Titre: Title:	Méthode de réduction dynamique de contraintes pour un programme linéaire
Auteur: Author:	Jérémy Omer
Date:	2006
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Omer, J. (2006). Méthode de réduction dynamique de contraintes pour un programme linéaire [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7731/

UNIVERSITÉ DE MONTRÉAL

MÉTHODE DE RÉDUCTION DYNAMIQUE DE CONTRAINTES POUR UN PROGRAMME LINÉAIRE

JÉRÉMY OMER
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(MATHÉMATIQUES APPLIQUÉES)
AVRIL 2006

Your file Votre référence
ISBN: 978-0-494-17962-8
Our file Notre référence
ISBN: 978-0-494-17962-8

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

MÉTHODE DE RÉDUCTION DYNAMIQUE DE CONTRAINTES POUR UN PROGRAMME LINÉAIRE

présenté par : OMER Jérémy
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. AUDET Charles, Ph.D., président
M. DESAULNIERS Guy, Ph.D., membre et directeur de recherche
M. ORBAN Dominique, Doct.Sc., membre et codirecteur de recherche
M. SOUMIS François, Ph.D., membre

À toute la smala

REMERCIEMENTS

Je souhaite tout d'abord remercier mon directeur Guy Desaulniers pour sa disponibilité, son écoute, ses orientations, et pour ses conseils et directives qui ont été indispensables à la rédaction de ce mémoire. Merci aussi pour le support financier sans lequel ma qualité de vie aurait été considérablement moins bonne pendant ma maîtrise.

Je tiens également à remercier mon codirecteur Dominique Orban pour son soutien, son soucis de clarté et pour l'apport majeur des articles qu'il m'a fournis.

J'adresse mes remerciements à François Soumis pour l'intérêt qu'il a toujours manifesté à l'égard de mes recherches et pour les avancées considérables que ses idées ont rendues possibles.

Je remercie Charles Audet qui a accepté de faire partie du jury de ce mémoire.

Enfin, je voudrais remercier François Lessard, Ismail Elhallaoui, Abdelmoutalib Metrane et Ann-Sophie Pepin pour avoir toujours répondu patiemment à mes questions.

RÉSUMÉ

L'algorithme du simplexe effectue de nombreuses itérations sans amélioration de la valeur de l'objectif pendant la résolution de programmes linéaires dégénérés. Nous proposons deux algorithmes de réduction dynamique de contraintes pour limiter les effets de la dégénérescence et diminuer la taille du problème à résoudre.

La méthode générale suivie par les deux algorithmes part d'une solution de base réalisable du problème à résoudre. Si cette solution possède certaines variables de base prenant une valeur nulle, des variables nulles et des contraintes sont retirées du problème afin de diminuer le nombre d'itérations dégénérées tout en conservant la réalisabilité du problème réduit (PL_R). La résolution de PL_R permet d'obtenir une solution réalisable pour le problème initial qui améliore la valeur de l'objectif. Le gain de temps occasionné par la méthode est double puisque PL_R est plus petit et moins dégénéré que le problème initial. Pour atteindre l'optimalité du programme linéaire en conservant la taille réduite des problèmes résolus, des variables et des contraintes sont réintégrées ou supprimées après chaque résolution de PL_R. En général, plus le programme linéaire est dégénéré et plus le problème réduit est petit, rendant ainsi la résolution efficace pour les problèmes très dégénérés.

Les deux algorithmes se différencient principalement par la structure de PL_R. Dans un cas, les contraintes du problème initial sont modifiées avant de former PL_R alors que dans l'autre, les contraintes et variables sont directement supprimées sans changement préalable. La deuxième méthode possède l'avantage de ne pas augmenter la densité de la matrice de contraintes, facteur déterminant de l'efficacité de la résolution d'un programme linéaire.

Les algorithmes ont été implantés en faisant appel au logiciel commercial CPLEX

pour la résolution de PL_R et les résultats sont comparés à une résolution par l'algorithme du simplexe primal de CPLEX. Les tests numériques ont été menés sur des problèmes maîtres obtenus en appliquant une méthode de génération de colonnes sur un problème de planification simultanée d'horaires d'autobus et de tournées de chauffeurs d'autobus et sur des problèmes conçus pour comporter de la dégénérescence primale et duale. Pour l'évaluation effectuée dans ce mémoire, la solution de base réalisable initiale est trouvée en résolvant une phase 1 de l'algorithme du simplexe ou en résolvant à l'optimalité le problème après avoir perturbé les coefficients de son objectif. Ceci permet d'obtenir des solutions initiales avec différents niveaux d'erreur pour évaluer les algorithmes. Pour le meilleur des deux algorithmes, une amélioration allant d'un facteur 2 à un facteur 10 a été observée par rapport à la résolution directe par l'algorithme du simplexe.

ABSTRACT

The primal simplex method makes a lot of degenerate iterations when solving a highly degenerate linear program. We propose two dynamic constraint reduction algorithms to limit the impact of degeneracy on the solution process while diminishing the size of the problem to be solved.

The general method followed by the two algorithms starts with a basic feasible solution of the problem. If some basis variables take a zero value, some variables and some constraints are removed from the problem to form a less degenerate problem which is still feasible. The solution of the reduced problem leads to a feasible solution for the initial problem which improves the value of the objective function. The interest of the method is due to the fact that the solved problem is less degenerate and smaller, two factors which help improving solution time. In order to obtain an optimal solution for the original problem, constraints and variables are added or removed from the reduced problem after solving each reduced problem. Generally speaking, the more the problem is degenerate, the smaller the reduced problem will be, enabling the solution process to be efficient on highly degenerate problems.

The two algorithms differ mostly by the structure of the reduced problem. The first algorithm changes the constraint matrix before removing variables and constraints while the second one doesn't. The positive point of the second method is that it doesn't raise the density of the constraint matrix, which would also slow down the solution process.

The implementation of the two algorithms uses a call to the simplex method of the commercial software CPLEX to solve the reduced problems and the computational

results are compared to a direct solution of the initial problem by CPLEX. The numerical experiments were conducted on master problems obtained by a column generation method applied on vehicle and crew scheduling problems and on problems generated to be primal and dual degenerate. To evaluate the two algorithms, initial basic solutions were found by solving a phase 1 of the simplex method or by solving the problem after modifying the costs of the variables. This process enables to get initial solutions with different levels of error to evaluate the algorithms. The results observed for the best algorithm improved the solution time by a factor of 2 to 10 when compared to a standard application of the simplex method.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

CHAPITRE 1 : INTRODUCTION

La programmation linéaire est un outil d'analyse mathématique utilisé pour la planification d'horaires de véhicules et d'équipages dans le domaine des transports, pour l'optimisation de la production de raffineries dans l'industrie du pétrole et pour de nombreuses autres applications. Une méthode reconnue pour résoudre les programmes linéaires (PL) est celle de l'algorithme du simplexe, introduite par Dantzig en 1947. Cette méthode requiert que le PL soit mis sous la forme standard suivante :

$$(\text{PL}) \equiv \begin{cases} \min\limits_{x} & z = c^T x \\ s.c & Ax = b \\ & x \geq 0, \end{cases}$$

où A est une matrice $m \times n$ de rang maximal, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ et $x \in \mathbb{R}^n$. A est appelée la matrice des contraintes, b le membre de droite des contraintes, c le vecteur de coût, z la valeur de l'objectif et x le vecteur des variables. Ce PL est le *problème primal.*

L'algorithme du simplexe est une méthode itérative qui trouve à chaque itération une solution réalisable de PL qui améliore la solution précédente à moins que ne survienne un phénomène appelé dégénérescence. En présence de dégénérescence, l'algorithme peut faire du sur-place. Ce phénomène de dégénérescence est présent dans de nombreuses applications pratiques. Le but du mémoire est de proposer une méthode itérative qui tire profit de la dégénérescence. La méthode s'appuie sur l'apparition de dégénérescence en cours d'exécution de l'algorithme du simplexe pour construire un problème réduit en supprimant des variables et des contraintes du PL. Le problème réduit est alors résolu par l'algorithme du simplexe pour améliorer la valeur de l'objectif. Le problème réduit est ensuite modifié et résolu par l'algorithme du simplexe itérativement, jusqu'à atteindre l'optimalité du PL initial.

Afin de bien comprendre la méthodologie proposée, voyons d'abord l'algorithme du simplexe et le concept de dégénérescence.

1.1 L'algorithme du simplexe

Dans cette section, les principes de base de l'algorithme du simplexe sont exposés. Le lecteur intéressé à avoir plus de détails peut consulter les ouvrages de Chvátal (1983) ou de Nash et Sofer (1996). Les preuves des théorèmes sont omises ci-bas et peuvent être retrouvées dans ces références.

1.1.1 Principe de l'algorithme primal du simplexe

L'algorithme du simplexe est une méthode itérative qui traite les problèmes de manière analogue à la résolution de systèmes d'équations linéaires ou à l'inversion de matrice par des éliminations de Gauss. Les variables sont séparées en deux groupes : m variables de base positives ou nulles et $n - m$ variables hors-base nulles. Puisque A est de rang maximal, les colonnes de A associées aux variables de base sont linéairement indépendantes et forment donc une base de $\mathbb{R}^m$.

Si on réordonne les variables et que l'on utilise une notation qui explicite ce partitionnement :

$$x = \begin{pmatrix} x_B \\ x_H \end{pmatrix}, \quad c = \begin{pmatrix} c_B \\ c_H \end{pmatrix}, \quad A = \left(\begin{array}{c|c} B & H \end{array} \right),$$

on réécrit le PL ainsi :

$$\begin{array}{lrl} \min\limits_{x} & z = c_B^T x_B + c_H^T x_H & \\ \text{s.c.} & B x_B + H x_H = & \text{b} \\ & x_B, x_H \geq & 0. \end{array}$$

Définition 1.1.1 *La solution obtenue en posant $x_B = B^{-1}b$ et $x_H = 0$ est appelée solution de base.*

Chaque solution de base réalisable correspond à l'intersection de m contraintes dont certaines peuvent être des contraintes de non-négativité, ce qui correspond à un point extrême du domaine réalisable.

Théorème 1.1.1 *S'il existe une solution optimale pour un PL, alors il en existe une qui soit une solution de base.*

Le théorème 1.1.1 implique qu'il est suffisant de parcourir les solutions de base réalisables pour trouver la solution optimale d'un PL. Il existe un nombre exponentiel de solutions de base. L'algorithme du simplexe ne les énumère pas toutes, il passe d'une solution de base à une autre voisine (i.e., ayant $m-1$ variables de base en commun) qui ne détériore pas la valeur de l'objectif. Pour changer de base, il faut trouver une variable pour entrer en base, la *variable entrante*, et une variable pour sortir de la base, la *variable sortante*.

Exprimons les variables de base en fonction des variables hors-base :

$$Bx_B + Hx_H = b \Rightarrow x_B = B^{-1}b - B^{-1}Hx_H.$$

On peut alors exprimer l'objectif en fonction de ces mêmes variables :

$$z = c_B^T x_B + c_H^T x_H = c_B^T B^{-1} b + (c_H^T - c_B^T B^{-1} H) x_H.$$

Puisque les variables hors-base sont nulles, le coût de la solution courante est $z = c_B^T B^{-1} b$.

Dans tout le mémoire, nous notons : u^j le $j^{ème}$ élément d'un vecteur u, M^j, la $j^{ème}$ colonne d'une matrice M et $M^{i,j}$ l'élément de M se situant sur la ligne i et la colonne j.

Définition 1.1.2 *Le vecteur $c_H^T - c_B^T B^{-1} H$ s'appelle le vecteur des coûts réduits des variables hors-base. Le coût réduit $\bar{c}^j$ d'une variable hors-base x^j est donné par :*

$$\bar{c}^j = c^j - c_B^T B^{-1} A^j.$$

En s'appuyant sur l'expression de la valeur de l'objectif en fonction des variables hors-base, on observe que si $\bar{c}^j < 0$, la valeur de l'objectif diminuera en augmentant la valeur de la variable x^j, c'est-à-dire en la choisissant comme variable entrante.

Théorème 1.1.2 *Pour une base B réalisable donnée, si aucune variable hors-base n'a un coût réduit négatif, alors la solution de base associée est optimale.*

Supposons qu'il existe au moins une variable de coût réduit négatif. Soit x^e la variable entrante. Afin de satisfaire les contraintes de non-négativité, la variable sortante doit être choisie de façon à ce que

$$x_B = B^{-1} b - B^{-1} A^e x^e \geq 0.$$

Ainsi, on augmente la valeur de x^e jusqu'à ce qu'une variable en base s'annule. La première variable en base qui s'annule est la variable sortante. L'indice de la variable sortante x_B^s est donc déterminé par

$$s \in \underset{i=1,\ldots,m}{\operatorname{argmin}} \{(B^{-1}b)^i / (B^{-1}A^e)^i | (B^{-1}A^e)^i > 0\}.$$

Cette dernière opération est appelée le *test des ratios*. Si $\{i \in \{1, \ldots, m\} \mid (B^{-1}A^e)^i > 0\}$ est vide, la valeur d'aucune variable en base ne diminue quand la valeur de la variable entrante augmente. On peut augmenter la valeur de x^e autant qu'on le souhaite.

Théorème 1.1.3 *Soit x^e la variable entrante choisie avec $\bar{c}^e < 0$. Si l'ensemble $\{i \in \{1, \ldots, m\} \,|(B^{-1}A^e)^i > 0\}$ est vide, le problème est non-borné.*

Une fois les variables entrante et sortante identifiées, un pivotage de Gauss est effectué afin de recalculer la base B et son inverse B^{-1}.

Les opérations allant du calcul du coût réduit jusqu'au pivotage forment une itération de l'algorithme du simplexe. Des itérations de l'algorithme du simplexe sont effectuées jusqu'à ce que tous les coûts réduits soient positifs ou nuls ou qu'une variable fasse apparaître le fait que le PL est non-borné. Plusieurs itérations de l'algorithme du simplexe sont effectuées pour illustrer le déroulement de l'algorithme et la dégénérescence dans la section 1.2.

L'algorithme du simplexe primal conserve la réalisabilité de la solution x au cours des itérations. Par contre, il est possible que $\bar{c} \ngeq 0$. Il existe aussi un algorithme du simplexe dual qui préserve $\bar{c} \geq 0$ mais n'assure pas la réalisabilité de x.

Les itérations que nous venons de décrire constituent la phase 2 de l'algorithme primal du simplexe.

1.1.2 La phase 1 de l'algorithme du simplexe primal

Afin de trouver une solution de base réalisable du PL, une phase 1 de l'algorithme du simplexe est en général effectuée. Une manière de procéder revient à ajouter un vecteur de variables artificielles a de taille m au problème, puis à résoudre le PL

$$\begin{array}{lll} \min\limits_{x} & \sum\limits_{i=1\ldots m} a^i & \\ \text{s.c.} & Ax + a = & \mathrm{b} \\ & x, a \geq & 0. \end{array}$$

On peut toujours reformuler le problème de façon à ce que $b \geq 0$ dans l'écriture d'un PL sous forme standard. Si on s'est placé dans cette situation, la solution telle que $x = 0$ et $a = b$ est une solution de base réalisable pour ce problème de la phase 1. On résout ce PL par des itérations de l'algorithme du simplexe. Si la valeur optimale de l'objectif est nulle, l'expression de l'objectif implique que $a = 0$, donc le vecteur de variables optimales x est réalisable pour le problème initial. Dans le cas contraire, il n'existe aucune solution réalisable de ce problème telle que $Ax + a = b$ avec $a = 0$, d'où le PL initial n'est pas réalisable.

1.1.3 Le problème dual

On introduit également le *problème dual* que l'on formule :

$$\begin{array}{ll} \max\limits_{\pi} & b^T\pi \\ \text{s.c.} & A^T\pi \leq \quad c, \end{array}$$

avec $\pi \in \mathbb{R}^m$. π est le vecteur des variables duales.

Théorème 1.1.4 (*Théorème de la dualité*) *Si le problème primal a une solution optimale x^*, alors le problème dual a une solution optimal π^* telle que*

$$c^T x^* = b^T \pi^*.$$

À chaque itération de l'algorithme du simplexe primal, on donne au vecteur de variables x la valeur $B^{-1}b$, où B est la base réalisable courante. Afin d'obtenir une solution optimale du problème dual en même temps que la solution optimale du problème primal, on attribue à chaque itération la valeur $c_B^T B^{-1}$ au vecteur des variables duales π^T pour respecter le théorème de la dualité. Le vecteur de coût réduit $\bar{c}$ se calcule donc aussi par :

$$\bar{c}^T = c^T - \pi^T A.$$

1.2 Dégénérescence dans l'algorithme du simplexe

Une fois la variable entrante x^e choisie, il arrive qu'il soit impossible d'augmenter la valeur de x^e. Puisque $x_B = B^{-1}b - B^{-1}A^e x^e$, augmenter la valeur de x^e amènera à donner une valeur négative à x^k si une des variables de base x^k est nulle et que $(B^{-1}A^e)^k > 0$. La valeur de l'objectif reste alors la même malgré le changement de base et, au moment de la mise à jour de B^{-1}, on fait ce que l'on appelle un *pivot dégénéré.*

Dans la suite du mémoire, nous appellerons *solution dégénérée* une solution de base dont certaines variables en base sont nulles, car le risque d'effectuer un pivot dégénéré à la prochaine itération de l'algorithme du simplexe ne se pose que lorsque des variables en base sont nulles. Lorsque des pivots dégénérés sont effectués en cours de résolution d'un PL par l'algorithme du simplexe, on dit que ce PL est dégénéré. L'algorithme du simplexe primal et la dégénérescence sont illustrés dans l'exemple ci-dessous.

Exemple : On considère le PL suivant

$$\begin{array}{ll} \min\limits_{x} & z = x^1 + x^2 + 4x^3 + x^4 + x^5 \\ \text{s.c.} & x^1 + 4x^3 + x^4 + 2x^5 = 1 \\ & x^2 + 4x^3 + 2x^4 = 1 \\ & x \geq 0. \end{array}$$

$x = (1, 1, 0, 0, 0)^T$ est une solution de base réalisable de coût $z = 2$ telle que $x_B = (x^1, x^2)$ et $x_H = (x^3, x^4, x^5)$. L'algorithme du simplexe part de cette solution. On décrit ci-dessous la suite des itérations jusqu'à atteindre l'optimalité.

- Itération 1.

  Les variables de base et l'objectif sont exprimées en fonction des variables hors-base.

$x^1 = 1 - 4x^3 - x^4 - 2x^5$

$x^2 = 1 - 4x^3 - 2x^4$

$z = 2 - 4x^3 - 2x^4 - x^5$

Le coût réduit $\bar{c}$ est déduit de l'expression de z et vaut $\bar{c} = (0, 0, -4, -2, -1)^T$. Le choix de la variable entrante se porte souvent sur la variable de coût réduit minimal. On prend donc $x^e = x^3$. Pour les deux variables en base le ratio $(B^{-1}b)^i/(B^{-1}A^e)^i$ vaut $\frac{1}{4}$. On peut donc choisir indifféremment l'une ou l'autre des deux variables en base pour sortir de la base. On prend $x^s = x^1$. La nouvelle solution et l'objectif valent $x = (0, 0, \frac{1}{4}, 0, 0)^T$ et $z = 1$ et les deux variables en base sont x^3 et x^2. La variable x^2 est une variable en base nulle. C'est donc une variable dégénérée.

- Itération 2.

  On répète le même processus.

  $x^3 = \frac{1}{4} - \frac{x^1}{4} - \frac{x^4}{4} - \frac{x^5}{2}$

  $x^2 = 0 + x^1 - x^4 + 2x^5$

  $z = 1 + 2x^1 - x^4 + x^5$

  On obtient $\bar{c} = (0, 2, 0, -1, 1)^T$. La variable entrante est donc $x^e = x^4$ et par le test des ratios, $x^s = x^2$.

  On est alors dans le cas où la variable sortante est une variable de base nulle. La variable x^4 rentre donc en base avec la valeur 0 et par suite, la valeur de z ne va pas diminuer. Par conséquent, on effectue un pivot dégénéré en calculant B^{-1} (ici B^{-1} n'est pas calculé explicitement, mais le pivot est quand même fait pour exprimer x_H et z en fonction de x_B).

- Itération 3.

  $x^3 = \frac{1}{4} - \frac{x^1}{2} + \frac{x^2}{4} - x^5$

  $x^4 = 0 + x^1 - x^2 + 2x^5$

  $z = 1 + x^2 - x^5$

  On déduit $\bar{c} = (0, 1, 0, 0, -1)$. Donc $x^e = x^5$ puis $x^s = x^3$. La solution devient $x = (0, 0, 0, \frac{1}{2}, \frac{1}{4})^T$, puis $z = \frac{3}{4}$. L'itération suivante mène à $\bar{c} \geq 0$ donc la solution courante est optimale. □

Lorsqu'un programme linéaire est très dégénéré, i.e., que beaucoup de pivots dégénérés sont effectués, la résolution est considérablement ralentie et il arrive dans certains cas que l'algorithme cycle. Il s'agit d'ailleurs de la seule situation dans laquelle la convergence de l'algorithme du simplexe n'est pas certaine. Toutefois, un choix judicieux des variables entrante et sortante assure la convergence théorique de l'algorithme. Des méthodes de perturbation, utilisées dans beaucoup de codes performants, réduisent aussi le nombre de pivots dégénérés et rétablissent des temps de résolution raisonnables.

Des théories développées plus récemment mettent à profit la dégénérescence pour réduire la taille du problème. Lorsque le problème est dégénéré, la solution de base courante a forcément, à un moment donné, des variables de base nulles. Soit $p < m$ le nombre de variables de base non-nulles de cette solution. Dit autrement, les contraintes du PL ont été satisfaites par l'action de p variables uniquement. Certaines contraintes ont donc un rôle redondant dans l'optimisation du problème, ce qui nous suggère la possibilité de construire une programme linéaire dont la valeur optimale de l'objectif sera la même que celle du PL en supprimant des contraintes et des variables. Ces opérations mènent à résoudre un problème plus petit et moins dégénéré que le PL initial et à en déduire la solution optimale de ce dernier. Le gain en temps est double puisque la taille d'un problème et l'occurrence de pivots dégénérés sont deux facteurs de grande influence sur le temps de résolution.

1.3 Contenu du mémoire

Une méthode se servant de la dégérescence pour réduire la taille des problèmes résolus a déjà été appliquée avec succès sur quelques classes de problèmes dont la structure particulière a été exploitée. Nous proposons une méthode fonctionnelle pour tous les programmes linéaires en nous appuyant sur les méthodes récemment développées.

Dans un premier temps, une revue de littérature est faite dans le chapitre 2 sur la dégénérescence et les moyens de la combattre, suivie d'une description des méthodes récentes d'agrégation de contraintes. Deux algorithmes de réduction dynamique de contraintes sont ensuite décrits dans les chapitres 3 et 4. Finalement, les résultats de la plus aboutie des deux méthodes sont présentés au chapitre 5.

CHAPITRE 2 : REVUE DE LITTÉRATURE

Notre méthode trouve son intérêt dans la résolution d'un programme linéaire dont la résolution fait apparaître de la dégénérescence et s'inspire des travaux déjà faits sur l'agrégation de contraintes. Dans la section 2.1, nous parcourons la littérature afin d'étudier la manière dont la dégénérescence peut être traitée. Puis nous exposons certaines méthodes de résolution de PL par agrégation de contraintes dans la section 2.2. Finalement, nous positionnons nos travaux de recherches par rapport aux travaux déjà effectués et annonçons la contribution scientifique de notre méthode de réduction dynamique.

2.1 Dégénérescence en programmation linéaire

Lorsqu'on utilise l'algorithme primal du simplexe, le phénomène de dégénérescence entraîne des itérations sans diminution de la valeur de l'objectif et, dans certains cas, l'algorithme bouclera. Nous allons présenter une revue de littérature des moyens mis en oeuvre pour combattre la dégénérescence dans la résolution d'un PL par la méthode du simplexe et, plus particulièrement, par l'algorithme primal.

2.1.1 Les règles de pivot

La mise en évidence par Hoffman (1953) d'exemples de PL pour lesquels la résolution par la méthode du simplexe boucle a poussé à s'interroger sur la convergence de l'algorithme. Un choix judicieux des variables entrante et sortante lorsqu'il y a ambiguité force la convergence de l'algorithme. Bland (1977) décrit une telle règle bien connue

pour sa simplicité. Elle consiste à prendre commme variable entrante, celle de plus petit indice parmi les variables de coût réduit négatif minimum, puis de faire sortir la variable de plus petit indice parmi celles de ratio minimum. La règle d'Edmonds et Fukuda, les règles de "Last in First out" et "the Most often selected", dont Terlaky et Sushong (1993) nous donnent les schémas, empêchent aussi l'algorithme du simplexe de boucler.

Cependant, ces règles de pivot ont eu un intérêt plus théorique que pratique ; même si elles évitent de boucler, elles n'améliorent pas tellement les performances de l'algortihme du simplexe pour tous les cas où il stagne sur un sommet dégénéré pendant de nombreuses itérations. En outre, comme en témoignent les tests réalisés par Gass et Vinjamuri (2004), les logiciels modernes de résolution de PL qui combattent la dégénérescence par des méthodes de perturbation, de bris d'égalités, et des procédures primales duales, ne bouclent jamais sur les exemples théoriques connus à difficultés. Il arrive encore que certaines applications fassent boucler l'algorithme du simplexe mais les responsables de ce comportement sont les incertitudes numériques. Nous prêterons une attention plus particulière aux méthodes de perturbation plus efficaces en général.

2.1.2 Les méthodes de perturbation

La méthode des perturbations a d'abord été introduite par Charnes (1952). À partir de n'importe quelle base réalisable B, le problème initial est perturbé une fois pour toutes en changeant le membre de droite b par

$$b_p = b + \sum_{j=1}^{m} \epsilon^j B^j,$$

où ϵ est une constante positive arbitrairement choisie. Exceptionnellement, dans cette expression de b_p et celle de x^* donnée ci-dessous, le terme ϵ^j ne repésente pas la $j^{ème}$

composante du vecteur ϵ mais ϵ élevé à la puissance j. Charnes démontre alors que, pour une valeur suffisamment petite de la constante ϵ, le nouveau problème n'est plus dégénéré et que, si elle existe, la solution optimale x^* du problème initial se déduit de la solution du problème perturbé x_p^* par

$$x^* = x_p^* - \sum_{j=1}^{m} \epsilon^j (B_p^{-1} A)^j,$$

où B_p est la base optimale du problème perturbé. Une analyse un peu plus poussée de la procédure mène à la conclusion qu'il n'est jamais besoin d'implémenter à proprement parler le problème perturbé. Pour une valeur de ϵ suffisamment petite, ϵ^j est négligeable devant les termes ϵ^k pour $k < j$, ce qui mène pendant le test des ratios à toujours prendre la variable de plus petit indice si une ambiguité est rencontrée. Cette méthode donne la convergence du simplexe mais elle n'est pas plus intéressante dans la pratique que les règles de pivots énumérées ci-dessus. En revanche, elle a inspiré d'autres implémentations plus payantes des perturbations.

Wolfe (1963) propose une méthode de perturbations *adhoc* dans laquelle on attend de trouver une solution dégénérée avant de modifier le problème. Supposons que l'on ait une base réalisable B, et une solution de base associée x_B dégénérée. On pose alors I, l'ensemble des indices des contraintes et

$$I_0 = \{i \in \{1, \ldots, n \mid (B^{-1}b)^i = 0\}, \quad I_0 \neq \emptyset.$$

I_0 est également l'ensemble des indices des variables de base dégénérées. Si on n'est pas à l'optimalité, il existe une variable hors-base de coût réduit négatif x^e susceptible d'entrer en base. Si on utilise le test du ratio, en entrant en base, x^e prendra la valeur :

$$x^e = \min_{i=1,\ldots,m} \{ (B^{-1}b)^i / (B^{-1}A)^{i,e} \mid (B^{-1}A)^{i,e} > 0 \}.$$

À moins que $(B^{-1}A)^{i,e} \leq 0$, $i \in I_0$, un pivot dégénéré va être effectué. On introduit alors le vecteur de perturbation ϵ, tel que ϵ^i soit une constante positive arbitraire

pour $i \in I_0$ et $\epsilon^i = 0$ sinon. Cette perturbation restera en vigueur tant que l'on restera sur le même point extrême, elle sera retirée en le quittant.
En effectuant alors la modification du membre de droite

$$b_0 = b + B\epsilon,$$

la solution de base devient

$$x_B = x_B + \epsilon,$$

puis la variable entrante prend la valeur :

$$x^e = \min_{i \in I_0} \{ (B^{-1}b_0)^i / (B^{-1}A)^{i,e} \mid (B^{-1}A)^{i,e} > 0 \}. \tag{2.1}$$

Le pivot qui suit est non-dégénéré. On continue ensuite à effectuer des pivots jusqu'à se retrouver dans l'une des trois situations suivantes :

- Toutes les variables ont un coût réduit positif ou nul. L'expression du coût réduit ne dépend pas du membre de droite donc la base trouvée B est optimale. On pose $x_B = B^{-1}b$ pour obtenir la valeur de la solution optimale.
- Aucune valeur ne peut être attribuée à la variable entrante en se référant à l'équation du ratio (2.1). Le prochain pivot se fera donc sur une des variables de base de $I - I_0$. Les pivots sur le problème perturbé n'ont introduit aucune dégénérescence supplémentaire sur les variables de $I - I_0$ puisque l'algorithme est resté sur le même point extrême. On peut alors revenir au problème initial avec la certitude d'effectuer un pivot non-dégénéré.
- Une solution de base dégénérée est produite avec $I_1 = \{i \,|(B^{-1}b_0)^i = 0\} \subset I_0$ et $(B^{-1}A)^{i,e} > 0$ pour au moins un élément de I_1. On appelle alors récursivement la procédure en remplaçant I_0 par I_1.

En suivant cette procédure, on ne fera que des pivots non-dégénérés, ce qui assure la stricte décroissance de l'objectif. De plus, on a toujours $|I_1| < |I_0| < m - 1$. Donc la profondeur de récursion est finie, d'où la preuve que l'algorithme se termine.

Le manque de problèmes très dégénérés dans les années qui ont suivi la présentation de l'algorithme de Wolfe ont amené à négliger cette procédure jusqu'à ce que Ryan et Osborne (1988) appliquent cette méthode à un problème fortement dégénéré de planification d'horaires de véhicules. Les résultats ont montré que la procédure était très efficace et nécessitait très peu de changements dans le code. Seulement 17 instructions ont été ajoutées pour implémenter la méthode ; l'augmentation de la durée des itérations est donc négligeable. Sur leur exemple, l'algorithme du simplexe révisé ne diminuait plus la valeur de l'objectif pendant 1500 itérations après l'itération 444, alors qu'en ajoutant la procédure de Wolfe, le problème a été optimisé à l'itération 526.

George et Osborne (1993) approfondissent l'analyse en remarquant que la récursivité de l'algorithme peut être supprimée. Lorsqu'un pivot sur le problème perturbé redonne une solution dégénérée, relancer (au besoin plusieurs fois) l'itération en changeant les ϵ^i de manière aléatoire, mais en conservant le même ensemble I_0 évite la récursion et donne la même convergence. En effet, en arithmétique exacte, la chance de retrouver un problème dégénéré, si les ϵ^i ont été choisis au hasard, est nulle. En relançant le problème plusieurs fois, la probabilité de trouver une solution dégénérée sera le produit de probabilités très faibles que l'on considère nul.

Finalement, Gass (1993) recense les moyens utilisés pour combattre la dégénérescence par quatre codes efficaces à ce temps là. Son étude traite MINOS 5.3 et MPSX/370 qui modifient la méthode de perturbations de Wolfe, SCIONIC dans lequel est implémentée une procédure de perturbations virtuelles et CPLEX qui s'aide également d'une méthode de perturbations. La gestion de la perturbation par CPLEX nous intéresse particulièrement car nous comparerons nos résultats à ceux obtenus par CPLEX et nous comptons également l'appeler pour optimiser nos problèmes réduits. Lorsque la valeur de l'objectif n'a pas été améliorée et qu'aucune variable artificielle n'a quitté la base pendant un nombre d'itérations T dépendant du nombre de contraintes du problème (pour $m = 10000$, on aura $T = 600$ par exemple), le problème est perturbé de

la manière suivante. Supposons les variables du problème bornées, i.e., $l^j \leq x^j \leq u^j$. Les bornes sont déplacées ainsi : $l^j - \epsilon Y^j \leq x^j \leq u^j + \epsilon X^j$ avec X^j et Y^j deux variables aléatoires uniformes dans [0,1] et ϵ une constante positive égale à 10^{-4} par défaut. Le problème est alors résolu jusqu'à optimalité en perturbant à nouveau si besoin. On enlève ensuite les perturbations pour vérifier la réalisabilité de la base ; elle l'est la plupart du temps, mais si ce n'est pas le cas, on repart de la base actuelle en enlevant les perturbations et en actualisant T et ϵ par $T = 2T$ et $\epsilon = \epsilon/10$. Dans le cas où une variable n'a pas de borne inférieure ou supérieure, on ne perturbe que la borne existante.

2.1.3 Les méthodes récentes de traitement de la dégénérescence

Deux algorithmes proposés par Barnes *et al.* (2002) et Pan (1998) s'appuient sur des caractérisations analogues de la dégénérescence pour la méthode duale et la méthode primale, respectivement. Le phénomène de dégénérescence étudié jusqu'ici est celui rencontré pendant la résolution d'un PL par l'algorithme du simplexe primal que l'on appelle aussi dégénérescence primale. On parle de dégénérecence duale lorsque des pivots dégénérés sont effectués dans la résolution d'un PL par l'algorithme du simplexe dual. La dégénérescence primale peut survenir lorsque des variables en base sont nulles alors que la dégénérescence duale peut survenir quand le coût réduit de certaines variables hors-base est nul.

Barnes *et al.* partent d'une solution duale réalisable π_0 construite de façon à ce que

$$E = \{A^j \mid \pi_0^T A^j = c^j\} \neq \emptyset.$$

E va ensuite jouer le même rôle que celui de la base dans l'algorithme dual usuel. Il faut donc considérer le système

$$Ex_E = b, \qquad x_E \geq 0. \tag{2.2}$$

Si le système 2.2 a une solution x_E^*, alors la solution obtenue en donnant la valeur x_E^* aux variables associées à E et 0 aux autres est réalisable et a même coût que la solution duale réalisable π_0 ; l'optimalité est donc atteinte. Sinon, on recherche une direction de descente ρ dans l'espace dual telle que :

$$E^T\rho \leq 0$$

$$b^T\rho > 0.$$

Il est démontré qu'une bonne direction est donnée par $\rho = (b - Ex_E^*)$ où x_E^* est la solution de

$$\min_x \quad ||b - Ex||^2 \quad s.c. \quad x \geq 0.$$

On actualise alors la solution duale π_0 par

$$\pi_0 = \pi_0 + t\rho,$$

avec $t = \min_{j|\rho A^j>0}(c^j - \pi_0 A^j)/(\rho A^j)$. Cet algorithme n'effectue jamais de pivot dégénéré et bat les algorithmes du simplexe primal et dual en nombre d'itérations sur tous les tests présentés par les auteurs.

La vision primale correspond plus à nos objectifs et mérite toute notre attention. Pan (1998) commence par élargir le concept de la base. Une base est redéfinie comme étant une sous-matrice de A composée d'un nombre quelconque s de colonnes linéairement indépendantes ; l'espace vectoriel engendré par la base doit contenir le membre de droite b. Si $s = m$, la base est dite pleine. Si $s < m$, la base est dite déficiente. Supposons que l'on dispose d'une base réalisable B et d'une matrice des variables hors-base N. On définit la liste des indices de base et hors-base respectivement par

$$J_B = \{j_1, \ldots, j_s\} \quad et \quad J_N = \{k_1, \ldots, k_{n-s}\},$$

où j_i, $i = 1, \ldots, s$, est l'indice de la $i^{ème}$ colonne de B, et k_j, $j = 1, \ldots, n-s$, l'indice de la $j^{ème}$ colonne de N. On réordonne les colonnes de A et les éléments de b, c et x de sorte que

$$A = [B, N], \qquad c^T = [c_B^T, c_N^T], \qquad x^T = [x_B^T, x_N^T],$$

et que PL soit équivalent à

$$\begin{array}{ll} \min\limits_{x} & z = c_B^T x_B + c_N^T x_N \\ s.c. & Bx_B + Nx_N = b \\ & x_B \geq 0, \qquad x_N \geq 0. \end{array}$$

On forme alors le tableau initial

$$\left[\begin{array}{cc|c} B & N & b \\ c_B^T & c_N^T & 0 \end{array}\right].$$

On fait des pivots sur le tableau afin que B soit triangulaire supérieure avec des éléments diagonaux non-nuls et que c_B s'annule. On obtient le nouveau tableau représentant PL :

$$\left[\begin{array}{cc|c} U & \bar{N} & \bar{b} \\ \bar{c}_B^T & \bar{c}_N^T & -z \end{array}\right] = \left[\begin{array}{cc|c} U_1 & \bar{N}_1 & \bar{b}_1 \\ 0 & \bar{N}_2 & \bar{b}_2 \\ 0 & \bar{c}_N & -z \end{array}\right]$$

Ce tableau est dit sous forme canonique. La partie nulle de la matrice qui se trouve juste en dessous de U_1 n'existe pas si $s = m$, l'algorithme devient dans ce cas analogue à l'algorithme du simplexe usuel. La ligne des coûts représente maintenant les coûts réduits des variables. Ensuite, les coefficients de la partie inférieure du tableau des variables non-nulles ayant été annulés, la réalisabilité de la solution impose que $\bar{b}_2 = 0$. On note $\bar{A}$ la matrice des coefficients associée à ce tableau. La solution de base associée $\bar{x}$ s'exprime maintenant par

$$\bar{x}_B = U_1^{-1}\bar{b}_1 \quad et \quad \bar{x}_N = 0.$$

Si $\bar{c}_N \geq 0$, l'optimalité est atteinte et sinon on sélectionne la variable entrante en prenant la variable d'indice k_q telle que

$$q \in \underset{j=1,\ldots,n-s}{\operatorname{argmin}} \{\bar{c}_{k_j}\}.$$

Deux situations peuvent alors se produire.

Cas 1 : $s = m$ ou $s < m$ et les éléments de $\bar{A}^{k_q,j}$, $j = s+1, \ldots, n$, sont nuls.
On choisit alors la variable sortante par le test des ratios et on ajuste J_B et J_N en

fonction. On effectue des pivots pour que la nouvelle matrice U_1 reste triangulaire supérieure. Dans ce cas, l'espace engendré par les colonnes de U contient toujours le vecteur b et il n'est donc pas nécessaire d'augmenter la dimension de la base. Une telle itération est appelée itération pleine.

Cas 2 : $s < m$ et l'un des éléments de $\bar{A}^{k_q,j}$, $j = s+1, \ldots, n$ est non nul.
Cette fois, si la valeur de x_{k_q} augmente, l'une des $m - s$ dernières contraintes sera violée. Il faut donc augmenter le nombre de colonnes de la base. On pose alors $s = s + 1$, on élimine par des pivots les colonnes $s + 1$ jusqu'à m de la ligne d'indice k_q de $\bar{A}$, en échangeant si besoin des lignes pour avoir le plus grand élément possible dans la diagonale (c'est l'élément sur lequel on pivotera s'il y a lieu). Puis on fait les changements nécessaires sur J_B et J_N afin que la variable d'indice k_q entre en base et la variable de plus petit ratio sorte. La procédure implique que la matrice U_1 reste triangulaire supérieure. Une telle itération est dite d'augmentation du rang.

Dans son algorithme, Pan part d'une solution réalisable mise en évidence par une phase 1 de l'algorithme du simplexe avec variables artificielles et prend comme base initiale la plus petite base associée à sa solution. Il suppose de plus que les pivots effectués dans les itérations pleines sont toujours non-dégénérées. Par suite, les pivots des itérations d'augmentation du rang sont non-dégénérés également et son algorithme se termine car la taille de la base ne diminue jamais. Cet algorithme a été testé sur plusieurs problèmes de librairie NetLib en le comparant à une phase 2 où l'inverse de la base est toujours calculé explicitement. Le nouvel algorithme obtient de meilleurs résultats tant au niveau du nombre d'itérations que du temps de calcul.

La vision de Pan a ceci de plaisant dans sa manière de traiter la dégénérescence qu'elle ne se limite pas à la combattre ou à l'éliminer. Elle en tire partie pour réduire la taille du problème et, par conséquent, diminuer le nombre d'itérations et le temps de calcul par itération. Un de nos objectifs est de profiter au maximum de la réduction de la taille du PL. On voudrait faire apparaître explicitement cette réduction et

également être capable de réduire dynamiquement le problème si de nouvelles dégénérescences apparaissent. Des recherches ont été menées sur l'agrégation dynamique de contraintes qui nous guideront dans notre gestion des variations de la taille du problème. La suite en présente une étude.

2.2 Agrégation dynamique de contraintes

Rogers *et al.* (1991) recensent des méthodes d'agrégation de contraintes proposées avant 1990. La plupart des agrégations sont statiques (on n'agrège qu'une seule fois en début de résolution), et ne conservent pas la réalisabilité du problème, ce qui mène à des solutions approchées. Mendelssohn (1982) et Shetty et Taylor (1987), par contre, développent des méthodes d'agrégation dynamiques dans lesquelles le nombre de variables est également réduit. La méthode de Mendelssohn s'applique aux PL rencontrés dans les processus de décisions markoviens. À chaque itération, un problème agrégé et plusieurs sous-problèmes sont résolus pour calculer des variables duales du problème initial. On utilise ensuite ces variables pour définir le prochain problème agrégé. Shetty et Taylor s'intéressent au cas général. Le problème est agrégé une fois au départ puis des contraintes sont désagrégées lorsqu'elles sont violées en cours de résolution. Il calcule également des bornes inférieures et supérieures de l'objectif au fil des itérations. Des tests sur des problèmes ayant jusqu'à 200 contraintes sont exécutés et mènent à une réduction importante du temps de calcul lorsqu'on se satisfait d'une solution approchée à 1%, sans pour autant rencontrer un grand succès quand on recherche l'optimalité.

Elhallaoui *et al.* (2005) proposent une méthode d'agrégation dynamique de contraintes pour résoudre la relaxation linéaire d'un problème de partitionnement d'ensemble. Ils se placent dans un contexte de problèmes de tournées de véhicules ou d'horaires

d'équipages dans lesquels on cherche à couvrir un ensemble de tâches, chaque tâche exactement une fois, à l'aide d'ensembles de chemins de coût minimal. La procédure utilise la génération de colonnes mais nous nous concentrerons ici sur l'algorithme sans aborder la génération de colonnes à laquelle nous n'aurons pas recours. À partir d'un ensemble C de chemins, on construit une partition $Q = \{W_1, \ldots, W_l\}$ de W comme suit. Pour $j = 1, \ldots, l$, $W_j = \{w_j^1, w_j^2, \ldots, w_j^{|W_j|}\}$ est un ensemble ordonné tel que tout chemin de C couvre w_j^1, w_j^2,...,$w_j^{|W_j|}$ consécutivement et dans cet ordre ou ne couvre aucune des ces tâches. Un chemin p est compatible avec Q si, pour $j = 1, \ldots, l$, p couvre toutes les tâches de W_j dans l'ordre de la numérotation des tâches de W_j ou n'en couvre aucune. Sinon, p est incompatible. Un problème agrégé PL_Q est déduit de la partition en ne gardant qu'une tâche par élément de Q et en éliminant toutes les variables incompatibles avec Q. L'algorithme de résolution se formule alors ainsi

1. Faire des pivots de la méthode du simplexe sur PL_Q.
2. Calculer les variables duales désagrégées.
3. Chercher les variables avec un coût réduit négatif.
4. Si de telles variables existent, continuer. Sinon, la solution est optimale, stop.
5. Si on doit changer de partition, continuer. Sinon, retourner à l'étape 1.
6. Mettre Q et PL_Q à jour puis aller à l'étape 1.

On effectue des pivots sur PL_Q jusqu'à atteindre un critère d'arrêt tel qu'une réduction de la fonction objectif insuffisante ou un trop grand nombre de variables dégénérées. Les pivots donnent une solution primale et une solution duale de PL_Q qui ne donne pas d'information sur toutes les variables du problème initial. Pour cette raison, les variables duales sont désagrégées par un problème minimisant le nombre de variables de coût réduit négatif que l'on peut ramener à un plus court chemin rapide à résoudre. On détermine alors si la partition doit être mise à jour par un test

basé sur le ratio entre le plus petit coût réduit des variables compatibles et le plus petit coût réduit des variables incompatibles. La mise à jour de la partition mène à une agrégation du problème quand $\frac{|Q|}{|W|}$ excède une quantité donnée et que la fonction objectif a diminué ou à une désagrégation. Dans les deux cas, une nouvelle partition Q est formée à partir d'une mise à jour de l'ensemble de chemins C. Si on agrège, C est réduit à l'ensemble des chemins associés aux variables de base non-dégénérées, et si on désagrège, on ajoute à C des chemins associés à un ensemble de variables incompatibles dont les coûts réduits sont les plus négatifs. Des tests sur les relaxations de problèmes de construction d'horaires de chauffeurs d'autobus ont donné lieu à une réduction du temps de calcul allant jusqu'à 80%.

Dans l'article suivant d'Elhallaoui *et al.*, une méthode multi-phase a été introduite pour choisir les variables à désagréger. Les chemins incompatibles avec la partition Q sont classés selon leur nombre d'incompatibilités. Le nombre d'incompatibilités d'un chemin représente le nombre de fois que l'ordre de parcourt des tâches suggéré par la partition Q est violé. La probabilité d'effectuer des pivots dégénérés après la désagrégation sera d'autant plus grande que le nombre d'incompatibilités des chemins associés aux variables désagrégées est grand. Le choix des variables à désagréger se fera donc dans l'ordre du nombre d'incompatibilités des chemins qui leur sont associés. Il ne sera en général pas nécessaire d'aller plus loin que 2 incompatilités pour trouver les chemins associés aux variables qui amélioreront la valeur de l'objectif.

2.3 Contribution personnelle

La revue que nous venons de faire révèle que l'intérêt de combattre la dégénérescence remonte aux débuts de l'algorithme du simplexe et n'a jamais faibli. L'évolution des méthodes a amené à constater que la dégénérescence était une opportunité pour réduire les temps de calcul et les nombres d'itérations.

Nous contribuons à ces avancées en développant deux algorithmes. Le premier part de l'algorithme primal de Pan (1998), mais nous réduisons explicitement la taille du problème en nombre de variables et en nombres de contraintes. Nous réduisons la taille du problème en cours de résolution si de la dégénérescence est créée et nous construisons une solution duale du problème initial dont les valeurs mènent au calcul d'un coût réduit ayant du sens. Le deuxième algorithme reprend le même schéma, mais on y change la méthode de réduction et on y ajoute quelques raffinements afin qu'il s'adapte mieux aux différentes classes de problèmes. Nous montrons enfin l'efficacité de ces algorithmes en les comparant à une résolution directe par le logiciel commercial CPLEX pour deux classes de problèmes dégénérés.

CHAPITRE 3 : LA RÉDUCTION DYNAMIQUE DE CONTRAINTES

Notre objectif initial était de généraliser la vision de Elhallaoui *et al.* (2005) au cas général des programmes linéaires quelconques. Le manque de structure du programme à résoudre nous encourage plutôt à partir de la vision de Pan (1998). Toutefois, nous nous démarquerons de son algorithme en réduisant le problème explicitement par une suppression de contraintes et de variables. Nous aimerions également nous détacher de son hypothèse de non-dégénérescence des pivots de ses itérations pleines et réduire la taille du problème lorsqu'une nouvelle dégénérescence est rencontrée pour continuer à profiter de la dégénérescence en cours de résolution. En outre, les variables duales qu'il calcule à chaque itération ont une valeur nulle pour toutes les contraintes dont le membre de droite est nul. Nous désirons déterminer des variables duales conduisant au calcul d'un coût réduit qui sera négatif pour moins de variables et évitera, nous l'espérons, de nombreux pivots du simplexe inutiles.

Ce chapitre se divise en quatre parties. La section 3.1 est consacrée au concept de base de la réduction dynamique de contraintes. Nous présentons notre algorithme de base et en expliquons le détail dans la section 3.2 et nous prouvons sa convergence dans la section 3.3. Puis, dans la section 3.4, nous décrivons l'implémentation de l'algorithme, les classes sur lesquelles nous l'expérimentons, et les résultats obtenus.

3.1 Concept de base

Notre objectif est de réduire la taille du problème en nombres de variables et de contraintes. Dans ce chapitre, on ne supprimera pas directement des contraintes du

problème initial PL. Il sera d'abord transformé en multipliant A et b par une matrice inversible C, ce qui ne change ni le domaine réalisable ni la valeur de l'objectif et donne donc un problème équivalent. Nous supprimerons par la suite des contraintes du nouveau problème noté PL_C.

Cet algorithme de résolution part d'une base réalisable B et d'une solution de base associée $x_B = B^{-1}b$. On suppose que la solution courante possède d variables nulles en base , $d > 0$. On définit les ensembles d'indices de lignes ordonnés dégénérés (N pour nul) et non-dégénérés (P pour positif), respectivement, par

$$I_N = \{j_1, \ldots, j_d\} \quad et \quad I_P = \{k_1, \ldots, k_{m-d}\},$$

où $x_B^{j_i} = 0$, $i = 1, \ldots, d$, et $x_B^{k_i} > 0$, $i = 1, \ldots, m-d$. Posons $C = B^{-1}$. I_N est l'ensemble des indices des contraintes de PL_C dont le membre de droite Cb est nul. À partir de la matrice C, nous définissons une matrice de compatibilité C_N et une matrice de réduction C_P dont les lignes seront les deux sous-ensembles de l'ensemble des lignes de C dont les indices sont respectivement dans les ensembles I_N et I_P. En notant C_k la ligne de C d'indice k, C_N et C_P s'écrivent

$$C_N = \begin{pmatrix} C_{j_1} \\ \vdots \\ C_{j_d} \end{pmatrix} \quad et \quad C_P = \begin{pmatrix} C_{k_1} \\ \vdots \\ C_{k_{n-d}} \end{pmatrix},$$

avec $C \in \mathbb{R}^{m \times m}$, $C_N \in \mathbb{R}^{d \times m}$ et $C_P \in \mathbb{R}^{(m-d) \times m}$. Nous avons donc $C_N b = 0$ et $C_P b > 0$. La matrice C_P donnera les contraintes du problème réduit tandis que C_N permettra de déterminer les variables qui seront conservées dans le problème réduit. Afin de connaître les variables présentes dans le problème réduit, nous définissons le concept de compatibilité.

Définition 3.1.1 *Soit M une matrice $m_1 \times m$ quelconque telle que $m_1 < m$. Une variable x^j du programme linéaire PL est dite compatible avec la matrice M si et seulement si*

$$MA^j = 0.$$

La variable x^j est incompatible si elle n'est pas compatible.

Exemple : Considérons les deux matrices M et A définies par

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad A = \begin{pmatrix} 5 & 12 & 4 & 3 & 0 & 9 \\ 0 & 0 & 6 & 0 & 10 & 15 \\ 0 & 1 & 5 & 0 & 12 & 5 \\ 4 & 2 & 0 & 1 & 6 & 13 \\ 12 & 0 & 5 & 0 & 6 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Pour connaître les variables compatibles avec M, on forme le produit MA :

$$MA = \begin{pmatrix} 0 & 0 & 6 & 0 & 10 & 15 \\ 0 & 1 & 5 & 0 & 12 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Seules les colonnes MA^1 et MA^4 sont nulles donc x^1 et x^4 sont compatibles avec M tandis que x^2, x^3, x^5 et x^6 sont incompatibles. □

Proposition 3.1.1 *Les variables positives de x_B sont compatibles avec C_N et les variables nulles de x_B sont incompatibles avec C_N.*

Preuve : Pour $i = 1, \ldots, m$, notons l_i l'indice de la colonne de A associée à la variable de base x_B^i. On a $CA^{l_i} = B^{-1}A^{l_i} = e_i$, $i = 1, \ldots, m$, avec e_i le $i^{ème}$ vecteur de la base canonique de $\mathbb{R}^m$. Or, $e_i^i = 1$ et $e_i^j = 0$ pour $j \neq i$, donc $C_N A^{l_i} \neq 0$ si $i \in I_N$ et $C_N A^{l_i} = 0$ si $i \in I_P$. Il vient que x_B^i est incompatible avec C_N si $x_B^i = 0$ et que x_B^i est compatible avec C_N si $x_B^i > 0$. □

En appliquant la définition 3.1.1, on détermine x_C, le vecteur des variables de PL compatibles avec C_N et x_I, le vecteur des variables incompatibles avec C_N. Le vecteur x_C contient toutes les variables en base non-dégénérées ainsi que certaines variables

hors-base et le vecteur x_I contient toutes les variables en base dégénérées ainsi que certaines variables hors-base. On note A_C et A_I les colonnes de A associées aux variables de x_I et de x_C, respectivement. Par construction de C_N, on a $C_N b = 0$ et, par compatibilité des variables de x_C, $C_N A_C = 0$. Par suite, le problème modifié PL_C s'écrit

$$(\mathrm{PL}_C) \equiv \begin{cases} \min\limits_{x} & c_C^T x_C + c_I^T x_I \\ s.c. & C_P A_C x_C + C_P A_I x_I = C_P b \\ & C_N A_I x_I = 0 \\ & x_C \geq 0,\ x_I \geq 0, \end{cases}$$

où c_I et c_C sont les coûts de x_I et x_C, respectivement. À partir du problème modifié PL_C, formons le problème réduit PL_R en posant $x_I = 0$ dans PL_C :

$$(\mathrm{PL}_R) \equiv \begin{cases} \min\limits_{x} & c_C^T x_C \\ s.c & C_P A_C x_C = C_P b \\ & x_C \geq 0. \end{cases}$$

Le problème PL_R est obtenu à partir d'une solution réalisable pour PL_C dont toutes les variables prenant des valeurs positives sont conservées dans PL_R. En donnant aux variables x_C les valeurs qu'elles avaient dans cette solution réalisable de PL_C, on obtient une solution réalisable de PL_R d'où PL_R est réalisable.

La réduction dynamique se fait en deux étapes. On commence par résoudre à l'optimalité le problème réduit PL_R, ce qui améliore l'objectif en conservant la réalisabilité de PL_C. La résolution permet le calcul d'une solution duale, d'une solution primale et d'un coût réduit de toutes les variables du problème initial. On ajuste ensuite les matrices de réduction et de compatibilité afin de réintégrer des variables de coût réduit négatif et au besoin de supprimer de nouvelles variables dégénérées, tout en assurant la conservation de la réalisabilité. Puis on résout le nouveau problème réduit qui donne une nouvelle solution primale et un nouveau coût réduit. L'algorithme se termine lorsque le coût réduit de chacune des variables est positif ou nul.

3.2 Description de l'algorithme

Le pseudo-code de l'algorithme de réduction dynamique de contraintes est donné dans l'algorithme 3.1 et il est commenté dans les paragraphes suivants. On commence par introduire quelques notations utiles :

- *augmentation*(C_N, C_P, I) met les matrices de réduction et de compatibilité à jour afin que les variables de l'ensemble I deviennent compatibles. L'ensemble I est formé de variables incompatibles de coût réduit strictement négatif.
- *dégénéré*(x_C) est un prédicat qui détermine si une solution réalisable x_C d'un problème réduit PL_R est suffisamment dégénérée pour qu'il soit intéressant de réduire à nouveau le problème.
- *réduction*(C_N, C_P) est une fonction qui construit les nouvelles matrices de réduction et de compatibilité pour que les variables dégénérées du problème réduit deviennent incompatibles.
- *optimal* est une constante initialisée à 0 qui prend la valeur 1 lorsqu'une solution optimale du programme linéaire est trouvée.
- Z_0 représente la valeur de l'objectif à la résolution précédente et est initialisée à ∞.

La première étape de l'algorithme 3.1 consiste à trouver une base réalisable de PL. Durant les tests numériques, deux méthodes sont utilisées pour déterminer une telle base. Elle peut être trouvée en effectuant une phase 1 de l'algorithme du simplexe primal. La deuxième option consiste à résoudre à l'optimalité un programme linéaire pour en tirer la base optimale puis à perturber sa fonction de coût pour ne plus être à l'optimalité. Cette deuxième option correspond à réoptimiser un programme linéaire suite à des perturbations du contexte.

Les étapes 1 à 5 décrivent la réduction initiale du problème et ont été explicitées dans la section 3.1. Les étapes 7 à 16 forment une itération majeure de notre algorithme. Leur description n'a été que survolée et nécessite des éclaircissements.

Algorithme 3.1 Algorithme de base

1: Déterminer une base réalisable B et une solution de base x_B.
2: $C = B^{-1}$
3: $Z_0 = \infty$, *optimal* $= 0$
4: Déduire C_N et C_P selon la dégénérescence de x_B.
5: Construire le problème réduit PL_R à partir de C_N et C_P.
6: **tant que** *optimal* $= 0$ **effectuer**
7: Résoudre PL_R pour obtenir une solution primale x_C^* de coût Z et une solution duale π_P^*.
8: Calculer $\bar{c}$, le coût réduit des variables de PL_C.
9: **si** $\bar{c} \geq 0$ **alors**
10: *optimal* $= 1$, $x = \begin{pmatrix} x_C^* \\ 0 \end{pmatrix}$ est optimal.
11: **si** $Z < Z_0$ et *dégénéré*(x_C) **alors**
12: $(C_N, C_P) =$ *réduction*(C_N, C_P)
13: Soit I, un ensemble non-vide de variables de coût réduit strictement négatif incompatibles avec C_N.
14: $(C_N, C_P) =$ *augmentation*(C_N, C_P, I)
15: Construire le problème réduit PL_R.
16: $Z_0 = Z$

La résolution de PL_R faite à l'étape 7 se fait par un appel au programme d'optimisation CPLEX. L'optimisation produit une solution primale x_C^* et une solution duale π_P^* de PL_R. Pour calculer le coût réduit de toutes les variables de PL_C à l'étape 8, la solution π_P^* doit être complétée en une solution duale π de PL_C. Nous discuterons la méthode de complétion du vecteur des variables duales dans la section 3.2.1.

À l'étape 12, on change les matrices C_N et C_P pour réduire le problème en éliminant la dégénérescence apparue dans la dernière résolution, lorsque l'objectif a diminué lors de la dernière optimisation et que le prédicat *dégénéré*(x_C) a décidé que la dégénérescence était trop importante pour la bonne progression de la procédure. Nous verrons un exemple de prédicat *dégénéré*(x_C) dans la section 3.2.2.

Les étapes 13 et 14 consistent en la mise à jour des matrices de compatibilité et de réduction afin de rendre compatibles certaines variables incompatibles de coût

réduit négatif. Pour ce faire, on commence par construire un ensemble I de variables de coût réduit négatif. Le choix des variables de I est décrit dans la section 3.2.2. On intègre ensuite à l'ensemble I_P les indices des lignes de CA sur les quelles les coefficients des colonnes associées aux variables de I ne sont pas toutes nulles. Par cette procédure, on ajoute au problème réduit des variables susceptibles de réduire la valeur de l'objectif lors de la résolution de PL_R dans la prochaine itération majeure de notre algorithme.

Finalement, le problème réduit PL_R est construit à l'étape 15 de la même manière que dans la section 3.1.

3.2.1 Calcul du coût réduit complet

Après la résolution de PL_R, on obtient une solution duale π_P^*. Or, pour valider l'optimalité de cette solution par rapport au problème complet, il faut vérifier que tous les coûts réduits sont positifs ou nuls. Ceci exige de connaître une solution duale π complète. Nous posons les variables duales associées aux contraintes de $C_P A$ égales à la solution duale π_P^*. Notons par π_N le vecteur des variables duales associées aux contraintes de $C_N A$. À partir de la solution duale complète, le coût réduit s'obtient par

$$\bar{c}^T = c^T - ((\pi_P^*)^T C_P A + \pi_N^T C_N A).$$

La solution duale complète π aura du sens si et seulement si son coût Z_D est égal au coût Z de la solution primale x. Z_D se calcule par :

$$Z_D = (C_P b)^T \pi_P^* + (C_N b)^T \pi_N.$$

Puisque $C_N b = 0$, le coût de π est le même que celui de π_P^* dans PL_R. La solution duale π_P^* étant optimale pour PL_R, on a $Z_D = Z$ quelque soit π_N. Toute manière

de compléter π nous mène au calcul d'un coût réduit des variables de PL_C qui a du sens. Il nous appartient donc de choisir le vecteur π_N qui aboutira au calcul du coût réduit qui nous arrangera le plus.

Dans l'article de Pan (1998), le calcul du coût réduit revient à fixer π_N à 0. Nous souhaitons améliorer la complétion du vecteur π en tenant compte des considérations suivantes. Une solution de base de PL_C peut toujours être formée en prenant comme variables en base l'ensemble des variables en base de la solution courante de PL_R et l'ensemble des variables supprimées de PL_C qui étaient en base juste avant d'être retirées du problème (variables incompatibles en base). Si des variables incompatibles en base devaient être réintégrées au problème réduit, plutôt que d'utiliser des variables artificielles pour les contraintes rajoutées, on pourrait rentrer ces variables directement en base. Leur coût réduit doit donc être nul. Comme nous l'avons vu dans la preuve de la proposition 3.1.1, la colonne de CA correspondant à une variable de base est un des vecteurs de la base canonique de $\mathbb{R}^m$. Pour chaque variable de base incompatible x^j, on a $CA^j = e^{i_j}$, $i_j \in \{1, \ldots, m\}$. Le coût réduit de x_j vaut donc

$$\bar{c}^j = c^j - \pi^{i_j}.$$

La variable x^j étant incompatible, la contrainte d'indice i_j fait partie des contraintes de $C_N A$. Donc la variable duale π^{i_j} fait partie des variables de π_N. Ainsi, la condition de nullité de $\bar{c}^j$ impose $\pi^{i_j} = c^j$. En outre, le processus de réduction implique qu'il y a toujours autant de variables incompatibles en base que de contraintes supprimées du problème. Le vecteur π_N est donc complètement déterminé par la condition d'annulation des coûts réduits des variables incompatibles en base. Si on pose c_{B_N} le coût de ces variables et que l'on réordonne convenablement les contraintes, on obtient

$$\pi = \begin{pmatrix} \pi_P^* \\ c_{B_N} \end{pmatrix}.$$

3.2.2 La réintégration de contraintes

Lorsqu'en fin d'optimisation de PL_R, le coût réduit, calculé à l'aide des variables duales $\pi = \begin{pmatrix} \pi_P^* \\ \pi_N \end{pmatrix}$, de certaines variables incompatibles est négatif, la solution peut ne pas être optimale. L'optimisation du problème force à réintégrer des variables précédemment incompatibles.

Comme les variables de coût réduit négatif peuvent faire diminuer la valeur de l'objectif, on décide de faire revenir dans le problème réduit l'ensemble I contenant un certains nombres de variables de coûts réduits les plus négatifs. Le nombre de variables de I est choisi en fonction de la taille du problème original pour avoir toutes les chances de faire rentrer les variables qui feront diminuer l'objectif de manière significative sans trop augmenter la taille du problème. L'appel à la fonction *augmentation* sert à réintégrer ces variables. On commence par déterminer les contraintes qui seront réintégrées au problème. Ces contraintes sont choisies de façon à ce que les variables de I soient compatibles avec C_N en fin d'augmentation. Pour cela, les deux matrices C_N et C_P sont mises à jour. Une variable x^j est compatible avec C_N si $C_N A^j = 0$. Pour chaque variable x^j de I, le vecteur $C_N A^j$ est donc parcouru et chaque ligne de C_N d'indice i telle que $(C_N A^j)^i \neq 0$ est retirée de C_N et ajoutée à la matrice C_P. Ainsi, en fin de mise à jour, toutes les variables de I sont compatibles avec C_N.

Le problème réduit que l'on formera à partir de C_N et C_P comportera plus de contraintes et de variables. Les variables compatibles avec C_N sont maintenant les variables compatibles avant l'appel de la fonction *augmentation*, les variables de I et toutes les variables dont l'incompatibilité résultait de lignes de C_N qui sont maintenant dans C_P.

3.2.3 Réapparition de dégénérescence lors de la résolution

Il arrive que la résolution de PL_R fasse réapparaître de la dégénérescence dans la solution, et on s'y attend même avec une forte probabilité après que des variables et contraintes ont été réintégrées au problème. Le rôle de la fonction *réduction* est alors de changer les matrices C_P et C_N de sorte que les variables compatibles dégénérées deviennent incompatibles avec C_N. Cette fonction n'est appelée que dans le cas où la valeur de l'objectif a diminué depuis la dernière réduction de PL_R (i.e. $Z < Z_0$) car, sans cette condition, l'algorithme peut boucler en ajoutant et éliminant toujours les mêmes variables et contraintes sans changer la valeur de l'objectif. Lorsque $Z = Z_0$, l'algorithme effectue une succession de réintégrations de variables et contraintes et de résolutions de PL_R. Le prédicat *dégénéré* décide ensuite si la réduction du problème est utile. L'appel de la fonction *réduction* que nous décrivons dans la suite a un coût en temps non négligeable, on ne peut pas se permettre de faire appel à cette fonction dès qu'une variable dégénérée apparaît. Le prédicat reviendra en général à faire le rapport entre le nombre de variables dégénérées et le nombre de contraintes de PL_R puis à le comparer à un ratio λ prédéfini. Nous prendrons en général λ entre 15% et 20%.

Si le prédicat *dégénéré* décide que l'on doit réduire PL_R, la réduction se fait de la même manière que celle de PL décrite dans la section 3.1. On obtient alors une matrice $C_R = B_R^{-1}$, une matrice de compatibilité C_{R_N} et une matrice de réduction C_{R_P} avec B_R la matrice de base à l'optimalité de PL_R. La matrice de contraintes devient donc $C_{R_P}C_PA$ et les variables compatibles sont maintenant celles qui sont compatibles avec C_N et avec $C_{R_N}C_P$. Finalement, les matrices du problème initial sont mises à jour par

$$C_P = C_{R_P}C_P \quad et \quad C_N = \begin{pmatrix} C_{R_N}C_P \\ C_N \end{pmatrix}.$$

On remarque que la fonction *réduction* est appelée avant la fonction *augmentation*. Cet ordre a été choisi afin d'éviter de réintégrer des variables qui risquent d'être immédiatement supprimées par l'appel à la fonction *réduction*.

3.3 Convergence

Dans cette section, nous prouvons une proposition sur la convergence de l'algorithme de base après un nombre fini de pivots de l'algorihme du simplexe. Les manipulations du problème réduit consistent en un nombre fini d'assignations et de résolutions de systèmes d'équations linéaires considérés rapides par rapport aux optimisations de PL_R. Nous nous limitons donc à étudier les pivots du simplexe faits au cours de notre algorithme.

Proposition 3.3.1 *Soit PL un programme linéaire sous forme standard réalisable. En supposant que l'algorithme du simplexe résolvant PL_R est muni d'un mécanisme d'anti-cyclage, l'application de l'algorithme 3.1 trouve une solution primale optimale x^* de PL ou prouve que PL est non-borné en un nombre fini de pivots du simplexe.*

Preuve : Si PL est réalisable, il est possible de trouver une solution de base réalisable au sens primal, puis de faire la réduction initiale de PL. Si on le munit d'un mécanisme d'anti-cyclage, l'algorithme primal du simplexe appelé à l'étape 7 converge en un nombre fini de pivots vers la solution optimale de PL_R. Soit x_C^* la solution optimale de PL_R et $\bar{c}$ le coût réduit des variables de PL_C. Deux cas peuvent alors se présenter.

1. Si $\bar{c} \geq 0$, la discussion faite dans la section 3.2.1 nous assure que la solution $x^* = \begin{pmatrix} x_C^* \\ 0 \end{pmatrix}$ est optimale pour PL_C et donc pour son problème équivalent PL.
2. Sinon ($\bar{c} \ngeq 0$), une augmentation de la taille du problème sera faite, précédée au besoin d'une réduction.

Nous allons montrer qu'après un nombre fini d'itérations majeures menant au cas 2, on trouvera $\bar{c} \geq 0$.

À chaque appel de la fonction *augmentation*, un nombre positif de variables est réintégré. Si il n'y a aucune réduction et que le cas 1 n'est pas rencontré, après un nombre fini d'augmentations, nous aurons $\mathrm{PL}_R = \mathrm{PL}$. La résolution de PL_R nous amènera alors dans le cas 1.

Ensuite, tous les appels à la fonction *réduction* correspondent à des solutions de base de PL différentes puisque l'on impose la stricte décroissance de la valeur de l'objectif entre deux réductions par la condition $Z < Z_0$. Le nombre de solutions de base d'un programme linéaire est fini. Il en va donc de même pour le nombre de réductions effectuées par l'algorithme. Notons r, le nombre maximal de réductions consécutives possibles. Deux réductions consécutives sont séparées au pire des cas par $m - 2$ augmentations car une contrainte au moins est réintégrée à chaque augmentation. Il faudra au plus $(m-2) * r$ itérations majeures, avant de se retrouver dans la situation où plus aucune réduction n'est possible. À partir de ce point, il est certain que l'algorithme trouve la solution optimale en un nombre fini d'itérations majeures si le programme linéaire en possède une. On remarque également que si jamais le problème est non-borné, l'algorithme le détectera en un nombre fini d'itérations majeures.

Finalement, l'optimisation de chaque itération majeure se fait en un nombre fini de pivots du simplexe et l'optimalité de PL est atteinte après un nombre fini d'itérations majeures. D'où la convergence de l'algorithme. □

3.4 Expérimentation avec l'algorithme de base

Dans cette section, nous décrivons les expériences exécutées avec l'algorithme décrit afin d'évaluer son efficacité. Nous présentons d'abord les caractéristiques de l'im-

plantation (sous-section 3.4.1) puis la classe des problèmes résolus par l'algorithme (sous-section 3.4.2), avant d'exposer les résultats de l'expérimentation (sous-section 3.4.3).

3.4.1 Caractéristiques de l'implantation

L'algorithme a été implanté en utilisant le langage de programmation C. Nous nous sommes beaucoup servis de la bibliothèque "Callable Library" de la version 9.1 du logiciel commercial CPLEX. La solution de base a été déterminée par une phase 1 de l'algorithme du simplexe. Cette même bibliothèque a été appelée pour effectuer les opérations suivantes :

- La résolution de la phase 1 de l'algorithme du simplexe.
- Les constructions des matrices de réduction et de compatibilité grâce à une fonction de résolution du système $Bx = a^j$.
- Les résolutions du PL réduit ainsi que toutes les saisies d'informations sur les solutions.
- Les modifications du PL réduit.

La taille de l'ensemble I des variables de coût réduit négatif a été fixée à 5% du nombre de variables mises à l'écart lors de la première réduction, et nous avons pris $\lambda = 20\%$.

3.4.2 Description des problèmes résolus

Le but de l'algorithme étant de tirer avantage de la dégénérescence des programmes que l'on résout, nous avons choisi de faire nos tests sur des problèmes très dégénérés. Comme les résultats de Elhallaoui *et al.* (2005) le laissent supposer, l'agrégation de contraintes se prête bien à la résolution des relaxations de problème de construction

d'horaires de chauffeurs d'autobus. Nous avons testé notre algorithme sur des problèmes maîtres obtenus en appliquant une méthode de génération de colonnes sur ces problèmes que nous noterons VCSP dans la suite pour "Vehicle and Crew Scheduling Problem". La définition du VCSP est présentée dans Haase *et al.* (2001) et nous en donnerons ici les grandes lignes.

Le VCSP consiste à déterminer simultanément les horaires de autobus et de chauffeurs accomplissant pour un coût minimum un ensemble de voyages limités dans le temps qui parcourent un ensemble de lignes de autobus, tout en respectant un ensemble de contraintes imposées par des conventions collectives et des règlements internes. Les horaires d'autobus consistent en une alternance de voyages avec limites de temps et de voyages à vide servant à repositionner les autobus. Tous les autobus sont supposés identiques et basés dans un seul dépôt. Les horaires des chauffeurs, appelées tournées, sont compliquées puisque des changements de chauffeurs peuvent avoir lieu au cours d'un voyage limité dans le temps. Dans nos problèmes de test, les tournées de chauffeurs contiendront des voyages sans coupure ou avec une seule coupure. Il est assez évident que des segments consécutifs dans un voyage d'autobus seront le plus souvent consécutifs dans la solution optimale des tournées de chauffeurs. Cette constatation a amené à considérer l'agrégation de contraintes. L'horaire des autobus est construit a posteriori, une fois l'horaire optimal des chauffeurs construit.

Les variables du problème maître représentent le nombre de chauffeurs, le nombre d'autobus et un ensemble de chemins de chauffeurs réalisables générés par des sous-problèmes. Les contraintes du problème se divisent en trois ensembles.

- Une majorité de contraintes de partitionnement d'ensemble imposent que les autobus arrivent au début des voyages, partent du terminus des voyages et que chaque segment de voyage d'autobus soit couvert une fois exactement par un chauffeur.
- D'autres contraintes assurent l'optimalité de l'horaire des autobus dérivé de la solution optimale des tournées de chauffeurs.

- Une contrainte qui dénombre les chauffeurs nécessaires à la construction de l'horaire.

3.4.3 Présentation des résultats

Les tests ont été réalisés sur quatre problèmes maîtres générés au cours d'une même résolution d'un VCSP. Nous nous sommes arrêtés là dans nos tests cause du temps de calcul relativement élevé pour la résolution par l'algorithme 3.1. Toutes les résolutions présentées dans le mémoire ont été effectuées sur une machine utilisant un processeur Intel Pentium 4 à 1.8GHz.

Les caractéristiques des problèmes ainsi que les durées et le nombre d'itérations de la résolution par le logiciel CPLEX sont regroupés dans le tableau 3.1. Nous recensons dans le tableau 3.2 les résultats de notre implémentation de l'algorithme de base sur ces 4 problèmes. Définissons ci-dessous des notations utilisées dans les tableaux de cette section ainsi que dans la toute la suite du mémoire. Les temps de calcul correspondent à des temps CPU et sont en secondes.

- t_{tot} : temps de calcul total pour l'instance résolue
- i_{tot} : nombre total d'itérations de l'algorithme du simplexe pour la résolution du problème
- τ_{opt} : taux de dégénérescence de la solution optimale que l'on définit comme le rapport entre le nombre de variables de base nulles et le nombre de contraintes m
- t_{p2} : temps de calcul comsommé par les itérations de l'algorithme du simplexe lors de la phase 2
- i_{p2} : nombre d'itérations de l'algorithme du simplexe pendant la phase 2 de l'algorithme du simplexe
- t_{plr} : temps de calcul total consommé par les itérations de l'algorithme du simplexe

Tableau 3.1 – Caractéristiques des VCSP tests et résolution directe par CPLEX

Nom	m	n	t_{tot}	i_{tot}	$\tau_{opt}(\%)$	t_{p2}	i_{p2}
VCSP1	2084	10343	617	96428	37	422	71128
VCSP2	2084	7337	368	65208	38	260	48563
VCSP3	2084	8795	492	78520	37	339	58122
VCSP4	2084	10150	617	89401	37	447	67880

Tableau 3.2 – Résultats de l'implémentation de l'algorithme 3.1

Nom	t_{tot}	t_{plr}	i_{tot}	i_{maj}	n_{red}	$Perf(\%)$
VCSP1	4792	4634	137239	9	2	-998
VCSP2	2540	2454	104465	10	2	-843
VCSP3	4474	4372	139287	9	2	-1190
VCSP4	6080	5927	141401	11	2	-1226

- *Perf* : performance de l'algorithme testé quantifiée par l'écart relatif entre le temps de la résolution directe par CPLEX et le temps de calcul consommé par CPLEX pendant l'exécution de l'algorithme 3.1. Son expression est donnée par $Perf = \frac{t_{plr}-t_{p2}}{t_{p2}}$.
- n_{red} : nombre de réductions du problème, incluant la réduction initiale
- i_{maj} : nombre d'itérations majeures de l'algorithme 3.1

Puisque nous partons de la solution trouvée en fin de phase 1, les résultats du tableau 3.2 sont à comparer avec les performances de CPLEX pour la phase 2. À la première lecture, on remarque que le nombre d'itérations a été multiplié par 2 alors que les durées des itérations d'algorithme du simplexe ont été multipliées par des valeurs proches de 10. Cette constatation va à l'opposé de ce qu'on espérait. Le problème a bien été réduit et sa dégénérescence a diminué mais le nombre d'itérations et la durée de chaque itération ont tous deux augmenté. Cette lenteur s'explique par le changement de la matrice de contraintes. La multiplication par l'inverse de la base augmente

significativement la densité de la matrice, ce qui ralentit fortement les pivots des itérations de l'algorithme du simplexe. L'augmentation du nombre d'itérations vient du fait que la réduction à partir d'une solution réalisable de mauvaise qualité n'a pas forcément de sens. Elle mène à faire des pivots pendant l'optimisation de PL_R qui n'auraient jamais dû avoir lieu. Hélas, une réduction significative demande d'exploiter la structure du problème à résoudre et nous nous intéressons ici au cas général.

La quasi-totalité du temps de calcul qui n'est pas consommé par CPLEX est consommé par les opérations de réduction du PL. Si on observe alors la durée des réductions du problème, celle-ci paraît ridicule comparé au temps d'exécution de l'algorithme 3.1. En revanche elle représentent plus d'un quart du temps résolution de la phase 2. En fait, nous avons remarqué que CPLEX était peu performant pour la résolution du système $Bx = A^j$. Les réductions postérieures à la réduction initiale sont donc très coûteuses en temps de calcul. Pourtant, ces dernières sont indispensables, la réduction effectuée a fait gagner près de 50% de temps de calcul pour les problèmes testés. Par contre, plus de réductions n'amélioraient pas l'efficacité.

3.4.4 Perspectives d'améliorations

La suite de notre exposé traitera d'un algorithme fortement inspiré de celui que nous venons de présenter et qui tire les enseignements de ses résultats décevants et surtout des causes de ce manque de rapidité. Le second algorithme n'augmentera plus la densité de la matrice de contraintes et les réductions seront faites de sorte à ne pas avoir à faire appel aux fonctions de CPLEX pour calculer la matrice de compatibilité. En outre, la matrice de compatibilité ne sera plus liée à une base entière de manière aussi forte que par B^{-1}. Nous voulons ainsi donner moins d'importance à un ensemble de variables qui n'en aura pas forcément dans la solution optimale. Par exemple, nous ne considèrerons plus du tout les variables de bases nulles pour la construction de la

matrice de compatibilité, car elles sont inutiles à la définition du sommet sur lequel on se trouve et apportent donc moins d'information sur la situation actuelle de la résolution.

CHAPITRE 4 : AMÉLIORATION DE L'ALGORITHME

En poursuivant un schéma similaire à l'algorithme présenté dans le chapitre 3, nous allons décrire une seconde méthode de résolution par réduction dynamique de contraintes. Les principaux défauts du dernier algorithme est qu'il augmentait la densité des programmes linéaires résolus et que le calcul de la matrice de compatibilité était trop longue à recalculer en cas de réductions en cours d'optimisation. Afin de les corriger, la méthode développée dans la suite de ce chapitre s'appuiera sur une matrice de compatibilité plus simple et les réductions ne modifieront plus les coefficients de la matrice des contraintes. Des lignes et des colonnes du problème initial seront directement retirées pour obtenir les problèmes réduits.

La section 4.1 est consacrée à la théorie de la méthode améliorée de réduction dynamique de contraintes. Nous présentons l'algorithme amélioré et en expliquons le détail dans la section 4.2 et nous prouvons sa convergence dans la section 4.3. Puis, dans la section 4.4, nous discutons des conséquences des changements apportés à l'algorithme.

4.1 Aspects théoriques

Repartons du problème initial PL :

$$(PL) \equiv \begin{cases} \min\limits_{x} & z = c^T x \\ s.c & Ax = b \\ & x \geq 0, \end{cases}$$

Les conditions de départ de l'algorithme se différencient légèrement du chapitre précédent par le fait que nous ne supposerons pas être en possession d'une solution

réalisable de PL et de la base qui lui est associée mais seulement d'une solution réalisable $\hat{x}$. À partir de cette solution, nous construisons les deux ensembles d'indices I_N et I_P pour lesquels $\hat{x}$ est nul ou positif, respectivement, ou encore,

$$I_N = \{i|\hat{x}^i = 0\} \quad et \quad I_P = \{i|\hat{x}^i > 0\}.$$

Soit p le nombre de variables prenant une valeur positive dans la solution $\hat{x}$. Nous nous plaçons dans le cas de la résolution de problèmes dégénérés, soit $p < m$. Dans nos tests numériques, les solutions de départ sont déterminées par une phase 1 de l'algorithme du simplexe ou par la solution du problème PL dont les coûts ont été perturbés au préalable. Nous ne connaissons pas de base réalisable de PL. Par contre nous savons que les variables à valeurs positives sont des variables de base. On note B_P la matrice des colonnes de A dont les indices sont dans I_P. La matrice B_P est un sous-ensemble de colonnes d'une matrice de base, d'où B_P est de plein rang. Si on effectue une triangularisation de Gauss sur B_P, $m - p$ lignes seront donc éliminées. Afin de simplifier les notations, on suppose que les $m - p$ lignes éliminées sont les dernières lignes de B_P . Soit L la matrice inversible dont la multiplication par B_P revient à faire ces pivots de Gauss, c'est-à-dire que

$$LB_P = \begin{pmatrix} U_P \\ 0 \end{pmatrix}.$$

Soit L_N l'ensemble des $m-p$ dernières lignes de L. L_N est la matrice de compatibilité grâce à laquelle le problème réduit va être construit. On détermine alors x_C, le vecteur des variables compatibles avec L_N, et x_I le vecteur des variables incompatibles avec L_N. On introduit également les sous-matrices de A A_{PC}, A_{PI}, A_{NC}, et A_{NI}, et les vecteurs c_I, c_C, b_P et b_N où les indices P et N se rapportent aux coefficients des p premières et des $m - p$ dernières contraintes, respectivement, et les indices C et I se rapportent aux coefficients des variables compatibles et incompatibles, respectivement. Avec ces notations le problème initial se reformule

$$\begin{array}{ll} \min\limits_{x} & c_C^T x_C + c_I^T x_I \\ \text{s.c.} & A_{PC}x_C + A_{PI}x_I = b_P \\ & A_{NC}x_C + A_{NI}x_I = b_N \\ & x_C \geq 0,\ x_I \geq 0. \end{array}$$

Le problème réduit PL_R s'obtient en retirant du problème toutes les lignes supprimées par les pivots de Gauss, ainsi que les variables incompatibles avec L_N. PL_R s'écrit

$$(\text{PL}_R) \equiv \left\{ \begin{array}{ll} \min\limits_{x} & c_c^T x_C \\ s.c & A_{PC}x_C = b_P \\ & x_C \geq 0. \end{array} \right.$$

Exemple : Illustrons la réduction d'un problème en considérant le programme linéaire suivant :

$$\begin{array}{ll} \min\limits_{x} & x^1 + 2x^2 + x^3 + 5x^4 \\ s.c. & 2x^1 + 4x^4 = 4 \\ & x^1 - x^2 + 3x^3 + 2x^4 = 2 \\ & x \geq 0. \end{array}$$

Nous partons de la solution primale réalisable $\hat{x} = (0, 0, 0, 1)^T$. Cette solution est dégénérée car une seule variable est positive alors que le problème a deux contraintes. La matrice des contraintes A et la base partielle B_P vallent

$$A = \begin{pmatrix} 2 & 0 & 0 & 4 \\ 1 & -1 & 3 & 2 \end{pmatrix}, \quad B_P = \begin{pmatrix} 4 \\ 2 \end{pmatrix}.$$

Notons B_{P1} la première ligne de B_P et B_{P2} sa deuxième. Nous éliminons la deuxième ligne de B_P par l'opération sur les lignes $B_{P2} \leftarrow B_{P2} - \frac{1}{2}B_{P1}$. On en déduit

$$L = \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{pmatrix},$$

puis on calcule la matrice de compatibilité :

$$L_N A = \begin{pmatrix} 0 & -1 & 3 & 0 \end{pmatrix}.$$

Les variables x^1 et x^4 sont les seules variables compatibles avec L_N. Il nous reste à enlever la deuxième contrainte et les deux variables incompatibles pour obtenir PL_R ci-dessous.

$$\begin{array}{lll} \min\limits_{x} & x^1 + 5x^4 & \\ s.c. & 2x^1 + 4x^4 & = 4 \\ & x \geq 0 & . \end{array}$$

□

Proposition 4.1.1 *Le problème PL_R obtenu en suivant la procédure décrite ci-dessus est réalisable et toute solution réalisable $\tilde{x}_C$ de PL_R donne une solution réalisable $\tilde{x} = \begin{pmatrix} \tilde{x}_C \\ 0 \end{pmatrix}$ de PL .*

Preuve : Par construction de L, $L_N B_P = 0$. Donc toutes les variables positives de la solution réalisable de départ $\hat{x}$ sont compatibles avec L. Ceci implique que $\hat{x}_I = 0$ et par suite $A_{PC}\hat{x}_C = b_P$. $\hat{x}_C$ est un solution réalisable de PL_R donc PL_R est réalisable.

Soit, ensuite, PL_{RV} le problème réduit en variables uniquement :

$$\begin{array}{lll} min & c_C^T x_C & (4.1) \\ s.c. & A_{PC}x_C = b_P & (4.2) \\ & A_{NC}x_C = b_N & (4.3) \\ & x_C \geq 0. & (4.4) \end{array}$$

Il est rapide de vérifier que toute solution réalisable $\tilde{x}_C$ de PL_{RV} donne une solution réalisable $\tilde{x}$ de PL par $\tilde{x} = \begin{pmatrix} \tilde{x}_C \\ 0 \end{pmatrix}$. Pour prouver que ce résultat est aussi vrai pour PL_R, nous montrons que PL_R et PL_{RV} sont équivalents, i.e., que les contraintes (4.3) sont redondantes. Notons A_C, la sous-matrice de A dont les colonnes correspondent aux variables compatibles x_C. On a en fait

$$A_C = \begin{pmatrix} A_{PC} \\ A_{NC} \end{pmatrix}.$$

Par construction de A_C, $L_N A_C = 0$, puis comme $\hat{x}_C$ est réalisable $L_N A_C \hat{x}_C = L_N b$ donc $L_N b = 0$. Par conséquent, les contraintes $L_N A_C x_C = L_N b$ s'écrivent $0 = 0$. En outre, par construction de la matrice L_N, la $i^{ème}$ contrainte de $L_N A_C \hat{x}_C = L_N b$ est une combinaison linéaire de la $i^{ème}$ contrainte de $A_{NC} x_C = b_N$ et des contraintes (4.2). Donc les contraintes (4.3) sont combinaisons linéaires des contraintes (4.2). Il vient que PL_R et PL_{RV} sont équivalents. □

Une fois le problème réduit déterminé, nous suivrons une procédure analogue à celle décrite dans le chapitre précédent. PL_R est d'abord résolu à l'optimalité. On en tire une solution primale réalisable pour PL et une solution duale partielle. Grâce à la solution duale partielle, on calcule le coût réduit de toutes les variables incompatibles. On réintègre alors au problème réduit les variables de coûts réduits les plus négatifs. On ajoute également les lignes dans lesquelles la matrice de compatibilité $L_N A$ a des coefficients non-nuls pour ces variables afin de toujours rester réalisable pour le problème initial. On résout à nouveau PL_R et on le réduit si sa solution primale est trop dégénérée. L'algorithme se termine lorsque le coût réduit de toutes les variables est positif ou nul.

4.2 Description de l'algorithme

Le pseudo-code de l'algorithme de réduction dynamique de contraintes est donné dans l'algorithme 4.1 et il est commenté dans les paragraphes suivants. Nous conservons le sens général donné aux fonctions *augmentation* et *réduction*, au prédicat *dégénéré* et aux variable Z_0 et *optimal* définis dans la section 3.2. Cependant le détail des deux fonctions et du prédicat change, ils seront explicités à la suite de l'agorithme. Nous introduisons, en outre, quelques nouvelles notations :

- *degen1* est un prédicat dont le rôle est de décider si la solution initiale est suffisamment dégénérée pour mériter d'être réduite.

- *degen2* est un prédicat qui détermine si la dégénérescence de la solution actuelle de l'algorithme du simplexe justifie que l'on arrête la résolution pour réduire le problème avant l'optimalité.
- *lent* intervient au cours de l'optimisation de PL_R pour décider si la résolution est trop lente et doit être interrompue pour réduire ou agrandir le problème.
- $optimal_R$ est une variable qui prend la valeur 1 quand la dernière solution calculée par l'algorithme du simplexe est optimale pour PL_R et 0 sinon. $optimal_R$ est initialisée à 0.

La solution primale initiale $\hat{x}_P$ provient de la phase 1 de l'algorithme du simplexe ou de la résolution préalable du problème PL dont on a perturbé les coûts. Les étapes 2 à 8 initialisent le problème réduit. Les étapes 10 à 26 forment une itération majeure de l'algorithme. Elle se compose d'un certain nombre d'itérations du simplexe, et d'une augmentation ou une réduction de la taille du problème. Nous clarifions dans la suite les points délicats de l'algorithme.

À l'étape 2, on applique le prédicat *degen1* car il est arrivé au cours de certains tests de remarquer que la dégénérescence n'apparaît pas toujours dès la fin de la résolution de la phase 1. Dans ce cas, réduire le problème alors que la solution de départ est très peu dégénérée résulte en une perte de temps importante, non seulement parce que le problème réduit ne sera pas plus rapide à resoudre que le problème initial mais aussi parce que l'obtention de la matrice de compatibilité $L_N A$ sera bien plus longue à obtenir pour des raisons que nous verrons dans le chapitre 5.

Les étapes 3 à 5 sont décrites dans la section 4.1 et décrivent la première réduction du problème dans le cas où la solution initiale est suffisamment dégénérée. Des détails sur les moyens utilisés pour l'élimination des lignes et le choix des lignes à éliminer seront donnés dans le chapitre 5.

Par rapport à l'algorithme de base, PL_R n'est pas directement résolu à l'optimalité. Dans les étapes 12 à 18, l'algorithme 4.1 lance la résolution par l'algorithme du

Algorithme 4.1 Algorithme amélioré

1: Déterminer une solution primale réalisable $\hat{x}$ de PL. Soit p le nombre de variables positives de $\hat{x}$.
2: **si** $degen1(\hat{x})$ **alors**
3: Construire la base partielle B_P.
4: Eliminer $m-p$ lignes de B_P et en déduire la matrice L.
5: Retirer de A les $m-p$ lignes éliminées en appliquant la multiplication par L ainsi que toutes les variables incompatibles avec L_N pour obtenir PL_R.
6: **sinon**
7: PL_R =PL.
8: $Z_0 = \infty$, $optimal = 0$
9: **tant que** $optimal = 0$ **effectuer**
10: $optimal_R = 0$
11: **tant que** $optimal = 0$ **effectuer**
12: Effectuer m itérations de l'algorithme du simplexe sur PL_R. Obtenir la solution primale $\tilde{x}_C$, la solution duale $\tilde{\pi}_P$, la valeur de l'objectif Z et le coût réduit cr_C des variables compatibles.
13: **si** $degen2(\tilde{x}_C)$ **alors**
14: $PL_R = reduction(PL_R)$
15: **si** $lent(cr_C)$ **alors**
16: Aller à l'étape 19
17: **si** $\tilde{x}_c$ est optimale pour PL_R **alors**
18: $optimal_R = 1$
19: **si** $degenere(\tilde{x}_C)$ et $Z < Z_0$ **alors**
20: $PL_R = reduction(PL_R)$
21: $Z_0 = Z$
22: Aller à l'étape 11.
23: Calculer le coût réduit cr_I des variables incompatibles et construire un ensemble I de variables à coût réduit négatif.
24: **si** $cr_I \geq 0$ et $optimal_R = 1$ **alors**
25: $optimal = 1, \quad x^* = \begin{pmatrix} x_P^* \\ 0 \end{pmatrix}$ est optimale.
26: $PL_R = augmentation(PL_R)$

simplexe puis reprend la main toutes les m itérations pour s'assurer qu'il n'est pas apparu de dégénérescence en cours de résolution et que la résolution améliore encore significativement la valeur de l'objectif. Une description plus précise du processus de résolution est abordée dans la section 4.2.1.

La section 4.2.2 est consacrée aux nouvelles méthodes de manipulation du problème réduit appelées par les fonctions *réduction* et *augmentation*, ainsi qu'au prédicat *dégénéré*.

L'étape 23 consiste à calculer les coûts réduits des variables incompatibles. Le coût réduit calculé dans la section 3.2.1 dépendait directement de la méthode de réduction. Avec les changements de procédure, le coût réduit ne peut plus être calculé de la même manière. Nous traiterons ce sujet ainsi que la construction de l'ensemble I dans la section 4.2.3.

Finalement, on teste l'optimalité de la solution primale $x^* = \begin{pmatrix} x_P^* \\ 0 \end{pmatrix}$ pour le problème PL à l'étape 24, et on termine l'algorithme dans le cas favorable.

4.2.1 Le processus de résolution du problème réduit

Après avoir obtenu une solution primale réalisable de PL et construit le problème réduit PL_R, l'algorithme cherche à améliorer la valeur de l'objectif. La résolution à l'optimalité de PL_R est la manière la plus simple d'y parvenir et c'est d'ailleurs ce qui se passera en pratique dans la plupart des itérations majeures. Toutefois, il existe des cas où il s'avère très payant d'inspecter la solution primale et le coût réduit des variables en cours d'optimisation.

Il arrive que, même si le problème est très dégénéré, la solution en fin de phase 1, elle, ne le soit pas du tout. L'optimisation de PL_R reviendrait dans ce cas à optimiser PL,

ni plus ni moins. Le prédicat *degen2* sert à déterminer dans quelle mesure il est apparu suffisamment de dégénérescence additionnelle au cours des dernières itérations de l'algorithme du simplexe pour qu'il vaille mieux réduire le problème que continuer sa résolution. Soit pos_1 le nombre de variables non-nulles de la solution primale courante de l'algorithme du simplexe, et pos_0, le nombre de variables non-nulles de la solution primale x_R au moment de la dernière réduction du problème. *degen2* compare pos_0 et pos_1 et le problème sera réduit si

$$pos_1 < \alpha * pos_0.$$

Le coefficient α varie en fonction du taux de dégénérescence de x_R, c'est-à-dire l'indicateur $\tau = 1 - pos_0/m$. Nous avons choisi ce critère pour éviter de réduire le problème aux moindres fluctuations de la dégénérescence du problème. Une réduction en cours de résolution est une pure perte de temps si on se rend compte que le taux de dégénérescence redescend ensuite à une valeur proche de celle atteinte avant réduction. De telles fluctuations sont souvent observées quand τ est proche du taux de dégénérescence de la solution optimale mais pas quand τ est beaucoup plus faible. Des tests ont révélé qu'il était avantageux de fixer $\alpha = 0.85$ pour $\tau < 0.85$ et $\alpha = 0.97$ pour $\tau \geq 0.85$. Ce choix revient à réduire le problème autant que possible lorsque peu de variables en base sont nulles et à se montrer plus vigilant quand beaucoup de variables en base sont déjà nulles pour ne pas écarter de bonnes variables du problème en le réduisant trop vite.

D'autre part, il n'est pas forcément non plus judicieux d'effectuer toutes les itérations de l'agorithme du simplexe qui mènent à l'optimalité. Lorsque le coût réduit des variables compatibles à coût réduit négatif devient proche de 0, il y a de fortes chances qu'il soit plus intéressant de considérer des variables incompatibles dont le coût réduit est plus négatif. Les itérations de l'algorithme du simplexe à venir ont des chances de représenter une pure perte de temps. Par le prédicat *lent*, la résolution est arrêtée lorsque le coût réduit des variables compatibles cr_C devient proche de 0.

Il est à noter qu'il est à présent possible d'arriver en fin d'itération majeure sans avoir atteint l'optimalité du problème réduit. Le test sur les coûts réduits des variables incompatibles n'est par conséquent plus suffisant, on ajoute l'obligation d'avoir résolu PL_R à l'optimalité pendant la résolution, c'est-à-dire, la condition $optimal = 1$.

4.2.2 Les manipulations du problème réduit

L'algorithme de réduction dynamique de contraintes demande une réintégration de variables et de contraintes dans la majorité des itérations majeures ou, dans certains cas, une réduction du problème. Ces deux opérations sont effectuées par les fonction *augmentation* et *réduction.*

La réduction du problème peut être effectuée à l'issue du test *degen2* décrit dans la section 4.2.1 mais également à la suite du prédicat *dégénéré.* Là où le prédicat *degen2* juge la dégénérescence de la solution courante de l'algorithme du simplexe en tant que solution primale du problème initial PL, le prédicat *dégénéré* signale une trop forte dégénérescence de la solution primale courante par rapport à PL_R. *dégénéré* décide que la fonction *réduction* doit être appliquée lorsque les réintégrations successives de variables et de contraintes ont mené à un problème réduit trop dégénéré dont l'optimisation redevient lente. En gardant les notations de la section précédente et en introduisant m_P, le nombre de contraintes de PL_R, une réduction est opérée lorsque

$$pos_1/m_P < \beta * (1 - \tau) = \beta * pos_0/m.$$

Le choix du coefficient β fera l'objet d'une discussion au chapitre 5. Sa valeur sera prise dans l'intervalle $[1.1, 1.9]$ en essayant de trouver un compromis entre taille du problème réduit et efficacité de la résolution. Réduire le problème trop tôt mène à un grand nombre d'itérations majeures sans parvenir à réintégrer les bonnes variables au problème réduit. Si on se contentait du prédicat *dégénéré* pour prendre la décision

de réduire, il serait possible de ne pas améliorer la valeur de l'objectif entre deux réductions et l'algorithme pourrait boucler. C'est pourquoi on ajoute la condition $Z < Z_0$, en actualisant Z_0 après chaque réduction (en pratique on impose même que la valeur de l'objectif ait diminué d'au moins un facteur $\frac{1}{1000}$).

La fonction *réduction* suit exactement le schéma de la première réduction décrite dans la section 4.1. Des opérations sur les lignes de la matrice des variables non-nulles de la solution primale $\tilde{x}_C$ de PL_R sont appliquées pour en éliminer $m_P - pos_1$ lignes. La matrice L_C analogue à L est déduite. Les variables du problème réduit sont alors les variables de PL_R compatibles avec L_{CN}, la matrice des $m_P - pos_1$ lignes éliminées de L_C. En définitive, les variables du prochain problème réduit sont les variables de PL compatibles avec la matrice de compatibilité

$$\begin{pmatrix} L_{CN} & 0 \\ L_N & \end{pmatrix},$$

où L_N est la matrice de compatibilité à l'issue de la dernière réduction. L_N est alors mise à jour en prenant la valeur de cette matrice de compatibilité. On enlève les $m_P - pos_1$ contraintes éliminées pour obtenir les contraintes du nouveau problème réduit. Si $optimal = 0$ et afin de tirer profit immédiatement de la petite taille du programme linéaire, on va directement le résoudre au lieu d'augmenter sa taille comme l'exige l'instruction de l'étape 22. Si $optimal = 1$, $\tilde{x}_C$ est déjà optimale pour le nouveau problème réduit.

Lorsque PL_R n'est pas réduit et que l'on n'est pas à l'optimalité de PL, l'amélioration de la valeur de l'objectif demande la réintégration de variables dans le problème. Les variables incompatibles réintégrées sont toutes les variables de l'ensemble de variables à coût réduit négatif I construit en se référant à la description faite dans la section 4.2.3. Pour que toute solution de PL_R reste réalisable, on doit ensuite réintégrer toutes les contraintes qui correspondent aux lignes de $L_N A$ où l'une des variables

de I a un coefficient non-nul. La méthode de Pan (1998) suggérait d'effectuer des pivots de Gauss sur $L_N A$ avant la réintégration des contraintes afin de ne réintégrer qu'une contrainte par variable de I. Cette méthode a été implantée mais elle ne diminuait pas beaucoup le nombre de contraintes réintégrées, par contre elle forçait à faire des pivots coûteux en temps de calcul. La matrice L_N est actualisée en retirant les lignes correspondant aux contraintes réintégrées. Il reste finalement à ajouter toutes les variables devenues compatibles à la suite des changements de la matrice compatibilité.

4.2.3 Le coût réduit des variables incompatibles

La complétion de la solution duale faite avec la méthode de base n'est plus possible pour plusieurs raisons. D'abord, nous ne partons plus d'une base réalisable donc, vouloir annuler les coûts réduits des variables qui auraient dû être en base n'a plus de sens. En outre, les coefficients du problème ne sont plus modifiés donc les variables compatibles n'ont plus leurs coefficients tous nuls dans les contraintes retirées. Par conséquent, le raisonnement qui avait amené à la propriété que la solution duale pouvait être complétée comme nous l'entendions n'est plus valable.

Toutes les manières sophistiquées de compléter la solution duale auxquelles nous avons pensé ont entraîné de longs calculs sans assurance d'améliorer la résolution. Le solution duale π est obtenue à partir de π_R par

$$\pi = \begin{pmatrix} \pi_R \\ 0 \end{pmatrix}.$$

Le coût réduit cr des variables est donné par

$$cr^T = \pi^T A.$$

L'ensemble I des variables incompatibles de coût réduit négatifs à réintégrer est ensuite construit en prenant un certain nombre γm de variables de coûts réduits

les plus négatifs. Le nombre de variables de I dépend du nombre de contraintes du problème afin d'avoir une homogénéité entre les résolutions de tous les problèmes. Nous réaliserons des tests dans la section 5.3.1 afin de déterminer la meilleure valeur de la constante γ.

4.3 Preuve de convergence

Nous énonçons à présent un résultat sur la convergence de l'algorithme 4.1 et nous présentons une preuve s'appuyant sur celle du théorème analogue sur la convergence de l'algorithme 4.1 donné dans la section 3.3.

Proposition 4.3.1 *Soit PL un programme linéaire réalisable sous forme standard réalisable. En supposant que l'algorithme du simplexe résolvant PL_R est muni d'un mécanisme d'anti-cyclage, l'application de l'algorithme 4.1 trouve une solution primale optimale x^* de PL ou prouve que PL est non-borné en un nombre fini de pivots du simplexe.*

Preuve : Si le processus de résolution n'avait pas changé, la convergence serait acquise en se référant à la preuve de convergence de l'algorithme de base dans la section 3.3. Montrons donc que le nouveau processus de résolution ne peut pas introduire de cyclage dans les itérations majeures.

Nous avons toujours la propriété voulant qu'après au plus $m-1$ itérations majeures consécutives sans réduction, on aura $\mathrm{PL}_R = \mathrm{PL}$. Puis à chaque itération majeure, on effectue au moins m itérations de l'algorithme du simplexe. Donc, à partir du moment où l'on a $\mathrm{PL}_R = \mathrm{PL}$, on se retrouve à l'optimalité après un nombre fini d'itérations majeures.

Ensuite, nous pourrions ajouter la condition $Z_0 < Z$ à la condition $degen2(\tilde{x}_C)$ à l'étape 13 avant de réduire le problème pour que le nombre maximal de réductions soit fini. Mais, la condition du prédicat implique en fait une amélioration de l'objectif. En effet, *degen2* décidera de procéder à une réduction si le nombre de variables de base nulles a augmenté depuis la dernière réduction. Pour que des variables de base non-nulles s'annulent pendant une itération de l'algorithme du simplexe, il faut que la variable entrante prenne une valeur strictement positive et donc qu'elle améliore la valeur de l'objectif. On conclut que le nombre de réductions au pire des cas est fini. Le processus de résolution ne peut donc cycler. Par conséquent, si PL est réalisable, l'application de l'algorithme 4.1 aboutit à une solution optimale. □

4.4 Conséquences des améliorations

La principale avancée est de ne plus changer les coefficients de la matrice de contraintes à chaque réduction. Les bénéfices de la réduction de taille de PL étaient minimes lorsque comparées aux effets de l'augmentation de la densité de la matrice de contraintes dans les expérimentations de l'algorithme 3.1. À présent, on est assuré que la résolution de PL en utilisant l'algorithme de réduction de contraintes ne résultera plus en une perte importante en terme de temps de calcul si on se compare à une résolution directe par l'algorithme du simplexe. Le calcul de la matrice de compatibilité est simple, une factorisation LU de la matrice B_P donne le résultat souhaité. Des logiciels commerciaux sont très performants pour réaliser ces opérations et nous profiterons de leur efficacité. L'ajout d'une condition avant la première réduction et les reprises de main en cours de résolution permettent de s'adapter à plusieurs classes de problèmes dégénérés. En effet, selon les classes de problèmes, les pivots dégénérés ne se manifestent pas au même moment de la résolution ou le nombre de variables nulles en base ne varie pas de la même façon, mais la dégénérescence a toujours pour conséquence un grand ralentissement de la résolution du problème.

CHAPITRE 5 : EXPERIMENTATION AVEC L'ALGORITHME AMÉLIORÉ

Nous nous intéressons maintenant à la manière dont l'algorithme 4.1 a été appliqué en pratique et aux résultats obtenus par cet algorithme. Dans la section 5.1, nous faisons une description des principales caractéristiques de notre code informatique, et des arrangements de la théorie nécessaires pour s'adapter aux particularités des deux logiciels CPLEX et UMFPACK. 5.2, nous décrivons les deux classes de problèmes utilisés pour les tests numériques. Enfin, nous présentons l'ensemble des résultats de notre méthode de réduction pour une résolution partant d'une solution de fin de phase 1 de l'algorithme du simplexe dans la section 5.3 et pour la résolution de problèmes perturbés dans la section 5.4.

5.1 Caractéristiques de l'implantation

L'intégralité du code a été écrit dans le langage C et nous avons utilisé la version 9.1 de CPLEX. En suivant nos observations faites lors des tests sur l'algorithme de base, nous avons réduit au maximum l'usage de CPLEX en dehors des résolutions par l'algorithme primal du simplexe. Il y est fait appel pour résoudre la phase 1 si besoin, pour la résolution des problèmes réduits et pour l'acquisition des données courantes de la résolution. Les réductions seront maintenant effectuées à l'aide de la librairie UMFPACK 4.4.

5.1.1 La librairie UMFPACK

Nous avons d'abord écrit un code pour éliminer les lignes de B_P en faisant directement les pivots de Gauss sur la matrice A entière. Cette fonction avait l'avantage de construire directement la matrice de compatibililité et de laisser le choix de l'élément sur lequel le pivot est effectué. Par contre, si la méthode fonctionnait parfaitement sur des petits programmes linéaires, nous avons vite rencontré des problèmes de performance et d'instabilité numérique en augmentant le nombre de lignes et de colonnes. Par conséquent nous avons décidé de faire appel à la librairie UMFPACK qui fournit un code écrit dans le langage C très performant pour effectuer des factorisations LU et résoudre des systèmes d'équations linéaires. D'ailleurs, le logiciel commercial Matlab utilise les fonctions d'UMFPACK.

Ce code n'est toutefois pas exactement adapté à nos besoins pour les raisons suivantes. La factorisation LU d'une matrice M de dimensions $m \times n$ renvoie les matrices P, L, U et Q, telles que

$$PLUQ = M,$$

où L est une matrice triangulaire inférieure de dimensions $m \times min(m,n)$, U est une matrice triangulaire supérieure de dimensions $min(m,n) \times n$, P est une matrice de permutations de lignes et Q une matrice de permutations de colonnes. Les permutations sur les lignes et les colonnes de M sont faites afin de choisir les pivots qui minimiseront les erreurs numériques et la densité de la matrice U. Nous désirons éliminer $m - p$ lignes de la matrice B_P de plein rang et de dimensions $m \times p$ avec $p < m$. La factorisation LU de la matrice B_P retournerait donc une matrice L de dimensions $m \times p$ et une matrice U de dimensions $p \times p$. Nous souhaitons pour notre part obtenir une matrice L carrée que l'on puisse inverser pour appliquer à A la même transformation qu'à B_P et une matrice U où les lignes annulées apparaissent. Nous avons donc choisi de factoriser la matrice $m \times m$

$$B = \left(\begin{array}{c|c} B_P & 0 \end{array} \right).$$

L et U sont maintenant deux matrices carrées d'ordre m. Les $m-p$ dernières lignes de U sont nulles et les $m-p$ colonnes nulles ajoutées n'ont rien changé aux pivots de Gauss et sont toujours nulles en fin de factorisation. Par contre, la matrice L n'est pas la même que celle du chapitre 4 que l'on obtiendrait ici par $(PL)^{-1}$. Il reste ensuite à résoudre les systèmes

$$PLx = A^j, \quad j \in I_N,$$

dont les solutions x sont les colonnes de la matrice de compatibilité liées aux variables nulles afin de déterminer les variables compatibles.

L'installation de UMFPACK n'a en revanche pas été complétée, car nous n'avons pas intégré de bonne librairie de calcul vectoriel et matriciel (CBLAS était recommandé) pour des raisons propres à notre système informatique. UMFPACK fonctionne tout de même mais il mentionne clairement que les calculs sont lents en l'absence d'une telle librairie. La lenteur n'est pas exagérée mais nous comparerons tout de même en priorité les performances de la résolution directe par CPLEX avec la partie du temps de calcul consommé par CPLEX dans la résolution par l'algorithme amélioré. Cette approximation n'est en général pas déraisonnable et permet de juger plus précisément du potentiel de l'algorithme 4.1.

5.1.2 Réglage des options de CPLEX

La théorie de notre algorithme repose entièrement sur l'hypothèse que la solution initiale ainsi que les solutions acquises à l'étape 12 de l'algorithme 4.1 sont réalisables. Or, le logiciel a une tolérance par défaut de 10^{-6} pour déterminer, en cours de résolution, si une solution est réalisable. Le code corrige cette erreur en réduisant la tolérance avant de conclure à l'optimalité mais il ne le fait pas en cas de reprise de main de l'utilisateur. Une réduction faite à partir d'une solution non-réalisable,

même à 10^{-6} près, conduit souvent à un problème réduit non-réalisable. Nous avons donc réduit la tolérance au minimum possible, i.e., 10^{-9}.

D'autre part, il y a aussi une tolérance par défaut de 10^{-6} pour conclure à l'optimalité d'une solution. Avec cette tolérance, il nous est arrivé de trouver des valeurs optimales de l'objectif différentes selon qu'on résolve directement par CPLEX ou qu'on utilise l'algorithme 4.1. Nous avons également fixé cette tolérance à 10^{-9} et avons rétabli la cohérence entre les deux méthodes.

Finalement, nous avons éteint le présolveur de CPLEX pour la résolution directe et pour l'algorithme de réduction dynamique de contraintes. Le présolveur a souvent de très bons résultats mais il change la structure de notre problème d'une manière que nous ne contrôlons pas. Plutôt que de compliquer nos analyses, nous résolvons directement les problèmes initiaux.

5.2 Description des problèmes de test

Afin d'apporter de la diversité aux tests, nous avons utilisé une classe de problèmes supplémentaire à celle des VCSP que l'on a décrite dans la section 3.4.2. Ces problèmes ont été introduits par Gonzaga (2003). Ils sont générés par une fonction codée sous le logiciel Matlab qui contrôle le nombre de contraintes m, le nombre de variables n, la dimension dim_d de la face optimale duale, la dimension dim_p de la face optimale primale et la densité de la matrice de contraintes. Les faces optimales primale et duale sont les ensembles de solutions optimales primales et duales d'un PL. Nous décrivons ci-dessous la construction d'un tel problème.

5.2.1 Construction des problèmes de Gonzaga

Gonzaga (2003) part de la constatation qu'en générant aléatoirement la matrice des contraintes, la probabilité d'observer de la dégénérescence primale et de la dégénérescence duale est nulle. La matrice doit donc respecter une certaine structure. Posons $r = m - \dim_d$ et $p = r + \dim_p$. La construction part d'une partition de l'ensemble des indices de colonnes en deux ensembles

$$B = \{1, \ldots, p\} \quad et \quad N = \{p+1, \ldots, n\}.$$

On crée alors les variables primales (x^*) et duales (s^*) optimales du problème à générer. s est en fait le vecteur des variables de surplus du problème dual

$$\begin{array}{lrl} \min\limits_{\pi} & b^T\pi & \\ \text{s.c.} & A^T\pi + s = & b \\ & s \geq & 0, \end{array}$$

mais comme A est de plein rang, chaque vecteur π correspond à un vecteur s unique. On peut donc parler de solution duale π ou s indifféremment. Les deux solutions optimales valent

$$x^* = \begin{pmatrix} x_B \\ 0 \end{pmatrix} \quad et \quad s^* = \begin{pmatrix} 0 \\ s_N \end{pmatrix},$$

où $x_B \in \mathbb{R}^p_+$ et $s_N \in \mathbb{R}^{n-p}_+$, sont générés aléatoirement dans l'intervalle $[10^{-4}, 1 + 10^{-4}]$.

Construisons alors la matrice des contraintes A. Nous définissons dans un premier temps l'ensemble de matrices ci-dessous.

- D_r et D_{m-r} sont deux matrices diagonales de dimensions r et $m - r$ dont tous les éléments diagonaux sont non-nuls.
- R_1 et R_2 sont deux matrices de dimensions respectives $(m - r) \times p$ et $m \times (n - p)$ ayant un élément non-nul par colonne sur une ligne prise au hasard.

- R_3 et R_4 sont deux matrices de dimensions respectives $(m-r) \times p$ et $m \times (n-p)$ générées aléatoirement et dont le nombre d'éléments non-nuls est déterminé de façon à ce que A ait la densité désirée.

Les éléments non-nuls de ces 6 matrices sont pris aléatoirement selon une loi uniforme dans l'intervalle $[-1, 1]$. Posons ensuite

$$A_1 = \left(\begin{array}{cc|cc} D_r & R_1 & 0 & \\ 0 & 0 & D_{m-r} & R_2 \end{array} \right) \quad \text{et} \quad A_2 = \left(\begin{array}{c|c} R_3 & \\ 0_{m-r} & R_4 \end{array} \right).$$

La matrice A est obtenu par

$$A = \left(\begin{array}{c|c} A_B & A_N \end{array} \right) = A_1 + A_2.$$

On termine la génération du problème en posant $b = Ax^*$ et $c = s^*$.

Il est démontré que les dimensions des faces optimales primales et duales satisfont bien la demande. Pour notre part, nous remarquerons que A_B la matrice des p premières colonnes de A est de rang r avec la probabilité 1 et que les $m-r$ derniers éléments de b sont nuls. Ecrivons $b = \begin{pmatrix} b_r \\ 0 \end{pmatrix}$ avec $b_r \in \mathbb{R}^r$. Puisqu'il existe $\hat{x}_r \in \mathbb{R}^r$ tel que $A_B \hat{x}_r = b_r$, le vecteur $\hat{x} = \begin{pmatrix} \hat{x}_r \\ 0 \end{pmatrix}$ est une solution réalisable du problème. Les r premiers éléments du vecteur de coût étant nuls, le coût de $\hat{x}$ est nul, et comme tous les éléments de c sont positifs, $\hat{x}$ est une solution optimale. Cette observation montre qu'il est possible de trouver une solution optimale dont le nombre de variables en bases nulles est $m - r = \dim_d$. Dans nos résultats, ce sera d'ailleurs, en général, le nombre de variables en base nulles dans la solution optimale.

5.2.2 Choix des problèmes à résoudre

Les caractéristiques de l'ensemble des problèmes de Gonzaga que nous utiliserons dans cette section sont répertoriées dans le tableau 5.1 Ces problèmes, référencés par

l'abréviation GON, ont été choisis pour renseigner sur l'impact d'une variation de dim_d (GON1,GON2,..., GON6), de la taille du problème (GON3, GON7,...,GON11), de m pour n constant (GON3,GON12,..., GON15) et de dim_p (GON9, GON16,..., GON19). Afin de mieux identifier les différents groupes de problèmes de Gonzaga, les problèmes GON3 et GON9 sont répétés plusieurs fois dans les tableaux de résultats. Les caractéristiques des VCSP sont inscrites dans le tableau 3.1 de la section 3.4.3. Toutes les données relatives aux problèmes de Gonzaga correspondent à des moyennes calculées sur la résolution de trois instances du même problème. Nous avons pris la densité de matrice de contraintes égale à 0.001 pour tous les problèmes de Gonzaga.

Nous allons maintenant présenter les performances de notre implantation de l'algorithme 4.1 pour la résolution des problèmes décrits dans les tableaux 5.1 et 3.1.

5.3 Expérimentation en partant d'une solution de phase 1

Dans cette section, nous exposons les résultats de l'algorithme amélioré lorsque la solution initiale réalisable est la solution retournée par CPLEX à la fin de la phase 1 de l'algorithme du simplexe primal.

Les résultats de la résolution directe par CPLEX des problèmes de Gonzaga sont donnés dans le tableau 5.2, ceux de la résolution directe des VCSP sont dans le tableau 3.1 de la section 3.4.3. La plupart des notations utilisées dans les tableaux de résultats ont été introduites dans la section 3.4.3. Nous en donnons trois autres ci-dessous. Tous les temps sont donnés en secondes.

- τ_{p1} taux de dégénérescence du problème en fin de phase 1 de l'algorithme du simplexe
- $\bar{m}$ nombre moyen de contraintes pendant la résolution des problèmes réduits
- $\bar{n}$ nombre moyen de variables pendant la résolution des problèmes réduits

Tableau 5.1 – Description des problèmes de test

Nom	m	n	dim_d	dim_p
GON1	3000	15000	0	0
GON2	3000	15000	300	0
GON3	3000	15000	900	0
GON4	3000	15000	1800	0
GON5	3000	15000	2700	0
GON6	3000	15000	2999	0
GON7	2500	12500	750	0
GON3	3000	15000	900	0
GON8	3500	17500	1050	0
GON9	4000	20000	1200	0
GON10	4500	22500	1350	0
GON11	5000	25000	1500	0
GON12	3000	9000	900	0
GON3	3000	15000	900	0
GON13	3000	30000	900	0
GON14	3000	60000	900	0
GON15	3000	120000	900	0
GON9	4000	20000	1200	0
GON16	4000	19970	1200	1600
GON17	4000	19849	1200	8000
GON18	4000	19777	1200	12800
GON19	4000	19691	1200	15999

5.3.1 Choix des constantes β et γ

La constante γ agit sur la taille de l'ensemble I, ensemble des variables de coût réduit négatif qui sont réintégrées en fin d'itération majeure. Nous avons $numretour = \gamma m$. Pour déterminer cette constante, nous résolvons les quatre VCSP par l'algorithme amélioré pour $\gamma = 0.01, 0.02, \ldots, 1.1, 1.2$ et nous consignons les moyennes des résultats trouvés dans le tableau 5.3.

Tableau 5.2 – Résultats de la résolution directe par CPLEX

Nom	t_{tot}	i_{tot}	$\tau_{opt}(\%)$	t_{p2}	i_{p2}	$\tau_{p1}(\%)$
GON1	547	54674	0	520	50869	0
GON2	437	56976	10	520	53253	2
GON3	388	95337	30	376	91785	7
GON4	895	212098	59.5	888	208332	12
GON5	1781	250313	90	1744	242096	11
GON6	696	55312	99.97	696	55312	99.97
GON7	82	37398	30.1	81	35958	11
GON3	388	95337	30	376	91785	7
GON8	757	113563	30	688	107749	5
GON9	2868	203416	30	2690	196102	3
GON10	6337	253589	30	6022	244225	2
GON11	11766	320281	30	11103	307742	1
GON12	185	53834	29.97	166	50154	7
GON3	388	95337	30	376	91785	7
GON13	1050	151140	30	1044	148184	8
GON14	2698	195718	30	2696	194316	11
GON15	4082	373774	30.03	4081	372935	14
GON16	1335	105529	30	1219	98704	4
GON17	297	35650	30	239	28894	9.3
GON18	80	16127	30	59	9500	18.7
GON19	4.6	2627	30	0.3	74	30

Pour $\gamma \leq 0.03$, l'algorithme réintègre trop peu de variables à chaque augmentation. Par suite, un grand nombre d'itérations majeures sont faites sans amélioration substantielle de la valeur de l'objectif. À l'inverse, quand $\gamma \geq 0.09$, trop de variables sont réintégrées à chaque augmentation, et la taille des problèmes réduits pourrait être plus petite tout en améliorant autant la valeur de l'objectif. En revanche pour $0.04 \leq \gamma \leq 0.08$, les résultats sont bons et similaires. Nous préférerons prendre $\gamma = 0.05$ parce que les résultats sont meilleurs qu'avec $\gamma = 0.04$ et que les temps de calcul trouvés pour les résolutions des quatre VCSP sont plus homogènes avec $\gamma = 0.05$ qu'avec $0.06 \leq \gamma \leq 0.08$. Ce dernier constat s'explique par le fait que moins

Tableau 5.3 – Influence du critère γ sur l'algorithme

γ	t_{plr}	t_{tot}	n_{red}	i_{maj}
0.01	263	318	6	70
0.02	222	285	8	44
0.03	212	285	7	33
0.04	213	269	7	25
0.05	200	254	7	23
0.06	201	253	6	20
0.07	213	264	6	18
0.08	201	248	6	16
0.09	227	280	6	16
0.10	223	274	6	14
0.11	226	286	7	16
0.12	226	298	7	15

d'itérations majeures sont faites pour $0.06 \leq \gamma \leq 0.08$ et, par conséquent, le hasard a plus d'impact sur la qualité de la solution avant réduction du problème.

Le choix des constantes α et β, définies respectivement dans les sections 4.2.1 et 4.2.2, décide des moments auxquels le problème est réduit. Leur rôle dans l'efficacité de l'algorithme est décisif. La valeur de la constante α a déjà été discutée dans la section 4.2.1. La majorité des réductions se fait pendant le processus de résolution de PL_R pour les problèmes GON et à l'optimalité du problème réduit pour les VCSP. Nous étudions donc le rôle de β par des tests sur les 4 VCSP. Nous faisons varier β dans l'intervalle [1.1, 1.9] et répertorions les informations relatives à la résolution des 4 VCSP dans le tableau 5.4. Les données contenues dans le tableau sont les moyennes des données recueillies pour les 4 résolutions.

Nous observons qu'à l'exception de la seconde gagnée sur t_{tot} quand $\beta = 1.4$, fixer $\beta = 1.5$ donne des meilleurs résultats sur tous les points. Ce sera donc la valeur que nous prendrons par la suite.

Tableau 5.4 – Influence du critère β sur l'algorithme

β	t_{plr}	t_{tot}	n_{red}	i_{maj}
1.1	251	278	3	16
1.2	255	279	3	15
1.3	227	268	5	20
1.4	202	257	7	23
1.5	191	258	9	26
1.6	203	309	14	33
1.7	201	309	16	34
1.8	197	308	17	34
1.9	197	308	17	34

5.3.2 Résolution des VCSP

Nous regroupons tous les résultats des VCSP dans le tableau 5.5. On observe une réduction du temps de calcul consommé par CPLEX allant de 42% à 55% en résolvant les VCSP par l'algorithme amélioré. Les temps de résolution ont été diminués par un facteur 20 si on se rapporte aux résultats de l'algorithme de base (voir tableau 3.2) et nous améliorons à présent les performances de la résolution directe par CPLEX (tableau 3.1). Nous notons que le nombre d'itérations du simplexe reste pourtant le même. Le progrès a surtout été fait par la réduction de la taille moyenne des problèmes résolus. En effet, le nombre de lignes de la matrice des contraintes a été réduit de 25% et le nombre de colonnes a été réduit de 50%. Par ailleurs, le temps de calcul consommé par les réductions a lui aussi diminué substantiellement par l'utilisation d'UMFPACK.

Tableau 5.5 – Résolution des VCSP à partir d'une solution de phase 1

Nom	t_{tot}	t_{plr}	i_{tot}	τ_{opt}	n_{red}	i_{maj}	$\frac{\tilde{m}}{m}(\%)$	$\frac{\tilde{n}}{n}(\%)$	$Perf(\%)$
VCSP1	312	232	72358	35	9	28	76	48	45
VCSP2	213	150	46705	36	10	25	75	51	42
VCSP3	233	182	55127	36	7	23	74	50	46
VCSP4	274	200	63286	37	9	28	74	46	55

5.3.3 Résolution des problèmes de Gonzaga

Les résultats de la résolution des problèmes de Gonzaga sont rapportés dans le tableau 5.6. Nous observons ici des améliorations dépassant les 90% en temps de calcul par rapport à la résolution directe par CPLEX, les meilleurs résultats ayant lieu lorsque τ_{opt} est grand ou que l'on augmente le nombre de variables sans toucher au nombre de contraintes.

Il convient d'expliquer la qualité de ces derniers résultats. La structure des problèmes générés est spéciale pour deux raisons. D'abord, le problème est construit de manière à ce que la valeur optimale de l'objectif soit égale à 0. Les variables non-nulles de la solution optimale ont donc toutes un coût nul. Par conséquent, s'il s'avère que la solution courante soit optimale et que l'on réduise le problème à ce moment là, le vecteur de coût des variables de base c_B sera nul, puis le vecteur des variables duales $\tilde{\pi}_P$, obtenu par

$$\tilde{\pi}_P^T = c_B^T B^{-1},$$

sera également nul. En complétant le vecteur des variables duales, nous obtenons $\pi = 0$, puis $cr = c$. Puisque $c \geq 0$, l'algorithme concluera immédiatement à l'optimalité. Cette propriété sauve un très grand nombre de pivots du simplexe dégénérés dans la résolution des problèmes les plus dégénérés et explique notamment la raison pour laquelle nous obtenons des améliorations de 100% pour certains problèmes. Il suffit

Tableau 5.6 – Résolution des problèmes de Gonzaga depuis une solution de phase 1

Nom	t_{tot}	t_{plr}	i_{tot}	τ_{opt}	n_{red}	i_{maj}	$\frac{\tilde{m}}{m}(\%)$	$\frac{\tilde{n}}{n}(\%)$	$Perf(\%)$
GON1	550	550	51484	0	0	1	100	100	-6
GON2	399	353	44516	10	3	4	96	87	15
GON3	159	108	25265	30	7	9	84	62	69
GON4	80	61	26206	60	5	15	77	50	93
GON5	140	111	41767	90	5	5	75	43	94
GON6	0.7	0	0	99.7	1	1	-	-	100
GON7	31	24	11256	30	5	6	81	58	70
GON3	159	108	25265	30	7	9	84	62	69
GON8	351	235	32291	30	11	21	87	46	66
GON9	1375	935	64366	30	9	10	89	69	65
GON10	3379	2466	99068	30	10	10	92	75	59
GON11	7088	5452	136992	30	10	11	95	81	51
GON12	88	62	17265	30	8	10	84	65	63
GON3	159	108	25265	30	7	9	84	62	69
GON13	376	247	40232	30	8	20	84	57	74
GON14	448	281	41704	30	7	9	81	48	90
GON15	495	273	118419	30	7	9	77	40	93
GON9	1375	935	64366	30	9	10	89	69	65
GON16	810	454	38072	30	9	9	87	68	63
GON17	158	80	10681	30	4	4	84	75	67
GON18	59	23	3584	30	3	4	84	87	61
GON19	5.8	0.1	2	30	1	1	70	88	67

pour cela que la solution initiale soit une solution optimale. Autrement dit, pour ce problème particulier, il s'avère que notre manière de compléter la solution duale soit la meilleure possible. Par cette remarque, nous touchons du doigt l'importance de trouver de bonnes valeurs pour les variables duales associées aux contraintes éliminées du problème. Ce problème paraît propre à chaque classe de PL et difficile à résoudre dans le cas général.

La seconde particularité des problèmes générés aléatoirement est cette fois défavorable à notre méthode de résolution. Le taux de dégénérescence des solutions du

problème réduit n'approche celui de la solution optimale que lorsqu'elle partage un grand nombre de variables de base avec la solution optimale. En général, il faut donc attendre en général un stade avancé de la résolution avant de réduire largement la taille du problème. Une conséquence de ce point est par exemple que la solution optimale de GON5 possède seulement 10% de variables non-nulles alors que le nombre moyen de contraintes des problèmes réduits est de 75% du nombre de contraintes du PL initial.

A la lumière de ces deux propriétés, nous commentons le comportement de l'algorithme en fonction des caractéristiques des problèmes.

Les six problèmes GON1, GON2,..., GON6, varient uniquement par le taux de dégénérescence de leur solution optimale. Conformément à nos attentes, plus le taux de dégénérescence de la solution optimale est grand, meilleurs sont les résultats. À la différence de la résolution des VCSP, le nombre d'itérations d'algorithme du simplexe est plus petit que dans la résolution directe mais la taille moyenne du problème ne diminue pas autant pour des valeurs de τ_{opt} comparables. Ces deux changements s'expliquent par la description faite ci-dessus des particularités des problèmes générés aléatoirement. Le problème n'est pas réduit avant de s'approcher de la solution optimale. Par contre, à partir du moment où l'on se rapproche de l'optimalité, la solution optimale est trouvée très rapidement. Le taux de dégénérescence influe quand même sur la taille moyenne du problème réduit qui décroît strictement quand le taux de dégénérescence augmente. Mais il est surtout important ici car de nombreux pivots du simplexe dégénérés sont épargnés par la réduction dynamique.

Ensuite, nous notons que les performances de la méthode de réduction dynamique de contraintes diminuent lorsque la taille du problème augmente (deuxième groupe de problèmes de Gonzaga dans les tableaux) et augmentent avec n lorsque m est fixé (troisième groupe de problèmes de Gonzaga). Pour expliquer ce comportement, il est

utile de se référer aux valeurs de τ_{p1} du tableau 5.2. En effet, τ_{p1} représente le taux de dégénérescence de la solution en fin de phase 1 et donc de la solution initiale de l'algorithme. Puis, étant donné que, pour ces problèmes, le taux de dégénérescence de la solution courante de l'algorithme du simplexe augmente en se rapprochant de la solution optimale, la valeur de τ_{p1} renseigne aussi sur la qualité de la solution initiale. Quand la taille du problème diminue ou que n augmente à m constant, τ_{p1} augmente sans pour autant allonger le temps de calcul consommé par la phase 1 par rapport à celui consommé par la phase 2 de l'algorithme du simplexe. Ces variations des caractéristiques du problème sont donc bénéfiques à la méthode de réduction dynamique sans profiter à la résolution directe par CPLEX, d'où les fluctuations de performances de l'algorithme.

L'étude de la variation de la dimension de la face optimale primale nous permet de nuancer notre analyse puisqu'ici, une variation de τ_{p1} ne change pas l'efficacité de l'algorithme. Nous expliquons ce phénomène par le fait que l'amélioration de la solution de fin de phase 1 se fait ici au prix de nombreuses itérations de simplexe supplémentaires. Ainsi, lorsque dim_p augmente, la phase 1 de l'algorithme du simplexe améliore autant la solution réalisable pour la phase 2 de CPLEX que pour notre méthode.

Nous avons vu dans cette section que la méthode de réduction dynamique de contraintes obtenait de bons résultats à partir d'une solution généré par une phase 1 de l'algorithme du simplexe. Nous aimerions à présent étudier l'impact de la qualité de la solution initiale en résolvant des PL plus ou moins perturbés à partir de la solution du problème initial.

5.4 Résolution d'un problème perturbé

Nous présentons ici les résultats de l'algorithme lorsque l'on résout un problème perturbé en partant de la solution optimale du problème initial. On perturbe un PL

en remplaçant le vecteur de coût c de la fonction objectif par le vecteur $\hat{c}$ tel que

$$\hat{c}^j = c^j + \delta c_{max} U$$

où U est une variable aléatoire uniforme dans le segment $[0, 1]$, δ est le *facteur de perturbation* et c_{max} est le plus grand élément du vecteur c.

Les variables comptant les nombres d'autobus et de chauffeurs des VCSP ont des coûts 1000 fois plus grands que les autres variables. La résolution d'un tel problème consiste donc en général à minimiser les nombres de chauffeurs et d'autobus, puis à minimiser le coût des tournées en fonction du résultat trouvé. À moins de perturber le problème par un facteur de perturbation énorme, la solution optimale du problème perturbé ne diffère que très peu de la solution du problème initial car les quantités optimales de chauffeurs et d'autobus restent les mêmes. Dans ce cas, la réduction du temps de calcul tourne autour de 98% mais la situation est trop favorable.

Afin de réaliser des expériences plus significatives, nous avons ramené le coût des variables comptant le nombre d'autobus et de chauffeurs à des valeurs du même ordre que les coûts des autres variables. Nous prenons donc un coût de 50, les coûts des autres variables variant entre 15 et 90. La structure du problème n'est pas changée, par contre, le problème sera beaucoup plus sensible aux perturbations de l'objectif.

Les résultats de l'algorithme 4.1 pour l'optimisation des VCSP sont donnés dans le tableau 5.7. Nous rapportons dans le tableau 5.8 les résultats de cet algorithme pour la résolution de GON3 et GON4 dont on a également perturbé les coefficients de l'objectif. Nous notons *Pert* le taux de perturbation de la solution finale, c'est-à-dire le rapport entre le nombre de variables positives dans la solution optimale mais nulles dans la solution initiale et le nombre total de variables positives dans la solution optimale. *Perf1* est l'indice de performance si on fournit à CPLEX la base optimale du problème initial et *Perf2* est l'indice de performance si on fournit uniquement

à CPLEX la base partielle contenant les variables positives de la solution optimale du problème initial. Nous faisons les deux comparaisons car notre algorithme utilise uniquement l'information relative aux variables positives de la solution optimale du problème initiale, mais en contrepartie, si on se place dans un contexte de réoptimisation, il est possible que l'on détienne la base optimale entière. Il faut de plus préciser que CPLEX tire très peu partie d'une base partielle en général, et il arrive même qu'il soit plus lent qu'en ne partant de rien. L'indice *Perf2* sera donc toujours meilleur que *Perf1*. Dans la suite, nos commentaires sur la performance de l'algorithme 4.1 font référence à *Perf1*.

Les performances de l'algorithme pour la résolution des VCSP s'étalent entre 18% et 78% et diminuent quand le taux de perturbation de la solution optimale augmente. Cette tendance se retrouve également pour la résolution des problèmes de Gonzaga. La réduction dynamique tire mieux profit d'une bonne solution initiale que CPLEX. Nous dévoilons ici un enjeu important de notre méthode. La recherche d'une bonne solution primale réalisable pour débuter la résolution dans les meileures conditions possibles. Toutefois cette recherche devra se faire différemment pour chaque classe de problème.

La perturbation des problèmes de Gonzaga ne change pas uniquement la solution optimale du problème, elle fait également chuter le taux de dégénérescence de la solution optimale. Cependant, les performances de la réduction dynamique n'en souffrent pas tellement. Pour GON3, l'amélioration devient moins bonne qu'en partant de la solution de phase 1 pour la plus grande valeur de δ uniquement. L'efficacité de notre algorithme pour la résolution de GON4 est toujours moins bonne qu'en partant de la solution de phase 1 mais on a tout de même $Perf1 \geq 61\%$. Cela résulte du fait que cette fois la solution initiale a beaucoup de variables en base nulles et, par conséquent, la réduction du problème est tout de suite intéressante.

Pour conclure, les résultats de la méthode de réduction dynamique de contraintes

Tableau 5.7 – Résultats pour les VCSP perturbés

Nom	δ	t_{tot}	t_{plr}	i_{tot}	$\tau_{opt}(\%)$	$Pert(\%)$	$Perf1(\%)$	$Perf2(\%)$
VCSP1	5	47	26	6725	34	29	78	95
	10	99	66	18064	35	30	65	88
	15	141	99	25943	36	33	45	82
	20	152	98	27397	35	39	48	82
	50	292	202	58689	36	55	32	59
	100	286	198	57808	36	60	33	58
VCSP2	5	41	26	6933	35	18	58	92
	10	100	68	18918	36	35	54	80
	15	131	94	27027	35	41	43	72
	20	157	118	33179	33	45	36	64
	50	240	168	51384	35	54	20	46
	100	237	163	54193	35	62	22	45
VCSP3	5	56	36	9255	35	21	63	92
	10	155	105	27779	35	40	71	78
	15	169	118	31299	36	32	39	73
	20	239	186	47088	34	50	22	57
	50	346	228	69101	34	42	18	45
	100	297	197	68221	33	72	20	47
VCSP4	5	78	49	13324	37	35	78	92
	10	166	126	32290	35	38	37	79
	15	246	184	50208	35	49	27	69
	20	245	179	49416	35	51	36	69
	50	370	255	75285	34	67	18	51
	100	337	232	65656	33	65	28	54

proposée sont également bons pour la résolution de problèmes perturbés et la qualité des résultats dépend étroitement de la qualité de la solution réalisable initiale de l'algorithme. Ainsi, il sera toujours intéressant de fournir à l'algorithme une bonne solution initiale, quitte à passer un peu de temps à trouver cette solution par une heuristique quand la possibilité se présente. Cependant, si la recherche d'une bonne solution initiale ne paraît pas avantageuse, résoudre le problème par l'algorithme en débutant par une phase 1 de l'algorithme du simplexe reste une alternative intéres-

Tableau 5.8 – Résultats pour les problèmes de Gonzaga perturbés

Nom	δ	t_{tot}	t_{plr}	n_{tot}	$\tau_{opt}(\%)$	$Pert(\%)$	$Perf1(\%)$	$Perf2(\%)$
GON3	5	38	22	6175	20	22	69	94
	10	75	41	10691	17	31	68	90
	15	87	42	13095	15	34	69	89
	20	100	55	15547	13	38	63	86
	50	187	118	26967	9	48	47	71
	100	259	180	37805	7	55	39	57
GON4	5	60	53	28003	19	55	88	93
	10	74	65	31534	15	58	86	91
	15	79	57	28629	14	61	87	89
	20	93	76	34079	13	63	83	85
	50	162	107	39655	11	69	75	76
	100	228	152	49322	10	72	61	64

sante si le problème est dégénéré.

CONCLUSION

La résolution de PL par l'algorithme primal du simplexe se heurte à des difficultés lorsque le PL à optimiser est dégénéré. Des méthodes de sélection des variables entrante et sortante et des méthodes de perturbations sont utilisées depuis longtemps pour réduire les pertes de temps engendrées par des pivots dégénérés. Récemment, des méthodes d'agrégation de contraintes ont été suggérées pour la résolution de PL dégénérés. Ces méthodes traitent la dégénérescence sous un nouvel angle et vont être d'autant plus rapides que les problèmes sont dégénérés. Les meilleurs résultats de cette méthode ont été obtenus par Ellhalaoui *et al.* (2005) pour la résolution de VCSP par génération de colonnes. L'objectif de ce mémoire était de proposer une méthode similaire s'appliquant à n'importe quel PL dégénéré.

Notre recherche nous a conduit à décrire deux algorithmes de résolution de programmes linéaires par réduction dynamique de contraintes dont la convergence théorique est assurée.

Nous avons d'abord développé un algorithme de base que nous avons implanté en nous servant du logiciel CPLEX pour réaliser les itérations de l'algorithme du simplexe et effectuer les opérations nécessaires à la réduction du problème. Puis nous avons résolu quatre problèmes maîtres obtenus en appliquant une méthode de génération de colonnes sur un VCSP. Les temps de calcul de l'algorithme proposé ont été jusqu'à 13 fois plus grand que ceux de la résolution directe par CPLEX.

Nous avons alors changé la méthode de réduction du problème et adapté le processus de résolution du problème réduit afin de récolter des informations en cours de résolution et être capable de reprendre le contrôle à CPLEX pour réduire ou agrandir

le problème. Nous avons encore utilisé CPLEX pour résoudre les problèmes réduits mais nous avons confié les opérations de la réduction au code commercial UMFPACK. Nous avons également introduit une classe de problèmes générés aléatoirement selon une méthode permettant de régler la taille et la densité de la matrice de contraintes ainsi que les dimensions des faces optimales primales et duales. Nous sommes partis de deux types de solutions primales réalisables. La première est une solution générée par une phase 1 de l'algorithme du simplexe. Pour trouver la seconde, nous avons d'abord optimisé le PL puis nous avons perturbé les coefficients de sa fonction objectif. Nous avons alors résolu les problèmes perturbés en partant de la solution optimale du problème perturbé. En partant d'une solution de phase 1, la réduction dynamique a amélioré les temps de calcul de la résolution directe jusqu'à 55% pour les VCSP, et jusqu'à plus de 90% pour les problèmes de Gonzaga quand le taux de dégénérescence de la solution optimale était supérieur à 60%. Cet algorithme a également permis de diviser par 4 et plus le temps de calcul consommé par CPLEX lorsque la solution initiale restait bonne.

Les résultats de la méthode de réduction dynamique pour la résolution de PL appartenant à deux classes de problèmes différentes montrent son efficacité pour combattre la dégénérescence. Il est cependant certain que cet algorithme mérite d'être expérimenté sur beaucoup d'autres classes de problèmes dégénérés afin de faire ses preuves comme généralisation de la méthode d'agrégation de contraintes d'Elhallaoui *et al.* L'analyse des résultats nous a appris, en outre, que cet algorithme gagnerait beaucoup à utiliser les structures particulières de chaque classe de problèmes pour le calcul du coût réduit des variables incompatibles et la recherche d'une bonne solution initiale réalisable. Enfin, un enjeu de recherches futures est d'améliorer le temps de calcul consommé par les réductions du problème qui ne devraient jamais excéder 10% du temps de calcul total pour ne pas gâcher les efforts faits sur le concept de la réduction dynamique de contraintes.

BIBLIOGRAPHIE

BARNES, E., CHEN, V., GOPALAKRISHNAN, B. et JOHNSON, E., (2002). A Least-Squares Primal-Dual Algorithm for Solving Linear Programming Problems. *Operations Research Letters*, 30, 289–294.

BLAND, R.G., (1977). New Finite Pivoting Rule for Simplex Method. *Mathematics of Operations Research*, 2, 103–107.

CHARNES, A., (1952). Optimality and Degeneracy in Linear Programming. *Econometrica*, 20, 160–170.

CHVÁTAL, V., (1983). *Linear Programming.* W.H. Freeman, New York.

DANTZIG, G.B., (1963). *Linear Programming and Extensions.* Princeton University Press, Princeton.

ELHALLAOUI, I., VILLENEUVE, D., SOUMIS, F. et DESAULNIERS, G., (2005). Dynamic Aggregation of Set Partitioning Constraints in Column Generation. *Operations Research*, 53(4), 632–645.

FLETCHER, R., (1998). A New Degeneracy Method and Steepest Edge-Based Conditioning for LP. *SIAM Journal on Optimization*, 4, 1038–1059.

GASS, S.I., (1993). Encounters with Degeneracy : A Personal View. *Annals of Operations Research*, 47, 335–342.

GASS, S.I. et VINJAMURI, S., (2004). Note : Cycling in Linear Programming Problems. *Computers & Operations Research*, 31, 303–311.

GEORGE, K. et OSBORNE, M.R., (1993). On Degeneracy in Linear Programming and Related Problems. *Annals of Operations Research*, 47, 345–359.

GONZAGA, C.C., (2003). Generation of Degenerate Linear Programming Problems. *Technical report, Dept. of Mathematics, Federal University of Santa Catarina, Brazil.*

HAASE, K., DESAULNIERS, G. et DESROSIERS, J., (2001). Simultaneous Vehicle and Crew Scheduling in Urban Mass Transit Systems. *Transportation Science*, 35, 286-303.

HOFFMANN, A.J., (1953). Cycling in the Simplex Algorithm. *National Bureau of Standards*, Report 2974.

MENDELSSOHN, R., (1982). An Iterative Aggregation Procedure for Markov Decision Process. *Operations Research*, 30, 62–73.

NASH, S.G. et SOFER, A., (1996). *Linear and Non-Linear Programming.* Mc Graw Hill, New York.

OUKIL, A., BEN AMOR, H., DESROSIERS, J. et EL GUEDDARI, H., (2004). Stabilized Column Generation Algorithm for Highly Degenerate Multiple-Depot Vehicle Scheduling Problems. *Les Cahiers du Gerad*, G-2004-75, HEC Montréal, Canada.

PAN, P.-Q., (1998). A Basis Deficiency-Allowing Variation of the Simplex Method for Linear Programming. *Computers & Mathematics with Applications*, 36(3), 33–53.

ROGERS, D., PLANTE, R., WONG, R. et EVANS, J., (1991). Aggregation and Disaggregation Techniques and Methodology in Optimization. *Operations Research*, 39(4), 543–582.

RYAN, D.M. et OSBORNE, M., (1988). On the Solution of Highly Degenerate Linear Programmes. *Mathematical Programming*, 41, 385–392.

SHETTY, C.M. et TAYLOR, R.W., (1987). Solving Large-Scale Linear Programs by Aggregation. *Computers & Operations Research*, 14, 385–393.

TERLAKY, T. et SUSHONG, Z., (1993). Pivot Rules for Linear Programming : A Survey on Recent Theoretical Developments. *Annals of Operations Research*, 46, 203–233.

WOLFE, P., (1963). A Technique for Resolving Degeneracy in LP. *SIAM Journal*, 2, 205–211.