp/eil22	> 0	9	25/25	> 0	8	25/25	> 0	6	25/25	> 0	44	25/25
cvrp/eil23	> 0	666	25/25	> 0	486	25/25	> 0	353	25/25	> 0	261	25/25
cvrp/eil30	> 0	12	25/25	> 0	10	25/25	> 0	7	25/25	> 0	113	25/25
cvrp/eil31	> 0	227	25/25	> 0	201	25/25	> 0	165	25/25	> 0	112	25/25
cvrp/eil33	1	36	25/25	1	25	25/25	1	16	25/25	1	156	25/25
cvrp/eil51	1	28	25/25	1	18	25/25	1	9	25/25	1	384	25/25
cvrp/eilA101	1 925	553	20/25	1 726	67	**22/25**	1 573	34	**22/25**	1 897	1 627	20/25
cvrp/eilA76	105	109	25/25	84	63	25/25	77	32	25/25	106	513	25/25
cvrp/eilB101	4 758	278	**13/25**	4 974	177	11/25	4 896	89	**13/25**	5 342	849	10/25
cvrp/eilB76	610	289	25/25	531	239	25/25	573	161	25/25	590	356	25/25
cvrp/eilC76	7	31	25/25	8	19	25/25	7	10	25/25	9	495	25/25
cvrp/eilD76	9	47	25/25	8	25	25/25	8	13	25/25	13	797	25/25
cvrp/gil262	7 201	4 740	0/25	7 201	1 666	0/25	7 201	752	0/25	7 201	11 795	0/25
Total (/400)			**358**			**358**			**360**			**355**

TABLE 2.1 – Performances de l'oracle branch-and-bound

rection PROP pourrait expliquer une résolution plus rapide du problème maître pour ces instances.

L'utilisation des stratégies de pricing de la configuration STEEP+BC diminue encore le nombre de nœuds de branchement évalués et domine les deux stratégies précédentes.

Malgré tout, aucune des stratégies proposées ne permet de résoudre les instances de la classe cvrp/gil262. Ces instances sont très difficiles à résoudre car elles induisent, pour l'algorithme de branch-and-price, un nombre trop important de nœuds de branchement et de résolution des sous-problèmes de pricing, bien que le temps de résolution de ces derniers soit de l'ordre de la milliseconde.

2.6.3 Comparaison de la formulation compacte et de la formulation étendue

Dans cette section, nous comparons les performances de la formulation étendue, résolue à l'aide de notre solveur spécialisé, basé sur le framework BaPCod [Van], avec celles de la formulation compacte (P) résolue par le solveur générique CPLEX 12.6 [CPL].

Dans le Tableau 2.2, nous rapportons time_tot le temps total en secondes passé à résoudre l'instance à l'optimum (limité à 2h) et time_root le temps total en secondes passé à résoudre la racine de l'arbre de branchement (c'est-à-dire la relaxation continue de (P) et de (P^M), respectivement). La colonne gap_opt correspond à l'écart d'optimalité à la fin de l'algorithme (pour les instances exeo, les deux formulations sont capables de trouver une solution optimale bien avant la fin du temps imparti). Par contraste, la colonne gap_root indique l'écart entre la valeur optimale de la relaxation continue de la formulation considérée et la valeur optimale de la solution entière de l'instance. Enfin, les colonnes nb_bb_nodes et time_per_node correspondent, respectivement, au nombre total de nœuds de branchement (dans l'algorithme de branch-and-bound pour la formulation compacte, ou branch-and-price pour la formulation étendue) évalués pour résoudre l'instance à l'optimum, et au temps moyen passé pour chacun de ces nœuds.

Les expérimentations montrent que chaque approche peut être pertinente selon le type d'instance résolu. En effet, la formulation compacte offre de bien meilleures performances pour la résolution des instances exeo, même si la formulation étendue permet également de résoudre toutes ces instances dans un temps raisonnable.

Le choix entre une formulation compacte ou étendue semble donc tenir à l'équilibre entre une relaxation continue beaucoup plus forte pour la formulation étendue — nous pouvons voir que l'écart d'optimalité est régulièrement fermé dès le nœud racine — contre un coût d'évaluation des nœuds de branchement plus élevé. Par exemple, pour l'instance exeo/171, le nombre nœuds de branchement évalués pendant la résolution de la formulation compacte est plus que compensé par la vitesse moyenne de résolution de ces nœuds. De manière analogue, bien que l'écart à la racine de la formulation étendue pour l'instance exeo/245 soit quasiment nul, l'évaluation du nœud racine requiert presque deux fois plus de temps que la résolution totale de l'instance par le solveur générique. En revanche, la formulation étendue permet de résoudre plus d'instances cvrp que la formulation compacte, bien que cette dernière soit mieux adaptée à la résolution des instances cvrp/eilA101.

Instance	time_tot		time_root		gap_opt = (pb-db)/db		gap_root = (pb*-lp)/lp		nb_bb_nodes		time_per_node	
	(P)	CG BB	(P)	CG BB	(P)	CG BB	(P)	CG BB	(P)	CG BB	(P)	CG BB
exeo/031	1	0	0	0.06	0%	0%	6.1%	0.0%	24	1	0	0
exeo/032	0	0	0	0.05	0%	0%	0.0%	0.0%	0	1	0	0
exeo/054	0	2	0	2.01	0%	0%	0.0%	0.0%	0	1	0	2
exeo/075	0	0	0	0.48	0%	0%	0.0%	0.0%	0	1	0	0
exeo/087	0	1	0	1.45	0%	0%	0.2%	0.0%	0	1	0	1
exeo/102	0	2	0	1.56	0%	0%	0.0%	0.0%	0	1	0	2
exeo/109	2	5	0	4.52	0%	0%	2.4%	0.0%	0	1	0	5
exeo/125	3	12	0	2.95	0%	0%	2.3%	0.0%	21	1	0	12
exeo/162	6	7	1	6.71	0%	0%	3.3%	1.5%	9	19	1	0
exeo/163	5	60	1	8.31	0%	0%	0.4%	0.0%	5	1	1	60
exeo/171	68	483	1	10.77	0%	0%	6.2%	0.4%	860	17	0	28
exeo/174	72	22	1	19.58	0%	0%	2.3%	1.5%	428	339	0	0
exeo/245	38	184	5	60.74	0%	0%	1.9%	$> 0\%$	11	3	3	61

Instance	(P)	CG BB
cvrp/att48	25/25	25/25
cvrp/eil7	25/25	25/25
cvrp/eil13	25/25	25/25
cvrp/eil22	25/25	25/25
cvrp/eil23	25/25	25/25
cvrp/eil30	25/25	25/25
cvrp/eil31	25/25	25/25
cvrp/eil33	25/25	25/25
cvrp/eil51	25/25	25/25
cvrp/eilA101	**25/25**	22/25
cvrp/eilA76	25/25	25/25
cvrp/eilB101	8/25	**13/25**
cvrp/eilB76	11/25	**25/25**
cvrp/eilC76	25/25	25/25
cvrp/eilD76	25/25	25/25
cvrp/gil262	0/25	0/25
Total	**344/400**	**360/400**

TABLE 2.2 – Comparaison de la formulation compacte et de la formulation étendue

2.6.4 Comparaison de méthodes génériques pour réduire la dégénérescence

Dans cette section, nous nous intéressons aux effets de deux méthodes génériques de stabilisation sur la dégénérescence observée dans la résolution de la formulation compacte. Le Tableau 2.3 regroupe les performances de l'algorithme de branch-and-price lorsque celui-ci est combiné à la stabilisation de Wentges et au pricing multiple.

Nous avons ici testé 4 configurations différentes. La colonne `std` correspond à un algorithme de génération standard. Les variables duales ne sont pas lissées et à chaque itération, le maître reçoit la colonne de plus petit coût réduit, tous centres confondus. La colonne `W` correspond au lissage dual proposé par Wentges [Wen97] en utilisant l'implémentation de [PSUV17], tandis que la colonne `M` correspond à un pricing multiple, c'est-à-dire qu'à chaque itération, nous ajoutons la meilleure colonne par centre au problème maître, et effectuons autant d'opérations de pivot que nécessaire pour trouver la nouvelle base optimale. Enfin, la colonne `W+M` combine le pricing multiple et le lissage dual de Wentges.

Pour chaque configuration, nous rapportons `tps_tot` le temps total en secondes passé à trouver la solution entière optimale (limité à 2h), `solved` le nombre d'instances résolues à l'optimum et `nb_col_gen_global` le nombre de fois que le sous-problème de pricing a été résolu (une itération consiste à résoudre le sous-problème associé à chaque centre dans V). De plus, nous rapportons `primal_degen` (resp. `dual_degen`), correspondant à la proportion d'itérations dégénérées lorsque la base courante est sous-optimale (resp. optimale) pour le problème maître à un nœud de branchement donné de l'arbre de branch-and-price. Ces deux cas sont confondus dans la colonne `any_degen` afin de mesurer la dégénérescence totale observée pendant la résolution. Enfin, la colonne `max_#_of_consec_degen` représente le nombre maximal d'itérations dégénérées consécutives.

Ces expérimentations montrent que les deux techniques génériques ont un impact positif significatif sur les performances de l'algorithme de branch-and-price. En particulier, on observe une diminution conséquente de la dégénérescence primale. Dans les configurations utilisant le pricing multiple, cela est probablement dû en grande partie au fait que chaque itération de génération de colonnes peut donner lieu à plusieurs pivots consécutifs. En revanche, la proportion de dégénérescence duale semble être la plus faible dans la configuration standard. Il semble cependant plus raisonnable de penser que le nombre d'itérations dégénérées à l'optimum est resté inchangé dans les autres configurations, tandis que le nombre total d'itérations a diminué, ce qui pourrait expliquer l'augmentation du ratio dans les configurations stabilisées.

2.6.5 Mesure des effets d'une agrégation de contraintes sur la dégénérescence du problème maître restreint

Nous souhaitons maintenant déterminer l'efficacité des techniques de stabilisation inspirées de l'agrégation des sommets du réseau. Nous nous plaçons dans le contexte de la résolution du modèle obtenu par décomposition de Dantzig-Wolfe, dans lequel une itération de génération de colonnes consiste à résoudre les sous-problèmes de tous les centres et renvoyer les meilleures colonnes correspondantes (configurations `M` et `M+W` dans le Tableau 2.3). Si l'itération précédente était dégénérée (la valeur optimale du maître

Instance	tps_tot				solved				nb_col_gen_global				primal_degen				dual_degen				any_degen				max # of consec degen			
	std	W	M	W+M	std	W	M	W+M	std	W	M	W+M	std	W	M	W+M	std	W	M	W+M	std	W	M	W+M	std	W	M	W+M
exco/031	0	0	0	0	1	1	1	1	78	74	15	17	42%	24%	13%	0%	10%	39%	20%	29%	53%	64%	33%	29%	25	29	3	5
exco/032	0	0	0	0	1	1	1	1	139	104	25	31	43%	64%	24%	16%	22%	10%	24%	39%	65%	74%	48%	55%	30	60	6	12
exco/054	1	0	0	0	1	1	1	1	486	209	81	56	56%	11%	11%	45%	1%	36%	33%	11%	57%	47%	44%	55%	71	76	27	21
exco/075	4	1	1	1	1	1	1	1	785	267	81	58	58%	55%	28%	38%	2%	3%	1%	2%	61%	58%	30%	40%	124	30	5	12
exco/087	8	2	3	2	1	1	1	1	1 051	347	168	107	59%	59%	18%	3%	4%	13%	36%	62%	63%	72%	55%	64%	139	120	61	66
exco/102	17	6	5	2	1	1	1	1	1 391	660	111	95	43%	20%	15%	34%	2%	50%	7%	27%	44%	70%	23%	61%	43	332	8	26
exco/109	22	9	7	4	1	1	1	1	1 344	573	168	151	48%	18%	14%	7%	1%	40%	21%	65%	49%	58%	36%	72%	69	231	36	98
exco/125	59	31	18	12	1	1	1	1	2 870	1 453	495	427	67%	42%	41%	27%	5%	21%	11%	19%	72%	63%	51%	46%	95	84	32	57
exco/162	51	25	9	6	1	1	1	1	1 572	700	150	106	35%	20%	15%	19%	9%	31%	15%	26%	45%	51%	30%	45%	146	218	22	28
exco/163	479	134	168	70	1	1	1	1	12 053	4 554	1 248	1 057	58%	37%	29%	25%	3%	25%	9%	24%	62%	62%	38%	49%	180	281	40	69
exco/171	2 022	1 400	1 108	615	1	1	1	1	57 863	25 803	13 709	10 493	68%	40%	44%	29%	6%	26%	8%	22%	74%	66%	52%	51%	148	197	40	82
exco/174	431	142	79	25	1	1	1	1	5 453	1 662	310	178	52%	16%	19%	14%	0%	17%	5%	34%	52%	33%	24%	48%	463	281	34	60
exco/245	7 203	376	7 205	410	0	1	0	1	20 001	4 339	819	1 057	67%	45%	10%	35%	0%	24%	0%	37%	67%	68%	10%	72%	345	809	12	266
Total	10 296	2 127	8 604	1 147	**12**	**13**	12	**13**																				
												Average	54%	35%	**22%**	**22%**	**5%**	26%	15%	31%	59%	60%	**36%**	53%				
Instance	**std**	**W**	**M**	**W+M**	**std**	**W**	**M**	**W+M**	**std**	**W**	**M**	**W+M**	**std**	**W**	**M**	**W+M**	**std**	**W**	**M**	**W+M**	**std**	**W**	**M**	**W+M**	**std**	**W**	**M**	**W+M**
cvrp/att48	2	1	1	1	25	25	25	25	823	517	171	185	34%	27%	21%	18%	5%	13%	10%	16%	39%	40%	31%	35%	58	50	12	18
cvrp/eil7	0	0	0	0	25	25	25	25	10	14	5	8	33%	25%	13%	16%	25%	20%	20%	15%	58%	44%	33%	32%	4	4	1	2
cvrp/eil13	0	0	0	0	25	25	25	25	48	61	20	32	23%	19%	6%	9%	12%	11%	9%	6%	35%	30%	15%	15%	5	6	2	2
cvrp/eil22	0	0	0	0	25	25	25	25	137	144	44	63	27%	21%	11%	8%	10%	10%	14%	14%	37%	32%	24%	21%	8	9	2	3
cvrp/eil23	0	0	0	0	25	25	25	25	139	129	44	53	24%	21%	13%	10%	11%	16%	13%	15%	35%	37%	26%	25%	15	20	5	6
cvrp/eil30	0	0	0	0	25	25	25	25	359	326	103	132	34%	29%	14%	14%	12%	15%	15%	15%	46%	44%	29%	30%	19	17	5	6
cvrp/eil31	0	1	0	0	25	25	25	25	210	233	68	104	22%	18%	8%	7%	8%	8%	10%	9%	31%	27%	19%	15%	8	10	2	3
cvrp/eil33	1	1	0	1	25	25	25	25	767	620	201	270	34%	26%	16%	14%	10%	11%	12%	12%	44%	38%	28%	26%	17	18	5	7
cvrp/eil51	2	1	1	1	25	25	25	25	869	537	163	189	31%	22%	12%	13%	6%	8%	11%	10%	37%	30%	22%	23%	25	21	4	7
cvrp/eilA101	2 018	1 840	1 845	1 720	20	20	19	22	145 187	125 736	65 142	70 905	56%	38%	20%	16%	4%	8%	7%	6%	60%	46%	27%	22%	82	49	21	16
cvrp/eilA76	176	164	59	77	25	25	25	25	26 763	23 290	5 735	9 202	46%	29%	13%	13%	6%	7%	10%	8%	52%	36%	22%	21%	27	22	6	7
cvrp/eilB101	5 806	5 772	4 789	4 939	6	6	14	11	327 500	310 486	174 126	214 897	45%	27%	12%	11%	5%	5%	8%	6%	50%	32%	19%	17%	50	33	12	11
cvrp/eilB76	1 315	944	470	516	25	24	25	25	112 882	80 767	31 179	39 394	38%	22%	10%	9%	5%	5%	7%	6%	43%	27%	17%	14%	23	19	6	6
cvrp/eilC76	19	14	7	7	25	25	25	25	4 145	3 073	795	1 063	44%	32%	14%	15%	9%	10%	13%	12%	53%	42%	27%	27%	29	25	8	10
cvrp/eilD76	24	15	8	8	25	25	25	25	4 977	2 833	787	985	47%	31%	15%	15%	6%	8%	11%	10%	53%	40%	26%	25%	46	33	11	12
cvrp/gil262	7 200	7 200	7 200	7 201	0	0	0	0	19 484	19 362	15 467	19 012	32%	24%	13%	10%	11%	8%	14%	10%	43%	32%	27%	20%	17	18	13	14
Total	16 564	15 954	14 380	14 470	**351**	**350**	**358**	358				**Average**	**37%**	**26%**	**13%**	**12%**	**8%**	**9%**	**11%**	**10%**	**45%**	**35%**	**24%**	**22%**				

TABLE 2.3 – Tests préliminaires de dégénérescence (**W** : stabilisation Wentges et **M** : pricing Multiple)

restreint n'a pas changé malgré les nouvelles colonnes), alors nous remplaçons les sous-problèmes de pricing par la résolution du problème de désagrégation duale. L'algorithme va donc chercher la meilleure solution duale (celle maximisant le coût réduit des colonnes) vérifiant les contraintes duales d'agrégation. Ensuite, l'algorithme résout les problèmes de pricing à partir de cette solution désagrégée.

Dans un premier temps, nous résolvons le problème de désagrégation duale par l'algorithme de sous-gradient décrit dans la sous-section 2.4.3. Les résultats de ces tests sont compilés dans le Tableau 2.4 et le Tableau 2.5. Chaque tableau est séparé en deux afin de pouvoir comparer les effets de la désagrégation duale lorsque la solution duale original est lissée (stabilisation de Wentges) ou non. De plus, pour chaque mesure, la colonne de gauche `no_agg` représente la configuration sans agrégation, et la colonne de droite `agg` la configuration avec agrégation. La colonne `tps_tot` correspond au temps total en secondes passé à résoudre l'instance à l'optimum entier (limité à 2h), tandis que la colonne `solved` indique si l'instance a pu être résolu à l'optimum dans le temps imparti. Comme dans le tableau précédent, les colonnes `primal_degen`, `dual_degen` et `any_degen` représentent la proportion d'itérations de génération de colonnes dégénérées selon si la solution basique courante est optimale ou non pour le problème maître, à un nœud de branchement donné de l'arbre de branch-and-price. De même, la colonne `max_cons_cg_degen` indique le nombre maximal d'itérations dégénérées consécutives de génération de colonnes. Les colonnes `nb_col_gen` et `nb_cols_tot` indiquent, respectivement, le nombre total d'itération de génération de colonnes (une itération consiste à résoudre le sous-problème pour chaque centre $j \in V$ ou bien à résoudre le problème de désagrégation duale), et le nombre de colonnes distinctes générées pendant la résolution de l'instance. Enfin, nous rapportons les performances spécifiques à l'algorithme de sous-gradient dans la colonne `subgradient_perfs` : `#calls` le nombre de désagrégations résolues ; `#it/call` le nombre moyen d'itérations de l'algorithme de sous-gradient par appel ; et `time/call` le temps moyen passé à résoudre le problème de désagrégation duale.

Sur le jeu d'instances `exeo`, l'algorithme permet de diminuer le nombre d'itérations dégénérées, passant de 39% à 24% sans lissage dual, et de 53% à 32% avec lissage. Cet effet est principalement dû à la capacité de l'algorithme à trouver des solutions duales réalisables permettant de prouver l'optimalité de la base courante. En effet, le nombre d'itérations dégénérées à l'optimum passe de 16% à 6% sans lissage dual en moyenne, et de 31% à 13% avec lissage, tandis que la dégénérescence primale passe de 23% à 18% dans les deux cas. Ces tests ont néanmoins permis de soulever des points problématiques. Premièrement, le nombre d'itérations de génération de colonnes et le nombre de colonnes distinctes générées n'est pas beaucoup affecté par l'utilisation d'une solution duale désagrégée, avec une diminution de 15% des itérations de génération de colonnes dans le meilleur cas, et le nombre de colonnes générées augmente même de 7% lorsque le lissage dual est activé. De plus, l'instance `exeo/171` semble particulièrement mal réagir à l'utilisation de solution duales désagrégées.

Les performances de l'algorithme sur les instances `cvrp` sont également négatives, avec une diminution du nombre d'instances résolues de 358/400 à 344/400, et un effet très marginal sur la dégénérescence, qui passe de 23% en moyenne à 19%. Notons que les itérations dégénérées sont beaucoup moins présente dans ce jeu d'instances que dans les instances `exeo`, ce qui peut justifier le manque d'efficacité de la stabilisation par désagrégation duale.

Sans stabilisation Wentges

Instance	tps_tot		solved		primal_degen		dual_degen		any_degen		maxConsCgDegen		nb_col_gen		nb_cols_tot		subgradient_perfs		
	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	#calls	#it/call	time/call
exeo/031	0	0	1	1	13.3%	21.4%	20.0%	7.1%	33.3%	28.6%	3	2	15	14	304	285	4	24.0	0.01
exeo/032	0	0	1	1	24.0%	46.2%	24.0%	7.7%	48.0%	53.8%	6	2	25	13	560	394	7	36.3	0.00
exeo/054	1	1	1	1	11.1%	12.2%	33.3%	4.1%	44.4%	16.3%	27	2	81	49	3 012	2 387	8	50.9	0.01
exeo/075	1	1	1	1	28.4%	7.8%	1.2%	3.1%	29.6%	10.9%	5	2	81	64	4 065	3 942	7	78.7	0.04
exeo/087	4	2	1	1	18.5%	5.3%	36.3%	2.6%	54.8%	7.9%	61	2	168	76	8 224	5 724	6	54.5	0.05
exeo/102	5	5	1	1	15.3%	8.0%	7.2%	1.1%	22.5%	9.1%	8	2	111	88	8 961	8 168	8	68.8	0.06
exeo/109	8	6	1	1	14.3%	7.8%	21.4%	2.0%	35.7%	9.8%	36	2	168	102	10 342	8 995	10	55.7	0.09
exeo/125	19	51	1	1	40.6%	31.8%	10.7%	13.0%	51.3%	44.7%	32	41	495	409	22 055	18 956	195	85.6	0.19
exeo/162	9	20	1	1	15.3%	19.6%	14.7%	8.0%	30.0%	27.5%	22	12	150	138	10 225	10 337	38	68.5	0.29
exeo/163	172	212	1	1	29.2%	16.4%	8.7%	3.0%	37.9%	19.4%	40	24	1 248	835	80 362	69 911	233	104.4	0.24
exeo/171	1 129	5 863	1	1	43.9%	37.6%	8.1%	14.2%	51.9%	51.8%	40	93	13 709	12 061	490 707	463 642	6 936	72.3	0.71
exeo/174	79	239	1	1	19.4%	4.1%	4.8%	3.1%	24.2%	7.2%	34	9	310	292	35 501	31 503	135	71.7	1.27
exeo/245	7 232	7 218	0	0	8.0%(⋆)	4.6%(⋆)	0.0%(⋆)	0.1%(⋆)	8.0%(⋆)	4.7%(⋆)	12(⋆)	2(⋆)	785(⋆)	766(⋆)	141 229(⋆)	138 703(⋆)	35	68.6	0.85

Avec stabilisation Wentges

Instance	tps_tot		solved		primal_degen		dual_degen		any_degen		maxConsCgDegen		nb_col_gen		nb_cols_tot		subgradient_perfs		
	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	#calls	#it/call	time/call
exeo/031	0	0	1	1	0.0%	0.0%	29.4%	7.7%	29.4%	7.7%	5	1	17	13	281	267	2	52.0	0.01
exeo/032	0	0	1	1	16.1%	36.4%	38.7%	9.1%	54.8%	45.5%	12	1	31	11	420	305	6	60.3	0.01
exeo/054	0	0	1	1	44.6%	11.1%	10.7%	3.7%	55.4%	14.8%	21	3	56	27	1 603	1 256	6	93.8	0.02
exeo/075	1	1	1	1	37.9%	15.8%	1.7%	5.3%	39.7%	21.1%	12	2	58	38	2 340	2 214	10	140.2	0.06
exeo/087	2	2	1	1	2.8%	30.4%	61.7%	6.5%	64.5%	37.0%	66	4	107	46	3 627	3 133	19	65.8	0.06
exeo/102	2	2	1	1	33.7%	12.2%	27.4%	6.1%	61.1%	18.4%	26	3	95	49	4 061	3 845	12	62.6	0.05
exeo/109	5	4	1	1	7.3%	21.7%	64.9%	6.5%	72.2%	28.3%	98	9	151	46	5 026	4 017	15	83.1	0.11
exeo/125	13	44	1	1	27.4%	29.3%	19.0%	15.1%	46.4%	44.5%	57	17	427	317	10 318	10 464	187	103.6	0.18
exeo/162	6	28	1	1	18.9%	1.7%	26.4%	50.0%	45.3%	51.7%	28	60	106	120	5 807	6 191	65	68.2	0.32
exeo/163	71	180	1	1	24.9%	13.3%	24.2%	23.1%	49.1%	36.4%	69	84	1 057	684	33 762	35 565	351	148.8	0.34
exeo/171	681	6 862	1	1	28.7%	32.1%	22.3%	17.9%	51.0%	50.1%	82	93	10 493	10 365	170 166	207 912	6 897	92.9	0.88
exeo/174	31	147	1	1	14.0%	13.1%	33.7%	11.9%	47.8%	25.0%	60	20	178	160	12 417	15 909	92	81.8	1.24
exeo/245	503	973	1	1	35.5%	25.0%	36.8%	10.5%	72.3%	35.5%	266	68	1 057	504	88 040	72 271	212	170.4	2.67

TABLE 2.4 – Effets de la désagrégation duale approchée par algorithme de sous-gradient (instances `exeo`)

Sans stabilisation Wentges

Instance	tps_tot		solved		primal_degen		dual_degen		any_degen		maxConsCgDegen		nb_col_gen		nb_cols_tot		subgradient_perfs		
	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	#calls	#it/call	time/call
cvrp/att48	1	2	25	25	20.7%	16.1%	10.0%	6.9%	30.7%	23.0%	12	6	171	149	2,928	2,728	44	101.9	0.03
cvrp/eil7	0	0	25	25	12.9%	13.5%	19.8%	16.2%	32.8%	29.7%	1	1	5	4	24	24	1	70.1	0.19
cvrp/eil13	0	0	25	25	6.0%	5.7%	9.3%	8.3%	15.3%	13.9%	2	1	20	20	87	86	4	54.3	0.25
cvrp/eil22	0	0	25	25	10.6%	8.0%	13.7%	10.1%	24.4%	18.1%	2	2	44	40	272	264	8	76.2	0.01
cvrp/eil23	0	1	25	25	12.9%	11.5%	13.3%	8.0%	26.2%	19.5%	5	2	44	38	392	350	10	97.1	0.12
cvrp/eil30	0	1	25	25	14.5%	13.6%	14.9%	10.8%	29.4%	24.4%	5	3	103	94	827	791	27	150.0	0.03
cvrp/eil31	0	1	25	25	8.3%	8.4%	10.3%	9.1%	18.7%	17.5%	2	2	68	67	352	353	13	62.9	0.09
cvrp/eil33	0	3	25	25	16.5%	13.5%	11.6%	9.4%	28.1%	22.9%	5	3	201	186	1,500	1,427	50	170.8	0.06
cvrp/eil51	1	3	25	25	11.5%	11.1%	10.6%	7.9%	22.1%	19.0%	4	4	163	155	2,358	2,292	37	147.6	0.06
cvrp/eilA101	1,845	2,370	19	18	20.3%	16.8%	6.8%	5.9%	27.1%	22.6%	21	13	65,273	18,001	1,050,063	316,860	4,114	168.4	0.46
cvrp/eilA76	61	761	25	24	12.8%	12.4%	9.7%	8.8%	22.5%	21.1%	6	6	5,735	5,916	24,011	34,353	1,280	170.9	0.52
cvrp/eilB101	4,844	5,715	13	6	11.7%	10.4%	7.8%	8.0%	19.6%	18.5%	12	12	167,906	21,935	1,127,425	167,272	4,064	137.6	1.27
cvrp/eilB76	500	2,293	25	21	9.8%	9.4%	7.4%	7.2%	17.3%	16.6%	6	6	31,179	21,275	52,765	42,927	3,561	90.9	0.55
cvrp/eilC76	7	42	25	25	13.7%	11.2%	13.2%	9.8%	26.9%	21.0%	8	6	795	724	8,404	8,243	174	217.2	0.20
cvrp/eilD76	8	57	25	25	15.2%	11.8%	10.6%	8.7%	25.8%	20.5%	11	6	787	807	12,238	13,075	187	236.6	0.25
cvrp/gil262	7,201	7,295	0	0	13.0%	3.4%	14.0%	13.4%	27.0%	16.8%	13	5	15,394	1,636	332,941	20,372	277	91.9	24.24

Avec stabilisation Wentges

Instance	tps_tot		solved		primal_degen		dual_degen		any_degen		maxConsCgDegen		nb_col_gen		nb_cols_tot		subgradient_perfs		
	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	#calls	#it/call	time/call
cvrp/att48	1	3	25	25	18.1%	17.6%	16.4%	6.9%	34.5%	24.6%	18	5	185	150	1,866	1,849	64	133.0	0.00
cvrp/eil7	0	0	25	25	16.4%	18.8%	15.3%	6.5%	31.7%	25.3%	2	1	8	6	23	24	3	38.4	0.00
cvrp/eil13	0	0	25	25	8.5%	7.8%	6.4%	4.6%	14.9%	12.5%	2	2	32	31	80	79	15	48.7	0.00
cvrp/eil22	0	0	25	25	7.6%	7.4%	13.8%	9.1%	21.4%	16.5%	3	2	63	58	215	216	27	105.2	0.00
cvrp/eil23	0	2	25	25	9.7%	10.9%	15.1%	7.9%	24.8%	18.8%	6	3	53	48	302	295	21	91.9	0.00
cvrp/eil30	0	2	25	25	14.4%	12.5%	15.2%	8.3%	29.7%	20.8%	6	3	132	117	596	589	56	152.8	0.00
cvrp/eil31	0	5	25	25	6.5%	6.8%	9.0%	7.9%	15.5%	14.7%	3	3	104	100	301	296	49	65.8	0.00
cvrp/eil33	1	8	25	25	14.1%	13.6%	11.8%	7.6%	25.9%	21.2%	7	4	270	258	1,148	1,107	123	184.5	0.00
cvrp/eil51	1	7	25	25	13.2%	11.6%	10.0%	7.3%	23.2%	18.9%	7	4	189	177	1,582	1,641	74	183.1	0.00
cvrp/eilA101	1,688	2,916	22	18	15.7%	15.6%	6.2%	5.7%	22.0%	21.3%	16	14	68,437	10,599	545,127	93,403	4,168	186.7	0.00
cvrp/eilA76	86	820	25	24	13.1%	12.2%	8.2%	5.8%	21.3%	18.0%	7	7	9,202	5,040	25,045	11,426	2,280	176.8	0.00
cvrp/eilB101	4,999	6,366	11	4	11.1%	9.9%	5.9%	5.7%	17.1%	15.6%	11	10	209,978	11,422	696,588	33,336	5,031	135.5	0.01
cvrp/eilB76	544	3,598	25	14	8.5%	9.1%	5.5%	5.7%	14.0%	14.8%	6	7	39,394	10,332	30,182	15,975	4,873	123.6	0.01
cvrp/eilC76	8	177	25	25	14.8%	14.8%	12.1%	8.5%	26.9%	23.3%	10	7	1,063	1,096	5,356	5,957	514	246.4	0.00
cvrp/eilD76	8	120	25	25	15.4%	14.8%	9.9%	7.1%	25.4%	21.9%	12	9	985	1,066	8,005	8,868	451	232.5	0.00
cvrp/gil262	7,200	7,345	0	0	10.8%	7.1%	10.3%	8.6%	21.1%	15.7%	14	4	17,089	569	100,559	8,312	235	121.3	0.25

TABLE 2.5 – Effets de la désagrégation duale approchée par algorithme de sous-gradient (instances `cvrp`)

Malgré tout, l'algorithme de sous-gradient ne semble pas consommer un temps de calcul excessif par exécution, ce qui semble indiquer que les défauts de la méthode se situent dans la méthode de désagrégation elle-même. Afin de pouvoir le vérifier, nous avons effectué des tests supplémentaires en remplaçant l'algorithme de sous-gradient par un algorithme de plan sécant. Celui-ci, bien que pouvant être long à converger, permet de toujours obtenir la meilleure solution duale désagrégée possible, c'est-à-dire celle maximisant le coût réduit des colonnes hors-base.

Dans ce nouvel ensemble de tests, nous avons exécuté l'algorithme spécifiquement sur les instances de la classe `exeo`, qui sont les plus favorables à la dégénérescence. Les performances en terme de temps ne présentent ici qu'un intérêt mineur, c'est pourquoi nous avons augmenté la limite de temps de calcul à 4h par instance.

Dans le Tableau 2.6, nous comparons le déroulement de l'algorithme de branch-and-price avec et sans utilisation de la solution duale stabilisée obtenue par résolution du problème de désagrégation duale par un algorithme de plans sécants. Dans ce tableau, les valeurs sont données par paires. Pour chaque mesure, la colonne de gauche `no_agg` représente la configuration sans agrégation, et la colonne de droite `agg` la configuration avec agrégation. Nous avons également comparé les performances avec et sans lissage dual (*i.e.* stabilisation de Wentges) ; en revanche, toutes les configurations utilisent le pricing multiple. Tout comme dans le tableau précédent, les colonnes `primal_degen`, `dual_degen` et `any_degen` contiennent la proportion d'itérations de génération de colonnes dégénérées, et la colonne `max_cons_cg_degen` représente le nombre maximal d'itérations dégénérées consécutives. Par ailleurs, nous rapportons dans `nb_col_gen` le nombre total d'itérations de génération de colonnes, et dans `nb_cols_tot` le nombre total de colonnes uniques générées pendant la résolution. Enfin, les deux dernières colonnes comparent le ratio entre le nombre d'itérations de génération de colonnes et le nombre de colonnes uniques générées lorsque l'agrégation de sommets est activée ou désactivée.

Lorsque l'une des deux configurations n'a pas résolu une instance à l'optimum, la ligne correspondante est marquée avec des ($\star$). En effet, la comparaison des performances n'est pas toujours pertinente dans ce cas, puisque l'algorithme de branch-and-price peut avoir été interrompu à différents niveaux de la résolution.

Les résultats obtenus sont cohérents avec ceux de l'algorithme de sous-gradient. En particulier, nous observons la même diminution de la dégénérescence duale, passant respectivement de 17% à 6% sans lissage dual. Lorsque la solution duale est lissée, la dégénérescence passe même de 31% à 7%, ce qui est sensiblement mieux que les 12% induits par l'algorithme de sous-gradient. De la même manière, la diminution de la dégénérescence primale est légèrement plus importante, passant de 20% à 12%. L'utilisation de la solution désagrégée confirme également le manque d'impact de la désagrégation sur le nombre de colonnes générées, qui ne diminue que de 13% dans la configuration sans lissage dual, et ne varie presque pas lorsque la solution duale est lissée au préalable. En revanche, le nombre d'itérations de colonnes diminue de manière beaucoup plus importante, pouvant aller jusqu'à 50% d'itérations en moins.

Pour pouvoir interpréter correctement ces résultats, il convient de noter quelques différences entre l'algorithme de sous-gradient et celui des plans sécants. Premièrement, la solution retournée par le sous-gradient est une approximation, car le nombre maximal d'itérations est limité. De plus, l'algorithme de plans sécants est capable de retourner toutes les colonnes générées correspondant au coût réduit minimum, lorsque l'on utilise

Sans stabilisation Wentges

Instance	primal_degen		dual_degen		any_degen		maxConsCgDegen		nb_col_gen		nb_cols_tot		ratioNbColGen	ratioNbColsTot
	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg		
exeo/031	13%	8%	20%	17%	33%	25%	3	2	15	12	304	277	80%	91%
exeo/032	24%	33%	24%	22%	48%	56%	6	2	25	9	560	229	36%	41%
exeo/054	11%	23%	33%	3%	44%	26%	27	2	81	39	3 012	1 982	48%	66%
exeo/075	28%	6%	1%	3%	30%	10%	5	2	81	63	4 065	4 122	78%	101%
exeo/087	18%	5%	36%	4%	55%	9%	61	3	168	74	8 224	5 940	44%	72%
exeo/102	15%	8%	7%	1%	23%	9%	8	1	111	89	8 961	8 605	80%	96%
exeo/109	14%	7%	21%	1%	36%	8%	36	1	168	101	10 342	9 512	60%	92%
exeo/125	41%	19%	11%	4%	51%	23%	32	3	495	309	22 055	21 435	62%	97%
exeo/162	15%	7%	15%	1%	30%	8%	22	2	150	104	10 225	10 655	69%	104%
exeo/163	29%(⋆)	11%(⋆)	9%(⋆)	2%(⋆)	38%(⋆)	13%(⋆)	40(⋆)	3(⋆)	1 248(⋆)	363(⋆)	80 362(⋆)	40 051(⋆)	29%(⋆)	50% (⋆)
exeo/171	44%(⋆)	21%(⋆)	8%(⋆)	5%(⋆)	52%(⋆)	27%(⋆)	40(⋆)	4(⋆)	13 709(⋆)	686(⋆)	490 707(⋆)	48 253(⋆)	5%(⋆)	10% (⋆)
exeo/174	19%	6%	5%	1%	24%	7%	34	2	310	179	35 501	28 038	58%	79%
exeo/245	8%(⋆)	1%(⋆)	0%(⋆)	0%(⋆)	8%(⋆)	1%(⋆)	12(⋆)	3(⋆)	785(⋆)	671(⋆)	141 229(⋆)	124 197(⋆)	85%(⋆)	88%(⋆)

Avec stabilisation Wentges

Instance	primal_degen		dual_degen		any_degen		maxConsCgDegen		nb_col_gen		nb_cols_tot		ratioNbColGen	ratioNbColsTot
	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg	no_agg	agg		
exeo/031	0%	0%	29%	8 %	29%	8%	5	1	17	13	281	267	76%	95%
exeo/032	16%	23%	39%	23%	55%	46%	12	3	31	13	420	267	42%	64%
exeo/054	45%	11%	11%	4%	55%	15%	21	2	56	27	1,603	1,335	48%	83%
exeo/075	38%	18%	2%	6%	40%	24%	12	2	58	34	2,340	2,187	59%	93%
exeo/087	3%	17%	62%	2%	64%	20%	66	1	107	41	3,627	3,161	38%	87%
exeo/102	34%	27%	27%	9%	61%	36%	26	2	95	22	4,061	2,390	23%	59%
exeo/109	7%	5%	65%	3%	72%	8%	98	1	151	37	5,026	3,585	25%	71%
exeo/125	27%	17%	19%	8%	46%	25%	57	3	427	237	10,318	14,113	56%	137%
exeo/162	19%	2%	26%	2%	45%	3%	28	1	106	59	5,807	5,573	56%	96%
exeo/163(⋆)	25%(⋆)	12%(⋆)	24%(⋆)	3%(⋆)	49%(⋆)	15%(⋆)	69(⋆)	2(⋆)	1,057(⋆)	261(⋆)	33,762(⋆)	24,203(⋆)	25%(⋆)	72%(⋆)
exeo/171(⋆)	29%(⋆)	20%(⋆)	22%(⋆)	7%(⋆)	51%(⋆)	27%(⋆)	82(⋆)	5(⋆)	10,493(⋆)	688(⋆)	170,166(⋆)	32,903(⋆)	7%(⋆)	19%(⋆)
exeo/174	14%	3%	34%	2%	48%	5%	60	2	178	97	12,417	13,082	54%	105%
exeo/245(⋆)	35%(⋆)	0%(⋆)	37%(⋆)	1%(⋆)	72%(⋆)	1%(⋆)	266(⋆)	2(⋆)	1,057(⋆)	137(⋆)	88,040(⋆)	22,439(⋆)	13%(⋆)	25%(⋆)

TABLE 2.6 – Effets de la désagrégation duale optimale (instances `exeo`)

la solution duale désagrégée. En comparaison, l'algorithme de sous-gradient ne retourne que la solution duale désagrégée elle-même, et nous ajoutons dans le maître les colonnes correspondant à une itération de pricing utilisant cette solution duale. Ainsi, l'algorithme de sous-gradient retourne exactement une colonne par centre, alors que l'algorithme de plans sécants n'a pas cette limitation.

Il ressort des expérimentations que les contraintes duales d'agrégation semblent souvent valides pour le problème dual, puisqu'elles permettent de trouver une solution duale désagrégée réalisable un nombre significatif de fois, permettant d'éviter de nombreuses itérations dégénérées lorsque l'algorithme a trouvé la solution primale basique optimale. L'impact sur les bases sous-optimales est cependant plus mitigé : le problème de désagrégation duale proposé ne permet pas souvent de générer un ensemble de colonnes induisant un pivot non-dégénéré. Enfin, le nombre de colonnes distinctes générées semble varier relativement peu. Pourtant, la stratégie de lissage dual (stabilisation de Wentges) permet généralement de diviser le nombre de colonnes générées par deux sur les instances `exeo`. Ce point laisse supposer que les colonnes générées par le problème de désagrégation duale ne sont pas véritablement de meilleure qualité que celles générées en utilisant la solution duale basique retournée par l'algorithme du simplexe.

2.7 Conclusion

Dans ce chapitre, nous avons étudié un nouveau problème de couverture des sommets d'un graphe par des arborescences, sous contraintes de capacité. Nous avons proposé plusieurs modèles mathématiques.

Chaque modèle permet de résoudre la plupart des instances considérées, mais les performances varient selon le type d'instance à résoudre. L'algorithme de branch-and-bound proposé semble le plus efficace en moyenne lorsqu'il est utilisé comme oracle pour résoudre le problème de pricing de la formulation étendue.

En revanche, la méthode d'agrégation de contraintes proposée ne permet pas d'accélérer les performances de l'algorithme de branch-and-price, même si elle mène à une diminution du nombre d'itérations dégénérées. Cela est dû essentiellement à la complexité du sous-problème de désagrégation dual, pour lequel l'investissement en temps de calcul n'est pas compensé par les bénéfices obtenus grâce à la solution duale désagrégée et aux colonnes générées.

Chapitre 3

Minimisation de la somme des rayons

Dans ce chapitre, nous présentons un problème de sectorisation que l'on modélise comme la recherche d'une couverture des sommets d'un graphe par des arborescences. Contrairement au chapitre précédent, le coût des secteurs est basé ici sur une distance maximale, à la manière du problème de p-centre. Nous présentons trois modèles de programmation linéaire en nombres entiers, deux compacts et un basé sur une reformulation de Dantzig et Wolfe. Pour ce dernier, une attention particulière est portée au sous-problème de génération de colonnes, pour lequel nous proposons plusieurs algorithmes. Des résultats numériques sont rapportés sur des instances de la littérature, et des instances réelles issues d'une application industrielle. Les expérimentations montrent que l'algorithme de branch-and-price est capable de résoudre le problème pour des instances beaucoup plus grandes que le modèle compact, qui reste limité à des tailles très faibles.

3.1 Introduction

Nous nous intéressons à un problème de couverture dans le contexte de la collecte de déchets. Notre problème appartient à la famille des problèmes de *sectorisation*. Comme dans tous les problèmes de sectorisation, on cherche à construire des secteurs **"connexes"**, **"compacts"** et **"équilibrés"**.

Nous reprenons ici la modélisation du chapitre précédent concernant les notions de connexité et d'équilibre. Ainsi, nous devons déterminer un centre j pour chaque secteur, ce centre étant choisi parmi les sommets couverts par le secteur. Pour être connexe, un secteur doit être une sous-arborescence de l'arborescence des plus courts chemins partants de j. Par ailleurs, tout secteur doit vérifier deux contraintes de sac à dos, chacune servant à représenter une capacité en terme de charge de travail (l'une pour le temps de traitement d'un secteur, et l'autre pour la quantité de déchets à y récupérer).

En revanche, nous adoptons une approche différente pour représenter la notion de compacité. La compacité peut être modélisée à la manière d'un problème de p-median [ZS83, MJN98], où l'on minimise la somme des distances de chaque sommets au centre de son secteur ou bien comme un problème de p-centre [RMF09, EARMD14] où l'on ne considère que la plus grande distance employée, tous secteurs confondus. Dans ce

chapitre, la fonction objectif de notre problème se rapproche de la modélisation p-centre, puisque nous considérons pour chaque secteur la distance maximale entre tout sommet du secteur et son centre. Par la suite, la plus grande distance au sein d'un secteur sera appelée le rayon du secteur. Cependant, nous représentons le coût total d'une solution comme la somme des rayons de tous les secteurs qui la composent, au lieu de considérer uniquement le secteur de plus grand rayon. En ce sens, notre modèle peut être comparé à ceux de [MCVO03] et [SdAFU14]. L'un des critères de [MCVO03] est la somme des rayons *relatifs* de chaque secteur généré, tandis que [SdAFU14] utilise la somme des diamètres, c'est-à-dire de la distance maximale entre toute paire de sommets du même secteur.

Dans notre problème, nous n'imposons pas de charge de travail minimale par secteur. Dans les instances issues de l'industrie que nous avons étudiées, l'équilibrage n'était pas une demande. Cependant, la charge de travail liée à chaque secteur est bornée supérieurement, il n'est donc pas possible de créer un secteur qui va produire une quantité de travail trop importante. Le nombre de secteurs est prescrit, et les secteurs sont soumis à deux contraintes de capacité. Par ailleurs, notre problème consiste à déterminer une couverture des sommets, et non une partition. Dans les cas où certains sommets appartiennent à beaucoup de plus courts chemins, cela favorise l'existence d'une solution réalisable.

Notre travail concerne des méthodes de résolution exactes basées sur la programmation mathématique pour résoudre notre variante du problème. On s'intéresse en particulier à une méthode de branch-and-price. La principale difficulté est la mise au point d'un oracle efficace pour le sous-problème associé à un centre, qui cherche à calculer le sous-arbre enraciné en ce centre qui minimise le coût réduit. Ce problème peut également être vu comme une extension du problème de sac-à-dos prenant en compte à la fois des contraintes de précédences, deux contraintes de capacités, et la distance au centre (rayon). Nous proposons plusieurs méthodes pour résoudre ce problème. La première repose sur la résolution pour chaque rayon possible d'une généralisation en deux dimensions du branch-and-bound développé dans [CS98]. La seconde méthode repose sur la résolution d'un programme dynamique, basé sur l'algorithme proposé dans [CS97], prenant en compte le rayon dans sa récurrence.

Nous validons numériquement nos différentes méthodes sur des instances tirées d'un cas pratique, ainsi que sur des jeux de données classiques issus de la littérature des problèmes de tournées de véhicules. Nos expérimentations montrent que la formulation basée sur la génération de colonnes est bien plus performante que les formulations compactes. Utiliser le branch-and-bound basé sur l'énumération des rayons semble être la meilleure manière de résoudre les sous-problèmes de génération de colonnes mais le programme dynamique permet de résoudre plus rapidement le problème un nombre important de fois.

Dans la section 3.2, nous présentons formellement le problème, ainsi que les différentes formulations étudiées dans le document. La section 3.3 est dédiée au sous-problème de génération de colonnes. Nous présentons nos résultats numériques dans la section 3.4 avant de conclure.

3.2 Description du problème et formulations mathématiques

Dans cette section, nous présentons formellement le problème étudié. Pour ce faire, nous nous abstrayons de l'application et définissons le problème comme la recherche d'une couverture d'un graphe par des arborescences.

3.2.1 Description formelle du problème

Tout comme dans le chapitre précédent, nous représentons la zone géographique à traiter par un graphe orienté et valué $G = (V, A, c)$ dans lequel chaque sommet $i \in V$ représente un point de collecte et chaque arc $(i, j) \in A$ représente une connexion de i à j de distance $c_{i,j}$. On note $n = |V|$. Pour tout sommet $i \in V$, nous associons également deux valeurs $w_i^1 \in \mathbb{N}$ et $w_i^2 \in \mathbb{N}$ qui correspondent respectivement au temps de traitement et à la quantité de déchets à traiter. La flotte dispose de K véhicules, tous limités par un temps de travail effectif maximal W^1 et une capacité W^2 de déchets pouvant être collectés. Le temps de travail W^1 ne comprend pas les temps de trajet.

Chaque *secteur* que nous cherchons à construire est défini par une paire (U, j), où $U \subseteq V$ est un ensemble de sommets, et $j \in U$ un de ses sommets appelé *centre*. Pour un secteur donné de centre j, on note $d_{i,j}$ la longueur du plus court chemin dans G allant de j à un sommet i et $\pi_j(i)$ le sommet précédant i sur ce chemin. S'il n'existe pas de chemin allant de j à i, alors $d_{i,j} = +\infty$ et par convention, on considère dans ce cas que $\pi_j(i) = j$. Ces plus courts chemins définissent ainsi une arborescence $T^j = (V, A^j)$ enracinée en j avec $A^j = \{(\pi_j(i), i), i \in V \setminus \{j\}\}$. La spécificité du problème que nous considérons ici est que nous définissons le **rayon** d'un secteur (U, j), défini par $\max_{i \in U} d_{i,j}$. D'autre part, pour un secteur (U, j), on a $i \in U \implies \pi_j(i) \in U$.

Dans la suite, si $T^j = (V, A^j)$ est une arborescence enracinée en $j \in V$ et $i \in V \setminus \{j\}$, on notera par $T^j(i)$ la sous-arborescence de T^j enracinée en i et contenant tous les descendants de i. De plus, pour un ensemble de sommets $U \subseteq V$ on notera $T^j[U] = (U, A^j \cap (U \times U))$ la sous-arborescence de T^j enracinée en j et contenant les sommets de U. On dira que l'ensemble U induit une sous-arborescence valide si $j \in U$ et $T^j[U]$ forme une composante connexe dans T^j.

Définition 2 (secteur valide, rayon)**.** Étant donné une arborescence $T^j = (V, A^j)$ enracinée en j, trois vecteurs $\boldsymbol{d}_{\cdot,j}$, $\boldsymbol{w}^1$, $\boldsymbol{w}^2$ de $\mathbb{R}_+^n$, et deux réels W^1 et W^2, un *secteur valide de centre* j est une paire (U, j), telle que $T^j[U]$ est une sous-arborescence valide et tel que $\sum_{i \in U} w_i^\ell \leq W^\ell$ pour $\ell = 1, 2$. Le *rayon* d'un secteur valide (U, j) est défini comme $r(U, j) = \max_{i \in U} d_{i,j}$.

Problème 9 (couverture par secteurs valides de coût minimum)**.** Étant donné un ensemble V de sommets, deux vecteurs $\boldsymbol{w^1}$ et $\boldsymbol{w^2}$ de $\mathbb{N}^n$, trois entiers W^1, W^2, et K, et pour chaque sommet $j \in V$, une arborescence T^j des plus courts chemins de j vers les autres sommets de V, et un vecteur $\boldsymbol{d}_{\cdot,j} \in \mathbb{R}^n$ de distances de plus courts chemins de j vers les sommets de V dans cette arborescence, trouver une couverture de V en K secteurs valides de centres différents qui minimise la somme des rayons des secteurs utilisés.

Le problème de couverture par secteurs valides est NP-difficile, car il est équivalent au problème de bin packing si on considère des arborescences de profondeur 1, des distances nulles et la capacité W^2 égale à $+\infty$.

3.2.2 Formulations compactes

Le problème de couverture par secteurs valides de coût minimum peut se modéliser à l'aide de la programmation linéaire en nombres entiers. Le premier modèle repose sur les variables naturelles du problème (choix des centres, affectation, et rayon). Soit $\boldsymbol{x} \in \{0,1\}^{n\times n}$ une matrice de variables de décision. Pour $i \neq j$, $x_{i,j}$ est égale à 1 si i est couvert par un secteur de centre j, 0 sinon. Par convention, $x_{j,j} = 1$ indique que j est utilisé comme centre d'un secteur. Soit $\boldsymbol{r} \in \mathbb{R}^n$ un vecteur de variables. Pour chaque sommet $j \in V$, la variable réelle r^j représente le rayon du secteur de centre j si j est le centre d'un secteur, 0 sinon. Le problème peut alors être modélisé de la manière suivante :

$$
(\mathrm{P}_0) = \begin{cases}
\min \displaystyle\sum_{j\in V} r^j & & & \\
\text{t.q.} \quad \displaystyle\sum_{j\in V} x_{j,j} \leq K, & & & (3.1)\\
\displaystyle\sum_{j\in V} x_{i,j} \geq 1 & \forall i \in V, & & (3.2)\\
\displaystyle\sum_{i\in V} w_i^\ell \, x_{i,j} \leq W^\ell & \ell = 1,2, & \forall j \in V, & (3.3)\\
x_{i,j} \leq x_{\pi_j(i),j} & \forall i \in V, i \neq j, & \forall j \in V, & (3.4)\\
r^j \geq d_{i,j}\, x_{i,j} & \forall i \in V, & \forall j \in V, & (3.5)\\
r^j \in \mathbb{R}^+ & & \forall j \in V, & (3.6)\\
x_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V. & (3.7)
\end{cases}
$$

La fonction objectif est égale à la somme des rayons. Notons que dans toute solution optimale, si j n'est pas utilisé comme centre d'un secteur, alors $r^j = 0$ implicitement. La contrainte (3.1) borne supérieurement le nombre de secteurs créés ; les contraintes (3.2) imposent la couverture de chaque sommet par au moins un secteur ; et les contraintes (3.3) assurent que les secteurs ne dépassent pas les capacités maximum. Les contraintes de précédences sont exprimées par (3.4). Les contraintes (3.5) assurent que les rayons sont correctement calculés.

Nous proposons maintenant une meilleure formulation du problème, qui repose sur des variables différentes. On s'inspire pour cela du modèle de [ELP04] pour le problème de p-centre. Pour chaque centre $j \in V$, on définit par σ_j l'ordre des sommets de V par distance croissante au centre j. Pour $i, j \in V$ donnés, soit $\Delta_{i,j}$ l'écart de distance au centre j entre le sommet i et son prédécesseur dans l'ordre σ_j. Nous pouvons définir cet écart de la manière suivante : $\Delta_{j,j} = 0$ et $\Delta_{\sigma_j(k),j} = d_{\sigma_j(k),j} - d_{\sigma_j(k-1),j}$ pour $k = 2, \ldots, n$. On peut maintenant reporter le coût d'un secteur sur plusieurs sommets, à savoir ceux qui se trouvent avant le sommet définissant le rayon dans l'ordre σ_j. Pour cela, nous

introduisons une nouvelle matrice $\boldsymbol{y} \in \{0,1\}^{n\times n}$ de variables binaires. Pour toute paire de sommets (i,j), $y_{i,j}$ prend la valeur 1 lorsque le rayon du secteur de centre j est supérieur ou égal à $d_{i,j}$, et 0 sinon. Ainsi, si j est sélectionné comme centre d'un secteur, et si le sommet $\sigma_j(i)$ définit le rayon du secteur j, nous avons $y_{j,j} = y_{\sigma_j(2),j} = \ldots = y_{\sigma_j(i),j} = 1$ et $y_{\sigma(i+1),j} = \ldots = y_{\sigma(n),j} = 0$.

Le problème peut alors être modélisé par le MIP suivant.

$$
(\mathrm{P}) = \begin{cases}
\min \displaystyle\sum_{j\in V}\sum_{i\in V} \Delta_{i,j}\, y_{i,j} & & & \\
\text{t.q.} \quad \displaystyle\sum_{j\in V} x_{j,j} \le K, & & & (3.8)\\
\displaystyle\sum_{j\in V} x_{i,j} \ge 1 & \forall i \in V, & & (3.9)\\
\displaystyle\sum_{i\in V} w_i^{\ell}\, x_{i,j} \le W^{\ell} & \ell = 1,2, & \forall j \in V, & (3.10)\\
x_{i,j} \le x_{\pi_j(i),j} & \forall i \in V, i \neq j, & \forall j \in V, & (3.11)\\
y_{\sigma_j(k),j} \le y_{\sigma_j(k-1),j} & \forall k \in \{2,\ldots,n\}, & \forall j \in V, & (3.12)\\
x_{i,j} \le y_{i,j} & \forall i \in V, & \forall j \in V, & (3.13)\\
x_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V, & (3.14)\\
y_{i,j} \in \{0,1\} & \forall i \in V, & \forall j \in V. & (3.15)
\end{cases}
$$

Le modèle ci-dessus diffère de (3.1)–(3.7) par sa fonction objectif, qui est calculée à partir des variables $\boldsymbol{y}$, et les contraintes (3.12) et (3.13). Les contraintes (3.12) garantissent que l'affectation des valeurs aux variables $\boldsymbol{y}$ est cohérente, et les contraintes (3.13) impliquent que si i appartient au secteur de centre j, alors le rayon de ce secteur est supérieur ou égal à $d_{i,j}$.

3.2.3 Formulation étendue

Dans cette section, nous montrons comment construire une formulation étendue de (P) en utilisant la décomposition de Dantzig-Wolfe.

Pour $j \in V$, on note $\mathcal{G}^j$ l'ensemble des points extrêmes de l'enveloppe convexe des solutions entières au système (3.10)–(3.15) associé à j. Notons qu'une solution associée au centre j peut représenter un secteur vide, dans lequel même le sommet j n'est pas couvert. Nous pouvons réécrire le modèle compact (P) en exprimant la valeur des vecteurs $\boldsymbol{x}$ et $\boldsymbol{y}$ comme une combinaison entière des éléments de $\mathcal{G}^j$. Étant donné $g \in \mathcal{G}^j$, soit $x^g_{i,j}$ (resp. $y^g_{i,j}$) la valeur de la variable $x_{i,j}$ (resp. $y_{i,j}$) dans g. Soit $\boldsymbol{\lambda}_j \in \{0,1\}^{|\mathcal{G}^j|}$ un vecteur de variables binaires tel que λ^g_j est égal à 1 si le point extrême g est utilisé dans la solution, 0 sinon. Le nouveau modèle peut être défini comme suit.

$$
(\mathrm{P}^M) = \begin{cases}
\min \displaystyle\sum_{j\in V}\sum_{i\in V} \Delta_{i,j} \left(\sum_{g\in\mathcal{G}^j} y^g_{i,j}\,\lambda^g_j\right) & & & & \\
\text{t.q.} \quad \displaystyle\sum_{j\in V}\left(\sum_{g\in\mathcal{G}^j} x^g_{j,j}\,\lambda^g_j\right) \le K, & & & (\kappa) & (3.16)\\
\displaystyle\sum_{j\in V}\left(\sum_{g\in\mathcal{G}^j} x^g_{i,j}\,\lambda^g_j\right) \ge 1 & \forall i\in V, & & (\boldsymbol{\phi}) & (3.17)\\
\displaystyle\sum_{g\in\mathcal{G}^j} \lambda^g_j = 1 & \forall j\in V, & & (\boldsymbol{\alpha}) & (3.18)\\
\lambda^g_j \in \{0,1\} & \forall g\in\mathcal{G}^j, & \forall j\in V. & & (3.19)
\end{cases}
$$

On appelle ce problème le problème maître. Le modèle obtenu possède un nombre polynomial de contraintes mais un nombre exponentiel de variables. Pour le résoudre efficacement, on a recours à une méthode de branch-and-price. Pour résoudre la relaxation linéaire de P^M, on résout initialement une restriction de ce modèle à un sous-ensemble de variables (on parle de problème maître restreint). On doit ensuite résoudre un sous-problème pour déterminer la variable non générée avec le coût réduit le plus négatif. Ce sous-problème peut se décomposer en n sous-problèmes (un par centre possible). En pratique, chaque sous-problème consiste à calculer le secteur correspondant au plus petit coût réduit, selon le centre considéré. Si aucun secteur n'est associé à un coût réduit négatif, alors la solution courante de la relaxation continue de (P^M) est optimale. Sinon, nous rajoutons les n colonnes générées au problème maître restreint (y compris les colonnes correspondant à un coût réduit positif ou nul).

En associant les variables duales κ, ϕ_i pour tout $i \in V$ et α_j pour tout $j \in V$ respectivement aux contraintes (3.16), (3.17) et (3.18), nous obtenons pour chaque centre le problème de pricing suivant.

$$
(\mathrm{P}^j) = \begin{cases}
\min \displaystyle\sum_{i\in V} \Delta_{i,j}\,y_{i,j} - \kappa\,x_{j,j} - \sum_{i\in V}\phi_i\,x_{i,j} - \alpha_j & & & \\
\Leftrightarrow \quad -\alpha_j - \max \quad \kappa\,x_{j,j} + \displaystyle\sum_{i\in V}\phi_i\,x_{i,j} - \sum_{i\in V}\Delta_{i,j}\,y_{i,j} & & & \\
\text{t.q.} \quad \displaystyle\sum_{i\in V} w^\ell_i\,x_{i,j} \le W^\ell & \ell = 1,2 & & (3.20)\\
x_{i,j} \le x_{\pi_j(i),j} & \forall i\in V, i\neq j, & & (3.21)\\
y_{\sigma_j(k),j} \le y_{\sigma_j(k-1),j} & \forall k\in\{2,\cdots,|V|\}, & & (3.22)\\
x_{i,j} \le y_{i,j} & \forall i\in V, & & (3.23)\\
x_{i,j} \in \{0,1\} & \forall i\in V, & & (3.24)\\
y_{i,j} \in \{0,1\} & \forall i\in V. & & (3.25)
\end{cases}
$$

En considérant ϕ_i comme étant un profit sur la couverture d'un sommet par un secteur, ce problème revient à chercher un secteur valide qui minimise le rayon tout en maximisant le profit des sommets couverts.

Pour obtenir une solution entière de (P^M), on utilise une méthode de branch-and-price, dont les éléments sont identiques à ceux décrits dans la section 2.5. Dans la section suivante, nous nous concentrons sur la résolution de la relaxation linéaire du modèle (P^M). Le problème maître restreint est résolu à l'aide d'un solveur de programmation linéaire. La difficulté du problème réside principalement dans la résolution des sous-problèmes de génération de colonnes, nous proposons donc plusieurs algorithmes pour les résoudre efficacement.

3.3 Méthodes de résolution du sous-problème de génération de colonnes

Dans cette section, nous allons nous intéresser au problème de pricing introduit dans la section précédente. Celui-ci se décompose en n sous-problèmes de pricing pour chacun desquels on fixe un sommet j de V comme étant le centre. Ainsi, pour chaque sommet j de V, on cherche à générer un point extrême g de $\mathcal{G}^j$ tel que le coût réduit de la variable associée λ_j^g de (P^M) soit minimisé. Pour simplifier les notations, l'indice j est omis dans les données du problème. Nous considérons également qu'il s'agit d'un problème de maximisation. Formellement, le problème à résoudre est le suivant.

Problème 10 (secteur valide maximisant la somme des profits moins le rayon)**.** Étant donné une arborescence $T = (V, A)$ enracinée en j, deux vecteurs $\boldsymbol{d}$, $\boldsymbol{p}$, de $\mathbb{R}^n$, deux vecteurs $\boldsymbol{w^1}$ et $\boldsymbol{w^2}$ de $\mathbb{N}^n$, ainsi que deux entiers W^1 et W^2, trouver un secteur valide (U, j) qui maximise $\sum_{i \in U} p_i - r(U, j)$.

Ce problème est une généralisation de deux variantes du problème de sac-à-dos étudiées dans la littérature. La première de ces variantes est appelée *Tree Knapsack Problem*. Elle est définie par une seule contrainte de sac-à-dos mais pas de rayons (ou, de manière équivalente, des rayons nuls). Dans [CS98], un algorithme de branch-and-bound a été proposé pour résoudre ce problème, tandis qu'une approche par programmation dynamique est présentée dans [CS97]. Les deux méthodes sont basées sur un parcours en profondeur (DFS) de l'arbre des précédences.

La seconde variante est appelée *Penalized Knapsack Problem*. Elle prend en compte une contrainte de sac-à-dos, mais pas de contraintes de précédence. Les rayons sont ici appelés des pénalités, c'est-à-dire que la plus forte pénalité associée à un objet appartenant à une solution est soustraite du profit total de cette solution. Dans [CR06], ce problème est résolu par un programme dynamique dans lequel les objets sont rangés par ordre de pénalité croissante. Cependant, les contraintes de précédence altèrent la structure de notre problème, nous empêchant de tirer parti d'un tel ordre.

Pour le sous-problème avec centre fixé, nous avons mis au point deux familles d'approches. La première repose sur l'énumération des valeurs de rayon $r(U, j)$ possibles. On doit alors résoudre itérativement une variante d'un problème de sac à dos avec contraintes de précédence. La deuxième famille d'approches consiste à attaquer directement le problème en calculant à la fois les sommets sélectionnés et le rayon. Notons

que, puisque le problème de sac à dos est faiblement NP-difficile, et que la complexité de l'algorithme de programmation dynamique décrit dans la section 3.3.2 a une complexité pseudo-polynomiale, alors le problème 10 est faiblement NP-difficile.

3.3.1 Résolution des sous-problèmes par énumération des rayons et branch-and-bound

Une première approche pour résoudre les sous-problèmes consiste à fixer la valeur du rayon de la solution, de manière à obtenir une variante d'un problème traité dans la littérature, nommé *Tree Knapsack Problem (TKP)*.

Problème 11 (sac à dos 2D avec contraintes de précédences en arborescence (TKP-2D))**.** Étant donné une arborescence $T = (V, A)$ enracinée en j, un vecteur $\boldsymbol{p}$ de $\mathbb{R}^n$, deux vecteurs $\boldsymbol{w^1}$ et $\boldsymbol{w^2}$ de $\mathbb{N}^n$, ainsi que deux entiers W^1 et W^2, trouver un secteur valide (U, j) qui maximise $\sum_{i \in U} p_i$.

On a au plus n valeurs possibles de rayon pour chaque centre. Pour chaque valeur r possible, on peut éliminer en prétraitement tous les sommets dont la distance au centre est supérieure à r. Pour un centre j et un rayon r donnés, le sous-problème revient à résoudre TKP-2D$(T; r) - r + \alpha_j$, où TKP-2D$(T; r)$ est le problème défini de la manière suivante.

$$
(\text{TKP-2D}(T^j; r)) = \begin{cases}
\max \displaystyle\sum_{i \in V} p_i \, x_i & & \\
\text{t.q.} \quad \displaystyle\sum_{i \in V} w_i^\ell \, x_i \leq W^\ell & \ell = 1, 2 & (3.26) \\
x_i \leq x_{\pi_j(i)} & \forall i \in V, i \neq j, & (3.27) \\
x_i = 0 & \forall i \in V, d_i > r & (3.28) \\
x_i \in \{0, 1\} & \forall i \in V & (3.29)
\end{cases}
$$

Le problème ci-dessus a déjà été étudié dans sa version 1D dans [CS97, CS98]. Un algorithme de programmation dynamique [CS97], ainsi qu'un algorithme de séparation et évaluation [CS98] ont été proposés. Ces algorithmes ont été adaptés au cas à deux dimensions dans [CGM+14].

L'algorithme de branch-and-bound que nous présentons est une adaptation de celui présenté par Cho et Shaw [CS98] pour le problème de sac-à-dos 1D avec contraintes de précédence en arborescence.

Dans [CS98], le schéma de branchement de l'algorithme de branch-and-bound consiste à sélectionner un sommet $i \in V$, puis construire deux sous-problèmes : un premier sous-problème dans lequel le sommet i n'appartient pas à la solution, et un second sous-problème dans lequel le sommet i appartient à la solution. La borne duale pour chaque nœud de l'arbre de branchement est obtenue par relaxation lagrangienne des contraintes de sac-à-dos en utilisant une recherche linéaire pour calculer les multiplicateurs lagrangiens optimaux. Cette recherche retire itérativement les sous-arbres de l'arbre de précédence enracinés en un objet critique. Cette séquence d'objets critiques est utilisée pour définir les décisions de branchement successives dans l'arbre de branch-and-bound.

Dans le cas à deux dimensions, nous employons le même schéma de branchement et la même méthode d'évaluation. Cependant, puisque la relaxation lagrangienne est appliquée à deux contraintes de sac-à-dos simultanément, il n'est plus possible de trouver les multiplicateurs lagrangiens optimaux par une recherche linéaire. En conséquence, nous utilisons un algorithme de descente pour optimiser la valeur des multiplicateurs. Afin de tirer pleinement parti de la stratégie de branchement présentée dans [CS98], l'algorithme de descente doit permettre de retirer itérativement des sous-arbres enracinés en des objets critiques, de la même manière que la recherche linéaire dans le cas à une dimension.

Commençons par formaliser le problème obtenu après relaxation des deux contraintes de capacité.

Problème 12 (sous-arbre de poids maximum)**.** Étant donné un arbre $T = (V, A)$ enraciné en j et un vecteur $\boldsymbol{p}$ dans $\mathbb{R}^n$, trouver un sous-arbre valide $T[U]$ maximisant $\sum_{i \in U} p_i$.

Pour des multiplicateurs $\boldsymbol{\mu} \in \mathbb{R}^2$ donnés, le problème est équivalent au problème de sous-arbre de poids maximum avec $\tilde{p}_i = p_i - \boldsymbol{\mu} \cdot \boldsymbol{w}_i$ pour tout $i \in V$. Ce problème peut être résolu en temps linéaire [CS98] en utilisant un programme dynamique. Un état de ce programme est défini par un sommet $i \in V$ et correspond au sous-arbre maximum de $T(i)$. Le programme dynamique peut alors être défini par

$$\beta(i) = \max\{0, \tilde{p}_i + \sum_{k:\pi(k)=i} \beta(k)\},$$

et la solution optimale est construite à partir de $\beta(j)$. En utilisant un parcours en profondeur inverse des sommets de T, tous les états du programme dynamique sont calculés itérativement.

Si nous limitons l'algorithme de descente à des directions augmentant la valeur des multiplicateurs lagrangiens, alors la solution de chaque relaxation est obtenue en retirant des sous-arbres de la solution précédente. Cette propriété découle du fait que les valeurs $\tilde{p}_i$ et $\beta(i)$ diminuent proportionnellement à $\boldsymbol{\mu}$. La solution optimale est donc immuable jusqu'à ce qu'au moins un des états du programme dynamique soit associé à une valeur $\beta(i^*)$ négative (ce qui détermine la longueur du pas à effectuer dans la direction choisie), auquel cas nous retirons le sous-arbre enraciné en l'objet critique i^*.

La séquence d'objets critiques produite par l'algorithme de descente détermine les décisions de branchement successives dans notre algorithme de branch-and-bound.

Nous pouvons finalement définir la direction de descente utilisée par l'algorithme. La direction la plus naturelle correspond à la violation des contraintes de capacité par la solution optimale de la relaxation lagrangienne, pour des multiplicateurs $\boldsymbol{\mu}$ donnés. Cependant, des tests préliminaires ont montré que ne tenir compte que d'une seule dimension de sac-à-dos, à savoir celle correspondant à la contrainte la plus violée, offre de meilleurs performances.

Lorsque nous résolvons (TKP-2D($T^j; r$)) pour un rayon r donné, nous obtenons une borne primale pour (P^j) pouvant être utilisée pour élaguer des nœuds de branchement pour les autres rayons. Nous pouvons également obtenir une borne duale pour tout rayon $r' < r$ en conservant l'ensemble de sommets couverts et en appliquant le nouveau rayon r' (sans tenir compte du rayon induit par cet ensemble de nœuds).

De ce fait, nous choisissons de nous concentrer sur l'obtention d'une bonne borne primale en commençant par évaluer le meilleur rayon *supposé*, c'est-à-dire le rayon qui a

été le plus souvent associé à une solution optimale de (P^j) lors des itérations précédentes. Ensuite, nous évaluons les rayons dans l'ordre décroissant afin d'obtenir des bornes duales. Les tests expérimentaux ont montré que cette stratégie améliorait les performances de l'algorithme de manière significative.

3.3.2 Résolution directe des sous-problèmes par Programmation Dynamique

Notre branch-and-bound est basé sur l'énumération des rayons possibles. Le programme dynamique présenté ci-dessous résout le sous-problème en calculant le rayon en même temps que les sommets sélectionnés. Nous présentons dans un premier temps le programme dynamique que l'on peut utiliser lorsque le rayon est connu, et qui est une extension en deux dimensions de celui proposé dans [CS97]. Dans un deuxième temps, nous montrons comment le calcul du rayon peut être pris en compte efficacement dans la récurrence, ainsi que plusieurs techniques qui améliorent la performance de l'algorithme.

3.3.2.1 Programme dynamique pour le problème TKP-2D

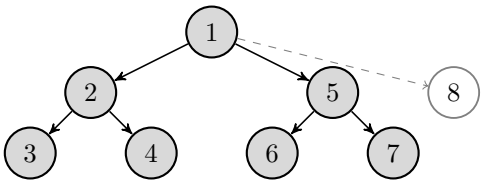
La version originale de ce programme dynamique a été proposée par Cho et Shaw [CS97] pour le cas à une seule dimension, en tant que variante du programme dynamique de [JN83], puis adapté dans [CGM+14] à deux dimensions de sac-à-dos. Nous rapportons le programme dynamique dans notre formalisme. Dans ce qui suit, nous considérons une arborescence $T = (V, A)$ enracinée en j. Pour simplifier les notations, on considère que les sommets de V sont numérotés dans un ordre DFS. Sous cette hypothèse, si on sélectionne un sommet $k \in V$, le prochain sommet à considérer sera le sommet $k+1$ (le prochain dans l'ordre DFS). En revanche, si on ne choisit pas le sommet k, aucun sommet de $T(k)$ ne peut être choisi non plus. Dans ce cas, le prochain sommet à considérer est $\eta(k) = \min\{i : i > k, i \notin T(k)\}$. Par convention, s'il ne reste plus de sommets à considérer après avoir choisi de ne pas prendre le sommet d'indice k, on pose $\eta(k) = n+1$. Le lien entre l'arbre de précédence et le graphe d'état du programme dynamique est rapporté Figures 3.1a et 3.1b.

En se basant sur cette structure de précédences, on peut définir les états du programme dynamique par des paires $(k, \boldsymbol{w})$, où $k \in \{1, \dots, n+1\}$ représente l'indice dans l'ordre DFS et $\boldsymbol{w} \in \{0, \dots, W^1\} \times \{0, \dots, W^2\}$ le vecteur de consommation de ressources. Puisque les profits sont issus de la valeur de variables duales, ils ne sont pas entiers. Par conséquent, seul un programme dynamique basé sur les poids est envisageable.

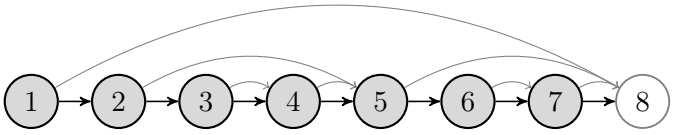
Dans ce qui suit, pour deux vecteurs $\boldsymbol{u}$ et $\boldsymbol{v}$ de même dimension, $\boldsymbol{u} \leq \boldsymbol{v}$ et $\boldsymbol{u}+\boldsymbol{v}$ s'effectuent composantes par composantes. En notant $\boldsymbol{W} = \begin{pmatrix} W^1 \\ W^2 \end{pmatrix}$, le programme dynamique s'écrit comme suit.

$$\psi^{\mathrm{CS}}(k, \boldsymbol{w}) = \begin{cases} 0 & \text{si } k = n+1, \\ \max\{p_k + \ \psi^{\mathrm{CS}}(k+1, \boldsymbol{w} + \boldsymbol{w}_k), \psi^{\mathrm{CS}}(\eta(k), \boldsymbol{w})\} & \text{si } k \leq n \text{ et } \boldsymbol{w} + \boldsymbol{w}_k \leq \boldsymbol{W}, \\ \psi^{\mathrm{CS}}(\eta(k), \boldsymbol{w}) & \text{sinon,} \end{cases}$$

où $\psi^{\mathrm{CS}}(k, \boldsymbol{w})$ représente la valeur de la meilleure solution utilisant uniquement des



(a) Un graphe de précédence (avec le sommet fictif 8 de profit et poids 0)



(b) Structure des successeurs correspondant au graphe de précédence de la figure 3.1a. Les arcs noirs sont les arcs $(k, k+1)$. Les arcs gris sont les arcs $(k, \eta(k))$.

FIGURE 3.1 – Lien entre arbre de précédence et structure des successeurs.

sommets de $\{k, \dots, n+1\}$ avec une consommation de ressources égale à $\boldsymbol{w}$. Une solution optimale est trouvée en calculant $\psi^{\text{CS}}(1, \begin{pmatrix} 0 \\ 0 \end{pmatrix})$.

Ce programme dynamique peut être résolu en calculant un plus long chemin dans un graphe acyclique, où les sommets sont les états et les arcs correspondent aux choix possibles dans la formule de récurrence. Ce graphe comporte de l'ordre de $\mathcal{O}(nW^1W^2)$ sommets et arcs. Les plus longs chemins issus du sommet $(1, \begin{pmatrix} 0 \\ 0 \end{pmatrix})$ peuvent être calculés en un temps de l'ordre de $\mathcal{O}(nW^1W^2)$. L'algorithme de Cho et Shaw [CS97] est une application directe de l'algorithme de Bellman sur ce graphe dans le cas où une seule contrainte de capacité est considérée.

D'un côté, les contraintes de précédence incitent à utiliser un ordre DFS des objets, afin de pouvoir intégrer ces contraintes directement dans la relation de récurrence. D'un autre côté, les meilleures méthodes de programmation dynamique appliquées au Penalized knapsack problem [DCPS17] traitent les objets dans l'ordre croissant des pénalités afin de ne pas requérir une dimension supplémentaire dans l'espace d'état. Cependant, ces deux ordres risquent souvent d'être contradictoires. Prenons l'exemple d'une instance de 4 objets $\{i_0, i_1, i_2, i_3\}$ telle que les précédences sont définies par $\pi(i_1) = \pi(i_2) = i_0$, $\pi(i_3) = i_2$ et telle que les rayons vérifient les inégalités $r(i_0) < r(i_2) < r(i_1) < r(i_3)$. Clairement, une telle instance d'admet aucun ordre DFS dans lequel les objets sont ordonnés selon un ordre croissant des rayons (pénalités). De ce fait, le programme dynamique de la section suivante sera basé sur un ordre DFS des objets ne tenant pas compte des rayons.

3.3.2.2 Extension du programme dynamique prenant en compte le rayon

Dans cette section, nous montrons comment étendre le programme dynamique ci-dessus au cas où le rayon entre en compte dans le calcul du coût de la solution. Nous obtenons ainsi une méthode pour résoudre directement (P^j). Pour simplifier les notations, on note r_i le rayon correspondant au sommet i.

L'idée générale derrière le programme dynamique suivant est d'adapter ψ^{CS} afin de tenir compte du rayon courant dans la solution partielle. Désormais, un état du programme est de la forme $(k, \boldsymbol{w}, r)$, avec k et $\boldsymbol{w}$ définis comme dans ψ^{CS}, et r représente le rayon de la solution partielle courante. Quand nous considérons l'objet k, le rayon n'évolue que si k est ajouté à la solution partielle et que $r_k > k$. Le cas échéant, nous mettons également à jour le coût de l'état généré afin de tenir compte de la pénalité correspondant au nouveau rayon.

Étant donné $x \in \mathbb{R}$, soit $(x)^+ = \max\{0, x\}$. Si $\boldsymbol{x}$ est un vecteur, alors $(\boldsymbol{x})^+$ est appliqué à chaque composante de $\boldsymbol{x}$. On considère que les sommets sont toujours dans un ordre DFS.

$$\psi^{\mathrm{R}}(k, \boldsymbol{w}, r) = \begin{cases} 0 & \text{si } k = n+1, \\ \max\Big\{ p_k + (r_k - r)^+ + \psi^{\mathrm{R}}(k+1, \boldsymbol{w} + \boldsymbol{w}_k, \max\{r, r_k\}), & \text{si } k \leq n \\ \qquad\qquad \psi^{\mathrm{R}}(\eta(k), \boldsymbol{w}, r) & \text{et } \boldsymbol{w} + \boldsymbol{w}_k \leq \boldsymbol{W}, \Big\} \\ \psi^{\mathrm{R}}(\eta(k), \boldsymbol{w}, r) & \text{sinon.} \end{cases}$$

Une solution optimale est obtenue en calculant $\psi^{\mathrm{R}}(1, \begin{pmatrix} 0 \\ 0 \end{pmatrix}, 0)$. Puisque le graphe correspondant n'a pas de circuit, nous utilisons un algorithme de label-setting pour résoudre le programme. Autrement dit, nous considérons les sommets de ce graphe dans un ordre topologique, et nous conservons un ensemble de solutions partielles non-dominées pour chaque sommet. Les solutions sont représentées par des labels. Un label ℓ_i est un tuple $(p_i, k_i, \boldsymbol{w}_i, r_i)$ contenant le profit du chemin partiel, la position dans l'ordre DFS, la consommation de ressource et le rayon courant. On notera ces valeurs $p(\ell_i)$, $k(\ell_i)$, $\boldsymbol{w}(\ell_i)$, $r(\ell_i)$. La complexité théorique de cet algorithme est $\mathcal{O}(|V|^2 W^1 W^2)$.

Nous pouvons maintenant définir des règles de dominance entre les labels qui vont permettre de réduire leur nombre et ainsi d'accélérer la résolution du programme dynamique. Soient deux labels ℓ_i et ℓ_j tels que $k(\ell_i) = k(\ell_j)$. Le label ℓ_i domine le label ℓ_j si au moins l'une des règles de dominance suivantes est vérifiée. Intuitivement, ℓ_i domine ℓ_j si ℓ_i a un profit plus élevé que ℓ_j, tout en consommant moins de ressources et en étant plus proche du centre.

Règle de dominance 1. Le label ℓ_i domine le label ℓ_j si $p(\ell_i) \geq p(\ell_j)$, $r(\ell_i) \leq r(\ell_j)$ et $\boldsymbol{w}(\ell_i) \leq \boldsymbol{w}(\ell_j)$

La deuxième règle de dominance indique que ℓ_i domine ℓ_j si ℓ_i a un meilleur profit et *compense mieux* le coût de son rayon que ℓ_j tout en consommant moins de ressources.

Règle de dominance 2. Le label ℓ_i domine le label ℓ_j si $p(\ell_i) \geq p(\ell_j)$, $p(\ell_i) - r(\ell_i) \geq p(\ell_j) - r(\ell_j)$ et $\boldsymbol{w}(\ell_i) \leq \boldsymbol{w}(\ell_j)$.

Proposition 3: La règle de dominance 2 est correcte.

Démonstration. Si la règle de dominance 2 est vérifiée et que $r(\ell_i) \leq r(\ell_j)$, alors la règle de dominance 1 est également vérifiée, donc ℓ_i domine ℓ_j.

Supposons alors $r(\ell_i) > r(\ell_j)$. ℓ_i est un meilleur label que ℓ_j, puisque son profit net $p(\ell_i) - r(\ell_i)$ est plus élevé et qu'il consomme moins de ressources. Pour justifier la conservation de ℓ_j, il faut partir du principe que nous allons lui ajouter un ensemble d'objets supplémentaires I. Soit $\oplus$ l'opération consistant à ajouter un ensemble d'objets à un label. Puisque $k(\ell_i) = k(\ell_j)$ et $\boldsymbol{w}(\ell_i) \leq \boldsymbol{w}(\ell_j)$, on peut également ajouter I à ℓ_i , puis comparer les deux labels $(\ell_i \oplus I)$ et $(\ell_j \oplus I)$. Le coût réel des deux nouveaux labels étant

$$\Big(p(\ell_i) + p(I) - \max\{r(\ell_i), r(I)\}\Big) \geq \Big(p(\ell_j) + p(I) - \max\{r(\ell_j), r(I)\}\Big),$$

alors $(\ell_i \oplus I)$ reste plus intéressant que $(\ell_j \oplus I)$. Nous pouvons en conclure que ℓ_i domine ℓ_j. □

La règle de dominance 2 est au moins aussi forte que la règle de dominance 1 et a le même coût de calcul. C'est donc elle qui est utilisée dans notre algorithme.

La validité du programme dynamique dérive du fait que les sommets sont évalués selon un ordre DFS de l'arbre des précédences. Tout ordre DFS permet de garantir la cohérence des indices $k + 1$ et $\eta(k)$, pour tout sommet $k \in V$. Notons qu'il existe un nombre exponentiel d'ordres DFS possibles. Parmi ceux-ci, nous choisissons l'ordre DFS dans lequel les sommets sont également ordonnés par ordre décroissant des rayons induits par leurs sous-arbres.

Pour certaines instances, l'utilisation du concept de rayon du sous-arbre pour construire l'ordre DFS permet de meilleures performances en pratique pour le programme dynamique. Une hypothèse plausible pour cette observation est que les états du programme dynamique sont associés à leur rayon final plus rapidement, entraînant un coût plus élevé pour les labels générés qui peuvent alors être filtrés plus tôt.

3.3.2.3 Filtrage lagrangien des labels

Le programme dynamique décrit dans cette section a un nombre doublement pseudo-polynomial d'états. Le nombre de labels à énumérer peut être très grand dans certains cas, d'autant que le nombre de dimensions limite la capacité de l'algorithme à éliminer des labels par dominance. L'objectif de notre algorithme de filtrage est de détecter des transitions dans le programme dynamique qui ne peuvent jamais être utilisées dans une solution optimale. Pour ce faire, nous utilisons une technique inspirée de la méthode SSDP (voir par exemple [IN94]), qui résout un problème relâché, puis réintroduit itérativement les contraintes dans le problème. Dans notre implémentation, nous appliquons la relaxation lagrangienne aux contraintes de capacité.

Pour un vecteur de multiplicateurs de lagrange $\boldsymbol{\zeta} \in \mathbb{R}^2$ donné, nous calculons le profit lagrangien $\hat{p}_k$ de chaque objet k, c'est-à-dire $\hat{p}_k = p_k - \boldsymbol{\zeta} \cdot \boldsymbol{w}_k$. La récurrence devient la suivante.

$$\psi^{\mathrm{LR}}(k, r) = \max\Big\{\hat{p}_k + (r_k - r)^+ + \psi^{\mathrm{LR}}(k+1, \max\{r, r_k\}), \quad \psi^{\mathrm{LR}}(\eta(k), r)\Big\} \tag{3.30}$$

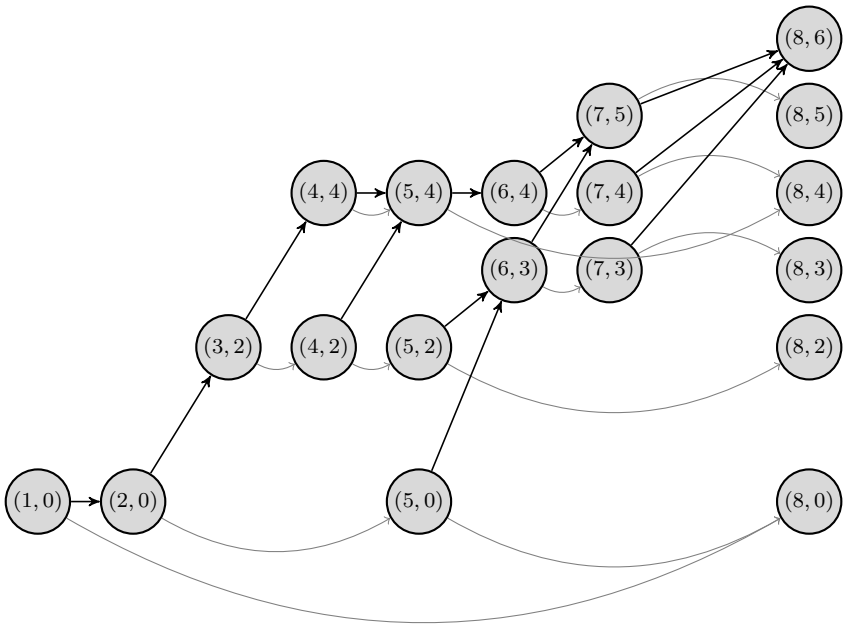


FIGURE 3.2 – Graphe de programmation dynamique obtenu lors de la relaxation lagrangienne. Les labels sont déterminés par la position dans le DFS et le rayon courant. Les arcs en gras correspondent à la sélection de l'objet courant. Les arcs en gris correspondent à ne pas sélectionner l'objet courant.

Nous construisons le graphe correspondant au programme dynamique (3.30). Chaque sommet du graphe est un état du programme dynamique. Les arcs correspondent aux décisions possibles à partir de chaque état (choisir un objet ou non). Un exemple d'un tel graphe est donné Figure 3.2, sur lequel les coûts associés à chaque arc est défini de la manière suivante. Chaque sommet (k, r) a deux arcs sortants. L'arc $a = \big((k, r), (k + 1, \max\{r, r_k\})\big)$, correspond à la sélection du sommet k à la solution, et a pour coût $\tilde{p}_a = \hat{p}_k + (r_k - r)^+$. L'arc $a' = \big((k, r), (\eta(k), r)\big)$ correspond à la non sélection de ce sommet, et n'engendre aucun coût supplémentaire, c'est-à-dire $\tilde{p}_{a'} = 0$.

La complexité de cet algorithme de programmation dynamique est $\mathcal{O}(|V|^2)$, puisque chacun des n sommets initiaux est associé, au maximum, à un label par rayon atteignable.

On suppose qu'on connaît une borne primale PB pour la valeur de la solution optimale. Par exemple, cette borne peut avoir été calculée à partir d'une solution réalisable. Pour un vecteur de multiplicateurs donné, on calcule pour chaque sommet (i, r) du graphe d'état deux valeurs $\omega^-((i, r))$ et $\omega^+((i, r))$, respectivement la valeur d'un plus long chemin du sommet $(1, 0)$ jusqu'au sommet (i, r) et celle d'un plus long chemin du sommet (i, r) jusqu'au sommet $(n + 1, 0)$. Pour un arc $a = ((i, r), (j, r'))$ donné, si $\omega^-((i, r)) + \tilde{p}_a + \omega^+((j, r')) < PB$, cet arc ne peut appartenir à une solution optimale. L'intérêt de cette fixation est qu'elle reste valide lorsqu'on résout le problème original avec toutes ses dimensions. En pratique, si on supprime l'arc partant du sommet (k, r) et correspondant à sélectionner le sommet k dans la phase de filtrage, on ne considérera pas les transitions issues de tout label $(p, k, \boldsymbol{w}, r)$ quelles que soient les valeurs de p et de

$\boldsymbol{w}$. Le même résultat s'applique aux transitions correspondant à ne pas sélectionner un sommet.

Les multiplicateurs lagrangiens $\boldsymbol{\zeta}$ sont optimisés par un algorithme de sous-gradient classique. Soit $\hat{\boldsymbol{p}}^t \in \mathbb{R}^n$ le vecteur des profits Lagrangiens, c'est-à-dire $\hat{p}_k^t = p_k - \boldsymbol{\zeta}^t \cdot \boldsymbol{w}_k$, et soit $\boldsymbol{x}^t$ la solution optimale obtenue après l'itération t de l'algorithme de sous-gradient. La direction de sous-gradient correspondante ∇^t est définie par la violation des contraintes de capacité, $\nabla^t = \begin{pmatrix} \boldsymbol{w}^1 \cdot \boldsymbol{x}^t - W^1 \\ \boldsymbol{w}^2 \cdot \boldsymbol{x}^t - W^2 \end{pmatrix}^+$ et le pas s^t est de la forme du pas de Polyak, $s^t = 2 \cdot \frac{0.1(\hat{\boldsymbol{p}}^t \cdot \boldsymbol{x}^t)}{\|\nabla^t\|_2^2}$, avec $0.1(\hat{\boldsymbol{p}}^t \cdot \boldsymbol{x}^t)$ une estimation de l'écart d'optimalité entre $\boldsymbol{\zeta}^t$ et les multiplicateurs optimaux $\boldsymbol{\zeta}^*$. L'algorithme est interrompu après deux itérations non-améliorantes, c'est-à-dire lorsque $\hat{\boldsymbol{p}}^t \cdot \boldsymbol{x}^t \leq \hat{\boldsymbol{p}}^t \cdot \boldsymbol{x}^{t-1}$.

3.3.2.4 Bornes de complétion : filtrage des labels par le coût

Chaque label du programme dynamique $(p, k, \boldsymbol{w}, r)$ correspond à une solution partielle vérifiant les contraintes de sac-à-dos et les contraintes de précédence du problème de pricing. Dans un programme dynamique, la manière classique de supprimer un label est d'utiliser des tests de dominance. Il est aussi possible d'utiliser une borne duale, comme dans un branch-and-bound. Étant donné une borne primale PB et un label ℓ, on peut filtrer ℓ si $p(\ell) + DB(\ell) \leq PB$, où $DB(\ell)$ correspond à un majorant sur la valeur qui va pouvoir être collectée entre ℓ et un état terminal, permettant donc de *compléter* le label ℓ.

L'efficacité de ce test dépend de la qualité du minorant $DB(\ell)$ et du temps pour l'obtenir. La méthode que nous utilisons réutilise les résultats calculés lors du filtrage lagrangien décrit plus haut. Soit $\boldsymbol{\zeta} \in \mathbb{R}_+^2$ un vecteur de multiplicateurs lagrangiens. Étant donné un label ℓ défini par le tuple $(p(\ell), k(\ell), \boldsymbol{w}(\ell), r(\ell))$, soit ℓ' le sommet du graphe correspondant à sa projection après relaxation des contraintes de capacité, défini par le triplet $(p(\ell) - \boldsymbol{\zeta} \cdot \boldsymbol{w}(\ell), k(\ell), r(\ell))$ et soit $v_{\boldsymbol{\zeta}}(\ell')$ la distance du plus long chemin allant de ℓ' à $(n+1, 0)$. Une borne duale $DB(\ell)$ sur la meilleure solution atteignable à partir de ℓ est alors donnée par $DB(\ell) = v_{\boldsymbol{\zeta}}(\ell') + \boldsymbol{\zeta} \cdot (\boldsymbol{W} - \boldsymbol{w}(\ell))$. Pour un couple $k(\ell), r(\ell))$ et une capacité résiduelle $\boldsymbol{W}_r = \boldsymbol{W} - \boldsymbol{w}(\ell)$ donnés, on peut calculer des multiplicateurs lagrangiens $\boldsymbol{\zeta}^*_{\boldsymbol{W}_r}$ optimaux. Cependant, il est plus efficace d'utiliser les multiplicateurs $\bar{\boldsymbol{\zeta}}^*$ obtenus à la fin de l'exécution de l'algorithme de sous-gradient décrit plus haut.

3.4 Résultats expérimentaux

Nos expérimentations ont deux objectifs. Le premier est de déterminer empiriquement quel est le meilleur oracle lorsque la génération de colonnes est utilisée. Le deuxième est de valider que la formulation étendue permet d'obtenir de meilleurs résultats que la formulation compacte originale.

3.4.1 Instances utilisées et configuration informatique

Deux ensembles d'instances ont été utilisés pour évaluer la performance des différentes méthodes proposées. Les instances `exeo` correspondent aux 13 instances issues de

données réelles fournies par la société eXEO Solutions. La taille de ces instances varie entre 31 points de collecte et 245 points de collecte. Les instances `cvrp` sont issues de la littérature scientifique. Nous avons transformé 16 instances du CVRP (*capacitated vehicle routing problem*) issues de la TSPLIB en mélangeant les consommations et en bruitant les distances. Chaque instance originale a donné 25 instances bruitées, pour un total de 400 instances dont la taille varie entre 7 et 262 points de collecte.

Afin de générer plusieurs instances à partir d'une seule instance de la TSPLIB, nous avons ajouté du bruit aléatoire à la longueur des arcs de la manière suivante. Pour (i, j) un arc donné de longueur $c_{i,j}$, nous générons la longueur $c'_{i,j} = c_{i,j} + r$ avec r un nombre réel aléatoire généré selon une distribution uniforme dans $[0, (c_{i,j})^{1.5}]$. De plus, nous générons une seconde contrainte de sac-à-dos en gardant la même capacité et en permutant les consommations des objets de façon aléatoire.

Les tests ont été effectués sur PlaFRIM (Plateforme Fédérative pour la Recherche en Informatique et Mathématiques), sur des machines équipées de processeurs *Dodeca-core Haswell Intel® Xeon® E5-2680 v3 @ 2,5 GHz* et de 128Go de RAM.

Les programmes linéaires et les formulations MIP compactes sont résolues en utilisant IBM ILOG CPLEX Optimization Studio 12.6.0 [CPL]. Le branch-and-price est implémenté dans le framework BaPCod 2.0.0 [Van], en utilisant en particulier les paramètres par défaut de stabilisation et la gestion de l'arbre de recherche dans le branch-and-price.

3.4.2 Analyse de l'oracle branch-and-bound

Nos premières expérimentations numériques ont pour objet la méthode de branch-and-bound pour résoudre le sous-problème de génération de colonnes. Nous cherchons à déterminer la meilleure configuration pour cet algorithme. Deux ingrédients ont un impact non négligeable sur l'efficacité de la méthode : la méthode utilisée pour résoudre le problème lagrangien, et l'ordre dans lequel on considère les rayons lors de l'énumération.

Ces résultats sont décrits dans le Tableau 3.1. Dans un premier temps, nous comparons deux directions de descente. La direction `PROP` correspond à la violation des contraintes de capacités projetée sur $\mathbb{R}^2_+$, tandis que la direction `STEEP` ne considère que la pénalité associée à la contrainte de capacité la plus violée. Dans un second temps, nous combinons la meilleure de ces deux directions avec un ordre des rayons particulier. Dans cet ordre, nous commençons par considérer le rayon associé au sous-problème ayant le plus souvent produit une solution optimale lors des itérations précédentes, puis nous énumérons les rayons par ordre décroissant. Cette configuration est appelée `STEEP+BR` dans le Tableau 3.1. Pour chaque configuration et pour chaque jeu d'instances, nous rapportons **time** le temps moyen en secondes passé à résoudre l'instance à l'optimum (limité à deux heures), **nodes/cg** le nombre moyen de nœuds de branchements générés par l'oracle et **solved** le nombre d'instances résolues en moins de deux heures.

Il apparaît que la direction utilisée a un impact important sur la performance de l'algorithme, puisque la direction `STEEP` nécessite 43% moins de nœuds de branchement en moyenne par rapport à la direction `PROP`. L'ordre révisé des rayons permet de réduire le nombre de nœuds de branchement de 80% supplémentaires. Cette réduction s'explique par le fait que l'algorithme trouve rapidement une borne primale forte, permettant alors de filtrer les autres rayons. Dans les sections qui suivent, les résultats associés à l'oracle branch-and-bound correspondent à la configuration `STEEP+BR`.

	PROP			STEEP			STEEP+BR		
Instance	**time**	**nodes/cg**	**solved**	**time**	**nodes/cg**	**solved**	**time**	**nodes/cg**	**solved**
exeo/031	> 0	776	1/1	> 0	344	1/1	> 0	284	1/1
exeo/032	> 0	46	1/1	> 0	46	1/1	> 0	19	1/1
exeo/054	1	15	1/1	> 0	15	1/1	> 0	10	1/1
exeo/075	67	51	1/1	70	51	1/1	65	31	1/1
exeo/087	31	279	1/1	31	279	1/1	26	185	1/1
exeo/102	50	109	1/1	50	109	1/1	48	71	1/1
exeo/109	1 471	4 328	1/1	1 478	4 328	1/1	765	1 648	1/1
exeo/125	7 200	1 860	0/1	7 200	1 859	0/1	7 200	720	0/1
exeo/162	212	1 697	1/1	212	1 697	1/1	178	1 127	1/1
exeo/163	7 200	888	0/1	7 200	893	0/1	7 200	418	0/1
exeo/171	7 200	7 978	0/1	7 200	7 980	0/1	7 200	1 729	0/1
exeo/174	7 200	8 000 257	0/1	1 288	8 963	**1/1**	635	1 998	**1/1**
exeo/245	7 200	6 494 471	0/1	7 200	39 808	0/1	7 200	28 641	0/1
	time	**nodes/cg**	**solved**	**time**	**nodes/cg**	**solved**	**time**	**nodes/cg**	**solved**
cvrp/att48	3 210	448	**17/25**	2 909	251	**17/25**	3 182	188	16/25
cvrp/eil7	> 0	107	25/25	> 0	92	25/25	> 0	46	25/25
cvrp/eil13	> 0	46	25/25	> 0	46	25/25	> 0	46	25/25
cvrp/eil22	> 0	475	25/25	> 0	408	25/25	> 0	143	25/25
cvrp/eil23	3	952	25/25	3	961	25/25	3	930	25/25
cvrp/eil30	1	625	25/25	1	529	25/25	1	197	25/25
cvrp/eil31	2	1 802	25/25	2	1 626	25/25	2	1 292	25/25
cvrp/eil33	32	3 514	25/25	23	2 556	25/25	12	893	25/25
cvrp/eil51	316	3 084	25/25	345	2 134	25/25	212	916	25/25
cvrp/eilA101	6 810	22 557	1/25	7 018	10 782	1/25	7 200	4 016	1/25
cvrp/eilA76	6 161	7 755	6/25	6 183	5 022	6/25	5 385	1 228	**12/25**
cvrp/eilB101	6 970	15 405	2/25	6 915	11 631	2/25	6 717	3 117	2/25
cvrp/eilB76	6 962	10 986	1/25	6 958	8 736	1/25	5 910	1 755	**9/25**
cvrp/eilC76	5 148	6 438	9/25	4 914	3 407	**11/25**	4 651	1 524	**11/25**
cvrp/eilD76	5 654	7 649	8/25	5 173	3 434	9/25	4 762	1 300	**13/25**
cvrp/gil262	7 355	792 422	0/25	7 200	404 621	0/25	7 200	26 934	0/25
Total (/400)			**244**			**247**			**264**

TABLE 3.1 – Performances de l'oracle branch-and-bound

3.4.3 Analyse de l'oracle par programmation dynamique

Dans cette section, nous déterminons numériquement la meilleure configuration à utiliser pour le programme dynamique permettant de résoudre le problème de génération de colonnes. Plusieurs méthodes permettent d'éliminer des états ne pouvant mener à une solution optimale : dominances, filtrage lagrangien et filtrage par coût.

La stratégie par défaut est notée `base`. Dans cette configuration, nous utilisons la règle de dominance 1, et n'appliquons ni filtrage lagrangien, ni filtrage par coût. De plus, les objets sont arrangés selon un ordre DFS arbitraire. À partir de cette configuration, nous pouvons complexifier l'algorithme avec les options suivantes : `dom` consiste à appliquer la règle de dominance 2 ; `filt` correspond à la méthode de filtrage lagrangien décrite dans la sous-sous-section 3.3.2.3 ; `sort` indique l'utilisation d'un ordre DFS des sommets tenant compte du rayon des sous-arbre ; les options `sgCB` et `stCB` sont associées à des bornes sur les labels calculées à partir de la relaxation lagrangienne des contraintes de capacité, utilisées dans le filtrage des labels par les coûts. La différence entre ces deux bornes est que `sgCB` utilise les multiplicateurs lagrangiens obtenus à la fin de la méthode de filtrage lagrangien, tandis que `stCB` fixe le multiplicateur d'une dimension à 0 et calcule le multiplicateur optimal pour la dimension restante. Dans la stratégie `all`, nous activons toutes les options.

Nous comparons également les performances de l'algorithme lorsque nous activons toutes les options sauf une. Par exemple, la configuration `\dom` utilise les options `filt`, `sort`, `sgCB` et `stCB` mais pas l'option `dom`.

Les performances de chaque configuration sont reportées Tableau 3.2. Pour chaque configuration et pour chaque jeu d'instances, nous rapportons **time** le temps moyen en secondes passé à résoudre l'instance à l'optimum (limité à deux heures), **labels/cg** le nombre moyen de labels du programme dynamique générés par l'oracle et **solved** le nombre d'instances résolues en moins de deux heures.

Il apparaît que toutes les techniques proposées permettent d'améliorer de manière significative la performance de l'algorithme de programmation dynamique. En particulier, deux configurations ressortent comme étant particulièrement efficaces : la configuration `all` semble plus efficace pour résoudre les instances de la classe `exeo`, tandis que la configuration `\sort` semble mieux adaptée pour résoudre les instances `cvrp`. Dans les deux configurations, l'algorithme génère environ 90% de labels en moins que la configuration `base`.

Dans la suite de cette section, les résultats associés à l'oracle de programmation dynamique correspondent aux configurations `all` (dénotée `DP1`) et `\sort` (dénotée `DP2`).

3.4.4 Comparaison des différents oracles

Nous présentons maintenant la comparaison des deux approches décrites dans les sections précédentes pour résoudre le sous-problème de génération de colonnes : branch-and-bound et programmation dynamique. Pour ce faire, deux dispositifs expérimentaux ont été mis en place. Le premier consiste à comparer le temps utilisé par le branch-and-price lorsqu'il utilise l'une ou l'autre des méthodes. Cette comparaison a le défaut de ne pas comparer les deux oracles sur les mêmes instances de sous-problèmes car en cas de solutions équivalentes, des colonnes différentes sont générées entre les deux méthodes.

Afin d'avoir une analyse plus fine des performances des deux méthodes, nous avons stocké les sous-problèmes correspondant à une exécution du processus complet de génération de colonnes (en utilisant le branch-and-bound). Pour chacune de ces instances de sous-problème, nous avons exécuté les deux algorithmes. Nous rapportons dans le Tableau 3.3 et les figures 3.3a–3.4f un résumé de ces expérimentations.

Dans Tableau 3.3, nous rapportons **time** le temps moyen en secondes passé à résoudre (P) avec l'algorithme de branch-and-price, **t_MP** le temps moyen en secondes passé à résoudre (P^M) (ce qui exclut donc le temps passé à résoudre les sous-problèmes (P^j)), et **solved** le nombre d'instances résolues en moins de deux heures.

Lorsqu'il s'agit d'obtenir un branch-and-price plus efficace, l'algorithme de branch-and-bound semble être à favoriser, puisqu'il obtient des résultats 1.7 fois plus rapides sur les instances `exeo` et permet de résoudre 21 instances `cvrp` supplémentaires. Les deux approches sont toutefois dans le même ordre de magnitude sur les instances utilisées.

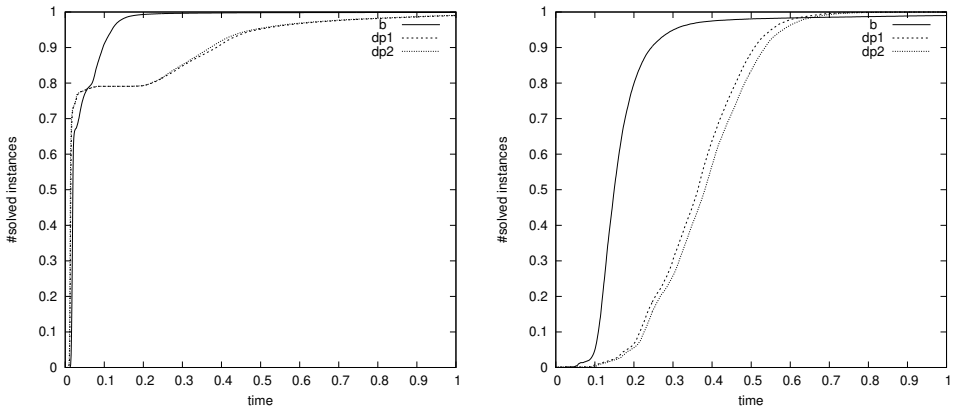
Nous présentons maintenant une comparaison plus fine des performances des deux oracles. D'un côté, les figures 3.3a–3.3c et 3.3d–3.3f représentent le nombre d'instances de pricing résolues par chaque oracle pour une limite de temps donnée. Par exemple, dans la figure 3.3a, la courbe correspondant à l'oracle de branch-and-bound montre que 90% des instances de pricing sont résolues en moins de 0.1 unités de temps, tandis que les 10% restant réclament jusqu'à 0.4 unités de temps. En comparaison, 20% des instances de pricing nécessitent plus de 0.2 unités de temps lorsque nous employons l'oracle basé sur le programme dynamique. Il est possible de simplifier ces résultats en considérant que plus la courbe est haute, plus l'algorithme est efficace.

En nous basant sur ces expérimentations, nous pouvons déduire que l'oracle de branch-and-bound résout généralement plus d'instances que n'importe quelle configuration de l'oracle de programmation dynamique, pour un temps limite donné. Notons cependant que la programmation dynamique semble être une approche plus robuste pour résoudre les instances dans `cvrp/eilB76` (figure 3.3e).

D'un autre côté, les figures 3.4a–3.4c et 3.4d–3.4f représentent le ratio des temps de résolution entre l'oracle de programmation dynamique et l'oracle branch-and-bound. Nous pouvons alors observer la proportion d'instances pour lesquelles le programme dynamique permet une résolution plus rapide que l'approche par branch-and-bound (et réciproquement), ainsi que la différence de vitesse. Par exemple, la figure 3.4a montre que l'oracle de programmation dynamique est plus rapide que l'oracle branch-and-bound plus de 75% du temps et peut aller jusqu'à être 9 fois plus rapide. En revanche, la méthode de branch-and-bound est au moins 5 fois plus rapide sur environ 20% des instances de pricing.

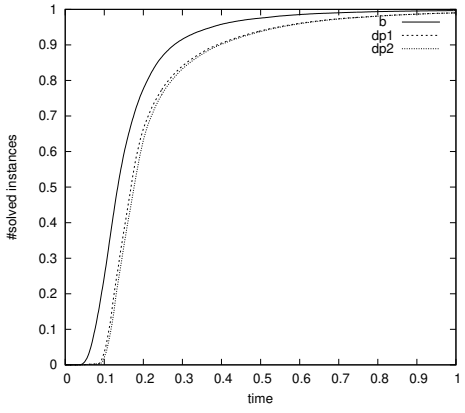
De la même manière que dans les tests précédents, l'oracle de branch-and-bound surclasse celui basé sur la programmation dynamique, sauf pour les instances de pricing issues de `exeo/075` (figure 3.4a) et `cvrp/eilB76` (figure 3.4e).

Dans les sections suivantes, nous utilisons uniquement l'oracle de branch-and-bound pour résoudre le sous-problème de génération de colonnes.

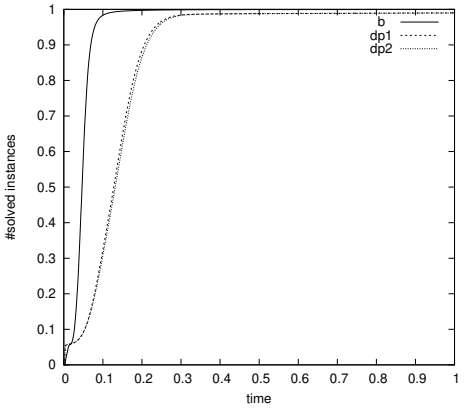


(a) exeo/075 walltimes

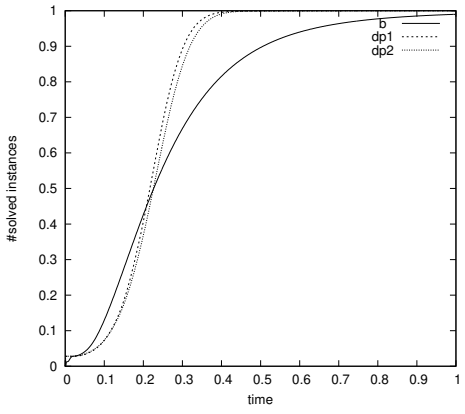
(b) exeo/162 walltimes



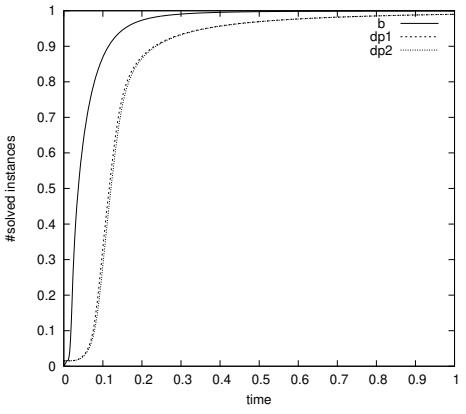
(c) exeo/174 walltimes



(d) cvrp/att48 walltimes

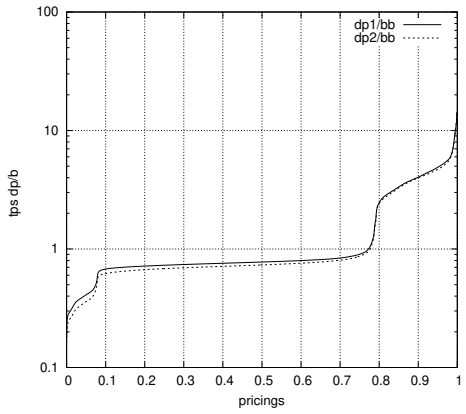


(e) cvrp/eilB76 walltimes

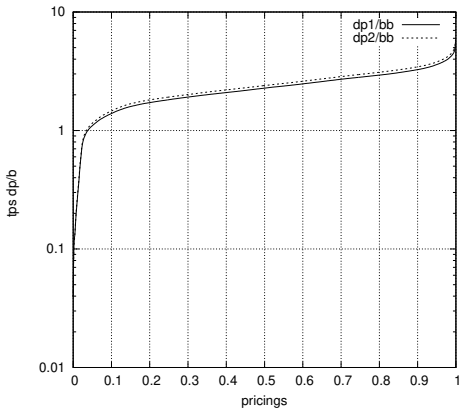


(f) cvrp/eilD76 walltimes

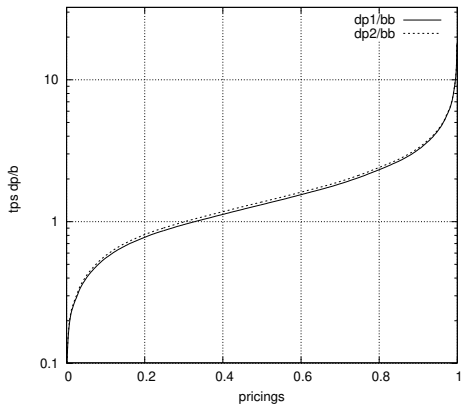
FIGURE 3.3 – Diagrammes de profiling des oracles, nombre d'instances résolues



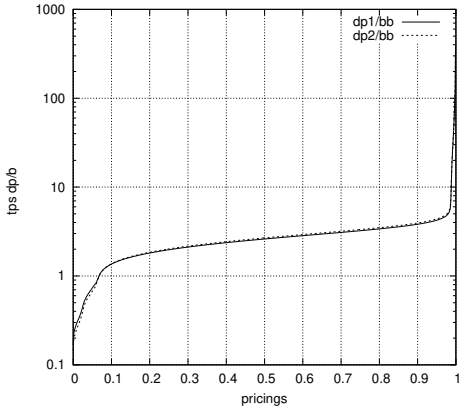
(a) exeo/075 time ratios



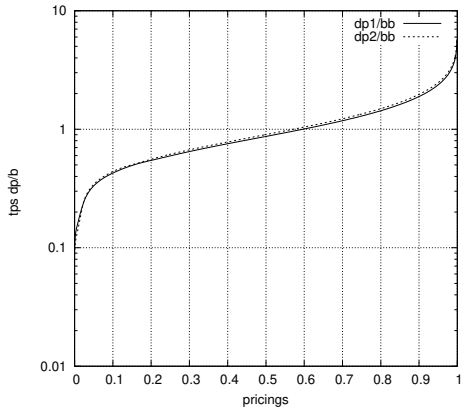
(b) exeo/162 time ratios



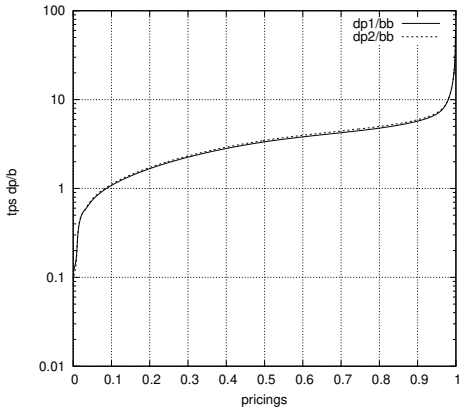
(c) exeo/174 time ratios



(d) cvrp/att48 time ratios

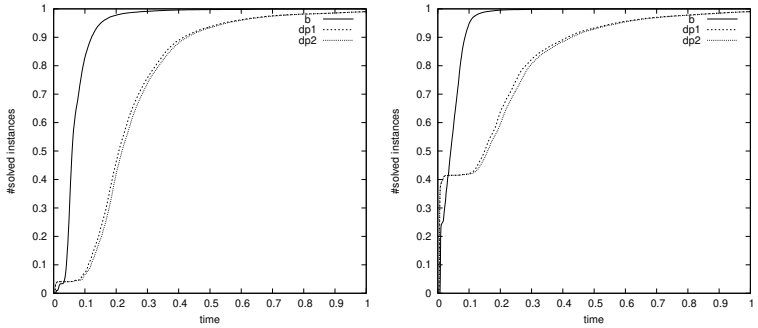


(e) cvrp/eilB76 time ratios



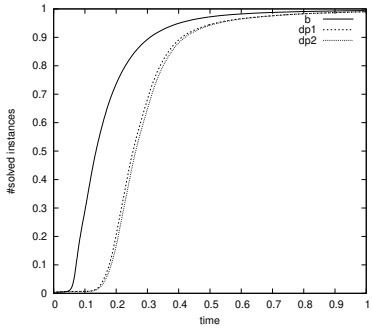
(f) cvrp/eilD76 time ratios

FIGURE 3.4 – Diagrammes de profiling des oracles, rapports de vitesse

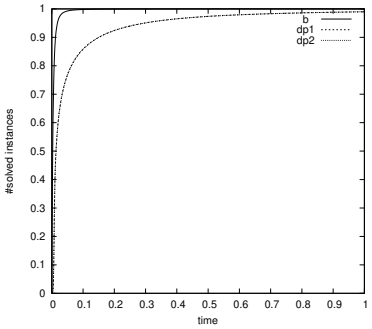


(a) exeo/087 walltimes

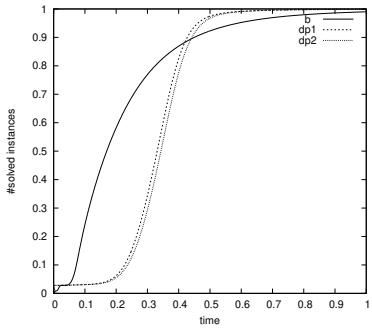
(b) exeo/102 walltimes



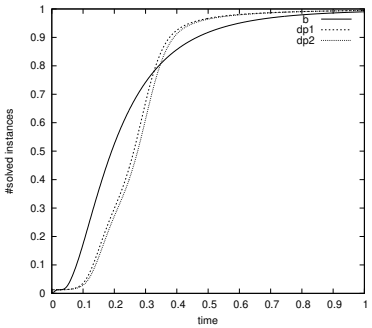
(c) exeo/109 walltimes



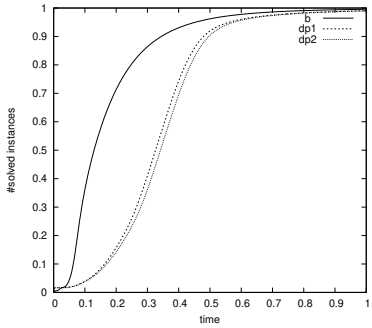
(d) cvrp/eilA101 walltimes



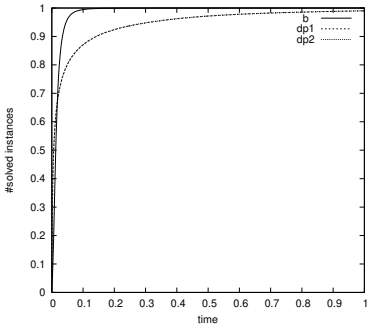
(e) cvrp/eilA76



(f) cvrp/eilB101 walltimes

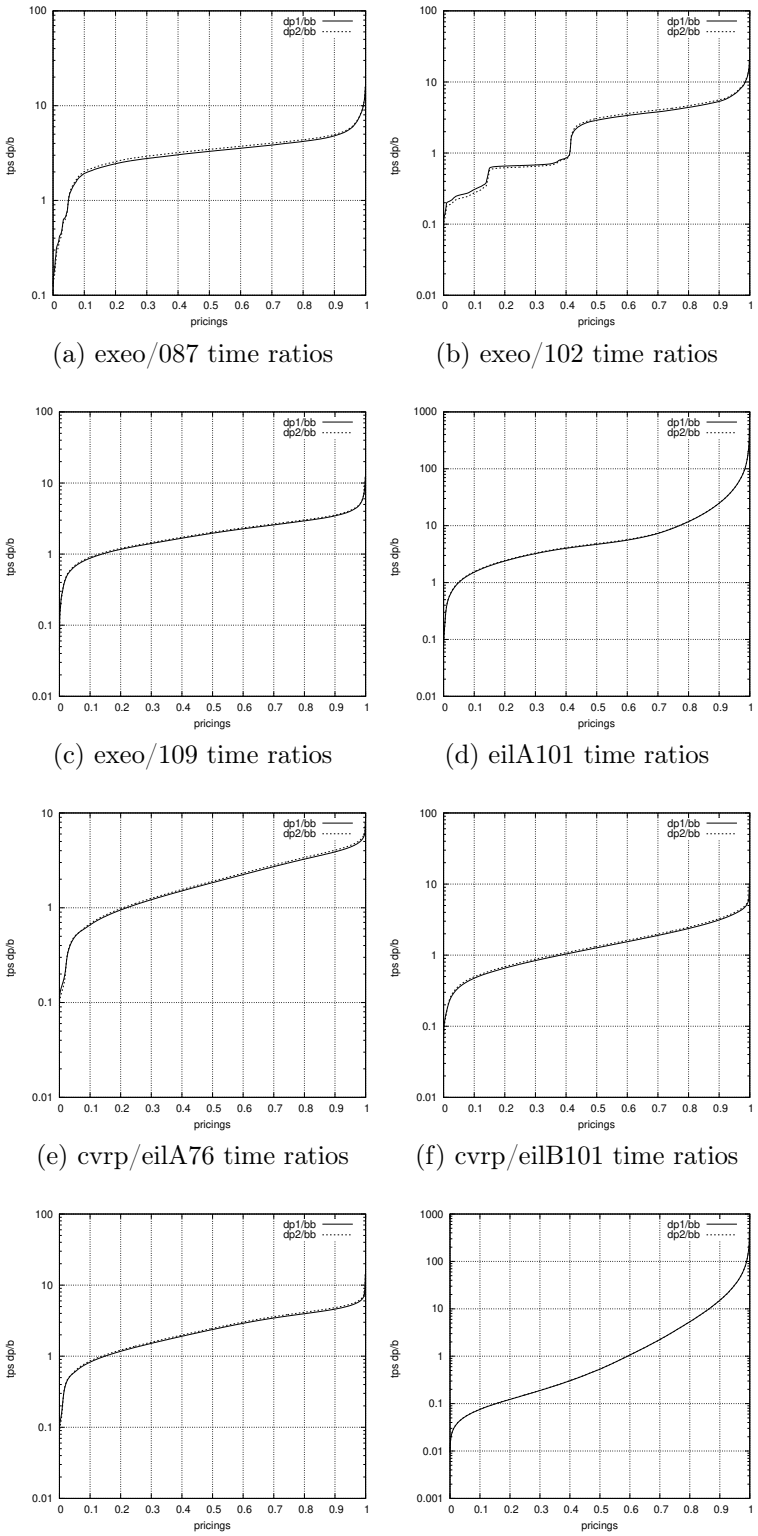


(g) cvrp/eilC76



(h) cvrp/gil262 walltimes

FIGURE 3.5 – Diagrammes de profiling supplémentaires des oracles, nombre d'instances résolues



(a) exeo/087 time ratios

(b) exeo/102 time ratios

(c) exeo/109 time ratios

(d) eilA101 time ratios

(e) cvrp/eilA76 time ratios

(f) cvrp/eilB101 time ratios

(g) cvrp/eilC76 time ratios

(h) cvrp/gil262 time ratios

FIGURE 3.6 – Diagrammes de profiling supplémentaires des oracles, rapports de vitesse

3.4.5 Comparaison de la formulation compacte et de la formulation étendue

Pour ces dernières expérimentations, nous validons l'intérêt de l'approche basée sur la génération de colonnes en comparant ses résultats avec ceux de la formulation compacte résolue avec un solveur commercial (CPLEX).

Le Tableau 3.4 permet de comparer les modèles compacts (P_0) et (P) avec la formulation étendue. Dans ce tableau, nous rapportons **time_tot** le temps total en secondes pour résoudre une instance à l'optimum (ou "-" si l'algorithme n'a pas convergé en deux heures), **time_root** le temps passé à la racine de l'arbre de branchement, qui correspond à la relaxation linéaire du modèle. L'écart à l'optimum, indiqué dans la colonne **gap_opt**, est $> 0\%$ si et seulement si l'algorithme n'a pas convergé en deux heures. Si l'algorithme n'a pas pu trouver une solution entière réalisable pendant ce laps de temps, alors cet écart vaut $+\infty$. La colonne **gap_root** indique la qualité de la relaxation continue de chaque modèle par rapport à la meilleure solution primale connue. Nous disposons de la valeur optimale pour toutes les instances `exeo`, à l'exception de `exeo/163` et `exeo/171`. Enfin, **nb_bb_nodes** indique le nombre de nœuds de branchement évalués, tandis que **time_per_node** est égal au temps moyen requis pour évaluer chacun de ces nœuds. Concernant les instances `cvrp`, nous ne rapportons que le nombre d'instances résolues à l'optimum en moins de 2 heures.

La génération de colonne est ainsi clairement plus efficace pour résoudre ces instances que la résolution des modèles compacts avec un solveur générique. En effet, même si chaque nœud de branchement est globalement plus coûteux à évaluer dans la formulation étendue, la qualité de la relaxation continue entraîne une réduction importante du nombre de nœuds nécessaires à la résolution des instances.

Notons cependant que, dans le cas de l'instance `exeo/245`, le gap à l'optimum obtenu à la fin du temps imparti est plus petit lorsque nous résolvons le modèle compact (P) que lorsque nous résolvons la formulation étendue. Ainsi, les méthodes de génération de colonnes semblent moins adaptées à cette instance car le temps nécessaire à la résolution du sous-problème est excessif.

3.5 Conclusion

Dans ce chapitre, nous avons étudié un nouveau problème de couverture des sommets d'un graphe par des arborescences, sous contraintes de capacité. Nous avons proposé plusieurs modèles mathématiques. Le plus efficace repose sur l'utilisation d'un algorithme de génération de colonnes, pour lequel nous avons proposé plusieurs algorithmes. L'algorithme de branch-and-bound proposé semble le plus efficace en moyenne lorsqu'il est utilisé comme oracle. Cependant, il existe de nombreuses instances de sous-problèmes pour lesquelles le programme dynamique est plus efficace.

Pour résoudre des instances de plus grande taille, plusieurs pistes sont à étudier. Des inégalités valides pourraient être proposées pour améliorer la borne duale dans le branch-and-price. Toutes les techniques permettant d'accélérer la convergence des méthodes lagrangiennes pourront aussi être mises à contribution.

Instance	base			\dom			\sort			\stCB			\sgCB			\filt			all		
	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved
exeo/031	0	377	1/1	0	65	1/1	0	71	1/1	0	79	1/1	0	68	1/1	0	172	1/1	0	**64**	1/1
exeo/032	0	1 008	1/1	1	74	1/1	1	**64**	1/1	1	143	1/1	1	83	1/1	0	259	1/1	1	71	1/1
exeo/054	1	1 999	1/1	0	141	1/1	0	**110**	1/1	0	171	1/1	0	199	1/1	1	476	1/1	0	132	1/1
exeo/075	700	2 957	1/1	115	282	1/1	123	277	1/1	108	309	1/1	120	396	1/1	139	554	1/1	111	**252**	1/1
exeo/087	1 224	26 588	1/1	87	2 542	1/1	103	2 447	1/1	100	3 848	1/1	89	2 938	1/1	139	4 464	1/1	80	**2 272**	1/1
exeo/102	2 472	22 875	1/1	162	2 020	1/1	157	1 754	1/1	161	2 343	1/1	175	2 757	1/1	231	3 564	1/1	148	**1 674**	1/1
exeo/109	7 200	61 037	0/1	1 588	2 363	1/1	1 708	2 428	1/1	1 749	4 480	1/1	1 490	2 637	1/1	3 821	8 962	1/1	1 192	**2 090**	1/1
exeo/125	7 200	31 105	0/1	7 200	1 686	0/1	7 200	**1 235**	0/1	7 200	3 068	0/1	7 200	1 944	0/1	7 200	5 386	0/1	7 200	1 559	0/1
exeo/162	2 303	38 191	1/1	353	1 894	1/1	472	**1 332**	1/1	481	5 024	1/1	317	2 136	1/1	514	7 005	1/1	358	1 687	1/1
exeo/163	7 200	85 964	0/1	7 200	5 128	0/1	7 200	4 411	0/1	7 200	8 095	0/1	7 200	7 174	0/1	7 200	8 787	0/1	7 200	**4 383**	0/1
exeo/171	7 200	105 997	0/1	7 200	5 645	0/1	7 200	4 865	0/1	7 200	6 957	0/1	7 200	6 075	0/1	7 200	28 573	0/1	7 200	**4 752**	0/1
exeo/174	7 200	179 300	0/1	1 247	14 559	1/1	1 346	13 140	1/1	1 246	16 682	1/1	1 019	14 173	1/1	3 719	35 935	1/1	992	**11 363**	1/1
exeo/245	7 200	291 000	0/1	7 200	176 685	0/1	7 200	**161 482**	0/1	7 200	206 605	0/1	7 200	216 524	0/1	7 200	258 996	0/1	7 200	204 946	0/1
	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved	time	labels/cg	solved
cvrp/att48	3 725	2 674	**17/25**	3 922	448	14/25	3 943	330	**17/25**	3 887	595	16/25	3 918	441	15/25	3 865	723	**17/25**	3 855	375	**17/25**
cvrp/eil7	0	17	25/25	0	5	25/25	0	5	25/25	0	7	25/25	0	5	25/25	0	9	25/25	0	5	25/25
cvrp/eil13	0	67	25/25	0	31	25/25	0	28	25/25	0	47	25/25	0	32	25/25	0	38	25/25	0	30	25/25
cvrp/eil22	4	624	25/25	3	225	25/25	2	163	25/25	2	488	25/25	3	229	25/25	4	244	25/25	2	213	25/25
cvrp/eil23	9	754	25/25	2	315	25/25	2	214	25/25	4	319	25/25	2	346	25/25	2	585	25/25	2	288	25/25
cvrp/eil30	1	3 225	25/25	1	901	25/25	1	650	25/25	1	1 515	25/25	1	1 028	25/25	1	984	25/25	1	815	25/25
cvrp/eil31	177	756	25/25	79	276	25/25	41	257	25/25	84	378	25/25	66	283	25/25	58	336	25/25	51	264	25/25
cvrp/eil33	5 328	5 687	25/25	568	3 061	25/25	475	1 978	25/25	904	4 019	25/25	598	3 155	25/25	821	2 848	25/25	520	2 590	25/25
cvrp/eil51	0	31 476	12/25	0	1 957	24/25	0	1 346	**25/25**	0	4 765	24/25	0	2 448	24/25	0	4 138	24/25	0	1 790	24/25
cvrp/eilA101	7 200	283 215	0/25	7 200	27 637	0/25	7 200	22 163	0/25	7 200	69 101	0/25	7 200	37 238	0/25	7 200	71 458	0/25	7 200	24 622	0/25
cvrp/eilA76	7 200	25 342	0/25	6 358	1 119	4/25	6 164	947	**7/25**	6 458	2 717	4/25	6 374	1 341	4/25	6 366	2 651	4/25	6 379	1 049	4/25
cvrp/eilB101	7 200	50 507	0/25	7 031	2 255	**2/25**	6 992	2 275	1/25	7 071	5 655	1/25	6 977	2 673	**2/25**	7 039	7 531	1/25	7 038	2 091	**2/25**
cvrp/eilB76	6 529	4 451	3/25	6 446	513	3/25	6 612	455	3/25	6 585	908	3/25	6 441	561	3/25	6 233	1 171	**4/25**	6 436	493	3/25
cvrp/eilC76	7 081	62 620	1/25	5 707	2 028	7/25	5 455	1 298	**8/25**	6 141	6 833	7/25	5 761	2 455	7/25	5 911	6 080	7/25	5 665	1 880	7/25
cvrp/eilD76	7 200	130 733	0/25	6 540	5 161	6/25	6 244	3 484	**7/25**	6 970	17 314	2/25	6 672	6 940	5/25	6 971	14 008	2/25	6 290	4 538	**7/25**
cvrp/gil262	7 200	536 548	0/25	7 200	198 712	0/25	7 200	183 170	0/25	7 200	252 277	0/25	7 200	196 805	0/25	7 200	237 222	0/25	7 200	187 191	0/25
Total (/400)			**208**			**235**			**243**			**232**			**235**			**234**			**239**

TABLE 3.2 – Performances de l'oracle de programmation dynamique

	BB			DP1			DP2		
Instance	**time**	**t_MP**	**solved**	**time**	**t_MP**	**solved**	**time**	**t_MP**	**solved**
exeo/031	> 0	> 0	1/1	> 0	> 0	1/1	> 0	> 0	1/1
exeo/032	> 0	> 0	1/1	1	> 0	1/1	1	> 0	1/1
exeo/054	> 0	> 0	1/1	> 0	> 0	1/1	> 0	> 0	1/1
exeo/075	65	11	1/1	111	23	1/1	123	23	1/1
exeo/087	26	1	1/1	80	4	1/1	103	4	1/1
exeo/102	48	8	1/1	148	15	1/1	157	13	1/1
exeo/109	765	20	1/1	1 192	44	1/1	1 708	45	1/1
exeo/125	7 200	128	0/1	7 200	238	0/1	7 200	154	0/1
exeo/162	178	2	1/1	358	7	1/1	472	7	1/1
exeo/163	7 200	1 305	0/1	7 200	551	0/1	7 200	476	0/1
exeo/171	7 200	127	0/1	7 200	126	0/1	7 200	101	0/1
exeo/174	635	26	1/1	992	25	1/1	1 346	28	1/1
exeo/245	7 200	145	0/1	7 200	4	0/1	7 200	5	0/1

	BB			DP1			DP2		
Instance	**time**	**t_MP**	**solved**	**time**	**t_MP**	**solved**	**time**	**t_MP**	**solved**
cvrp/att48	3 182	795	16/25	3 855	404	**17/25**	3 943	357	**17/25**
cvrp/eil7	> 0	> 0	25/25	> 0	> 0	25/25	> 0	> 0	25/25
cvrp/eil13	> 0	> 0	25/25	> 0	> 0	25/25	> 0	> 0	25/25
cvrp/eil22	> 0	> 0	25/25	> 0	> 0	25/25	> 0	> 0	25/25
cvrp/eil23	3	> 0	25/25	2	> 0	25/25	2	> 0	25/25
cvrp/eil30	1	> 0	25/25	2	> 0	25/25	2	> 0	25/25
cvrp/eil31	2	> 0	25/25	1	> 0	25/25	1	> 0	25/25
cvrp/eil33	12	1	25/25	51	1	25/25	41	1	25/25
cvrp/eil51	212	25	**25/25**	520	18	24/25	475	16	**25/25**
cvrp/eilA101	6 960	157	**1/25**	7 200	13	0/25	7 200	16	0/25
cvrp/eilA76	5 385	513	**12/25**	6 376	164	4/25	6 164	148	7/25
cvrp/eilB101	6 716	323	**2/25**	7 038	129	**2/25**	6 991	109	1/25
cvrp/eilB76	5 910	429	**9/25**	6 436	181	3/25	6 612	162	3/25
cvrp/eilC76	4 651	425	**11/25**	5 665	148	7/25	5 455	155	8/25
cvrp/eilD76	4 762	531	**13/25**	6 290	104	7/25	6 244	115	7/25
cvrp/gil262	7 200	61	0/25	7 200	5	0/25	7 200	5	0/25
Total (/400)			264			239			243

TABLE 3.3 – Comparaison des oracles branch-and-bound et programmation dynamique

Instance	time_tot			time_root			gap_opt = (pb-db)/db			gap_root = (pb*-lp)/lp			nb_bb_nodes			time_per_node		
	(P_0)	(P)	CG BB	(P_0)	(P)	CG BB	(P_0)	(P)	CG BB	(P_0)	(P)	CG BB	(P_0)	(P)	CG BB	(P_0)	(P)	CG BB
exeo/031	45	25	**> 0**	**> 0**	**> 0**	> 0	0.0%	0.0%	0.0%	242.6%	87.8%	**0.0%**	2574	1 759	1	0	**0**	0
exeo/032	21	3	**> 0**	**> 0**	> 0	> 0	0.0%	0.0%	0.0%	143.5%	26.3%	**3.9%**	931	215	9	**0**	**0**	0
exeo/054	393	6	**> 0**	> 0	1	**> 0**	0.0%	0.0%	0.0%	126.3%	5.2%	**0.0%**	2885	38	1	**0**	0	0
exeo/075	-	260	**65**	2	**1**	2	29.0%	0.0%	0.0%	128.6%	25.1%	**7.4%**	5659	1 054	145	1	**0**	0
exeo/087	-	2 981	**26**	5	**4**	5	47.7%	0.0%	0.0%	171.0%	66.5%	**2.9%**	19882	5 023	21	**0**	1	1
exeo/102	-	4 126	**48**	8	12	**5**	57.0%	0.0%	0.0%	154.8%	39.5%	**3.3%**	1212	2 613	21	6	**2**	2
exeo/109	-	-	**765**	**12**	19	18	214.4%	82.2%	**0.0%**	271.4%	101.8%	**3.5%**	1672	5 983	157	4	**1**	5
exeo/125	-	-	-	**27**	30	32	∞	126.3%	**0.5%**	293.7%	120.9%	**8.0%**	1367	3 019	1 677	5	**2**	4
exeo/162	-	-	**178**	71	178	**59**	∞	143.1%	**0.0%**	319.2%	67.0%	**0.9%**	429	520	17	17	14	**10**
exeo/163	-	-	-	83	223	**39**	218.9%	60.4%	**12.6%**	160.5%	50.8%	**12.8%**	470	541	309	15	**13**	23
exeo/171	-	-	-	**98**	300	156	∞	∞	**3.5%**	310.6%	69.1%	**5.6%**	310	181	288	**23**	40	25
exeo/174	-	-	**635**	**97**	275	205	∞	∞	**0.0%**	251.8%	74.6%	**1.1%**	107	433	21	67	**17**	30
exeo/245	-	-	-	**306**	1 233	576	72.4%	**14.8%**	67.5%	140.3%	26.6%	**2.9%**	151	17	4	**48**	424	2 788

Instance	(P_0)	(P)	CG BB
cvrp/att48	5/25	16/25	16/25
cvrp/eil7	25/25	25/25	25/25
cvrp/eil13	25/25	25/25	25/25
cvrp/eil22	25/25	25/25	25/25
cvrp/eil23	25/25	25/25	25/25
cvrp/eil30	25/25	25/25	25/25
cvrp/eil31	25/25	25/25	25/25
cvrp/eil33	25/25	25/25	25/25
cvrp/eil51	0/25	0/25	25/25
cvrp/eilA101	0/25	0/25	1/25
cvrp/eilA76	0/25	0/25	12/25
cvrp/eilB101	0/25	0/25	2/25
cvrp/eilB76	0/25	0/25	9/25
cvrp/eilC76	0/25	0/25	11/25
cvrp/eilD76	0/25	0/25	13/25
cvrp/gil262	0/25	0/25	0/25
Total	**180/400**	**191/400**	**264/400**

TABLE 3.4 – Performances de la formulation étendue

Conclusions et perspectives

Dans ce manuscrit, nous avons étudié deux problèmes, issus d'un besoin industriel, de couverture d'un graphe par des arborescences. La différence entre les deux problèmes réside essentiellement dans le calcul du coût d'une solution. Le premier problème, traité dans le chapitre 2, est défini par une fonction objectif analogue à celle du problème du p-median. Le second problème, traité dans le chapitre 3, est quant à lui défini par une fonction objectif proche de celle du problème du p-centre. À notre connaissance, ces problèmes n'avaient pas encore été traités dans la littérature scientifique, mais ils appartiennent à la famille des problèmes de sectorisation.

Nous avons comparé des approches de résolution exactes basées sur des formulation étendues, obtenues par la décomposition de Dantzig-Wolfe. Nous avons également proposé des modèles compacts, puis mesuré les performances d'un solveur générique pour leur résolution.

Dans le chapitre 2, nous avons montré que l'algorithme de branch-and-price appliqué à la formulation étendue permet de résoudre un grand nombre d'instances lorsque nous utilisons l'oracle de branch-and-bound pour résoudre les sous-problèmes. Par ailleurs, l'utilisation d'un solveur générique sur la formulation compacte permet également d'obtenir de bonnes performances. Le choix entre la formulation compacte et la formulation étendue dépend du type d'instance résolu. Bien que la méthode d'agrégation de contraintes proposée permette de réduire le nombre d'itérations de génération de colonnes dégénérées, celle-ci ne suffit pas à accélérer l'algorithme de branch-and-price.

Notre approche a permis de résoudre à l'optimum la plupart de nos instances. Il serait intéressant de mesurer les performances de notre méthode sur des instances de plus grande taille, de l'ordre du millier de sommets. Pour résoudre de telles instances, ainsi que les instances `cvrp/gil262`, plusieurs pistes peuvent être envisagées.

Premièrement, la résolution des sous-problèmes pourrait être accélérée par l'application d'heuristiques primales — soit pour effectuer un pricing heuristique, soit pour construire une solution initiale pour une méthode exacte. L'oracle de branch-and-bound peut également être amélioré par l'utilisation de bornes duales plus fortes que la borne Lagrangienne issue de la relaxation des contraintes de capacité, notamment pour l'évaluation des nœuds de branchement de l'oracle. Par exemple, nous pourrions considérer la relaxation Lagrangienne d'une des deux contraintes de sac à dos, afin d'obtenir un problème de TKP à une seule dimension. Une autre possibilité menant à un problème de TKP à une dimension consiste à combiner les deux dimensions de sac à dos en une seule à l'aide d'une relaxation surrogate. Notons cependant que, pour les instances `cvrp/gil262`, notre approche permet de résoudre le sous-problème de pricing en 0.17

secondes en moyenne, soit 0.7 millisecondes pour chaque sous-problème (P^j) associé à un centre j. Si un oracle plus performant pourrait effectivement permettre de traiter des instances de plus grande taille, cela semble moins susceptible de permettre une résolution des instances `cvrp/gil262`, pour lesquelles le temps de résolution des sous-problèmes est déjà très court.

Une deuxième approche possible consisterait à implémenter d'autres schémas d'agrégation de contrainte afin de réduire le nombre d'itérations de génération de colonnes dégénérées. Notamment, un schéma d'agrégation plus proche de la méthode IPS, tenant compte des contraintes de convexité et de cardinalité du problème maître, pourrait permettre d'éliminer presque systématiquement toute dégénérescence, à condition de disposer d'un algorithme efficace pour résoudre le problème de désagrégation duale (`DDP`). Cette approche est néanmoins limitée car la présence de dégénérescence dépend de la structure des instances résolues. Par exemple, les instances `cvrp/gil262` présentent une dégénérescence relativement faible, avec 20% à 35% d'itérations dégénérées en moyenne.

Au-delà de leur utilité contre le phénomène de dégénérescence, les méthodes d'agrégation de contraintes peuvent être employées pour accélérer la résolution du problème maître en réduisant explicitement la taille de la base. Ainsi, chaque itération de génération de colonnes nécessiterait potentiellement moins d'opérations de pivots pour trouver la base optimale du nouveau problème maître restreint. Cela amène cependant une difficulté dans le cadre de la génération de colonnes. En effet, une agrégation du maître à la manière de l'algorithme DCA nécessite de régulièrement maintenir l'ensemble des colonnes ajoutées au maître restreint afin de ne considérer que les colonnes compatibles avec l'agrégation courante. Sans cela, le problème maître agrégé pourrait admettre des solutions qui seraient pourtant non-réalisables pour le problème original, car elles ne vérifieraient pas toutes les contraintes de couverture. Malgré tout, ce type d'approche semble peu susceptible de permettre de traiter les instances `cvrp/gil262`. En effet, le temps passé pendant l'exécution de notre méthode de branch-and-price est consacré essentiellement (près de 96% du temps en moyenne) à la résolution des sous-problèmes de pricing.

Enfin, une approche plus prometteuse serait de modifier la formulation du problème afin de renforcer la relaxation continue. Une manière de procéder consiste à rajouter des contraintes valides du problème de set-covering au problème maître. La contrepartie à cette approche est que ces contrainte peuvent altérer la structure des sous-problèmes.

Les expériences du chapitre 3 montrent que l'algorithme de branch-and-price surpasse les approches de résolution utilisant une formulation compacte du problème. Nous avons également comparé les oracles de branch-and-bound et de programmation dynamique dans le contexte de la génération de colonnes, et avons montré que l'algorithme de branch-and-bound permet d'obtenir de meilleurs performances, globalement. Cependant, il existe un nombre significatif d'instances pour lesquelles l'approche par programmation dynamique est plus indiquée.

Actuellement, notre approche est limitée à la résolution de deux tiers des instances considérées (273/413 instances résolues à l'optimum). Il y a donc plusieurs manières envisageables pour améliorer l'algorithme et résoudre les instances restantes.

Tout comme pour le problème précédent, nous pouvons considérer l'amélioration des oracles utilisés pour résoudre le problème de pricing. Par exemple, nous avons montré

que la méthode basée sur le branch-and-bound pouvait être améliorée simplement en changeant l'ordre de résolution des rayons, notamment en commençant par un rayon pour lequel nous estimons avoir une forte probabilité d'induire une solution optimale du sous-problème. De la même manière, nous avons pu améliorer les performances de l'oracle basé sur le programme dynamique à l'aide de techniques de filtrage Lagrangien et de bornes de complétion. Des stratégies similaires pourraient être testées afin d'étudier leurs effets sur les performances des oracles.

Par ailleurs, nous avons montré que selon le type d'instance, il pouvait être préférable d'utiliser l'oracle de branch-and-bound ou bien celui de programmation dynamique. En particulier, l'algorithme de branch-and-bound semble plus rapide en moyenne, mais nécessite parfois un temps d'exécution anormalement long, contrairement à celui basé sur la programmation dynamique, qui est plus lent en moyenne mais plus stable. Si nous étions capables de prédire à l'avance l'oracle à utiliser en fonction de la structure du problème, nous pourrions bénéficier des avantages respectifs des deux méthodes. Cela peut être implémenté en parallélisant les deux algorithmes, et en interrompant le pricing dès que l'un d'eux a convergé. Une telle synchronisation nécessite cependant une implémentation efficace et doit limiter au maximum le temps passé à initialiser les sous-processus, car il s'agit ici de paralléliser des tâches potentiellement très courtes (parfois de l'ordre de la milliseconde).

Enfin, d'autres variantes du problème de sectorisation pourraient être étudiées. Par exemple, nous pourrions considérer le diamètre des secteurs (distance entre toute paire de sommets appartenant à un même secteur) dans la fonction objectif, à la place de leur rayon. La méthode de décomposition appliquée aux autres variantes serait toujours valide, mais produirait des sous-problèmes beaucoup plus complexes.

Bibliographie

[AE18] Z. Ales and S. Elloumi. Compact milp formulations for the p-center problem. In *International Symposium on Combinatorial Optimization*, pages 14–25. Springer, 2018.

[ASDF10] M. Albareda-Sambola, J. A. Díaz, and E. Fernández. Lagrangean duals and exact solution to the capacitated p-center problem. *European Journal of Operational Research*, 201(1) :71–81, 2010.

[AW09] K. M. Anstreicher and L. A. Wolsey. Two "well-known" properties of subgradient optimization. *Mathematical Programming*, 120(1) :213–220, 2009.

[BA00] F. Barahona and R. Anbil. The volume algorithm : producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3) :385–399, 2000.

[BDD12] P. Benchimol, G. Desaulniers, and J. Desrosiers. Stabilized dynamic constraint aggregation for solving set partitioning problems. *European Journal of Operational Research*, 223(2) :360–371, 2012.

[Bel54] R. Bellman. The theory of dynamic programming. Technical report, RAND Corp Santa Monica CA, 1954.

[Ber12] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II.* Athena Scientific, 2012.

[Ber17] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. I.* Athena Scientific, 2017.

[BHMM02] R. Baldacci, E. Hadjiconstantinou, V. Maniezzo, and A. Mingozzi. A new method for solving capacitated location problems based on a set partitioning approach. *Computers & Operations Research*, 29(4) :365–386, 2002.

[Bix12] R. E. Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

[BKP93] J. Barilan, G. Kortsarz, and D. Peleg. How to allocate network centers. *Journal of Algorithms*, 15(3) :385–415, 1993.

[BLM+08] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical programming*, 113(2) :299–344, 2008.

[BT97] D. Bertsimas and J. Tsitsiklis. *Introduction to linear programming.* Athena Scientific, 1997.

[Cal13] H. Calık. *Exact solution methodologies for the p-center problem under single and multiple allocation strategies.* PhD thesis, Bilkent university, 2013.

[CGM+14] F. Clautiaux, J. Guillot, Y. Magnouche, P. Pesneau, D. Trut, and F. Vanderbeck. Clustering problem for waste collection. Technical report, University of Bordeaux, INRIA Research team ReAlOpt., 2014.

[CPL] Ibm ilog cplex optimization studio cplex.

[CR06] A. Ceselli and G. Righini. An optimization algorithm for a penalized knapsack problem. *Operations Research Letters*, 34(4) :394–404, 2006.

[CS97] G. Cho and D. X. Shaw. A depth-first dynamic programming algorithm for the tree knapsack problem. *INFORMS Journal on Computing*, 9 :431–438, 1997.

[CS98] G. Cho and D. X. Shaw. The critical-item, upper bounds, and a branch-and-bound algorithm for the tree knapsack problem. *Networks*, 31 :205–216, 1998.

[CT13] H. Calik and B. C. Tansel. Double bound method for solving the p-center location problem. *Computers & Operations Research*, 40(12) :2991–2999, 2013.

[Dan49] G. B. Dantzig. Programming in a linear structure, 1949.

[Das95] M. S. Daskin. *Network and Discrete Location : Models, Algorithms and Applications*. John Wiley, New York, 1995.

[DCPS17] F. Della Croce, U. Pferschy, and R. Scatamacchia. Dynamic programming algorithms, efficient solution of the lp-relaxation and approximation schemes for the penalized knapsack problem. *arXiv preprint arXiv :1702.04211*, 2017.

[DL05] J. Desrosiers and M. E. Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

[DW60] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1) :101–111, 1960.

[EARMD14] M. G. Elizondo-Amaya, R. Z. Ríos-Mercado, and J. A. Díaz. A dual bounding scheme for a territory design problem. *Computers & Operations Research*, 44 :193–205, 2014.

[ELP04] S. Elloumi, M. Labbé, and Y. Pochet. A new formulation and resolution method for the p-center problem. *INFORMS Journal on Computing*, 16(1) :84–94, 2004.

[EMDS11] I. Elhallaoui, A. Metrane, G. Desaulniers, and F. Soumis. An improved primal simplex algorithm for degenerate linear programs. *INFORMS Journal on Computing*, 23(4) :569–577, 2011.

[EMSD10] I. Elhallaoui, A. Metrane, F. Soumis, and G. Desaulniers. Multi-phase dynamic constraint aggregation for set partitioning type problems. *Mathematical Programming*, 123(2) :345–370, 2010.

[EVSD05] I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. *Operations Research*, 53(4) :632–645, 2005.

[FH08] K. Fleszar and K. S. Hindi. An effective vns for the capacitated p-median problem. *European Journal of Operational Research*, 191(3) :612–622, 2008.

[GDL16] J. B. Gauthier, J. Desrosiers, and M. E. Lübbecke. Tools for primal degenerate linear programs : Ips, dca, and pe. *EURO Journal on Transportation and Logistics*, 5(2) :161–204, 2016.

[Geo74] A. M. Geoffrion. *Lagrangean relaxation for integer programming*, pages 82–114. Springer Berlin Heidelberg, Berlin, Heidelberg, 1974.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and intractability : a guide to NP-completeness*. WH Freeman and Company, San Francisco, 1979.

[Gon12] J. Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3) :587–601, 2012.

[Hak64] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations research*, 12(3) :450–459, 1964.

[Hak65] S. L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations research*, 13(3) :462–475, 1965.

[HFV99] S. Hanafi, A. Freville, and P. Vaca. Municipal solid waste collection : An effective data structure for solving the sectorization problem with local search methods. *INFOR : Information Systems and Operational Research*, 37(3) :236–254, 1999.

[HWS+65] S. W. Hess, J. B. Weaver, H. J. Siegfeldt, J. N. Whelan, and P. A. Zitlau. Nonpartisan political redistricting by computer. *Operations Research*, 13(6) :998–1006, 1965.

[IN94] T. Ibaraki and Y. Nakamura. A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76 :72–82, 1994.

[JN83] D. S. Johnson and K. A. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research*, 8(1) :1–14, 1983.

[Kal15] J. Kalcsics. Districting problems. In *Location Science*, pages 595–622. 2015.

[KH79a] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. i : The p-centers. *SIAM Journal on Applied Mathematics*, 37(3) :513–538, 1979.

[KH79b] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. ii : The p-medians. *SIAM Journal on Applied Mathematics*, 37(3) :539–560, 1979.

[Kha80] L. G. Khachiyan. Polynomial algorithms in linear programming. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 20(1) :51–68, 1980.

[LD60] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica : Journal of the Econometric Society*, pages 497–520, 1960.

[Lem01] C. Lemaréchal. Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer, 2001.

[LS04] L. A. N. Lorena and E. L. F. Senne. A column generation approach to capacitated p-median problems. *Computers & Operations Research*, 31(6) :863–876, 2004.

[Luk74] J. A. Lukes. Efficient algorithm for the partitioning of trees. *IBM Journal of Research and Development*, 18(3) :217–224, 1974.

[MB84] J. M. Mulvey and M. P. Beck. Solving capacitated clustering problems. *European Journal of Operational Research*, 18(3) :339–348, 1984.

[MCVO03] L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. District design for arc-routing applications. *Journal of the Operational Research Society*, 54(11) :1209–1221, 2003.

[MJN98] A. Mehrotra, E. L. Johnson, and G. L. Nemhauser. An optimization based heuristic for political districting. *Management Science*, 44(8) :1100–1114, 1998.

[OC94] I. H. Osman and N. Christofides. Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research*, 1(3) :317–336, 1994.

[Pan98] P.-Q. Pan. A basis-deficiency-allowing variation of the simplex method for linear programming. *Computers & Mathematics with Applications*, 36(3) :33–53, 1998.

[Pan14] P.-Q. Pan. *Linear programming computation.* Springer, 2014.

[Pol67] B. T. Polyak. A general method for solving extremal problems. In *Doklady Akademii Nauk*, volume 174, pages 33–36. Russian Academy of Sciences, 1967.

[Pol69] B. T. Polyak. Minimization of nonsmooth functionals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 9(3) :509–521, 1969.

[PS18] U. Pferschy and R. Scatamacchia. Improved dynamic programming and approximation results for the knapsack problem with setups. *International Transactions in Operational Research*, 25(2) :667–682, 2018.

[PSUV17] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 2017.

[RMF09] R. Z. Ríos-Mercado and E. Fernández. A reactive grasp for a commercial territory design problem with multiple balancing requirements. *Computers & Operations Research*, 36(3) :755–776, 2009.

[RS70] C. S. ReVelle and R. W. Swain. Central facilities location. *Geographical analysis*, 2(1) :30–42, 1970.

[SdAFU14] L. Silva de Assis, P. M. França, and F. L. Usberti. A redistricting problem applied to meter reading in power distribution networks. *Computers & Operations Research*, 41 :65–75, 2014.

[Sho85] N. Z. Shor. *Minimization Methods for Non-differentiable Functions.* Springer-Verlag, Berlin, Heidelberg, 1985.

[SPS04] M. P. Scaparra, S. Pallottino, and M. G. Scutellà. Large-scale local search heuristics for the capacitated vertex p-center problem. *Networks : An International Journal*, 43(4) :241–255, 2004.

[Van] F. Vanderbeck. Bapcod – a branch-and-price generic code. Technical report, University of Bordeaux, INRIA Research team ReAlOpt.

[Van05] F. Vanderbeck. Implementing mixed integer column generation. In *Column generation*, pages 331–358. Springer, 2005.

[Van14] R. J. Vanderbei. *Linear programming.* Springer, 2014.

[VW10] F. Vanderbeck and L. A. Wolsey. Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer, 2010.

[Wen97] P. Wentges. Weighted dantzig–wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2) :151–162, 1997.

[Wol98] L. A. Wolsey. *Integer Programming.* Wiley-Interscience, 1998.

[ZS83] A. A. Zoltners and P. Sinha. Sales territory alignment : A review and model. *Management Science*, 29(11) :1237–1256, 1983.