Titre: Title:	Un algorithme du simplexe primal amélioré pour des programmes linéaires dégénérés
Auteur: Author:	Vincent Raymond
Date:	2009
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Raymond, V. (2009). Un algorithme du simplexe primal amélioré pour des programmes linéaires dégénérés [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8510/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8510/
Directeurs de recherche: Advisors:	François Soumis, & Dominique Orban
Programme: Program:	Mathématiques de l'ingénieur

UNIVERSITÉ DE MONTRÉAL

UN ALGORITHME DU SIMPLEXE PRIMAL AMÉLIORÉ POUR DES PROGRAMMES LINÉAIRES DÉGÉNÉRÉS

VINCENT RAYMOND
DÉPARTEMENT DE MATHÉMATIQUES
ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR (Ph.D.)
(MATHÉMATIQUES DE L'INGÉNIEUR)
AVRIL 2009

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-89013-4

Our file Notre référence
ISBN: 978-0-494-89013-4

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

UN ALGORITHME DU SIMPLEXE PRIMAL AMÉLIORÉ POUR DES PROGRAMMES LINÉAIRES DÉGÉNÉRÉS

présentée par : RAYMOND Vincent

en vue de l'obtention du diplôme de : Philosophiæ Doctor

a été dûment acceptée par le jury d'examen constitué de:

M. DESAULNIERS, Guy, Ph.D., président

M. SOUMIS, François, Ph.D., membre et directeur de recherche

M. ORBAN, Dominique, Doct. Sc., membre et codirecteur de recherche

M. GAMACHE, Michel, ing., Ph.D., membre

M. MAHEY, Phillipe, Doctorat, membre externe

Remerciements

Je remercie M. François Soumis, directeur de recherche, pour son aide et son encadrement. Par ses connaissances, sa rigueur et sa disponibilité, il a su me faire progresser dans mon cheminement académique. Je le remercie également pour la confiance qu'il m'a démontrée, et ce, du début jusqu'à la fin de mes études.

Je remercie également M. Francois Lessard pour sa patience et son rigoureux travail de programmation. Ses programmes informatiques m'ont permis de me lancer dans l'aventure de la programmation en me basant sur un code informatique solide.

Je désire, par ailleurs, remercier M. Adbelmoutalib Metrane qui a contribué techniquement à certains aspects de mes recherches.

Je tiens enfin à remercier M. Dominique Orban, codirecteur de recherche, pour son aide à la rédaction de mes articles.

Résumé

Cette thèse traite de la présence de dégénérescence lors de la résolution de programmes linéaires. L'une des méthodes les plus populaires et les plus efficaces pour résoudre des problèmes linéaires est la méthode du simplexe. Cette méthode permet, pour une problème de minimisation, de diminuer la fonction objectif d'un programme linéaire non dégénéré à chaque itération. Par contre, lorsque ce programme est dégénéré, certaines itérations ne permettent pas de diminuer la valeur de la solution. Il se peut même que la méthode du simplexe cycle. Toutefois, plusieurs méthodes ont été proposées par différents auteurs pour éliminer le risque de cycler.

Afin d'améliorer la performance de la méthode du simplexe lors de la résolution de problèmes linéaires dégénérés, nous utilisons une méthode qui permet à la méthode du simplexe d'effectuer des pivots dans un problème réduit moins dégénéré. Ce dernier problème contient, en théorie, les lignes associées aux variables non dégénérées ainsi que les variables qui ne violent pas les contraintes enlevées lorsqu'elles entrent en base. Évidemment, une résolution successive de différents problèmes réduits est nécessaire afin d'obtenir la solution optimale. La méthode utilisée permet de modifier en cours de résolution le problème réduit en résolvant un problème complémentaire. Mentionnons que la construction de ces deux derniers problèmes nécessite la multiplication de la matrice inverse de la base par la matrice des contraintes.

Nous présentons d'abord une amélioration de la méthode du problème réduit en modifiant les stratégies et les choix des paramètres de plusieurs aspects de l'algorithme. Que ce soit le nombre d'itérations effectuées sur le problème réduit ou la méthode de résolution du problème complémentaire, toutes les étapes importantes

de l'algorithme sont paramétrées et calibrées. De plus, certains éléments théoriques, tels que l'ajout de contraintes indépendantes au problème réduit, sont appliqués à la méthode. Les résultats numériques obtenus avec la méthode améliorée du problème réduit affichent un facteur de diminution du temps total de résolution de plus de 12 sur des instances d'affectations d'une flotte d'avions comparativement au logiciel commercial CPLEX.

Nous proposons ensuite une généralisation de la méthode du problème réduit. En effet, l'implémentation de cette méthode récente ne convenait pas aux problèmes linéaires où les variables sont bornées inférieurement et supérieurement. Nous modifions donc la théorie ainsi que le programme informatique pour que ce dernier accepte les problèmes avec variables bornées. De plus, la programmation informatique de la première version de la méthode utilisait le logiciel UMFPACK pour effectuer les calculs nécessaires à la création des problèmes réduits et complémentaires. Toutefois la résolution de ces deux derniers problèmes est effectuée avec le logiciel commercial CPLEX. Ainsi, on se retrouve avec un algorithme qui passe d'un logiciel à l'autre tout au long de sa résolution, ce qui peut causer des problèmes d'incompatibilité entre les tolérances. Nous proposons une façon de construire les problèmes réduits et complémentaires en utilisant le logiciel CPLEX. Enfin, nous présentons une nouvelle formulation théorique qui permet de débuter l'algorithme avec une solution réalisable initiale plutôt qu'avec une solution de base réalisable initiale. Les résultats numériques montrent une meilleure performance de notre algorithme sur les problèmes dont les variables sont bornées supérieurement. Cela vient du fait qu'une variable, dont la valeur atteint sa borne supérieure, est traitée comme une variable nulle. De ce fait, la dégénérescence augmente ainsi que la performance de notre algorithme.

Tel que mentionné ci-haut, la construction du problème réduit demande une multiplication de deux matrices, ce qui peut être très coûteux en terme de temps si le problème à résoudre comporte un nombre élevé de contraintes et de variables.

La dernière partie de la thèse présente une version de la méthode du problème réduit qui travaille sur les données originales du problème. En effet, nous avons élaboré une règle d'évaluation, nommée *positive edge*, qui permet de construire le problème réduit en n'utilisant que les coefficients originaux de la matrice des contraintes. En d'autres mots, le critère du *positive edge* identifie les variables, dont l'entrée en base permet d'effectuer un pivot non dégénéré de la méthode du simplexe. Ainsi, il est maintenant possible de résoudre des problèmes de très grande taille (exemple : 100 000 contraintes et 450 000 variables) à l'aide de la méthode du problème réduit. Ce nouveau critère est une percée importante dans le domaine de la programmation linéaire.

En somme, les principales contributions de cette thèse sont : une meilleure répartition du temps de résolution entre les trois différentes sous-procédures de notre algorithme ; une généralisation aux problèmes bornés de la théorie du problème réduit ; une implémentation de l'algorithme qui utilise un seul logiciel commercial (CPLEX) ; une réécriture de la théorie permettant de partir d'une solution initiale plutôt que d'une base initiale ; et le développement d'un nouveau critère d'évaluation qui permet d'identifier si un pivot est dégénéré.

Abstract

This dissertation addresses the problem of degeneracy in linear programs. One of the most popular and efficient method to solve linear programs is the simplex algorithm. This method allows to strictly reduce the objective function at each iteration when the problem is not degenerate. However, non decreasing pivots can be performed when the problem is degenerate. In this case, the simplex method may cycle. In practice, many rules have been proposed to eliminate the risk of cycling.

To increase the performance of the simplex method on degenerate problem, we used an algorithm that executes pivots on a reduced problem. That is, on a problem that contains only the rows associated with the non degenerate variables and only the variables that can enter into the basis without violating the eliminated rows. Obviously, several reduced problems must be solved to reach the optimal solution. The method used allows to modify the reduced problem by solving a complementary problem. Note that the construction of these two problems requires to multiply the constraint matrix by the inverse basis matrix.

We first present an amelioration of the reduced problem method by adding parameters on many problem parts. The overall effect of these improvements is a better balance between the time spent on the three main subprocesses of our algorithm. The computational experiments give a reduction factor of the total computing time of 12 on fleet assignment instances compared to the commercial software CPLEX.

We then propose a generalization of the reduced problem method. In previous works, the implementation of the recent method does not allow to handle the linear programs that contain upper and lower bounds on variables. We thus modified the theory

and the implementation to handle them. Moreover, the implementation used two softwares: one to create the reduced and the complementary problems and one to solve them. We propose a way to construct and solve both problems with one software (CPLEX). We present a new theory formulation to allow the reduced problem method to start with an initial feasible solution instead of an initial basic feasible solution. The computational experiments show that our algorithm takes advantage of starting with a good initial solution while CPLEX is less predictable.

As we mentioned above, the construction of the reduced problem needs to multiply two matrices. When these matrices have a large size, the multiplication may be longer in computational time than the solution of the whole problem. The last part of this thesis proposes a new pricing criterion that identify a variable that can be entered into the basis with a non degenerate pivot. We call it the *positive edge* criterion. In other words, this pricing rule enables the construction of the reduced problem by using the original constraint coefficients. Thus, with the positive edge criteria, we can apply the reduced problem theory on large-scale problems (example: 100 000 constraints and 450 000 variables).

In short, the main contributions of this dissertation are: a better balance between the time spent on the three main subprocesses of the reduced problem algorithm; a generalization of the reduced problem theory for the linear programs with bounded variables; an implementation that uses only one software; a rewriting of the theory that allows to start our algorithm with an initial feasible solution instead of an initial basic feasible solution; a proposition of a new pricing rule that identifies a non degenerate pivot when one exists.

Table des matières

Liste des tableaux

Liste des figures

Introduction

L'algorithme du simplexe proposé par Dantzig (1949), au milieu du siècle dernier, est devenu un outil très utilisé pour la résolution de problèmes de programmation linéaire. Cet algorithme itératif permet de faire diminuer, pour un problème de minimisation, la valeur de la fonction objectif à chaque itération lorsqu'aucune dégénérescence n'est présente. Par contre, en présence de dégénérescence, la diminution de la valeur de la fonction objectif n'est pas assurée à chaque itération.

Dans cette thèse, il est question de la dégénérescence lors de la résolution d'un programme linéaire avec l'algorithme du simplexe. De ce fait, avant de présenter les objectifs de cette thèse, nous présentons une description détaillée de l'algorithme du simplexe ainsi que du principe de dégénérescence.

0.1 Algorithme du simplexe

Cette section expose l'algorithme du simplexe tel que présenté par Chvátal (1983). Pour simplifier la présentation de l'algorithme, nous omettons les preuves des théorèmes. Toutefois, il est possible de les retrouver dans Chvátal (1983) ou dans Nash et Sofer (1996). Nous débutons par la description de l'algorithme primal du simplexe. Ensuite, nous présentons l'algorithme dual et le principe de dégénérescence.

0.1.1 L'algorithme primal du simplexe

Soit un problème de programmation linéaire sous forme standard

$$\min_{x \in \mathbb{R}^n} c^\top x \quad s.c. \quad Ax = b,\ x \geq 0, \tag{LP}$$

où $c \in \mathbb{R}^n$ est le vecteur des coûts, A est la matrice $m \times n$ des contraintes, $b \in \mathbb{R}^m$ est le membre de droite et x est le vecteur des variables.

L'algorithme du simplexe est une méthode itérative finie qui résout des problèmes de programmation linéaire de manière analogue à la résolution d'un système d'équations linéaires. Plus précisément, l'algorithme du simplexe sépare les variables en deux groupes : les variables de base et celles hors base. On définit un ensemble de variables de base comme étant m variables de valeur non-négative d'une solution réalisable au LP. Ces variables sont linéairement indépendantes et forment donc une base de $\mathbb{R}^m$. Les variables hors-base sont les $n - m$ variables restantes et ont une valeur nulle.

Soit B l'ensemble des indices des variables de base et H l'ensemble des indices des variables hors-base, LP peut être réécrit :

$$\min_{x \in \mathbb{R}^n} c_{\text{B}}^\top x_{\text{B}} + c_{\text{H}}^\top x_{\text{H}} \quad s.c. \quad A_{\text{B}} x_{\text{B}} + A_{\text{H}} x_{\text{H}} = b,\ x_{\text{B}}, x_{\text{H}} \geq 0.$$

Une solution de base est une solution obtenue en posant $x_{\text{B}} = A_{\text{B}}^{-1} b = \bar{b}$ et $x_{\text{H}} = 0$. Chaque solution de base réalisable correspond à un point extrême du domaine réalisable.

Théorème 1 *S'il existe une solution optimale pour LP, alors il en existe une qui soit une solution de base.*

Pour trouver une solution optimale, le théorème 1 nous indique qu'il suffit de parcourir les points extrêmes du domaine. Il existe un nombre exponentiel de points extrêmes mais l'algorithme ne les parcourt pas tous. Il passe d'un point extrême à un autre en remplaçant une variable de base par une autre tout en ne détériorant pas la valeur de la fonction objectif.

En exprimant les variables de base en fonction des variables hors-base ($x_{\text{B}} = A_{\text{B}}^{-1}b - A_{\text{B}}^{-1}A_{\text{H}}x_{\text{H}}$), la fonction objectif peut se réécrire

$$z = c_{\text{B}}^{\top}A_{\text{B}}^{-1}b + (c_{\text{H}}^{\top} - c_{\text{B}}^{\top}A_{\text{B}}^{-1}A_{\text{H}})x_{\text{H}}.$$

La valeur de la fonction objectif de la solution courante est $z = c_{\text{B}}^{\top}A_{\text{B}}^{-1}b$ puisque $x_{\text{H}} = 0$. Le vecteur $\bar{c} = c_{\text{H}}^{\top} - c_{\text{B}}^{\top}A_{\text{B}}^{-1}A_{\text{H}}$ s'appelle le vecteur des coûts réduits. En augmentant une variable de coût réduit négatif, par son entrée en base, on remarque que la valeur de la fonction objectif diminue.

Théorème 2 *Pour une base A_B réalisable donnée, si toutes les variables hors-base ont un coût réduit non-négatif, alors la base A_B est optimale pour LP.*

S'il existe une variable de coût réduit négatif, on peut entrer cette variable en base. Soit e l'indice de la variable entrante. Pour sortir une variable de la base et respecter les contraintes de non-négativité, la variable sortante doit être choisie de manière à ce que

$$x_{\text{B}} = A_{\text{B}}^{-1}b - A_{\text{B}}^{-1}A_e x_e \geq 0$$

On peut donc augmenter la variable entrante jusqu'à ce qu'une variable en base deviennent nulle. Le choix de l'indice s de la variable sortante est donc

$$s \in \underset{i=1,\ldots,m}{\operatorname{argmin}} \left\{ \frac{(A_{\text{B}}^{-1}b)_i}{(A_{\text{B}}^{-1}A_e)_i} \mid (A_{\text{B}}^{-1}A_e)_i > 0 \right\}.$$

Si l'ensemble $\{i = 1, \ldots, m \mid (A_B^{-1}A_e)_i > 0\}$ est vide, le problème est non-borné puisque la variable entrante peut être augmentée jusqu'à l'infini. Par contre, si l'ensemble précédent est non-vide, un pivotage de Gauss est effectué pour calculer la nouvelle base et son inverse. Le choix des variables entrante et sortante ainsi que le pivotage constitue une itération. L'algorithme effectue des itérations jusqu'à ce que le problème soit optimal (aucune variable de coût réduit négatif n'est identifiée) ou jusqu'à ce qu'une variable entrante indique que le problème est non-borné. L'algorithme primal du simplexe conserve la faisabilité primale ($x \geq 0$) tout au long de son exécution.

Notons que l'algorithme primal du simplexe doit débuter avec une solution primale de base réalisable. Toutefois, si aucune n'est connue, une phase I de l'algorithme peut être effectuée en ajoutant au problème un vecteur de variables artificielles a de taille m et en résolvant

$$\min z_a = \sum_{i=1}^{m} a \quad s.c. \quad Ax + a = b,\ x, a \geq 0. \tag{0.1}$$

La solution de base initiale correspond à la solution $a = b$. Si le problème (LP) est réalisable, alors la solution optimale de (0.1) est $a^* = 0$, sa valeur optimale est $z_a^* = 0$ et la solution de base $x_B = A_B^{-1}b$ est réalisable pour LP.

0.1.2 L'algorithme dual du simplexe

Le problème LP peut être vu comme le modèle primal. Le modèle dual de ce même problème est

$$\max_{\pi \in \mathbb{R}^m} b^\top \pi \quad \text{subject to } A^\top x \leq c. \tag{0.2}$$

Un théorème important de la théorie de la programmation linéaire est le *théorème de la dualité* suivant.

Théorème 3 *Si le problème primal a une solution optimale x^*, le problème dual a une solution optimale π^* telle que $c^\top x^* = b^\top \pi^*$.*

Tel que mentionné dans la section précédente, l'algorithme primal conserve la faisabilité de la solution primale en maintenant $x_B = \bar{b} \geq 0$ tout au long de l'algorithme. Les coûts réduits de certaines variables sont négatifs tant que le problème n'est pas à l'optimalité. Pour l'algorithme dual, c'est l'inverse : tous les coûts réduits sont non-négatifs et certains membres de droite sont négatifs jusqu'à ce que l'on atteigne l'optimalité. De plus, contrairement à l'algorithme primal, l'algorithme dual sélectionne la variable sortante avant de sélectionner la variable entrante.

L'algorithme dual débute avec une solution de base duale réalisable. Il sélectionne la variable sortante en choisissant une ligne r telle que $\bar{b} < 0$. La variable sortante est la variable de base associée à la ligne r. La variable entrante est sélectionnée de façon à ce que tous les coûts réduits des variables restent non-négatifs. Pour ce faire, on doit choisir l'indice e de la variable entrante en calculant le test de ratio suivant :

$$e \in \underset{j=1,\ldots,m}{\operatorname{argmin}} \left\{ \frac{-\bar{c}_j}{\bar{a}_{rj}} \mid \bar{a}_{rj} < 0 \right\}.$$

Le pivot se fait de façon identique à celui du pivot primal. L'algorithme dual se termine lorsque tous les membres de droite ($\bar{b}$) sont non-négatifs.

0.1.3 La dégénérescence

La dégénérescence d'un programme linéaire survient lorsque le nombre de variables non nulles est strictement inférieur à la taille de la base. Dans ce cas, il arrive qu'il soit impossible d'augmenter la valeur de la variable entrante. En effet, si une variable x^k en base est nulle et que $(A_B^{-1}A_e)^k > 0$, où l'indice supérieur k représente la ligne associée à la variable k, alors augmenter x_e donnerait une valeur négative à la variable x^k puisque $x_B = A_B^{-1}b - A_B^{-1}A_e x_e$. La variable x_e entre donc en base avec une valeur nulle. La valeur de la fonction objectif reste par conséquent inchangée. Un pivot de ce type est appelé un pivot dégénéré.

Lorsque plusieurs pivots dégénérés sont effectués, la performance de l'algorithme est diminuée et la convergence de l'algorithme n'est pas assurée puisqu'il y a un risque de cyclage. Certaines règles de pivots (Bland, 1977; Fukuda, 1982; Terlaky et Sushong, 1993) ont été développées pour assurer la convergence de l'algorithme. Toutefois, ces règles n'améliorent pas la performance de l'algorithme mais permettent seulement de s'assurer de converger vers la solution optimale en un nombre fini d'itérations.

0.2 Objectifs de la thèse

Dans le but d'améliorer la performance de la résolution des problèmes dégénérés, Pan (1998) propose de travailler itérativement sur une suite de problèmes réduits en variables et en contraintes. Plus précisément, Pan (1998) propose d'éliminer les contraintes associées aux variables dégénérées. Pour que ce nouveau problème reste cohérent, les variables qui, une fois entrées en base, violeraient les contraintes enlevées, sont éliminées. Pour atteindre l'optimalité, Pan (1998) modifie le problème réduit en

ajoutant des variables et des contraintes enlevées. Le choix de ces variables se fait en fonction de leur coût réduit. Puisque les variables duales sont inconnues sur les contraintes enlevées, une valeur nulle leur est imposée. Notons que la construction du problème réduit nécessite la multiplication de l'inverse de la base et de la matrice des contraintes.

El Hallaoui *et al.* (2007) ont amélioré la méthode de Pan. Ils ajoutent la résolution d'un problème pour sélectionner les variables à entrer dans le problème réduit. Ce problème, appelé *problème complémentaire*, permet d'entrer un groupe de variables qui assure une diminution de la valeur de la fonction objectif. L'implémentation de cette méthode appelée *Improved Primal Simplex* (IPS) est toutefois de base dans l'article précédent. Le facteur de diminution du temps total de résolution de IPS comparativement à CPLEX est de 2.29 sur des problèmes d'horaires de chauffeurs d'autobus et sur des problèmes de répartition de flottes d'avions.

Le premier objectif de cette thèse est de développer et d'implémenter IPS de manière à obtenir tout le potentiel théorique de cette méthode. Nous paramétrons tout d'abord certains aspects de l'algorithme et nous ajoutons une sous-procédure qui permet d'accroître l'efficacité de l'augmentation du problème réduit. De plus, nous étudions plusieurs méthodes pour résoudre le problème complémentaire. Les résultats obtenus lors de nos tests sur des instances de répartition de flottes d'avions démontrent l'efficacité de notre méthode. Notons que ces instances proviennent d'un autre groupe d'instances que celles utilisées par El Hallaoui *et al.* (2007). En effet, nous obtenons un facteur de diminution du temps global de résolution de 12 par rapport au logiciel commercial CPLEX.

Le deuxième objectif de ce présent travail est de généraliser à la fois la théorie mathématique et l'implémentation de IPS. Les versions de Pan (1998) et d'El Hallaoui *et al.* (2007) ne permettent pas de manipuler les problèmes qui contiennent des bornes

inférieures et supérieures sur les variables. Notre version de l'algorithme le peut. Ensuite, IPS doit débuter avec une base réalisable initiale pour pouvoir construire le problème réduit ; notre algorithme permet de commencer avec une solution réalisable initiale. De plus, le programme informatique de IPS utilise deux logiciels : UMFPACK (Davis et Duff, 1997) pour la construction des problèmes réduits et complémentaires et CPLEX pour la résolution de ces problèmes. Notre algorithme n'utilise qu'un seul logiciel, soit CPLEX.

Le troisième objectif est de résoudre des problèmes de très grande taille avec la méthode IPS. Il faut donc construire le problème réduit en évitant de multiplier l'inverse de la base par la matrice des contraintes. Pour ce faire, nous développons un nouveau critère d'évaluation des variables. Ce critère possède une complexité de calcul égale à celui du calcul du coût réduit. Non seulement ce critère permet la construction du problème réduit mais indique, de façon générale, si une variable entrera en base avec une valeur nulle ou positive. L'élaboration de ce nouveau critère est une avancée significative et importante dans le domaine de la programmation linéaire.

Cette thèse est organisée comme suit. Le premier chapitre est une revue critique de la littérature. Nous y présentons les méthodes itératives qui résolvent des problèmes réduits en variables, réduits en contraintes et réduits en variables et en contraintes. Le deuxième chapitre est l'organisation de la thèse. Le troisième chapitre présente les améliorations de IPS. Le chapitre suivant expose la généralisation de IPS. Le dernier chapitre présente la résolution de problèmes dégénérés de très grande taille à l'aide de la méthode du problème réduit. Cet objectif a été réalisé grâce à l'élaboration d'un nouveau critère permettant d'identifier les pivots non dégénérés. Enfin une discussion générale et une conclusion sont présentées dans la dernière partie de cette thèse.

Chapitre 1

Revue de la littérature

Il est possible de diminuer la taille d'un programme linéaire en utilisant deux principales stratégies : réduire le nombre de variables ou réduire le nombre de contraintes. On peut également combiner ces deux stratégies, ce qui permet de diminuer encore plus la taille du problème. Toutefois, dans tous les cas, on doit résoudre itérativement plusieurs problèmes plus petits. Le temps de résolution global est donc la somme des temps de résolution de chaque petit problème. Pour qu'une méthode soit efficace, il faut que le temps de résolution d'une itération multiplié par le nombre d'itérations soit inférieur au temps de résolution du problème original. La première section de cette revue de la littérature présente ces deux stratégies.

La présence de dégénérescence lors de la résolution d'un programme linéaire diminue la performance de l'algorithme du simplexe. Toutefois des méthodes de perturbations ont été proposées pour diminuer l'effet négatif de la dégénérescence sur le temps de calcul de l'algorithme. Plus récemment, des méthodes qui réduisent en variables et en contraintes les problèmes dégénérés ont été développées. La deuxième section de cette revue de la littérature présente les méthodes qui se consacrent à la dégénérescence.

1.1 Reduction du problème

La première partie de cette section présente les méthodes permettant de réduire en contraintes des progammes linéaires. La seconde partie présente les méthodes pour réduire les programmes linéaires en variables.

1.1.1 Réduction du nombre de contraintes

Une des premières idées proposées pour réduire le nombre de contraintes d'un programme linéaire est de relaxer (d'enlever) des contraintes et de résoudre le problème restreint (Geoffrion, 1968). S'il n'existe aucune solution réalisable à ce dernier problème, alors le problème original est irréalisable. Par contre, s'il existe une solution réalisable et que cette solution satisfait les contraintes relaxées, alors cette solution est optimale pour le problème original. Toutefois, dans le cas où la solution au problème restreint est réalisable mais qu'elle viole certaines contraintes relaxées, on ajoute celles-ci au problème restreint et on réoptimise. Cette méthode permet d'obtenir une amélioration significative du temps de résolution lorsqu'un grand nombre de contraintes qui ont été relaxées ne sont pas réintroduites.

La relaxation lagrangienne (Geoffrion, 1974; Fisher, 1981) est une technique qui transfère les contraintes (généralement un sous-ensemble) dans la fonction objectif en utilisant des multiplicateurs de Lagrange. La valeur de la solution de cette relaxation est donc une borne inférieure (problème de minimisation) sur la valeur optimale du problème original. On peut trouver la valeur des multiplicateurs (la solution au problème dual lagrangien) en utilisant un algorithme pour les problèmes convexes non différentiables tels l'algorithme du sous-gradient (Held et Karp, 1971) ou par une méthode de faisceaux (Hiriart-Urruty et Lemaréchal, 1991).

La relaxation succédanée (*surrogate relaxation*) (Glover, 1968) est une méthode duale qui remplace toutes les contraintes par une ou quelques contraintes substituts. Ces dernières sont des combinaisons linéaires non négatives des contraintes du problème. Le problème dual succédané consiste à trouver les coefficients des combinaisons linéaires qui produisent la meilleure borne. Karwan et Rardin (1979) ont montré dans leur article que cette approche est similaire en plusieurs points à la relaxation lagrangienne. Un de ces points est que la relaxation succédanée ne fournit pas, ou très rarement, une solution primale réalisable. Comme pour la relaxation lagrangienne, la relaxation succédanée peut être combinée à une méthode de faisceaux.

Différentes méthodes d'agrégation de contraintes ont été proposées avant les années 1990 (Rogers *et al.*, 1991) mais celles-ci sont, dans la plupart des cas, statiques. En effet, les agrégations présentées sont exécutées au début de l'algorithme et restent inchangées tout au long du processus de résolution. De plus, ces méthodes donnent des solutions approximatives qui ne sont pas nécessairement réalisables. Enfin, ces études analysent l'erreur sur la solution causée par l'agrégation et discutent des bonnes et des mauvaises agrégations.

Par ailleurs, certaines études proposent des méthodes d'agrégation dynamique. C'est le cas des travaux effectués par Mendelssohn (1982) et Shetty et Taylor (1987). Tout d'abord, Mendelssohn (1982) présente une méthode d'agrégation de contraintes pour les problèmes linéaires provenant d'un processus de décision markovien. L'algorithme résout à chaque itération un problème où les contraintes sont agrégées et une série de sous-problèmes linéaires qui permet d'estimer les valeurs pour toutes les variables duales du problème original. Ces valeurs sont ensuite utilisées pour définir le problème agrégé de la prochaine itération. Malheureusement, Mendelssohn (1982) ne présente aucun résultat de sa méthode sur des problèmes pratiques.

Quant aux travaux de Shetty et Taylor (1987), leur méthode d'agrégation a été développée pour les programmes linéaires généraux. Leur méthode agrège les contraintes une fois, puis désagrège successivement les contraintes lorsque celles-ci sont violées par la solution du problème agrégé. Des bornes supérieures et inférieures sont calculées et servent de critère d'arrêt de la procédure. Enfin, les résultats exposés dans leur étude montrent une réduction substantielle du temps de calcul pour trouver une solution approximative mais démontrent l'inefficacité (aucune réduction notable du temps de calcul) de la méthode à trouver une solution optimale. Ces tests ont été exécutés sur des problèmes comportant 200 contraintes.

Nous avons présenté dans cette première section les différentes études portant sur la réduction du nombre de contraintes dans les problèmes linéaires. Dans la section suivante, nous présentons les recherches discutant de la réduction du nombre de variables.

1.1.2 Réduction du nombre de variables

La méthode de génération de colonnes permet de diminuer la taille d'un problème en réduisant le nombre de variables. Proposée par Dantzig et Wolfe (1960) et développée par Gilmore et Gomory (1961, 1963), cette méthode consiste à résoudre une séquence de problèmes maîtres restreints (PMR) et de sous-problèmes. Pour définir un PMR, ils utilisent un sous-ensemble de variables. Ils résolvent le PMR, puis ils recherchent, en résolvant un sous-problème, les variables à coût réduit négatif en utilisant la solution duale du PMR.

Cette méthode permet, grâce à l'utilisation d'un sous-ensemble de variables, de résoudre des problèmes qui n'auraient pu être résolus directement à cause du trop grand nombre de variables. En effet, certains problèmes ont un nombre de variables

tellement élevé qu'ils ne peuvent être traités directement dans une procédure de résolution.

La durée globale de la méthode de génération de colonnes est la somme des temps passés à chaque itération et peut donc être diminuée en utilisant l'une ou l'autre des approches suivantes. La première consiste à diminuer la durée de résolution de chaque itération, tandis que la seconde consiste à réduire le nombre d'itérations (pour le problème maître ou pour le sous-problème). Évidemment, pour diminuer globalement le temps de résolution, on ne doit pas trop augmenter le nombre d'itérations (respectivement le temps de résolution de chaque itération) pour la première approche (resp. deuxième approche).

Dans le but de diminuer le temps de résolution de chaque itération, Gamache *et al.* (1998) présentent une technique d'accélération qui réside dans la résolution de sous-problèmes restreints ou dans la résolution de sous-ensembles des sous-problèmes à chaque itération. Cette technique, appelée *partial pricing*, est souvent utilisée lorsqu'une large proportion du temps de calcul est investie dans la résolution des sous-problèmes. Par contre, puisque l'information obtenue des sous-problèmes est plus faible, le nombre d'itérations peut augmenter.

Par ailleurs, Kelley (1960) propose une approche duale permettant de trouver des bornes supérieures au problème maître en résolvant une séquence de problèmes maîtres restreints et de sous-problèmes qui génèrent des coupes dans l'espace dual. Cependant, cette méthode possède de piètres propriétés de convergence théorique. En effet, la faible convergence découle du manque de garantie significative que l'on a sur la diminution de l'espace dual obtenue par l'ajout des nouvelles coupes générées. Cette approche est similaire à la méthode de génération de colonnes mais d'un point de vue dual.

Il est possible de diminuer le nombre d'itérations de la méthode de génération de colonnes en utilisant la méthode de faisceaux (Hiriart-Urruty et Lemaréchal, 1991), la *méthode analytique de plans coupants centrés* (ACCPM) (Goffin et Vial, 1990, 1999) ou n'importe quelle méthode de points intérieurs. Certaines de ces méthodes ont été appliquées et sont présentées plus loin.

Freling *et al.* (1999) proposent de résoudre approximativement chaque problème maître restreint en utilisant la relaxation lagrangienne. L'avantage de cette méthode est qu'elle utilise environ le même nombre d'itérations que la méthode de génération de colonnes si la solution duale approximative est suffisamment près de la solution optimale. Par contre, la relaxation lagrangienne ne calcule pas directement de solution primale au problème restreint. On sait que la solution primale est utilisée pour trouver une solution heuristique entière réalisable en plus d'être employée pour définir les règles de branchement lorsque la génération de colonnes est utilisée avec une méthode de *branch-and-price*. Pour contrer cette déficience, Barahona et Anbil (2000) ont développé un algorithme de sous-gradient qui calcule une solution primale approximative.

Lorsque l'on résout le problème maître avec la relaxation lagrangienne, Brooks et Geoffrion (1966) ont démontré que la valeur de la solution optimale obtenue est la même que celle obtenue par la génération de colonnes. La relaxation lagrangienne est alors une formulation duale du problème de génération de colonnes où l'ensemble des colonnes de cette dernière formulation est remplacé implicitement par l'optimisation des sous-problèmes. Une convergence asymptotique peut être obtenue en utilisant une méthode de descente qui choisit le dernier sous-gradient, calculé à l'aide d'un sous-problème, comme direction.

Goffin et Vial (1990, 1999) se sont inspirés des travaux de Kelley (1960) pour développer l'ACCPM qui possède de meilleures propriétés de convergence que celle de

Kelley (1960). Dans ACCPM, une fonction barrière logarithmique est maximisée sur l'ensemble dual permettant ainsi d'obtenir les prochaines valeurs des multiplicateurs lagrangiens à utiliser pour résoudre les sous-problèmes.

Dans le même ordre d'idées, du Merle *et al.* (1999) ont développé une méthode afin d'obtenir une meilleure convergence pratique aux problèmes séquentiels de résolution des problèmes duaux. Leur approche est basée sur la méthode de faisceaux décrite par Hiriart-Urruty et Lemaréchal (1991), qui ajoute au modèle linéarisé du sous-problème lagrangien une pénalité quadratique centrée à l'itération duale courante. Toutefois, du Merle *et al.* (1999) utilisent plutôt une fonction linéaire par morceaux comme pénalité. L'avantage est qu'il est beaucoup plus facile de résoudre le problème maître utilisant une pénalité linéaire par morceaux car on peut se servir de l'algorithme du simplexe. De plus, en prenant avantage de l'information *a priori* sur les solutions duales optimales du problème maître, du Merle *et al.* (1999) obtiennent une réduction importante du temps de résolution global.

1.2 Dégénérescence

Cette section de la revue de la littérature se concentre sur la présence de la dégénérescence lors de la résolution avec l'algorithme du simplexe.

1.2.1 Méthodes de perturbations

Introduite par Charnes (1952), la méthode de perturbations modifie les membres de droite de manière à ce que toutes les variables de base soient non nulles. Charnes (1952) propose de perturber les membres de droite dès le début de l'algorithme.

Avec ces perturbations, l'auteur démontre que le nouveau problème ne peut pas être dégénéré. Cette méthode prouve la convergence de l'algorithme du simplexe mais ne permet pas d'améliorer la performance de celui-ci.

Wolfe (1963) propose plutôt d'attendre que l'algorithme se retrouve avec une solution dégénérée avant de perturber les membres de droite. Soit I_0 l'ensemble des indices des variables de base nulle. Si l'optimalité n'est pas atteinte, une variable de coût réduit négatif sera sélectionnée comme variable entrante (x_e). Si $(A_B^{-1}A)_e^i > 0$ pour $i \in I_0$, alors le test de ratio identifiera une ligne $i \in I_0$ comme ligne de pivot. Dans ce cas, l'algorithme effectuera un pivot dégénéré. Dans une telle situation, Wolfe (1963) propose de perturber le membre de droite :

$$b_0 = b + A_B\epsilon$$

où $\epsilon^j = 0$ si $j \notin I_0$ et ϵ^j est une constante positive arbitraire si $j \in I_0$. Cette perturbation reste en vigueur tant que les pivots se font sur le même point extrême. La solution de base devient $x_B = x_B + \epsilon$ et le prochain pivot est non dégénéré. Si ce pivot crée à nouveau une solution dégénérée, il suffit d'appeler la méthode de perturbations récursivement. En effectuant ces perturbations tout au long de l'algorithme, ce dernier n'effectue aucun pivot dégénéré.

Plus récemment, George et Osborne (1993) ont montré que la récursivité de l'algorithme n'est pas nécessaire. En effet, il est possible d'obtenir la même convergence en relançant l'itération avec de nouvelles perturbations lorsqu'un pivot sur le problème perturbé donne une solution dégénérée. Finalement, Gass (1993) présente une étude sur les différentes méthodes de perturbations utilisées par quatre logiciels de programmation linéaire.

Bien que les méthodes de perturbations éliminent le cyclage, elles augmentent le nombre de points extrêmes du polyèdre initial. Avec ces méthodes, la performance de l'algorithme du simplexe n'est pas vraiment améliorée.

1.2.2 Réduction des problèmes dégénérés

Cette section se concentre sur la réduction de contraintes et de variables d'un problème dégénéré.

Pan (1998) propose d'utiliser une base réduite en variables et en contraintes. Pour ce faire, on débute avec une solution de base initiale où l'on identifie les p variables non nulles. Ensuite, on enlève les $m-p$ contraintes associées aux variables dégénérées. On se retrouve donc avec une base $p \times p$ non dégénérée. Évidemment, pour conserver la faisibilité du problème, on doit enlever toutes les variables *incompatibles*. Une variable est dite *incompatible* (respectivement *compatible*) si elle ne peut (resp. elle peut) entrer en base et prendre une valeur positive sans violer les contraintes enlevées précédemment. Le problème réduit contient donc la base $p \times p$ et les variables compatibles.

Lorsque le problème réduit est défini, la méthode calcule à chaque itération le coût réduit de toutes les colonnes. Pour ce faire, Pan (1998) utilise les variables duales du problème réduit et donne arbitrairement une valeur nulle aux variables duales des contraintes enlevées. Si une variable incompatible est sélectionnée pour entrer dans la base, alors on doit réintroduire dans le problème réduit une ou plusieurs contraintes précédemment enlevées. Pan (1998) obtient un facteur de diminution de 4 à 5 pour sa méthode comparativement à un algorithme du simplexe que lui-même a programmé. Mentionnons que pour obtenir la matrice des coefficients du

problème réduit, la méthode multiplie l'inverse de la base par la matrice originale des contraintes. Cette façon de faire peut s'avérer laborieuse dans le cadre de la résolution de problèmes de très grande taille.

Omer (2006) a implanté l'agorithme de Pan (1998) en le modifiant quelque peu. La modification la plus importante est que sa méthode permet de réduire, en cours de résolution, le problème réduit si celui-ci est trop dégénéré. Par contre, l'implantation d'Omer (2006) n'est pas assez efficace et ne permet donc pas de comparer le temps de calcul de sa méthode avec un algorithme du simplexe commercial. Néanmoins, le nombre d'itérations qu'effectue l'algorithme permet de percevoir le potentiel de la méthode.

Il est toutefois important de noter qu'une majorité des problèmes testés par Pan (1998) et Omer (2006) ont la particularité d'avoir des valeurs optimales nulles pour les variables duales, ce qui leur permet d'obtenir de bons résultats numériques puisqu'ils calculent les coûts réduits des variables incompatibles en supposant des valeurs nulles pour les variables duales des contraintes enlevées. Les problèmes testés biaisent donc leurs résultats.

Dans la même veine, El Hallaoui *et al.* (2005) ont développé une méthode d'agrégation dynamique de contraintes (DCA) pour les problèmes de partitionnement. Leur méthode part également d'une solution de base pour réduire le problème original. Cependant, la réduction se fait différemment. En effet, leur méthode conserve une seule ligne de chaque groupe de contraintes qui deviennent identiques pour les colonnes des variables en base, dont la valeur est différente de zéro. Cette étape est appelée l'agrégation. De plus, contrairement à Pan (1998), la méthode de El Hallaoui *et al.* (2005) effectue des itérations, appelées mineures, qui considèrent seulement les variables compatibles avec la base. Lorsque le problème réduit est à l'optimalité, leur algorithme désagrège les variables duales en résolvant un sous-

problème, nommé problème complémentaire (CP), qui est en fait un problème de plus court chemin. Ensuite, les valeurs des variables duales désagrégées sont utilisées pour calculer les coûts réduits des variables incompatibles. Enfin, leur algorithme permet, en changeant l'agrégation courante, de réduire la dimension du problème réduit en cours de résolution si celui-ci devient trop dégénéré.La réduction du temps de calcul global sur ces problèmes est de plus de 80% par rapport à la génération de colonnes seule.

Afin d'améliorer DCA, El Hallaoui *et al.* (2008b) ont développé la méthode multi-phases d'agrégation dynamique de contraintes (MPDCA). Cette dernière est similaire à DCA sauf pour la sélection des variables incompatibles qui doivent entrer dans le problème réduit. En effet, MPDCA effectue plusieurs phases lors de la résolution de CP. Dans la première phase, seules les variables ayant une incompatibilité sont considérées. Si aucune variable n'est sélectionnée, l'algorithme passe à la deuxième phase où les variables ayant au plus deux incompatibilités sont ajoutées dans CP. S'il n'y a toujours pas de variables sélectionnées, alors la troisième phase ajoute toutes les variables incompatibles (pour une définition plus détaillée sur le nombre d'incompatibilités d'une variable, nous référons le lecteur à El Hallaoui *et al.* (2008b)) Cet ajout permet de garder le problème réduit moins dégénéré. Enfin, ils ont intégré leur méthode à la méthode de génération de colonnes et l'ont testée sur des problèmes d'horaires de chauffeurs d'autobus en milieu urbain. Le facteur de réduction du temps total de ces problèmes est de plus de 23 comparativement au temps de la méthode de génération de colonnes (El Hallaoui *et al.*, 2008a).

À la suite de ces recherches, El Hallaoui *et al.* (2007) se sont inspirés de la méthode de Pan (1998) et de DCA pour développer un algorithme appelé *Improved Primal Simplex* (IPS). La contribution principale de leur algorithme concerne la désagrégation des variables duales. Puisque la procédure de DCA qui désagrège les variables duales ne peut être utilisée dans le contexte de l'agrégation de Pan (1998), El Hallaoui *et al.*

(2007) ont proposé une autre approche. Au lieu de désagréger les variables duales pour ensuite calculer les coûts réduits, El Hallaoui *et al.* (2007) propose de résoudre un problème complémentaire qui sélectionne directement les variables incompatibles à entrer dans le problème réduit.

Dans leur article, El Hallaoui *et al.* (2007) ont démontré qu'une solution optimale au problème réduit est optimale pour le problème original si et seulement si le problème complémentaire est non réalisable. De plus, ils ont démontré que la valeur de la fonction objectif diminue strictement lorsque l'on entre en base toutes les variables sélectionnées lors de la résolution du problème complémentaire.

Toutefois, les résultats théoriques de El Hallaoui *et al.* (2007) sont obtenus dans le contexte où le problème réduit est toujours non dégénéré. En pratique, il est beaucoup trop coûteux en terme de temps de calcul de réduire le problème réduit à chaque fois que la dégénérescence apparaît. De plus, lorsque le problème réduit est dégénéré, la diminution de l'objectif lors de l'entrée en base des variables incompatibles n'est pas assurée. Enfin, l'implémentation de El Hallaoui *et al.* (2007) utilise deux logiciels, ce qui a comme inconvénients de donner lieu à certaines erreurs de tolérance. Même si ces erreurs ne remettent pas en cause l'exactitude de la méthode, elles diminuent quelque peu son efficacité.

Cette revue de la littérature a présenté les différentes études portant sur la réduction du nombre de contraintes, sur la réduction du nombre de variables et sur la réduction simultanée du nombre de contraintes et de variables d'un programme linéaire. Dans la première section, nous avons présenté la relaxation lagrangienne et la relaxation succédanée en plus de présenter des méthodes d'agrégation statique et dynamique de contraintes. La deuxième section, quant à elle, a présenté la méthode de génération de colonnes ainsi que la méthode de Kelley (1960). Enfin, nous avons présenté la

méthode du problème réduit en variables et en contraintes proposée par Pan (1998) et améliorée par El Hallaoui *et al.* (2005, 2007) et Omer (2006).

Chapitre 2

Organisation de la thèse

Dans cette thèse, nous nous penchons sur la résolution par la méthode du simplexe de problèmes linéaires dégénérés. Il est bien connu que la résolution de ce type de problèmes amène l'algorithme du simplexe à effectuer des pivots qui ne diminuent pas toujours la valeur de l'objectif.

Dans la revue de la littérature précédemment exposée, nous avons présenté des méthodes qui permettent d'améliorer la performance de la méthode du simplexe en réduisant la taille des problèmes linéaires originaux. Notons que la principale idée lors de la réduction de la taille des problèmes linéaires est de résoudre itérativement plusieurs problèmes réduits. Pour réduire la taille des problèmes, certains auteurs proposent de réduire le nombre de contraintes, d'autres proposent de diminuer le nombre de variables.

Pour les problèmes dégénérés, Pan (1998) proposent de réduire à la fois le nombre de variables et le nombre de contraintes. Quelques auteurs ont amélioré cette méthode. Toutefois, comme nous l'avons mentionné, les méthodes exposées présentent certains inconvénients et ne sont pas adaptées aux problèmes linéaires généraux dégénérés. De plus, l'implémentation de ces méthodes est simpliste.

À partir de ces informations, le premier objectif de cette thèse est d'implémenter une version améliorée et efficace de la méthode IPS. Dans le chapitre suivant, nous proposons une version améliorée de la méthode IPS de El Hallaoui *et al.* (2007). Ce chapitre décrit la méthode de Pan (1998) et l'implémentation de El Hallaoui *et al.* (2007). Nous présentons ensuite les contributions de notre recherche. Ces contributions sont l'ajout de paramètres permettant de répartir adéquatement le temps global de résolution dans les trois sous-procédures de l'algorithme. De plus, nous ajoutons un élément théorique qui sert à diminuer la dégénérescence des problèmes réduits. Les résultats numériques démontrent l'efficacité de notre implémentation. En effet, un facteur de diminution du temps total de résolution de 12 est observé sur des instances de répartition de flottes d'avions.

Une des lacunes de IPS est que cette méthode n'est pas adaptée à tous les types de problèmes, et ce, autant sur le plan théorique que pratique. Dans le chapitre 4, nous généralisons IPS. Effectivement, nous modifions la théorie pour que cette méthode puisse manipuler les problèmes avec variables bornées. Ensuite, nous simplifions l'implémentation de la méthode par le retrait de l'utilisation du logiciel UMFPACK (Davis et Duff, 1997) en n'utilisant que le seul logiciel commercial CPLEX. De plus, nous montrons qu'il est possible de débuter l'algorithme avec une solution initiale plutôt qu'avec une base initiale. Enfin, nous montrons que notre algorithme tire presque toujours profit d'une solution initiale, contrairement à CPLEX qui peut ou non tirer profit d'une solution initiale.

La méthode que propose Pan (1998) nécessite la multiplication de deux matrices. À cause de cette multiplication, la méthode du problème réduit peut difficilement résoudre les problèmes de très grande taille. L'objectif principal de cette thèse est de trouver une façon de construire le problème réduit sans avoir à multiplier la matrice inverse de la base par la matrice des contraintes. Dans le dernier chapitre, nous proposons une façon d'y arriver. En effet, nous proposons un critère d'évaluation des

variables qui indique si un pivot sera dégénéré. Ce critère est une avancée importante dans le domaine de la programmation linéaire. Avec ce critère, il est maintenant facile en terme de calcul de construire le problème réduit. Ainsi, la méthode proposée par Pan (1998) peut s'appliquer aux problèmes de grande taille. Avec une implémentation de base, nous obtenons un facteur de diminution de 4.75 sur des instances réelles de planification de la main-d'oeuvre fournies par la compagnie AD OPT de Montréal.

En conclusion de cette thèse, nous revenons sur les limites des recherches antérieures, nous définissons nos objectifs de thèse, nous exposons brièvement nos réalisations et nos résultats de recherches et nous présentons des pistes de recherches futures.

Chapitre 3

A New Version of the Improved Primal Simplex for Degenerate Linear Programs

Vincent Raymond, François Soumis et Dominique Orban (2009). *Computers and Operations Research*, doi : 10.1016/j.cor.2009.03.020.

La méthode IPS développée par El Hallaoui *et al.* (2007) est un algorithme de réduction dynamique de contraintes qui permet d'accélérer le temps de résolution de problèmes dégénérés. L'idée principale de leur algorithme est de résoudre plusieurs problèmes réduits en variables et en contraintes qui sont très peu dégénérés. Leurs résultats numériques démontrent une réduction par un facteur de plus de 3 sur certains types de problèmes de répartition de flottes d'avions comparativement à la méthode du simplexe implanté dans le logiciel commercial CPLEX.

Cet article présente des stratégies d'amélioration et un calibrage de certains paramètres pour IPS. Ces paramètres permettent de distribuer le temps de résolution total entre chaque sous-procédure de manière à soutirer tout le potentiel de l'algorithme. De plus, nos recherches nous ont permis de choisir une meilleure méthode pour résoudre

le problème complémentaire proposé par El Hallaoui *et al.* (2007). Nous ajoutons de plus un mécanisme qui permet d'augmenter le problème réduit avec le minimum de lignes nécessaires pour l'ajout des variables sélectionnées dans le problème complémentaire. Enfin, pour un groupe d'instances, notre version d'IPS (IPS-2) atteint un facteur de réduction du temps total de résolution de plus de 12 comparativement à CPLEX.

Le travail de programmation a été effectué en langage C. Le programme informatique utilise les logiciels CPLEX version 9.1.3 et UMFPACK. Nos tests ont été effectués sur des instances de répartition de flottes d'avions et sur des problèmes d'horaires de chauffeurs d'autobus.

A New Version of the Improved Primal Simplex for Degenerate Linear Programs

VINCENT RAYMOND, FRANÇOIS SOUMIS AND DOMINIQUE ORBAN

École Polytechnique de Montréal and GERAD

January 2009

Abstract

The Improved Primal Simplex (IPS) algorithm (El Hallaoui *et al.*, 2007) is a dynamic constraint reduction method particularly effective on degenerate linear programs. It is able to achieve a reduction in CPU time of over a factor of three on some problems compared to the commercial implementation of the simplex method CPLEX. We present a number of further improvements and effective parameter choices for IPS. On certain types of degenerate problems, our improvements yield CPU times lower than those of CPLEX by a factor of twelve.

Keywords: Linear Programming, Primal Simplex, Degeneracy.

3.1 Introduction

We consider the solution of linear programs in standard form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \; c^T x \quad \text{subject to } Ax = b, \; x \geq 0, \tag{LP}$$

where $c \in \mathbb{R}^n$ is the cost, A is the $m \times n$ constraint matrix and $b \in \mathbb{R}^m$ is the right-hand side. We are particularly interested in the so-called degenerate problems, on which the simplex algorithm (Dantzig, 1949) is likely to encounter degenerate pivots and possibly cycle.

In the presence of degeneracy, a common strategy to ensure convergence consists in imposing specific rules on the choice of a new basis, e.g., Bland (1977); Fukuda (1982); Terlaky and Sushong (1993). A different approach, proposed by Charnes (1952) and developed by Wolfe (1963) and Ryan and Osborne (1988), perturbs the right-hand side associated to degenerate variables. Although all strategies alleviate cycling, they do not improve the performance of the simplex algorithm on degenerate problems.

On the other hand, Pan (1998) uses a *reduced* basis; one with a smaller number of variables and constraints. His method starts with a initial basic feasible solution guess and identifies its p nonzero components. The $m - p$ constraints corresponding to the zero variables are removed, to leave a $p \times p$ non degenerate basis. To preserve feasibility, variables that cannot become basic without violating one of the $m - p$ eliminated constraints must be temporarily removed from the problem. Such variables are said to be *incompatible*—a definition which will be generalized in §3.2.1. The resulting *reduced problem* is solved and the reduced costs are computed by means of its dual variables. By convention, dual variables of (LP) corresponding to the $m - p$ eliminated constraints are set to zero. At the next iteration, if an incompatible

variable is to become basic, one of the eliminated constraints must be reintroduced to the reduced problem. When compared to his own primal simplex, Pan (1998) reports gains of a factor of 4-5.

Omer (2006) improves on the method of Pan (1998) by allowing further reductions while the reduced problem is being solved. This *dynamic* reduction method appears promising in terms of number of iterations. However, its implementation does not lend itself to comparisons with commercial implementations of the simplex algorithm. It is interesting to note that most problems tested by Omer (2006) and Pan (1998) have vanishing optimal dual variables and this characteristic gives advantage to their algorithms.

In the same vein, El Hallaoui *et al.* (2005) developed a *dynamic constraint aggregation* (DCA) method for partitioning problems. The method groups constraints that become identical with respect to nonzero basic variables, a stage referred to as *aggregation.* In the reduction phase, a single row per constraint group remains in the reduced problem. Once the reduced problem is solved, dual variables are recovered by *disaggregating* which corresponds to solving a shortest-path problem. The dual variables then allow to compute the reduced costs of incompatible variables. As Omer (2006), DCA makes provision for reaggregation, which allows to further reduce the reduced problem if the latter becomes more degenerate.

The *multi-phase dynamic constraint aggregation* (MDCA) (El Hallaoui *et al.*, 2008b), further improves on the DCA. The MDCA examines incompatible variables about to be added to the reduced problem in order of their number of incompatibilities. First, variables with a single incompatibility are considered. If none of them is selected to become basic, attention turns to variables with two incompatibilities. If again none of those is selected, all incompatible variables are added to the reduced problem. On a set of large bus driver scheduling problems in a column-generation framework, MDCA

reduces the solution time by a factor of more than five over the classical column-generation method.

The *Improved Primal Simplex* (IPS) (El Hallaoui *et al.*, 2007) combines ideas from Pan (1998) and DCA. Instead of disaggregating the dual variables to compute the reduced costs, the incompatible variables that are to become basic are obtained from the solution of a *complementary problem.* El Hallaoui *et al.* (2007) show that when the reduced problem is not degenerate, the objective function decreases at each iteration. In practice, reducing the problem until all degeneracy disappears is computationally costly. Partial reduction, however, yields good results.

In this paper, we provide a number of strategies and parameter values that speed up IPS. We pay attention to the number of variables in the reduced problem and hence, on the number of variables selected by the complementary problem. We also discuss how the later should be solved. To some extent, we optimize the number of further reductions of the reduced problem and propose a procedure that only adds independent rows to the reduced problem. Compared to CPLEX, we obtain a reduction of CPU time of a factor of twelve on some instances of *fleet assignment* (FA) problems.

The rest of this paper is organized as follows. §3.2 summarizes the results of Pan (1998) and El Hallaoui *et al.* (2007). In §3.3 we compare different methods to find a feasible solution to the complementary problem and to solve it. In §3.4, we present refinements of IPS. Numerical experimentations on realistic instances are reported in §3.5. Finally, we conclude in §3.6.

All tests in this paper are performed on a 2.8GHz PC running Linux and all comparisons are made with CPLEX 9.1.3.

3.1.1 Notation

If $x \in \mathbb{R}^n$ and $\mathrm{I} \subseteq \{1, \ldots, n\}$ is an index set, we denote x_{I} the subvector of x indexed by I. Similarly, if A is an $m \times n$ matrix, we denote A_{I} the $m \times |\mathrm{I}|$ matrix whose columns are index by I. If $\mathrm{J} = \{1, \ldots, n\} \setminus \mathrm{I}$, we allow ourselves to write $x = (x_{\mathrm{I}}, x_{\mathrm{J}})$ even though the indices in I and J may not appear in order. In the same way, we denote with superior indices the subset of rows.

The vector of all ones with dimension dictated by the context is denoted e and we note $A^{-\top}$ the inverse of the transpose of a square matrix A.

3.2 Background

In this section, we summarize the key properties of dynamic constraint aggregation methods developed by Pan (1998) and by El Hallaoui *et al.* (2007). We start with the construction of the reduced problem introduced by Pan (1998). Next, we review the disaggregation of dual variables of El Hallaoui *et al.* (2007) and recall the complete algorithm.

3.2.1 Reduced Problem

Let A_j be the jth column of A and B indexes basic variables for (LP). Let $\bar{x} = A_{\mathrm{B}}^{-1} b = A_{\mathrm{B}}^{-1} A x$ be a degenerate feasible point with $p > 0$ nonzero components, which, without loss of generality, we assume to be its first p components. Denote

$Q = A_{\mathrm{B}}^{-1}$ and partition

$$Q = \begin{bmatrix} Q^{\mathrm{P}} \\ Q^{\mathrm{Z}} \end{bmatrix}, \tag{3.1}$$

where Q^{Z} is the *compatibility matrix* formed of the last $m - p$ rows of Q. We have

$$\bar{x} = \begin{bmatrix} Q^{\mathrm{P}} b \\ Q^{\mathrm{Z}} b \end{bmatrix},$$

and therefore,

$$Q^{\mathrm{Z}} b = 0. \tag{3.2}$$

We begin with the following definition.

Definition 1 *Let M be an $m_1 \times m$ matrix with $m_1 < m$. The jth variable of (LP), x_j, is said to be* compatible *with M if and only if $MA_j = 0$.*

From Definition 1, we let $\mathrm{C} \subseteq \{1, \ldots, n\}$ denote the indices of variables that are compatible with Q^{Z} and $\mathrm{I} = \{1, \ldots, n\} \setminus \mathrm{C}$. Thus, x_{C} and x_{I} are the subvectors of x of compatible and incompatible variables, respectively.

We partition the cost vector c accordingly into c_{C} and c_{I}, and the columns of A into A_{C} and A_{I}. Upon premultiplying the equality constraints of (LP) by Q, we obtain $QAx = Qb$, which may be rewritten

$$\begin{bmatrix} Q^{\mathrm{P}} Ax \\ Q^{\mathrm{Z}} Ax \end{bmatrix} = \begin{bmatrix} Q^{\mathrm{P}} A_{\mathrm{C}} x_{\mathrm{C}} + Q^{\mathrm{P}} A_{\mathrm{I}} x_{\mathrm{I}} \\ Q^{\mathrm{Z}} A_{\mathrm{C}} x_{\mathrm{C}} + Q^{\mathrm{Z}} A_{\mathrm{I}} x_{\mathrm{I}} \end{bmatrix} = \begin{bmatrix} Q^{\mathrm{P}} b \\ 0 \end{bmatrix},$$

where we used (3.2). Since by definition x_{C} is compatible with Q^{Z}, we have $Q^{\mathrm{Z}} A_{\mathrm{C}} x_{\mathrm{C}} = 0$. Upon imposing $x_{\mathrm{I}} = 0$, we obtain the *reduced problem*

$$\underset{x_{\mathrm{C}}}{\text{minimize}}\ c_{\mathrm{C}}^{\mathrm{T}} x_{\mathrm{C}} \quad \text{subject to } Q^{\mathrm{P}} A_{\mathrm{C}} x_{\mathrm{C}} = Q^{\mathrm{P}} b,\ x_{\mathrm{C}} \geq 0. \tag{RP}$$

(RP) is potentially much smaller than (LP) and it is obtained from a degenerate basic feasible solution. It only depends on the compatible variables. Note that by construction, if x_{C} is feasible for (RP), then $(x_{\mathrm{C}}, 0)$ is feasible for (LP). This concept of compatibility is the same as in Pan (1998), i.e, the compatible variables can enter a basis for (RP) without violating the constraints of (LP) that have been omitted.

3.2.2 Complementary Problem

Assume x^*_{C} is optimal for (RP). Let P index the compatible basic variables, V index the compatible nonbasic variables and I index the incompatible variables, that is

$$\mathrm{P} = \{i \in \mathrm{C} \mid x^*_i \textit{ basic}\}, \quad \mathrm{V} = \{i \in \mathrm{C} \mid x^*_i \textit{ nonbasic}\}, \quad \text{and} \quad \mathrm{I} = \{1, \ldots, n\} \setminus \mathrm{C}.$$

Here, compatibility is understood with respect to Q^z that has been used to calculate the previous (RP). Then x^*_{C} is also optimal for (LP) if and only if all reduced costs, i.e., corresponding to all variables, compatible or not, are nonnegative. In other words, there must exist dual variables $\pi \in \mathbb{R}^m$ such that

$$c_j - A_j^\top \pi = 0 \quad \text{for all } j \in \mathrm{P}, \tag{3.3a}$$

$$c_j - A_j^\top \pi \geq 0 \quad \text{for all } j \in \mathrm{V}, \tag{3.3b}$$

$$c_j - A_j^\top \pi \geq 0 \quad \text{for all } j \in \mathrm{I}. \tag{3.3c}$$

From El Hallaoui *et al.* (2007, Proposition 3.1), if π satisfies (3.3a), it also satisfies (3.3b). The second set of conditions thus becomes redundant and it may be removed. El Hallaoui *et al.* (2007) propose to maximize the smallest reduced cost of incompatible variables while satisfying the zero values of the reduced cost of the

basic variables. Thus, they propose to solved the following problem:

$$\underset{\pi,y}{\text{maximize}}\ y \quad \text{subject to } c_{\text{P}} - A_{\text{P}}^{\top}\pi = 0,\ c_i - A_i^{\top}\pi \geq y,\ \forall i \in \text{I}. \tag{3.4}$$

By definition of P, the p columns of A_{P} are linearly independent and, possibly after row permutations, we may partition

$$A_{\text{P}} = \begin{bmatrix} A_{\text{P}}^{\text{P}} \\ A_{\text{P}}^{\text{N}} \end{bmatrix}$$

so A_{P}^{P} is nonsingular. The rows of A_{P}^{P} are the original rows of the matrix A associated with the nonzero variables. The rows of A_{N}^{P} are the other. The same partitioning, applied to A_{I} and π, yields

$$A_{\text{I}} = \begin{bmatrix} A_{\text{I}}^{\text{P}} \\ A_{\text{I}}^{\text{N}} \end{bmatrix} \quad \text{and} \quad \pi = \begin{bmatrix} \pi^{\text{P}} \\ \pi^{\text{N}} \end{bmatrix}.$$

Introducing this notation into the first set of constraints of (3.4), we obtain

$$c_{\text{P}} - (A_{\text{P}}^{\text{P}})^{\top}\pi^{\text{P}} - (A_{\text{P}}^{\text{N}})^{\top}\pi^{\text{N}} = 0,$$

and we may thus extract

$$\pi^{\text{P}} = (A_{\text{P}}^{\text{P}})^{-\top}c_{\text{P}} - (A_{\text{P}}^{\text{P}})^{-\top}(A_{\text{P}}^{\text{N}})^{\top}\pi^{\text{N}}.$$

Substituting the latter into the second set of constraints, (3.4) may be rewritten as the *complementary problem*

$$\underset{\pi,y}{\text{maximize}}\ y \tag{CP}$$

$$\text{s.t.}\quad y - \left(A_{\text{P}}^{\text{N}}(A_{\text{P}}^{\text{P}})^{-1}A_i^{\text{P}} - A_i^{\text{N}}\right)\pi^{\text{N}} \leq \left(c_i - ((A_{\text{P}}^{\text{P}})^{-1}A_i^{\text{P}})^{\top}c_{\text{P}}\right),\ \forall i \in \text{I}.$$

The following property (El Hallaoui *et al.*, 2007) justifies the use of (CP).

Proposition 1 *Let x_C^* be an optimal solution of (RP) and let y^* be an optimal solution of (CP). Then $(x_C^*, x_I^*) = (x_C^*, 0)$ is an optimal solution of (LP) if and only if $y^* \geq 0$.*

It is informative to consider the dual of (CP), the following *simplified dual*

$$\underset{v}{\text{minimize}} \ \left(c_{\text{I}} - ((A_{\text{P}}^{\text{P}})^{-1} A_{\text{I}}^{\text{P}})^{\top} c_{\text{P}}\right)^{\top} v \qquad \text{(SD)}$$

$$s.t. \quad \left(A_{\text{P}}^{\text{N}}(A_{\text{P}}^{\text{P}})^{-1} A_{\text{I}}^{\text{P}} - A_{\text{I}}^{\text{N}}\right) v = 0, \ e^{\top} v = 1, \ v \geq 0.$$

Note that (SD) possesses $m - p + 1$ equality constraints. Note also that from now, we refer to the pair (CP)-(SD) as the *complementary problem.* The next result (El Hallaoui *et al.*, 2007) links (SD) with (LP).

Theorem 4 *Let x be a (non-optimal) basic feasible point of (LP) with the corresponding basis A_B and with $p \leq m$ positive components. Let S be the index set of the positive components in a solution v of (SD) ($|S| \leq m - p + 1$). Let w be the convex combination of columns of A whose coefficients are the v_j, that is, $w = \sum_{j \in S} v_j A_j$. Then:*

1. *the variables in S can enter the basis A_B with positive value, decreasing the objective value of (LP);*
2. *w is linearly dependent with the columns corresponding to the p positive components of x.*

The algorithm proposed by El Hallaoui *et al.* (2007) is summarized in Algorithm 3.1. Note that the authors give a procedure to build a new basis at step 5 of Algorithm 3.1.

Algorithm 3.1 The Improved Primal Simplex

1: Choose an initial basis B.
2: Form and solve the reduced problem (RP).
3: Form and solve (SD). Let y^* be its optimal value.
4: If $y^* \geq 0$, the current solution $(x_C^*, 0)$ solves (LP).
5: Otherwise, build a new basis B' using the solution of (SD) and return to step 2.

3.3 Solving the Complementary Problem

El Hallaoui *et al.* (2007) find feasible solutions to (SD) with the primal simplex. The simplified dual (SD) must be solved at each iteration of the algorithm, which can be too costly. Moreover, all equality constraints of (SD) have a zero right-hand side, except for one. The primal simplex algorithm applied to (SD) is thus likely to perform a large number of degenerate pivots. In this section, we compare different solution methods for finding feasible solutions for (SD) and also for solving (SD) to optimality.

From Proposition 1, the objective value of (SD), which is equal to the objective value y of (CP), remains negative until optimality of (LP) is reached. An optimal solution v of (SD) produces an optimal value $y = \left(c_I - ((A_P^P)^{-1}A_I^P)^\top c_P\right)^\top v < 0$ and is thus such that

$$\left(A_P^N(A_P^P)^{-1}A_I^P - A_I^N\right)v = 0, \quad e^\top v = 1 \quad \text{and} \quad v \geq 0. \tag{3.5}$$

Then $\bar{v} = -v/y$ still satisfies $\bar{v} \geq 0$ and

$$\left(c_I - ((A_P^P)^{-1}A_I^P)^\top c_P\right)^\top \bar{v} = -1, \quad \left(A_P^N(A_P^P)^{-1}A_I^P - A_I^N\right)\bar{v} = 0. \tag{3.6}$$

Instead of computing an optimal solution of (SD), we find a $\bar{v}$ satisfying the latter conditions. Observe that if there is no feasible solution to (3.6), then (SD) has no feasible solution with a negative objective value. Moreover, if $\bar{v}$ is a feasible solution to (3.6), then $v = -y\bar{v}$ is a feasible solution with a negative objective value to (SD). We now compare various methods capable of identifying such a feasible point.

The first of the algorithms compared below is that of Filiz *et al.* (2001) which is a variant of the pivot algorithm of Terlaky and Sushong (1993). It seeks a feasible point for a degenerate linear program. In addition to pivoting on positive elements of primal feasible rows—as in the primal simplex—on negative elements of primal infeasible rows—as in the dual simplex—and to performing combinations of the two—as in the criss-cross method—the algorithm of Filiz *et al.* (2001) pivots on positive and negative elements of degenerate rows.

The second algorithm is the primal simplex method implemented in CPLEX. We use two versions of it. The first uses the steepest edge criterion and the second uses the default criterion for a variable to become basic. Both are tested with and without the presolved phase. Recall that the default criterion implemented in CPLEX is hybrid and combines the minimum reduced cost criterion with the *devex* criterion, an estimate of the steepest-edge criterion.

The third algorithm is the dual simplex method implemented in CPLEX. With this algorithm, (SD) should be solved to optimality since the first feasible point encountered will be optimal for (SD). Solving (SD) to optimality may result in a selection of better reduced cost variables to enter (RP). We compare the primal and the dual simplex to find an optimal solution to (SD).

Tables 3.1 and 3.2 report the computational effort on the first (SD) encountered for each of the instances described in §3.4.1. Table 3.1 compares four methods for finding a feasible solution of (SD). Table 3.2 compares two methods for finding an optimal solution of (SD). In the tables, *var* is the number of variables selected, i.e., the number of positive variables in a solution to (SD), and *np* is the number of pivots performed. We do not report CPU time for the algorithm of Filiz *et al.* (2001) since our implementation is quite simplistic. We do not report the number of pivots performed

Table 3.1: Comparison of four methods for finding a feasible solution to (SD).

	Filiz		Steepest Edge			Hybrid			Presolve	
inst	np	var	np	time	var	np	time	var	time	var
VCS1	240	2	1120	0.80	2	1748	0.30	39	0.03	2
VCS2	111	2	1003	0.40	2	760	0.07	9	0.02	2
VCS3	158	2	823	0.32	2	1027	0.11	20	0.03	2
VCS4	160	2	983	0.50	2	1540	0.19	38	0.02	2
VCS5	204	2	1133	0.46	6	982	0.10	33	0.03	2
VCS6	88	2	936	0.46	2	1797	0.29	60	0.02	2
VCS7	181	3	1211	0.73	2	1628	0.25	11	0.02	2
VCS8	203	2	1200	0.72	21	158	0.03	3	0.04	2
VCS9	100	4	1219	0.73	4	1937	0.36	76	0.05	2
VCS10	203	2	2	0.01	21	196	0.02	3	0.02	2
AVG	164	2.3	804	0.51	6.4	858	0.17	29.2	0.03	2.0

by the presolve in Table 3.1 because it is able to identify a feasible point before performing a single pivot with the primal algorithm.

It appears from Tables 3.1 and 3.2 that the primal simplex with presolve emerges as the most efficient both in terms of CPU time and number of pivots for finding a feasible point, while the dual simplex dominates in the search for an optimal solution. Apart from CPU time, another desirable characteristic is to identify combinations of a small number of variables to add to (RP): it appears that all methods identify few variables except for the primal simplex with hybrid criterion.

Finally, although the presolve followed by the primal simplex with hybrid criterion identifies a feasible point rapidly, it may not be faster than the dual simplex to solve (LP) overall. Effectively, finding an optimal solution to (SD) may be slower but the selected variables have a better combined reduced cost. It could result in a faster decrease of the objective value in the next reduced problem. In the next section, we compare both methods.

Table 3.2: Comparison of two methods for finding an optimal solution to (SD).

	Hybrid			Dual		
inst	np	time	var	np	time	var
VCS1	3164	0.36	2	110	0.13	2
VCS2	141	0.05	4	109	0.07	4
VCS3	1744	0.12	2	129	0.09	2
VCS4	2519	0.20	2	91	0.09	2
VCS5	3011	0.73	2	307	0.16	3
VCS6	225	0.08	2	153	0.12	2
VCS7	265	0.07	2	203	0.13	2
VCS8	3348	0.30	2	282	0.19	2
VCS9	3836	0.44	2	223	0.20	2
VCS10	144	0.05	2	123	0.08	2
AVG	1840	0.24	2.2	173	0.13	2.3

3.4 New Strategies and Parameter Values to Speed Up IPS

The improvements to IPS that we now describe comprise three parts: solving (RP), reducing (RP) and solving (SD). We will call *major iteration* a solution of (RP) followed by a solution of (SD). In our exposition below, we allow interruptions in the solution of (RP) to further reduce it. The resulting improved IPS is named IPS-2.

The solution of any linear program via IPS, CPLEX or IPS-2 starts with an initial basis computed by the Phase-1 algorithm of CPLEX. The CPU times reported in the present paper are measured in seconds and exclude the computation of the initial basis.

In the following, we start by presenting our test problem set in §3.4.1. These problems are *combined vehicle and crew scheduling problems in public transit* (VCS). In §3.4.2, we evaluate the impact of the number of variables entering (RP) and we investigate

different numbers of reductions of (RP) in §3.4.3. We propose rules to only add independent constraints to (RP) in §3.4.4. In §3.4.5, we then present test results obtained with the strategies of this section. Finally, we state the complete IPS-2 algorithm in §3.4.6.

3.4.1 VCS Data Set

To test different strategies independently, we have selected a number of instances of VCS, which exhibit important degeneracy. The problems were generated by El Hallaoui *et al.* (2007) using the random generator of Haase *et al.* (2001).

Table 3.3 reports the number of constraints and variables of each instance as well as the average proportion of degenerate basic variables in (RP) encountered in the course of the iterations of Algorithm 3.1. Additional test problems will be introduced in §3.5 to validate the complete algorithm.

Table 3.3: Characteristics of VCS instances.

instance	constraints	variables	degeneracy	instance	constraints	variables	degeneracy
VCS1	2084	10343	44%	VCS6	2084	8308	48%
VCS2	2084	6341	45%	VCS7	2084	8795	47%
VCS3	2084	6766	45%	VCS8	2084	9241	47%
VCS4	2084	7337	48%	VCS9	2084	10150	50%
VCS5	2084	7837	48%	VCS10	2084	6327	45%

3.4.2 Number of Variables Selected in (SD)

We know from Theorem 4, that entering the variables of the solution to (SD) in (RP) decreases the value of the objective function. However, multiple major iterations may be executed. It turns out that we can reduce the number of major iterations by entering more variables at a time into (RP). To do so, we must solve several instances of (SD). In this section, we give a systematic procedure for inserting the desired number of variables.

Let (SD^k) be the kth such simplified dual problem we solve, with $(SD^0) = (SD)$. Let Ω_k be the index set of the nonzero components of a solution v^k to (SD^k), and let us write $\Omega^k = \bigcup_{s=0}^{k} \Omega_k$. Also, let δ be a parameter whose value we will discuss below.

At the kth step of our procedure, if (SD^k) is feasible and if $|\Omega^k| < \delta$, we add the constraints

$$v_i \leq 0 \quad \text{for all } i \in \Omega_k$$

to (SD^k), thus forming the problem (SD^{k+1}). Hence, (SD^k) is simply the problem (SD) with the additional constraints $v_i \leq 0$, for all $i \in \Omega^{k-1}$. Notice also that the sets $\Omega_0, \Omega_1, \ldots, |\Omega^k|$ are mutually disjoint, by construction.

We need an additional notation to formulate the next proposition. Let L_k be the set of rows of A having at least one nonzero entry in the columns $j \in \Omega_k$.

Proposition 2 *Consider Ω_s and Ω_t, two set of variables as described above. If $L_s \cap L_t = \emptyset$, then the variables of index in $\Omega_s \cap \Omega_t$ can enter the basis of (RP) with positive values.*

PROOF: The columns indexed by Ω_s can first enter the basis of (RP) with positive values, by Theorem 4. In the process, the rows of (RP) that have their right-hand side modified necessarily have a nonzero entry in the columns indexed by Ω_s, that is, these rows are in L_s. Thus, the right-hand side values of the rows in L_t are unchanged, because $L_s \cap L_t = \emptyset$. Hence, the columns indexed by Ω_t can, too, enter basis of (RP) with positive values.

Remark 1 *By construction, the sets Ω_k are pairwise disjoint. While $\Omega_s \cap \Omega_t = \emptyset$ is obviously necessary for $L_s \cap L_t = \emptyset$ to hold, it is not sufficient. When A is sparse, however, L_s and L_t will often be disjoint. Indeed, the sets Ω_k are typically quite small, so that, in a sparse matrix, the corresponding sets L_k are themselves small compared to m, the total number of rows. Hence, in practice, the procedure described above is a simple and efficient way of generating multiple sets of variables that can be entered into (RP) within a single major iteration.*

From now, to simplify notation, let (SD) = SD^k and let $\Omega = \bigcup_k \Omega_k$ where k is the number of simplified dual problems solved.

For small values of δ, the next reduced problem (RP) presents little degeneracy but a potentially large number of (RP) must be solved. For large values of δ, the next reduced problem may be more degenerate and its resolution may be slow because of the potentially large number of degenerate pivots. We present the impact of variation of the δ value in §3.4.5.

3.4.3 Number of Reductions

After a preset number σ of pivots on the reduced problem, our algorithm attempts to further reduce it. For small values of σ, this means that the solution to (RP) is interrupted at a stage where each pivot decreases the objective significantly. On the other hand, for large σ, the solution is at a stage where the decrease of the objective per pivot is lower. This must be taken into account together with the reduction time when choosing an appropriate value of δ.

In IPS, $\sigma = 0.3m$. We believe that it is profitable to reduce less often (RP) since the reduction time is costly. As a side effect, the degeneracy of (RP) is slightly higher than that with IPS. However, degeneracy may be decreased by eliminating dependent rows before they enter the reduced problem. We describe how to do so in the next section.

Moreover, the value of σ is directly in relation with the value of δ. We present the impact of variation of the σ value for different values of δ in §3.4.5.

3.4.4 Adding only independent constraints to (RP)

An important improvement in the algorithm of El Hallaoui *et al.* (2007) over that of Pan (1998) is that the reduced problem (RP) may be further reduced in the course of the iterations. This allows to solve a smaller-dimensional problem, to lower the cost per pivot and the degeneracy of (RP). However, reducing is costly. It is possible to lower the number of further reductions by only allowing independent constraints to enter (RP). This also contributes to a lower-dimensional and less degenerate problem.

The general idea behind IPS is to obtain a reduced problem by removing dependent constraints. We now propose a method that only allows independent rows to enter (RP). This strategy is called *independent rows augmentation* (IRA)

Recall that for a set of solutions v to (SD), we denote by Ω the index set of its nonzero components. Consider the $(m-p) \times (m-p)$ matrix

$$\bar{A}_\Omega = \left[\left(A_{\mathrm{P}}^{\mathrm{N}} (A_{\mathrm{P}}^{\mathrm{P}})^{-1} A_{\mathrm{I}}^{\mathrm{P}} - A_{\mathrm{I}}^{\mathrm{N}} \right)_\Omega \quad 0 \right],$$

made of the columns of the constraint matrix indexed by Ω and completed with columns of zeros. Let $PLUQ$ be the matrices obtained after applying the LU factorization process to $\bar{A}_\Omega$, where L is a lower triangular matrix, U is an upper triangular matrix and P and Q are permutation matrices. We then use the U to enter the independent rows — those rows of U where there is at least one nonzero coefficient in the row.

In our tests below, the LU factorization is computed with UMFPACK (Davis and Duff, 1997).

3.4.5 Evaluation of Improvements

In this section, we present the results of three different strategies that we explained previously. First, we use the primal algorithm to find feasible solutions of (SD). Second, we use the dual algorithm to find optimal solutions of (SD). Third, we add the IRA strategy to the latter strategy. Moreover, we use different values for σ and δ. Table 3.4 gives the results on average for the ten VCS instances. The paramater *it* stands for the number of major iterations, *arp* is the average CPU time spent on the reduced problem per major iteration, *acp* is the average CPU time spent on the

complementary problem per major iteration and *time* is the solution time in seconds.

Table 3.4: Average results on VCS problems of three methods for different values of δ and σ.

		Primal Feasible				Dual Optimal				Dual Optimal with IRA			
δ/m	σ/m	it	arp	acp	time	it	arp	acp	time	it	arp	acp	time
0.01	0.2	69.3	0.57	0.06	106.0	60.7	0.48	0.20	83.0	56.9	0.43	0.22	72.9
	0.6	50.9	1.11	0.05	75.9	48.5	0.99	0.13	68.5	48.1	0.85	0.15	64.3
	1.0	47.1	1.53	0.03	85.8	41.9	1.60	0.10	81.0	44.4	1.19	0.11	69.7
	1.4	42.6	2.14	0.04	101.0	40.0	1.84	0.08	86.2	42.7	1.35	0.10	73.7
0.05	0.6	17.3	1.76	0.11	62.6	15.7	1.63	0.28	53.3	14.7	1.48	0.30	45.6
	1.0	15.6	2.52	0.09	58.5	13.6	2.27	0.24	49.8	13.0	2.06	0.26	42.8
	1.4	14.1	3.22	0.09	60.5	12.8	2.89	0.21	51.8	12.3	2.80	0.20	46.8
	1.8	13.2	4.01	0.08	66.1	11.4	3.83	0.18	55.9	12.0	3.10	0.20	47.5
0.10	1.0	11.3	2.89	0.16	58.9	9.6	2.96	0.35	52.2	9.5	2.43	0.38	43.2
	1.4	10.1	3.64	0.15	58.8	9.1	3.48	0.32	50.7	8.7	3.07	0.33	43.0
	1.8	9.5	4.41	0.14	59.1	8.8	4.13	0.29	52.9	8.5	3.51	0.31	43.9
	2.2	9.4	4.97	0.13	60.7	8.7	4.35	0.26	51.7	8.0	4.13	0.29	44.8
0.20	1.4	8.7	4.28	0.23	60.7	8.3	3.97	0.43	57.1	7.8	3.42	0.45	45.9
	1.8	8.4	4.64	0.22	59.6	8.1	4.53	0.39	57.0	7.0	3.98	0.45	44.2
	2.2	7.5	5.50	0.24	61.0	7.3	5.24	0.41	56.6	6.8	4.53	0.41	44.1
	2.6	7.0	6.39	0.21	60.3	6.4	6.37	0.39	57.0	6.7	4.85	0.39	46.1

It is worth noting that the value $\delta = 0.2m$ imposes virtually no limit on the number of variables selected, which is to say that the search continues until (SD) becomes infeasible. In contrast, with $\delta = 0.01m$, the number of variables selected is more drastically constrained. As a result, the number of major iterations decreases while the average time spent solving (SD) increases as long as the δ value increases.

When the value of σ increases, the average solution time of the reduced problem increases and the number of major iterations decreases. However, the value of σ must be set in function of the value of δ. That is, if we increase the number of selected variables to enter in (RP), we must increase the value of σ to execute more pivots in (RP). Nevertheless, we see in Table 3.4 that the value of σ has less and less influence on the total computing time as long as the value of δ increases. Lastly, all strategies is most effective when $\delta = 0.05m$ and $\sigma = 1.0m$. However, setting $\delta = 0.1m$ or $\delta = 0.2m$ with an adequate value of σ results in interesting total CPU time.

The average time spent to solve (SD) to optimality with the dual algorithm is greater than solving (SD) to feasibility with the primal algorithm. However, variables of optimal solutions of (SD) have better reduced cost, which results in a smaller number of major iterations and a smaller average time spent in (RP). Lastly, it appears from Table 3.4 that solving (SD) to optimality with the dual algorithm lowers the total computing time regardless of the value of δ.

Our mechanism for selecting sets of variables to enter into (RP) takes advantage of the dual simplex to solve (SD). Indeed, when imposing an upper bound on variables previously selected, the solution to the previous reduced problem remains dual feasible. It only violates the few newly added constraints and in consequence, few iterations are necessary to find an optimal solution to the new reduced problem. The results of Table 3.4 confirm this observation.

The rightmost part of Table 3.4 reports our result with the strategy IRA when solving (SD) to optimality. The average solution time of the reduced problem decreases, which is caused by a diminution of the degeneracy. In practice, we observe an average degeneracy of the reduced problem of 12% with the IRA strategy compared to 15% without it. This leads to fewer further reductions and fewer major iterations and therefore, to a lower total CPU time.

In conclusion, we set $\delta = 0.05m$ and $\sigma = m$, we solve (SD) to optimality with the dual algorithm and we add the IRA strategy.

3.4.6 Algorithm

We are now in a position to state the complete algorithm. We denote k the number of CPLEX pivots in the current resolution and denote B' the current basis of (RP). Our algorithm is stated as Algorithm 3.2.

Algorithm 3.2 The Improved Primal Simplex 2

1: Choose an initial basis B. Initialize σ and δ.
2: Construct the reduced problem (RP). (see §3.2.1)
3: Perform pivots on (RP) until optimality is reached or σ pivots are performed. (see §3.4.3)
4: If (RP) was not solved to optimality, set $B = B'$, $k = 0$, and go back to step 2. (see §3.4.3)
5: Construct (SD). Let y^* be its optimal value. (see §3.2.2)
6: While $|\Omega| < \delta$, $y^* < 0$ and (SD) is feasible, solve (SD) to optimality. (see §3.4.2)
7: If $|\Omega| = 0$, STOP, the current solution is optimal for (LP). (see proposition 1)
8: Compute the LU factorization of $\bar{A}_\Omega$. (see §3.4.4)
9: Identify and add columns and rows to (RP), and return to step 3. (see theorem 4)

At Step 1, we start with a basis B and we initialize σ and δ. Following the previous results, we set $\delta = 0.05m$ and $\sigma = m$. Step 2 is the construction of the reduced problem explained in §3.2.1. We then perform pivots on the reduced problem until optimality or until σ pivots are done at Step 3. At Step 4, if σ pivots are executed, we set the current basis as B and we go back to Step 2. Step 5 is the construction of the complementary problem presented in §3.2.2. Step 6 selects variables that will enter in the reduced problem by solving (SD). The constraint $y^* < 0$ allow not to select variables of a non-negative solution. Step 7 is the optimality criteria: if no variable is selected in Step 6, stop, the current solution of (RP) is optimal for (LP). Step 8 selects the independent rows to enter in (RP) (IRA strategy). At Step 9, we augment the reduced problem with the selected variables and rows, then, we go back to Step 3.

3.5 Results

In this section, we report numerical experimentations with all strategies described above and parameter choices calibrated on our VCS instances. Below, we run tests on both the VCS and the *fleet assignment* (FA) instances. Their characteristics are given below.

3.5.1 FA Data Set

Fleet assignment problems consist in maximizing the profits of assigning a type of aircraft to each flight segment over an horizon of one week. The paths of the aircraft must respect maintenance conditions and availability. Our problem instances are generated from real data with 2505 flight segments and four types of aircraft. Those problems have one set partitioning constraint per flight segment, one availability constraint per aircraft type, and one flow conservation constraint between flight sequences.

Our instances have a higher proportion of degenerate variables than those of El Hallaoui *et al.* (2007). In their article, the authors label their FA problems with FA1 to FA5. To avoid confusion, we label our FA problems with FA6 to FA12. Table 3.5 gives the number of constraints and variables of each instance along with the average proportion of degenerate variables encountered with Algorithm 3.2.

Table 3.5: Characteristics of FA instances.

instance	constraints	variables	degeneracy	instance	constraints	variables	degeneracy
FA6	5067	17594	68%	FA13	5159	25746	65%
FA7	5159	20434	59%	FA14	5159	22641	71%
FA8	5159	21437	65%	FA15	5182	23650	63%
FA9	5159	23258	66%	FA16	5182	23990	64%
FA10	5159	24492	66%	FA17	5182	24282	65%
FA11	5159	24812	66%	FA18	5182	24517	65%
FA12	5159	24645	66%	FA19	5182	24875	65%

3.5.2 Computational Experiments

Algorithm 3.2 was tested on both the VCS and the FA instances. The results are reported in Tables 3.6 and 3.8. Table 3.7 compares CPLEX and IPS on the FA instances. In the tables, *np* is the number of pivots, *it* is the number of major iterations *rp* is the solution time for (RP), *cp* is the solution time for (SD), *t* is the total time spent reducing problems, *nb* is the number of reductions, *c/i2* the gain factor of IPS-2 versus CPLEX and *i/i2* the gain factor IPS-2 versus IPS. All times are in seconds.

We observe that IPS-2 decreases the total number of pivots by over 55% compared to IPS. It also significantly decreases the number of major iterations and the time spent solving (SD), while slightly decreasing time spent solving (RP). It thus seems preferable to perform a large number of pivots on few reduced problems than few pivots on numerous reduced problems.

Regarding the VCS instances, IPS-2 decreases the total running time by a factor of 2.07 and 3.53 compared to IPS and CPLEX respectively. On the FA instances, the factors are 4.13 and 12.30. Notice also that IPS is quite unstable on our FA instances. Indeed, its solution times exhibit a large variance, because the number of degenerate pivots perform on the numerous (SD) solutions is very unpredictable.

Table 3.6: Results for VCS instances.

	IPS					IPS-2							
inst	np	it	rp	cp	t	np	it	rp	cp	r(nb)	t	i/i2	c/i2
VCS1	64923	26	33	27	119	27533	9	27	5	25(9)	58	2.05	3.60
VCS2	28266	19	22	11	58	18794	8	19	2	11(6)	32	1.81	3.09
VCS3	41955	20	23	12	65	21714	9	20	2	13(7)	36	1.80	3.30
VCS4	51347	24	24	17	71	23265	9	20	3	12(7)	35	2.02	3.74
VCS5	60262	27	28	22	98	25419	9	21	4	14(7)	40	2.45	4.25
VCS6	63613	27	28	26	97	26094	9	25	4	17(7)	46	2.11	3.30
VCS7	65509	27	30	26	101	25597	9	23	5	17(7)	45	2.24	3.71
VCS8	64610	26	30	27	99	28171	10	24	4	15(7)	44	2.25	3.98
VCS9	76656	29	33	36	131	32503	11	28	6	25(9)	60	2.18	3.20
VCS10	39050	19	19	12	55	17946	8	19	2	10(5)	31	1.77	3.13
AVG	55619	24	27	22	89	24703	9	23	4	16(7)	43	2.07	3.53

Table 3.7: Results for FA instances (CPLEX and IPS).

	CPLEX		IPS					
inst	np	t	np	it	rp	cp	t	c/i
FA6	78316	269	53853	18	21	62	106	2.53
FA7	88958	289	25069	9	19	21	64	4.51
FA8	94532	419	27093	8	15	26	65	6.44
FA9	121197	498	48987	21	18	67	120	4.15
FA10	146313	654	40821	16	20	53	108	6.06
FA11	131282	546	74042	22	29	170	249	2.19
FA12	133910	617	91259	26	29	172	253	2.43
FA13	141237	696	111952	31	43	230	340	2.05
FA14	140003	662	197522	16	78	431	549	1.21
FA15	129507	577	27987	11	16	29	75	7.69
FA16	193244	947	30399	13	16	35	77	12.29
FA17	141940	612	57202	17	23	89	149	4.10
FA18	137712	665	57640	18	23	93	155	4.29
FA19	131830	641	53534	10	19	351	410	1.56
AVG	121499	578	64067	17	24	131	194	2.98

Table 3.8: Results for FA instances (IPS-2).

	IPS-2							
inst	np	it	rp	cp	r(nb)	t	i/i2	c/i2
FA6	28402	10	19	8	9(3)	36	2.94	7.69
FA7	24497	7	21	9	15(3)	45	1.42	6.28
FA8	26794	9	17	15	12(3)	45	1.44	9.31
FA9	34113	11	22	15	16(3)	53	2.26	9.40
FA10	25028	7	18	12	15(3)	46	2.34	14.22
FA11	29678	6	21	11	17(3)	49	4.98	10.92
FA12	32220	7	23	12	17(3)	52	5.16	12.59
FA13	34618	10	24	16	18(3)	57	4.03	12.21
FA14	46930	13	29	21	14(4)	64	8.57	10.34
FA15	22109	6	17	7	11(2)	35	2.14	16.49
FA16	24824	7	18	9	12(2)	39	1.97	24.28
FA17	37264	11	24	15	14(3)	52	2.87	11.77
FA18	30771	8	20	15	15(3)	51	3.04	13.04
FA19	24794	5	19	7	13(2)	39	10.51	16.44
AVG	30860	8	21	12	14(3)	47	4.13	12.30

3.6 Conclusion

In this article, we present improvements for IPS (El Hallaoui *et al.*, 2007). More precisely, we choose a suitable algorithm to solve the complementary problem and we propose a strategy to obtain a predetermined number of variables to enter in the reduced problem. Furthermore, we propose a method to enter only independent rows in (RP) and we execute less reductions of (RP). The overall effect of these improvements is a better balance between the times spent on the three main subprocess of IPS (resolution of (RP), reduction of (RP) and resolution of (SD)) resulting in a more efficient use of the computing power.

Computational experiments show the efficiency of our strategies and calibration of parameters. In fact, on the combined vehicle and crew scheduling in public transit

instances, our algorithm (IPS-2) worked 3 times faster than CPLEX, and it solved our fleet assignment instances 12 times faster than CPLEX. This performance of our algorithm on the FA problems illustrates its robustness; the parameters used for these instances were actually those that had been adjusted to solve the VCS problems.

In future research, it would be interesting to modify IPS-2 so that it can handle problems with lower and upper bounds. Likewise, the use of UMFPACK could be replaced by the use of CPLEX to execute the reduction, so the user could use only one software.

Chapitre 4

Improved Primal Simplex Method Version 3 : Cold Start, Generalization for Bounded Variables Problems, New Implementation

Article écrit par Vincent Raymond, François Soumis et Abdelmoutalib Metrane; soumis pour publication à *European Journal of Operational Research.*

La méthode *Improved Primal Simplex* (IPS) a été proposée par El Hallaoui *et al.* (2007) et améliorée par Raymond *et al.* (2009b) (voir le chapitre 3). Dans cet article, nous réécrivons la théorie de façon à commencer l'algorithme avec une solution initiale plutôt qu'une base initiale. Cette amélioration permet d'augmenter la polyvalence de IPS. En effet, il est maintenant possible d'utiliser la stratégie *heuristic first - optimization second.*

Nous généralisons la théorie de manière à ce que les problèmes comportant des bornes supérieures sur les variables puissent être résolus par IPS. Non seulement nous augmentons la polyvalence de l'algorithme mais nous montrons que les variables qui atteignent leurs bornes supérieures sont traitées comme des variables nulles. Ainsi, il y a la dégénérescence des variables nulles et celles des variables à leurs bornes supérieures. Puisque notre algorithme tire profit de la dégénérescence, la performance de IPS sur ce type de problème augmente comparativement à CPLEX.

Nous simplifions de plus l'implémentation de IPS en remplaçant les appels aux modules de UMFPACK par des appels aux modules de CPLEX. Ceci permet de ne travailler qu'avec un seul logiciel. Nous montrons à l'aide de résultats numériques que la réduction d'un même problème est plus rapide avec CPLEX qu'avec UMFPACK.

Toutes ces améliorations permettent d'obtenir un facteur de diminution du temps total de résolution de plus de 20 par rapport à la résolution primale de CPLEX sur des problèmes de répartition de flottes d'avions avec bornes supérieures. Le travail de programmation a été effectué en langage C. Le programme informatique utilise le logiciel CPLEX version 9.1.3. Nos tests ont été effectués sur des instances de répartition de flottes d'avions et sur des problèmes d'horaires de chauffeurs d'autobus.

Improved Primal Simplex Method version 3: cold start, generalization for bounded variable problems and a new implementation

VINCENT RAYMOND, FRANÇOIS SOUMIS
AND ABDELMOUTALIB METRANE
École Polytechnique de Montréal and GERAD

February 2009

Abstract

The *Improved Primal Simplex* (IPS) method has been proposed by El Hallaoui *et al.* (2007). We rewrite the theory of IPS for a cold start with an initial feasible solution instead of an initial basic feasible solution. This allows us to use a *heuristic first - optimization second* strategy. We generalize the algorithm so that it can handle bounds on variables. We show that variables at upper bounds increase degeneracy, and consequently, increase performance of IPS compared to CPLEX. We simplify the implementation by replacing the UMFPACK (Davis and Duff, 1997) procedure by certain modules of the CPLEX library. This allows the user to work with only one software package. We obtain a reduction factor of solution time of 20 on fleet assignment instances with bounded variables.

Keywords: Linear Programming, Primal Simplex, Degeneracy.

4.1 Introduction

We consider the solution of a linear programs in standard form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \; c^T x \quad \text{subject to} \; Ax = b, \; x \geq 0, \tag{LP}$$

where $c \in \mathbb{R}^n$ is the cost-vector, A is the $m \times n$ constraint matrix and $b \in \mathbb{R}^m$ is the right-hand side. We are particularly interested in so-called degenerate problems on which the simplex algorithm of Dantzig (1949) is likely to perform degenerate pivots.

Our paper is organized as follows. In §4.2 we present the *Improved Primal Simplex* (IPS) proposed by El Hallaoui *et al.* (2007) and developed by Raymond *et al.* (2009b) (IPS-2) and we present the theory to handle bounds in the simplex method. In §4.3, we rewrite the IPS theory so that the algorithm can start with an initial feasible solution instead of an initial basic feasible solution and we generalize IPS to problems with bounded variables to take advantage of the degeneracy of variables that are at their upper bounds. We simplify the implementation of IPS by removing a procedure from UMFPACK (Davis and Duff, 1997). This new version of IPS is called IPS-3. Computational experiments are given in §4.4 and in §4.5 we present our conclusions.

4.2 Background

In this section, we present the IPS algorithm and some theoretical properties developed by El Hallaoui *et al.* (2007). A brief sub-section on how to handle bounds in the simplex method is also given. We start with the presentation of IPS summarized in Algorithm 4.1.

Algorithm 4.1 The Improved Primal Simplex (El Hallaoui *et al.*, 2007).

1: Choose an initial basis B for (LP).
2: Form and solve the reduced problem (RP).
3: Form and solve the dual of the complementary problem (SD). Let y^* be its optimal value.
4: If $y^* \geq 0$, the current solution of (RP) is optimal for (LP).
5: Otherwise, construct a new basis B' using the solution of (SD) and return to Step 2.

At Step 1 of Algorithm 4.1, the method starts with a degenerate basic feasible solution. To obtain this basis, the authors execute a phase I of the simplex algorithm on (LP). At Step 2 the reduced problem RP is formed according to the theory of Pan (1998). Note that IPS carries out the reduction by means of the software package UMFPACK. Then at Step 3, the complementary problem (CP) is formed and its dual (SD) is solved. We recall from El Hallaoui *et al.* (2007) in §4.3.2 the construction of this problem. Step 4 is the optimality condition of IPS. The authors prove that if the value of the objective function of (CP) is nonnegative, then the optimal solution of RP is optimal for (LP). Finally, Step 5 constructs a new basis for the reduced problem. El Hallaoui *et al.* (2007) proved that the objective function value associated with B' is strictly lower (minimization problem) than the one associated with the optimal basis of (RP). We refer the reader to El Hallaoui *et al.* (2007) for the technicalities of the last two steps. We note that IPS does not exploit upper bounded variables.

The computational experiments of IPS show the efficiency of this method. More precisely, it obtains a reduction factor of the objective function value of less than 2 for *combined vehicle and crew scheduling problems in public transit* (IPS) problems and more than 3 for fleet assignment problems. More recently, the improved version of IPS, called IPS-2 (Raymond *et al.*, 2009b), obtains a reduction factor of more than 3 and more than 12 respectively for the previous two problem types.

4.2.1 Bounded Variables in LP

We consider a linear program with bounded variables in standard form

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ c^T x \quad \text{subject to } Ax = b,\ l \leq x \leq u, \tag{BLP}$$

where l and u are the vectors of the lower and the upper bounds respectively.

At a given iteration, the simplex algorithm identifies the variable x_i to enter the basis by taking into consideration the lower bound and the upper bound of this variable. If the value of the variable decreases to its lower bound, the algorithm considers $y_i = x_i - l_i$. On the other hand, if the new value of x_i reaches its upper bound, the algorithm considers $y_i = u_i - x_i$. The standard treatment of the $y_i \geq 0$ constraint allows us to deal with lower and upper bounds on x_i. Therefore, when $x_i = l_i$ or $x_i = u_i$, x_i can be basic or non basic.

4.2.2 Notation

If $x \in \mathbb{R}^n$ and $\mathrm{I} \subseteq \{1,\ldots,n\}$ is an index set, we denote by x_{I} the subvector of x indexed by I. Similarly, if A is an $m \times n$ matrix, we denote by A_{I} the $m \times |\mathrm{I}|$ matrix whose columns are indexed by I. Let $\mathrm{J} = \{1,\ldots,n\} \setminus \mathrm{I}$, we write $x = (x_{\mathrm{I}}, x_{\mathrm{J}})$ even though the indices in I and J may not appear in order. Moreover, we denote with upper indices the subset of rows associated with the indexed variables set.

The jth column of A is denoted by A_j and we denote by $A^{-\mathsf{T}}$ the inverse of the transpose of square matrix A.

4.3 Contributions

This article presents three improvements of IPS. First, instead of using an initial basic feasible solution to reduce the problem as in El Hallaoui *et al.* (2007), we present in §4.3.1 a new reduction method using only a feasible degenerate solution. This new reduction is well adapted to the *heuristic first - optimization second* approach (Boubaker *et al.*, 2008) that starts with a heuristic algorithm and finishes with a mathematical programming optimization.

Secondly, instead of using UMFPACK, we present in §4.3.3 a procedure using only CPLEX to construct the reduced and the complementary problems. Doing all the computation with the same program avoids possible numerical tolerance incompatibilities and allows the user to work with only one software package.

Finally, §4.3.4 presents the generalization to bounded variables. The proposed algorithm also removes the basic variables at their upper bounds from the reduced problem.

Computational experiments for each of these improvements are reported in §4.4.

4.3.1 Reduced Problem

Let $\bar{x}$ be a feasible solution of (LP) and P be the index set of the $p < m$ nonzero variables of $\bar{x}$, i.e.,

$$\mathrm{P} = \{i \in \{1, ..., n\} \mid \bar{x}_i > 0\},$$

where $\bar{x}_i$ may be integer or not. We construct a basis A_B of (LP) such that the first p variables are the variables of P and the last $m - p$ variables are artificial (denoted by

the index set N). Without loss of generality, we assume that the p rows associated with the variables of P are the first p rows of (LP). In the same way, we assume that the $m-p$ rows associated with the $m-p$ artificial variables of N are the last $m-p$ rows of (LP). Thus, we can write

$$A_{\mathrm{B}} = \begin{bmatrix} A_{\mathrm{P}}^{\mathrm{P}} & 0 \\ A_{\mathrm{P}}^{\mathrm{N}} & A_{\mathrm{N}}^{\mathrm{N}} \end{bmatrix}.$$

Suppose that the $A_{\mathrm{P}}^{\mathrm{P}}$ matrix is non singular. It is the case when the p variables of P are linearly independent. If not, it is possible to find a new solution from $\bar{x}$ such that the nonzero variables are linearly independent by solving

$$\underset{x \in \mathbb{R}^m}{\text{minimize}} \; c_{\mathrm{P}}^{\top} x \quad \text{subject to } A_{\mathrm{B}} x = b, x \geq 0. \tag{4.7}$$

Note that this problem is solved anyway in the CPLEX reduction procedure (see §4.3.3). Thus, the reduced problem is always created from a linearly independent solution. The rows of the submatrix $A_{\mathrm{P}}^{\mathrm{P}}$ are the rows associated with the non zero variables of the solution of (4.7).

Denote by Q the inverse matrix of A_{B} and partition

$$Q = \begin{bmatrix} Q^{\mathrm{P}} \\ Q^{\mathrm{Z}} \end{bmatrix} = \begin{bmatrix} (A_{\mathrm{P}}^{\mathrm{P}})^{-1} & 0 \\ -A_{\mathrm{P}}^{\mathrm{N}}(A_{\mathrm{P}}^{\mathrm{P}})^{-1} & A_{\mathrm{N}}^{\mathrm{N}} \end{bmatrix} = A_{\mathrm{B}}^{-1}, \tag{4.8}$$

where Q^{Z} is the *compatibility matrix* formed by the last $m-p$ rows of Q. We have

$$\bar{x} = \begin{bmatrix} Q^{\mathrm{P}} b \\ Q^{\mathrm{Z}} b \end{bmatrix}, \text{ and therefore, } Q^{\mathrm{Z}} b = 0. \tag{4.9}$$

We begin with the following definition.

Definition 2 *The jth variable of (LP), x_j, is said to be* compatible *if and only if* $Q^z A_j = 0$.

From Definition 2, we let $C \subseteq \{1, \ldots, n\}$ denote the indices of variables that are compatible and $I = \{1, \ldots, n\} \setminus C$. Thus, x_C and x_I are the subvectors of x of compatible and incompatible variables, respectively. We partition the cost vector c accordingly into c_C and c_I, and the columns of A into A_C and A_I. The partitioning that we applied to A_B can be generalized to A_C and A_I and yields

$$A_I = \begin{bmatrix} A_I^P \\ A_I^N \end{bmatrix} \quad \text{and} \quad A_C = \begin{bmatrix} A_C^P \\ A_C^N \end{bmatrix}. \tag{4.10}$$

Upon premultiplying the equality constraints of (LP) by Q, we obtain $QAx = Qb$, which may be rewritten

$$\begin{bmatrix} Q^P Ax \\ Q^z Ax \end{bmatrix} = \begin{bmatrix} ((A_P^P)^{-1} A_C^P) x_C & ((A_P^P)^{-1} A_I^P) x_I \\ (-A_P^N (A_P^P)^{-1} A_C^P + A_C^N) x_C & (-A_P^N (A_P^P)^{-1} A_I^P + A_I^N) x_I \end{bmatrix} = \begin{bmatrix} Q^P b \\ 0 \end{bmatrix},$$

where we used equations (4.8)-(4.9). Since by definition x_C is compatible, we have $(-A_P^N (A_P^P)^{-1} A_C^P + A_C^N) x_C = 0$. Upon imposing $x_I = 0$, we obtain the *reduced problem*

$$\underset{x_C}{\text{minimize}} \; c_C^\top x_C \quad \text{subject to } ((A_P^P)^{-1} A_C^P) x_C = Q^P b, \; x_C \geq 0. \tag{RP}$$

(RP) is potentially much smaller than (LP) and is obtained from a degenerate primal basis. It only depends on the compatible variables—those that can enter a basis for (RP) without violating the constraints of (LP) that have been omitted. Note that by construction, if x_C is feasible for (RP), then $(x_C, 0)$ is feasible for (LP).

With this new theoretical presentation of the reduced problem, we show that we can use an initial feasible solution of (LP) instead of using an initial basic feasible solution

to reduce the problem. This particularity allows the use of a heuristic method to obtain the values of the nonzero variables of a feasible solution. Moreover, a heuristic feasible solution has more chances to be closer to an optimal solution than the classical phase I solution. Furthermore, the computational time of finding a heuristic initial solution can be much less than a phase I procedure.

4.3.2 Complementary Problem

In this section, we present the construction of the complementary problem as explained by El Hallaoui *et al.* (2007). Note that a complementary problem is created from a basic feasible solution of the reduced problem and contains only incompatible variables. Let $\bar{x}_{\mathrm{C}}$ be the current feasible solution for (RP). Here, compatibility is understood with respect to Q^{z} that has been used to calculate the previous reduced problem.

Recall that P indexes the nonzero variables of the current solution of the reduced problem. Assume that the reduced problem is not degenerate (if it is, then reduce it). P can be considered as the indices of the compatible basic variables. Let V indexes the compatible nonbasic variables and I indexes the incompatible variables, i.e.,

$$\mathrm{P} = \{i \in \mathrm{C} \mid \bar{x}_i \; \textit{basic}\}, \quad \mathrm{V} = \{i \in \mathrm{C} \mid \bar{x}_i \; \textit{nonbasic}\}, \quad \text{and} \quad \mathrm{I} = \{1, \ldots, n\} \setminus \mathrm{C}.$$

Then $\bar{x}_{\mathrm{C}}$ is also optimal for (LP) if and only if all reduced costs (i.e., corresponding to all variables, compatible or not) are nonnegative. In other words, there must exist

dual variables π such that

$$c_j - A_j^\top \pi = 0 \quad \text{for all } j \in \mathrm{P}, \tag{4.11a}$$

$$c_j - A_j^\top \pi \geq 0 \quad \text{for all } j \in \mathrm{V}, \tag{4.11b}$$

$$c_j - A_j^\top \pi \geq 0 \quad \text{for all } j \in \mathrm{I}. \tag{4.11c}$$

It is easy to see that constraints (4.11b) are satisfied when $\bar{x}_{\mathrm{C}}$ is optimal for (RP). In this case, constraints (4.11b) are redundant and can be removed. In the other case, when $\bar{x}_{\mathrm{C}}$ is not optimal, we can handle them subsequently in the next reduced problem.

To find a negative reduced cost set of incompatible variables, El Hallaoui *et al.* (2007) propose to solve the following problem:

$$\underset{\pi,y}{\text{maximize}} \; y \quad \text{subject to } c_{\mathrm{P}} - A_{\mathrm{P}}^\top \pi = 0, \; c_i - A_i^\top \pi \geq y, \; \forall i \in \mathrm{I}. \tag{4.12}$$

By using the same partitioning as A_{P} (see equations (4.10)), we partition the vector of dual variables $\pi = \begin{bmatrix} \pi^{\mathrm{P}} \\ \pi^{\mathrm{N}} \end{bmatrix}$. We use this notation to rewrite the first set of constraints of (4.12):

$$c_{\mathrm{P}} - (A_{\mathrm{P}}^{\mathrm{P}})^\top \pi^{\mathrm{P}} - (A_{\mathrm{P}}^{\mathrm{N}})^\top \pi^{\mathrm{N}} = 0,$$

and we may thus express

$$\pi^{\mathrm{P}} = (A_{\mathrm{P}}^{\mathrm{P}})^{-\top} c_{\mathrm{P}} - (A_{\mathrm{P}}^{\mathrm{P}})^{-\top} A_{\mathrm{P}}^{\mathrm{N}} \pi^{\mathrm{N}}.$$

We may rewrite (4.12) as the *complementary problem* by substituting π^{P}:

$$\underset{\pi,y}{\text{maximize}} \; y \tag{CP}$$

$$\text{s.t.} \quad y - \left(A_{\mathrm{P}}^{\mathrm{N}}(A_{\mathrm{P}}^{\mathrm{P}})^{-1} A_i^{\mathrm{P}} - A_i^{\mathrm{N}}\right) \pi^{\mathrm{N}} \leq \left(c_i - ((A_{\mathrm{P}}^{\mathrm{P}})^{-1} A_i^{\mathrm{P}})^\top c_{\mathrm{P}}\right), \; \forall i \in \mathrm{I}.$$

The following property (El Hallaoui *et al.*, 2007) justifies the use of (CP).

Proposition 3 *Let x_C^* be an optimal solution of (RP) and let y^* be an optimal solution of (CP). Then $(x_C^*, x_I^*) = (x_C^*, 0)$ is an optimal solution of (LP) if and only if $y^* \geq 0$.*

We consider the dual of (CP), the *simplified dual*

$$\underset{v}{\text{minimize}} \ \left(c_{\text{I}} - ((A_{\text{P}}^{\text{P}})^{-1} A_{\text{I}}^{\text{P}})^{\top} c_{\text{P}}\right)^{\top} v \qquad \text{(SD)}$$

$$s.t. \quad \left(A_{\text{P}}^{\text{N}}(A_{\text{P}}^{\text{P}})^{-1} A_{\text{I}}^{\text{P}} - A_{\text{I}}^{\text{N}}\right) v = 0, \ e^{\top} v = 1, \ v \geq 0.$$

Note that (SD) possesses $m - p + 1$ equality constraints. From now on, we refer to the *complementary problem* as the pair (CP) and (SD).

We mention that the matrix $A_{\text{P}}^{\text{N}}(A_{\text{P}}^{\text{P}})^{-1} A_{\text{I}}^{\text{P}} - A_{\text{I}}^{\text{N}}$ of the complementary problem is the lower right-hand matrix of QA obtained by the previous reduction. Thus, we do not need to compute this matrix at each solution of (SD). We just have to update the matrix after each modification (augmentation or reduction) of the reduced problem.

4.3.3 CPLEX Reduction

To avoid possible numerical tolerance incompatibilities between UMFPACK and CPLEX, we use the latter to execute the reduction.

Algorithm 4.2 describes the method of reducing problems that do not contain bounds on variables. Step 1 and Step 2 are clear. At Step 3, we solve the temporary problem. The optimal basic feasible solution of this problem contains the nonzero variables of

Algorithm 4.2 CPLEX reduction of a problem that does not contain bounds on variables.

1: Create a temporary CPLEX problem (LPtmp) that contains only the nonzero variables.
2: Add to (LPtmp) an identity matrix that represents artificial variables.
3: Solve (LPtmp).
4: The rows associated with basic artificial variables must be removed.
5: Use the basis inverse of (LPtmp) to construct (RP) and (SD).

the current solution of (LP), that is (A_{P}), and the artificial variables associated with the rows that are not associated with a nonzero basic variable, that is (A_{N}). At Step 4, we identify the subsets of rows: A^{N} are the rows associated with artificial variables and A^{P} are the rows associated with nonzero basic variables. At Step 5, we construct the reduced and complementary problems. Since the required Q matrix is given by the current inverse basis of the temporary problem, we can compute QA to create both problems. We might add that the computational time to find Q is relatively insignificant compared with that needed to calculate QA.

4.3.4 Bounds

We mentioned previously that we generalized our algorithm to handle problems with bounded variables. As we explained in §4.2.1, variables that are at their upper bounds can be handled as degenerate variables. Thus, since these problems may have more degeneracy, their reduced problems may be smaller. Our algorithm must be modified in the reduction process and in the construction of (SD).

We define the index sets L and U:

$$L = \{i \in C \cup I \mid \bar{x}_i = l_i\} \quad \text{and} \quad U = \{i \in C \cup I \mid \bar{x}_i = u_i\}$$

where $\bar{x}_i$ is the current value of x_i.

To take into account the bounds of the variables, we must add some steps in Algorithm 4.2. The procedure to reduce problems that contain bounds with CPLEX is summarized in Algorithm 4.3.

Algorithm 4.3 CPLEX reduction of problems that contain bounds on variables.

1: Create a temporary CPLEX problem (LPtmp) that contains only the nonzero variables.
2: Change the bounds of variables in (LPtmp):
$l_i \leq x_i \leq l_i$ for all $i \in L$.
$u_i \leq x_i \leq u_i$ for all $i \in U$.
3: Make nonbasic the variables x_i for all $i \in L \cup U$ in (LPtmp).
4: Add to (LPtmp) an identity matrix that represents artificial variables.
5: Solve (LPtmp).
6: The rows associated with basic artificial variables must be removed.
7: Use the basis inverse of (LPtmp) to construct (RP) and (SD).

Steps 1, 4, 5 and 6 of Algorithm 4.3 are the same as in Algorithm 4.2. Step 2 assures that the optimal solution of (LPtmp) is the same as the feasible solution given. Step 3 allows maximizing the number of artificial variables in the basis, that is, allows completing reduction of the problem.

Step 7 is executed as follows. If a variable x_i with $i \in L$ is incompatible, instead of deleting it from (RP), we "remove" it by changing its bounds in (RP) such that $l_i \leq x_i \leq l_i$. Thus, the values of these incompatible variables cannot be changed in the reduced problem as zero incompatible variables. Moreover, by changing bounds, CPLEX will take into account the values of x_i in the right-hand side of the reduced problem. The complementary problem is constructed as usual for this type of variable.

In the same way, if a variable x_i with $i \in U$ is incompatible, instead of deleting it from (RP), we "remove" it by changing its bounds in (RP) such that $u_i \leq x_i \leq u_i$. However, these variables are modified in the complementary problem. Since the

theoretical substitution of this type of variable is $y_i = u_i - x_i$, we take into account the negative relation between y_i and x_i by multiplying the coefficients of this variable in (SD) by -1. Note that u_i and l_i in the substitution are present indirectly in (SD) by the modification of the bounds in (RP).

We then obtain the bounded reduced problem (BRP)

$$\underset{x}{\text{minimize}} \sum_{i \in C} c_i x_i + \sum_{j \in I \cap L} c_j x_j + \sum_{k \in I \cap U} c_k x_k \qquad \text{(BRP)}$$

$$\text{s.t.} \sum_{i \in C} (A_{\text{P}}^{\text{P}})^{-1} A_i^{\text{P}} x_i + \sum_{j \in I \cap L} (A_{\text{P}}^{\text{P}})^{-1} A_j^{\text{P}} x_j + \sum_{k \in I \cap U} (A_{\text{P}}^{\text{P}})^{-1} A_k^{\text{P}} x_k = Q^{\text{P}} b$$

$$\begin{aligned} l_i \leq x_i \leq u_i \quad & \text{for all} \quad i \in C \\ l_j \leq x_j \leq l_j \quad & \text{for all} \quad j \in I \cap L \\ u_k \leq x_k \leq u_k \quad & \text{for all} \quad k \in I \cap U \end{aligned}$$

and the associated simplified dual problem (ASD)

$$\underset{v}{\text{minimize}} \sum_{i \in I \setminus U} \left(c_i - ((A_{\text{P}}^{\text{P}})^{-1} A_i^{\text{P}})^{\top} c_{\text{P}} \right) v_i - \sum_{j \in I \cap U} \left(c_j - ((A_{\text{P}}^{\text{P}})^{-1} A_j^{\text{P}})^{\top} c_{\text{P}} \right) v_j \qquad \text{(ASD)}$$

$$\sum_{i \in I \setminus U} \left(A_{\text{P}}^{\text{N}} (A_{\text{P}}^{\text{P}})^{-1} A_i^{\text{P}} - A_i^{\text{N}} \right) v_i - \sum_{j \in I \cap U} \left(A_{\text{P}}^{\text{N}} (A_{\text{P}}^{\text{P}})^{-1} A_j^{\text{P}} - A_j^{\text{N}} \right) v_j = 0$$

$$\sum_{i \in I} v_i = 1$$

$$v \geq 0.$$

When a variable x_i with $i \in I \cap (L \cup U)$ is chosen to enter (RP), we enter the rows as usual and we enter the variable by re-initializing the bounds of x_i since the latter variable is already in (RP).

4.4 Computational Experiments

This section presents computational experiments that were obtained with IPS-3. More precisely, we present a comparison of reduction times obtained by CPLEX and UMFPACK. We compare solution time of CPLEX when starting with an initial basic feasible solution or with an initial feasible solution. Then, we present a sensitivity analysis as a function of the quality of the initial solution. Finally, we present computational experiments on problems with bounded variables. Characteristics of the instances used are presented in §4.4.1 and §4.4.5.

Before the presentation of the results, we define the different versions of IPS to avoid confusion. IPS has been proposed and developed by El Hallaoui *et al.* (2007) and is the basic algorithm. IPS-2 has been developed by Raymond *et al.* (2009b) and contains various improvements. Here, we present IPS-3, a version based on IPS-2 which has the improvements that we mentioned in §4.3. All the results in this paper have been computed with IPS-3.

4.4.1 Vehicle and Crew Scheduling Data Set

We selected a number of VCS instances which exhibit important degeneracy. These instances were generated by El Hallaoui *et al.* (2007) using a random generator of Haase *et al.* (2001). They obtain various VCS instances by applying a column

generation algorithm on different random data set and by collecting different restricted master problems. These instances were also used by Raymond *et al.* (2009b).

Table 4.1 reports the number of constraints and variables of each instance as well as the average percentage of degenerate basic variables in (RP) encountered in the course of the iterations of Algorithm 4.1.

Table 4.1: Characteristics of VCS instances.

instance	constraints	variables	degeneracy	instance	constraints	variables	degeneracy
VCS1	2084	10343	44%	VCS6	2084	8308	48%
VCS2	2084	6341	45%	VCS7	2084	8795	47%
VCS3	2084	6766	45%	VCS8	2084	9241	47%
VCS4	2084	7337	48%	VCS9	2084	10150	50%
VCS5	2084	7837	48%	VCS10	2084	6327	45%

4.4.2 CPLEX Reduction Instead of UMFPACK Reduction

When we compare a CPLEX reduction to the same UMFPACK reduction, the time reduction is significant. Moreover, the implementation of IPS-3 is simplified since it uses only one software package. However, the use of CPLEX instead of UMFPACK to create the reduced problem and the complementary problem results in a relatively small gain in the total computing time.

We present in Table 4.2 the CPLEX and the UMFPACK reduction time of the first reduced problem encoutered for each VCS instance. All the times are in seconds. We see that the CPLEX reduction is 1.48 times faster than the UMFPACK reduction.

Table 4.2: Reduction times of UMFPACK and CPLEX for VCS instances.

instance	UMFPACK	CPLEX	instance	UMFPACK	CPLEX
VCS1	6.58	3.87	VCS6	4.49	3.45
VCS2	3.53	2.34	VCS7	4.72	3.45
VCS3	3.69	2.54	VCS8	4.27	3.16
VCS4	3.83	2.64	VCS9	5.55	3.53
VCS5	4.60	2.84	VCS10	3.47	2.41
		AVERAGE		4.47	3.02

4.4.3 Initial Feasible Solution Instead of Initial Basic Feasible Solution

Beginning with an initial feasible solution instead of an initial basic feasible solution increases the generality of IPS-3. Indeed, the method can begin with an initial feasible solution obtained by a heuristic or begin with an initial basic feasible solution. CPLEX may also start with an initial feasible solution. However, it must construct a basic feasible solution from the given initial solution.

We present in Table 4.3 the solution time of CPLEX and IPS-3 when they start with initial solutions obtained from the phase I basis. The times are again in seconds.

Table 4.3: CPLEX and IPS-3 solution times when starting with an initial solution.

instance	CPLEX	IPS-3	reduction factor	instance	CPLEX	IPS-3	reduction factor
VCS1	252	58	4.34	VCS6	176	45	3.91
VCS2	105	33	3.18	VCS7	185	44	4.20
VCS3	135	35	3.85	VCS8	214	44	4.86
VCS4	163	35	4.65	VCS9	246	58	4.24
VCS5	171	39	4.38	VCS10	124	30	4.13
			AVERAGE		177	42	4.17

The average solution times of CPLEX and IPS-3 are 177 and 42 seconds respectively. The reduction factor is 4.17 on average. Note that the times of IPS-3 are similar to those of IPS-2 (see Raymond *et al.* (2009b)) since the latter does not include the phase I time used to find the initial basis.

In short, starting with an initial feasible solution increases the performance time of CPLEX while augmenting the generality of IPS. The performance time of the latter is decreased when the initial feasible solution is given since few computational time is needed to obtain an initial basic feasible solution (phase I) or to complete it.

4.4.4 Solution Time as a Function of the Quality of the Initial Solution

As we stated in §4.3.1, the theory allows us to begin with an initial feasible solution. This subsection presents the results of IPS-3 on VCS instances when our algorithm starts with different initial solutions. These initial solutions have been generated by CPLEX and their costs are from 0.5 percent to 7 percent more than the optimal value. To generate these initial basic feasible solutions, we first find an optimal solution. We then execute CPLEX from phase I and write the basis in a file when we reach a solution whose objective value has the predetermined gap value compared to the optimal solution.

In Figures 4.1 and 4.2, the lines correspond to the ten VCS instances. As we show in these figures, our method is more stable when compared to CPLEX. Indeed, we see that the results of IPS-3 are almost always better when the initial solution is better. We can surely say that IPS-3 takes advantage of a good initial solution. By contrast,

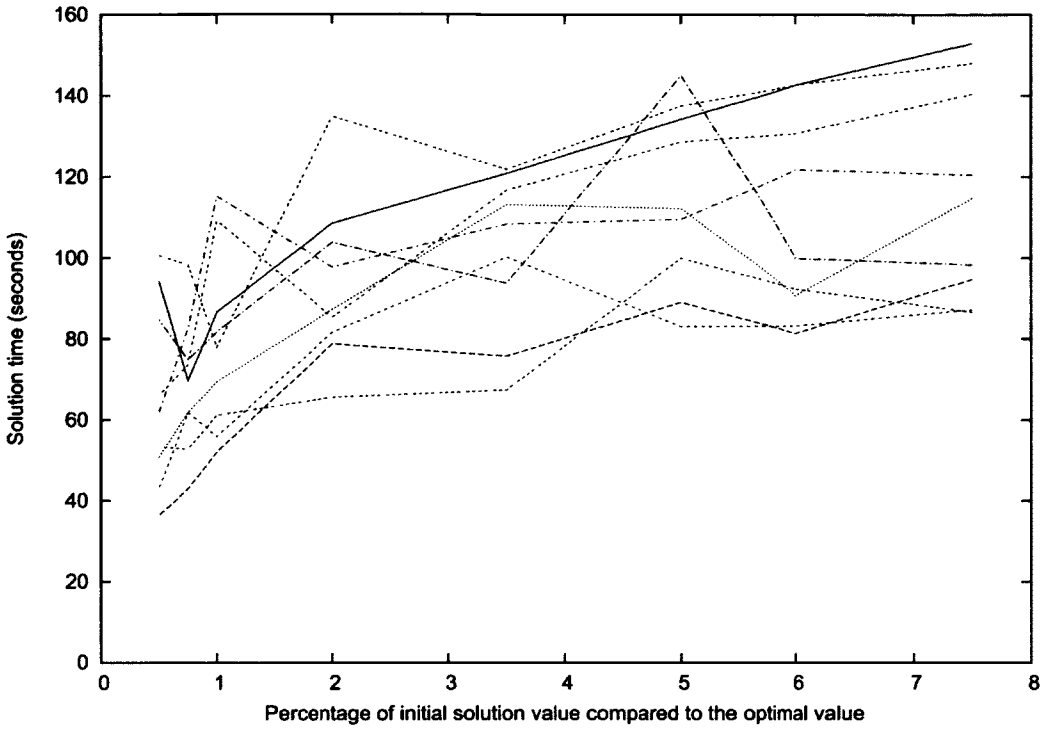

Figure 4.1: Results of CPLEX on VCS instances

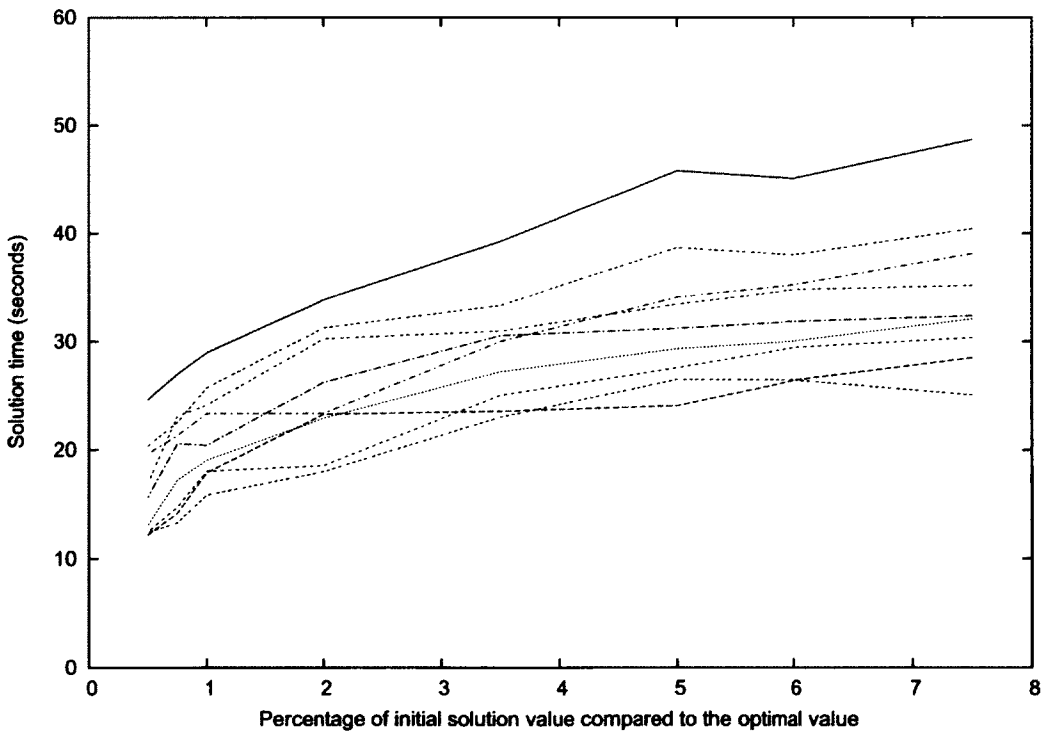

Figure 4.2: Results of IPS-3 on VCS instances

the results with CPLEX on VCS instances show that the initial solution can handicap the method.

4.4.5 Fleet Assignment With Bounded Variables Data Set

Fleet assignment (FA) problems consist in maximizing the profits of assigning a type of aircraft to each flight segment over an horizon of one week. The paths of the aircraft must respect maintenance conditions and availability. Our problem instances are generated from real data with 2505 flight segments and four types of aircraft. The variables are associated with flight sequences between maintenance. Those problems have one set partitioning constraint per flight segment, one availability constraint per aircraft type, and one flow conservation constraint between flight sequences at the maintenance base.

Those problems are not so large but they represent some typical master problems that need to be solved at each iteration of a column generation algorithm embedded in a *branch & bound* procedure. These instances were used by Raymond *et al.* (2009b).

To test IPS-3 on problems that contain bounds on variables, we modify our FA instances. We add an upper bound of 1 on each variable, i.e., $x_i \leq 1,\ i = 1, ..., n$. The resulting instances are called *upper bounded fleet assignement* (UBFA) problems. We choose these instances instead of the VCS because there is a significant number of variables with value of 1 in optimal solutions of FA instances. Consequently, adding upper bounds of 1 on variables in FA instances increases degeneracy.

Table 4.4 gives the number of constraints and variables in each instance along with the average percentage of degenerate variables ($x_i = 0$) and the average percentage of variables that are at their upper bounds ($x_i = 1$) encountered with IPS-3.

Table 4.4: Characteristics of UBFA instances.

instance	const.	variables	degeneracy $x_i = 0$	degeneracy $x_i = 1$	instance	const.	variables	degeneracy $x_i = 0$	degeneracy $x_i = 1$
UBFA6	5067	17594	68%	12%	UBFA13	5159	25746	65%	15%
UBFA7	5159	20434	59%	11%	UBFA14	5159	22641	71%	15%
UBFA8	5159	21437	65%	14%	UBFA15	5182	23650	63%	13%
UBFA9	5159	23258	66%	14%	UBFA16	5182	23990	64%	13%
UBFA10	5159	24492	66%	14%	UBFA17	5182	24282	65%	14%
UBFA11	5159	24812	66%	14%	UBFA18	5182	24517	65%	14%
UBFA12	5159	24645	66%	14%	UBFA19	5182	24875	65%	14%

4.4.6 Results for UBFA Data Set

We used UBFA instances to test IPS-3. To start the algorithms, we find initial feasible solutions through the phase I of CPLEX. These initial solution values are at 1.5 percent of the optimal solution values on average.

Table 4.5: Results on UBFA instances.

instance	CPLEX	IPS-3	reduction factor	instance	CPLEX	IPS-3	reduction factor
UBFA6	581	43	13.51	UBFA13	1362	62	21.97
UBFA7	643	40	16.08	UBFA14	1590	76	20.92
UBFA8	690	38	18.16	UBFA15	1242	45	27.60
UBFA9	1083	58	18.67	UBFA16	1340	57	23.51
UBFA10	948	69	13.74	UBFA17	924	60	15.40
UBFA11	1384	50	27.68	UBFA18	1049	56	18.73
UBFA12	1731	61	28.38	UBFA19	1169	62	18.85
				AVERAGE	1124	55.5	20.23

Table 4.5 presents the computational time in seconds for CPLEX and IPS-3 for solving the instances. The reduction factor is the CPLEX time divided by the IPS-3 time. These factors show that on average IPS-3 is 20 times faster than CPLEX for solving problems that contain bounds on variables. This reduction factor is very significant

when such problems need to be solved thousands of times in a column generation scheme embedded in a *branch & bound* procedure.

These impressive performances of our algorithm can be explained as follow. First, we saw in §4.3.4 that the variables whose values are equal to their upper bounds are handled like zero variables. In other words, the addition of upper bounds in FA instances increases degeneracy. As a result, there are more degenerate pivots in the CPLEX solution and greater computational time. Secondly, our algorithm takes advantage of degeneracy. As a result, we treat smaller reduced problems and we obtain better solution time relative to CPLEX. Starting with an initial feasible solution instead of an initial basic feasible solution also can be a part of the explanation as well as the reduction now executed by CPLEX.

4.5 Conclusion

We proposed algorithm IPS-3: a theoretical and implementational generalization of IPS. First, we rewrote the theory so that the algorithm can start with a feasible solution instead of a basic feasible solution. We can claim that IPS-3 takes advantage of initial solutions whereas CPLEX is less predictable. Moreover, we simplified the implementation by using CPLEX instead of UMFPACK to create the reduced and the complementary problems. Then, we added procedures to handle problems with bounded variables. The computational experiments on fleet assignment instances with bounded variables show the efficiency of IPS-3. Indeed, we obtain on average a reduction factor of 20 on the total computing time compared to CPLEX.

Chapitre 5

A Pricing Criterion to Identify Non Degenerate Pivots

Ce chapitre propose un nouveau critère d'évaluation des variables. Ce critère, nommé *positive edge*, identifie les variables qui permettront d'effectuer un pivot non dégénéré, s'il existe, de la méthode du simplexe. En d'autres mots, ce critère permet d'identifier les variables *compatibles* (voir définition 1 dans §3.2.1). Ainsi, il est maintenant possible de construire le problème réduit (voir §3.2.1) en ne travaillant qu'avec les coefficients originaux de la matrice des contraintes.

La complexité de la méthode de Pan (1998) pour identifier les variables compatibles est de $O(m^2n)$. La complexité du *positive edge* est similaire à celle du calcul du coût réduit, soit $O(mn)$. De plus, il est possible de combiner le *positive edge* avec un autre critère d'évaluation, le *steepest edge* par exemple, pour réduire significativement le temps de calcul de ce dernier.

Le *positive edge* nous a permis de résoudre des problèmes de très grande taille. Les résultats préliminaires de notre algorithme de base démontrent le potentiel de notre critère. En effet, sur des problèmes réels de planification de la main-d'oeuvre de très grande taille (100 000 contraintes, 450 000 variables), nous obtenons un facteur de

diminution du temps total de résolution de 4.75 par rapport à CPLEX. Le *positive edge* est une avancée importante dans le domaine de la programmation linéaire.

5.1 Introduction

We consider a linear program in standard form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \; c^T x \quad \text{subject to } Ax = b, \; x \geq 0, \tag{LP}$$

where $c \in \mathbb{R}^n$ is the cost-vector, A is the $m \times n$ constraint matrix and $b \in \mathbb{R}^m$ is the right-hand side vector. We are particularly interested in so-called degenerate problems, on which the simplex algorithm (Dantzig, 1949) is likely to encounter degenerate pivots and possibly cycle. To avoid cycling, many pivot rules and perturbations methods have been proposed, e.g., Charnes (1952); Wolfe (1963); Bland (1977); Fukuda (1982); Ryan and Osborne (1988); Terlaky and Sushong (1993). However, these rules and methods do not improve the performance of the simplex algorithm on degenerate problems. Another way of avoiding degenerate pivots is using the steepest edge criterion. This criterion computes the improvement of the cost function for each entering variable, that is, the step size multiplied by the reduced cost. This criterion selects a non degenerate pivot (step size > 0) if it exists. However, the steepest edge uses significant computational time.

To improve the performance of resolution of degenerate problems, Pan (1998) proposed using a *reduced problem* with a smaller number of variables and constraints. His method starts with an initial basic feasible solution and identifies its p nonzero components. The $m-p$ constraints corresponding to the zero variables are temporarily removed, to leave a $p \times p$ non degenerate basis. To preserve feasibility, variables that cannot become basic without violating one of the $m - p$ eliminated constraints must be also removed from the problem. The variables that are not removed are said to be *compatible*. Those variables that are removed are said to be *incompatible*—these definitions will be detailled in §5.2. Note that since the $p \times p$ basis of the reduced

problem is non degenerate, that is, the number of non zero variables are p, the next pivot will be non degenerate. The resulting reduced problem is solved and the reduced costs are computed by means of its dual variables. In Pan's method, dual variables of (LP) corresponding to the $m - p$ eliminated constraints are set to zero. At the next iteration, if an incompatible variable is to become basic, one of the eliminated constraints must be reintroduced in the reduced problem. When compared to his own primal simplex, Pan reports gains of a factor of four to five. However, his program cannot compete with commercial simplex algorithms.

In the same vein, El Hallaoui *et al.* (2005) developed a *dynamic constraint aggregation* (DCA) method for partitioning problems. The method groups constraints that become identical with respect to nonzero basic variables, a stage referred to as *aggregation.* In the reduction phase, a single row per constraint group remains in the reduced problem. As its name implies, DCA allows further reductions while the reduced problem is being solved. Once the reduced problem is solved, dual variables are recovered by *disaggregating* which corresponds to solving a shortest-path problem. The dual variables then allow computing the reduced costs of incompatible variables. To improve their method, the authors developed the *multi-phase dynamic constraint aggregation* method (MPDCA) (El Hallaoui *et al.*, 2008b). The goal of MPDCA is to add to the reduced problem the incompatible variables in order of their number of incompatibilities. On a set of large bus driver scheduling problems in a column-generation framework, MPDCA reduces the solution time by a factor of more than 23 over the classical column-generation method.

The *Improved Primal Simplex* method (IPS) (El Hallaoui *et al.*, 2007) combines ideas from Pan (1998) and DCA to solve (LP). Instead of disaggregating the dual variables to compute the reduced costs, the incompatible variables that are to become basic are obtained from the solution of a *complementary problem.* This latter problem contains all the incompatible variables and its coefficient matrix is created at the

same time as the reduced problem. El Hallaoui *et al.* (2007) show that when the reduced problem is not degenerate, the objective function decreases at each iteration. In practice, reducing the problem until all degeneracy disappears is computationally costly. Partial reduction, however, yields good results.

As we will detail it in §5.2, the reduced problem and the complementary problem work with a modified constraint matrix. This matrix is obtained by multiplying B^{-1} by A, where B^{-1} is the $m \times m$ inverse matrix of a basis of (LP). The complexity of computing this modified matrix is $O(m^2 n)$ (see §5.3). The results of Raymond *et al.* (2009b) show that, on medium-sized instances ($m \approx 5000$, $n \approx 24\ 000$), using several modified constraint matrices to solve (LP) is faster than the simplex algorithm. However, on large-scaled problems ($m \approx 100\ 000$, $n \approx 450\ 000$), constructing the reduced problem and the complementary problem is too costly compared to the simplex algorithm itself.

This paper seeks to use the idea of IPS, that is, to give a certain priority to non degenerate pivots, on the original constraint matrix instead of on the modified one. In fact, we develop the *positive edge* pivot rule that we apply on the initial constraint matrix that allows to select a non degenerate pivot when it exists. Obviously, as in IPS, we need to execute some degenerate pivots to reach optimality.

This paper is organized as follows. §5.2 presents the reduced problem and the compatibility concept of Pan (1998). In §5.3, we present our positive edge pivot rule. Basic algorithm and preliminary results are reported in §5.4. Finally, we discuss future research and conclude in §5.5.

5.1.1 Notation

If $x \in \mathbb{R}^n$ and $\mathrm{I} \subseteq \{1, \ldots, n\}$ is an index set, we denote by x_{I} the subvector of x indexed by I. Similarly, if A is an $m \times n$ matrix, we denote by A_{I} the $m \times |\mathrm{I}|$ matrix whose columns are index by I. If $\mathrm{J} = \{1, \ldots, n\} \setminus \mathrm{I}$, we write $x = (x_{\mathrm{I}}, x_{\mathrm{J}})$ even though the indices in I and J may not appear in order. Moreover, we denote with upper indices the subset of rows associated with the set of indexed variables.

5.2 Reduced Problem

In this section, we summarize the construction and the key properties of the reduced problem proposed by Pan (1998) and developed by El Hallaoui *et al.* (2007).

Let A_j be the jth column of A and B be the index set of basic variables for (LP). Let $\bar{x} = A_{\mathrm{B}}^{-1}b = A_{\mathrm{B}}^{-1}Ax$ be a degenerate feasible point with $p > 0$ nonzero components, which, without loss of generality, we assume to be its first p components. Let P be the index set of the non zero variables, that is, $\mathrm{P} = \{i \in \{1, ..., n\} \mid \bar{x}_i > 0\}$, $Q = A_{\mathrm{B}}^{-1}$ and partition

$$Q = \begin{bmatrix} Q^{\mathrm{P}} \\ Q^{\mathrm{Z}} \end{bmatrix}, \tag{5.13}$$

where Q^{Z} is the *compatibility matrix* formed by the last $m - p$ rows of Q. We have

$$\bar{x} = \begin{bmatrix} Q^{\mathrm{P}}b \\ Q^{\mathrm{Z}}b \end{bmatrix}, \text{ and therefore, } Q^{\mathrm{Z}}b = 0. \tag{5.14}$$

We begin with the following definition.

Definition 3 *The jth variable of (LP), x_j, is said to be* compatible *if and only if* $Q^z A_j = 0$.

From Definition 3, we let $\mathrm{C} \subseteq \{1, \ldots, n\}$ denote the indices of variables that are compatible and $\mathrm{I} = \{1, \ldots, n\} \setminus \mathrm{C}$. Thus, x_C and x_I are the subvectors of x of compatible and incompatible variables, respectively.

We partition the cost vector c accordingly into c_C and c_I, and the columns of A into A_C and A_I. Upon premultiplying the equality constraints of (LP) by Q, we obtain $QAx = Qb$, which may be rewritten

$$\begin{bmatrix} Q^\mathrm{P} Ax \\ Q^\mathrm{Z} Ax \end{bmatrix} = \begin{bmatrix} Q^\mathrm{P} A_\mathrm{C} x_\mathrm{C} + Q^\mathrm{P} A_\mathrm{I} x_\mathrm{I} \\ Q^\mathrm{Z} A_\mathrm{C} x_\mathrm{C} + Q^\mathrm{Z} A_\mathrm{I} x_\mathrm{I} \end{bmatrix} = \begin{bmatrix} Q^\mathrm{P} b \\ 0 \end{bmatrix},$$

where we used (5.14). Since by definition x_C is compatible with Q^Z, we have $Q^\mathrm{Z} A_\mathrm{C} x_\mathrm{C} = 0$. Upon imposing $x_\mathrm{I} = 0$, we obtain the *reduced problem*

$$\underset{x_\mathrm{C}}{\text{minimize}} \; c_\mathrm{C}^\top x_\mathrm{C} \quad \text{subject to } Q^\mathrm{P} A_\mathrm{C} x_\mathrm{C} = Q^\mathrm{P} b, \; x_\mathrm{C} \geq 0. \tag{RP}$$

Notice that the concept of compatibility is the same as Pan (1998), i.e, the compatible variables can enter into the basis for (RP) without violating the constraints of (LP) that have been omitted. Problem (RP) is potentially much smaller than (LP) and is obtained from a degenerate primal basis. It only depends on the compatible variables. Note that by construction, if x_C is feasible for (RP), then $(x_\mathrm{C}, 0)$ is feasible for (LP).

5.3 Positive Edge Rule

The identification of the compatible variables using the method of §5.2 has a complexity of $\mathrm{O}(m^2 n)$. In fact, the method multiplies an $(m-p) \times m$ matrix (Q^Z) by

an $m \times n$ matrix (A). Even if A is sparse with a density λ_a, the complexity remains $O(\lambda_a m^2 n)$ (see §5.3.2). For large-scale problems, this method should be more costly, in terms of computing time, than the solution of (LP) with the simplex method.

To identify the columns that are compatible with the current degenerate basis of (LP) without multiplying Q^z by A, we developed the *positive edge* rule. This rule allows to know (with a very small error probability) if a variable will enter into the basis with a positive value without modifying the rows associated with the zero variables. If such a variable has a negative reduced cost, this variable will strictly decrease the objective function (minimisation problem) when entered in basis. That is why we called this method a pivot rule. However, we can use the positive edge criterion to identify compatible variables (as described in §5.2) from the original coefficients of the constraint matrix.

Let w^z be a row vector of $(m - p)$ components and $w = w^z Q^z$. We know from Definition 3 that $Q^z A_c = 0$. A necessary condition to identify a compatible variable is $wA_c = w^z Q^z A_c = 0$. Can we choose w^z such that this condition becomes sufficient? The next section answers this question. We will explain that most of the time, when choosing w^z with some particularities, if $wA_j = 0$ then the variable x_j is compatible. The positive edge criterion consists to compute wA_j to know if the variable j is compatible.

5.3.1 Reliability of the Positive Edge Criterion

We study in this section the reliability of the positive edge criterion to identify compatible variables.

From numerical computing with IEEE floating point arithmetic with single precision (Overton, 2001), the standard 32-bit representation of a word may be numbered from 0 to 31, left to right. The first bit is the sign bit, the next eight bits are exponent bits, and the final 23 bits are the numerator fraction. The set of finite real numbers can be represented with the simple form $2^{c-24}d$ with c and d integers and $1-2^7 < c < 2^7$ and $-2^{24} < d < 2^{24}$.

Every representation of d in $]-2^{24}, 2^{24}[$ and c in $]1-2^7, 2^7[$ has the same probability. A *floating point number with distribution* U^2 is a number $w = 2^{c-24}d$ where d and c are uniform in $]-2^{24}, 2^{24}[$ and $]1-2^6, 2^6[$ respectively. Notice that we select exponent in the interval $]1-2^6, 2^6[$ to be able to do computation without overflow. The computer standardizes (translates such that the first bit of d is 1) to lose information as little as possible during the operation. With these translations, the probability of the i^{th} bit is no longer equal to $\frac{1}{2}$. With the translated representation, the following proposition becomes more complex. However, to obtain simplified valid results, we consider the calculation with the non-translated representation of w but with sufficient bits such that no information is lost. If, after the calculation, we translate and truncate to 32-bit, we obtain the same result as the translated w.

We first prove the two following lemmas.

Lemma 1 *Let d be a positive integer. The addition of an integer with d-bits uniformly distributed with any d-bit integer (without any information on its distribution) is an integer uniformly distributed.*

In fact, let m_1 be a l-bit number uniformly distributed without exponents and m_2 be any l-bit integer. This means that the probability of the i^{th} bit of m_1 is $P\left(i^{th} \text{ bit } = 1\right) = \frac{1}{2}$. From Table 5.1, the probability of the i^{th} bit of $m_1 + m_2$ of

being 0 or 1 is $\frac{1}{2}$ for any number m_2 even if we do not know the remainder of $(i-1)^{th}$ bit.

Table 5.1: Probability distribution of $m_1 + m_2$

	m_1	m_2	(m_1+m_2) without remainder	$(i-1)^{th}$ bit remainder	(m_1+m_2) with remainder	i^{th} bit remainder
P(i^{th} bit $=0$)	$\frac{1}{2}$	a	$\frac{1}{2}a + \frac{1}{2}(1-a) = \frac{1}{2}$	b	$\frac{1}{2}b + \frac{1}{2}(1-b) = \frac{1}{2}$	$\frac{a+b}{2}$
P(i^{th} bit $=1$)	$\frac{1}{2}$	1-a	$\frac{1}{2}(1-a) + \frac{1}{2}a = \frac{1}{2}$	1-b	$\frac{1}{2}(1-b) + \frac{1}{2}b = \frac{1}{2}$	$1 - \frac{a+b}{2}$

□

Lemma 2 *The result obtained from the multiplication of a floating point number with distribution U^2 and a non-zero number (without any information on its distribution) is also a floating point number with distribution U^2.*

Multiplication is based on addition. Let $x_1 = 2^{c_1-24}d_1$ and $x_2 = 2^{c_2-24}d_2$. We have $x_1x_2 = 2^{c_1+c_2-48}d_1d_2$. For the d_1d_2 part, the example of the multiplication of an integer uniformly distributed with the binary number 1010 gives an integer uniformly distributed.

				a_1	a_2	$\cdots$	a_{21}	a_{22}	a_{23}
$\times$						1	0	1	0
				0	0	$\cdots$	0	0	0
$+$			a_1	a_2	$\cdots$	a_{21}	a_{22}	a_{23}	
$+$		0	0	$\cdots$	0	0	0		
$+$	a_1	a_2	$\cdots$	a_{21}	a_{22}	a_{23}			
				23 bits			truncated area		

From Lemma 1, this finite sum gives 0 or 1 at each bit $i \in \{1, \ldots, 23\}$ with the same probability $\frac{1}{2}$ because the probability $P(a_i = 1) = \frac{1}{2}$ for all i and these bits are independent. We can also show from Lemma 1 that $c_1 + c_2$ is an integer uniformly

distributed for the first six bits of its exponent, because c_1 and c_2 are two integers uniformly distributed. Then $w_i^z(Q^zA)_i$ is a floating point number with distribution U^2 for all i. We can assume that $(Q^zA)_i = 2^{c_i-24}d_i$ with $1-2^6 < c_i < 2^6$. Since $k \leq 2^{64}$ and the exponent bits of x_1 and x_2 satisfy $1-2^6 < c_1, c_2 < 2^6$, then we do not have overflow. □

Proposition 4 *Let A_j be an incompatible column $(Q^zA_j \neq 0)$ and let w^z be a row vector of $(m-p)$ floating point number with distribution U^2 in a 32-bit computer with $w_i^z \neq 0$. The probability that $w^zQ^zA_j = 0$ is less than $\frac{1}{2^{144}}$.*

Proof: We assume that $(Q^zA)_i \neq 0$ for $i \in \{1, \ldots, k\}$ where k is the number of non zero coefficients of (Q^zA).

If $k = 2$, then $P(w^z(Q^zA) = 0) = P(w_1^z(Q^zA)_1 + w_2^z(Q^zA)_2 = 0)$. The probabilities of having $w_1^z(Q^zA)_1 = -w_2^z(Q^zA)_2$ is the number of representation of the value $w_1^z(Q^zA)_1$ on all the possible values of a floating point number with distribution U^2 (except the value zero). By taking into account the assumptions described previously and by computing the bit sign with the 23-bit of the numerator, there is $2^{24}-2$ possibilities of numerator number. The number of possibilities of the exponent is $2^{2^7-2} = 2^{126}$. The number of representation of the value zero is the number of possibilities of exponent. The number of representation of a nonzero value is 1 plus the number of zero before the first 1 of n plus the number of zero after the last 1 of n. Thus, the maximum number of representation of a nonzero value is 22. Consequently, $P(\sum_i^{k=2} w_i^z(Q^zA)_i = 0) \leq 23/[(2^{24}-2)(2^{126}) - 2^{126}]$.

When $k = 3$, there are two cases. If $\sum_j^{k=2} w_j^z(Q^zA)_j = 0$, then $P(\sum_j^{k=3} w_j^z(Q^zA)_j = 0) = 0$. If $\sum_j^{k=2} w_j^z(Q^zA)_j \neq 0$, then $P(\sum_j^{k=3} w_j^z(Q^zA)_j = 0) = P(\sum_i^{k=2} w_i^z(Q^zA)_i = 0)$. Thus, if we apply these facts recursively, we obtain $P(\sum_i^{k>2} w_i^z(Q^zA)_i = 0) \leq$

$P(\sum_i^{k=2} w_i^z (Q^z A)_i = 0)$. Consequently,

$$P(w^z Q^z A = 0) \leq \frac{22}{[(2^{24}-2)(2^{126})] - 2^{126}} \leq \frac{1}{2^{144}}.$$

□

Let summarize the positive edge criterion:

1. Let $w^z \neq 0$ be a row-vector of floating point numbers with distribution U^2.
2. Compute $w = w^z Q^z$.
3. Compute wA_j.
4. If $wA_j = 0$ then A_j is compatible with an error probability of $\frac{1}{2^{144}}$.

In the case of an integration of the positive edge rule into a simplex method, the positive edge criterion can be combined with the best negative reduced cost criterion to identify a pivot that will strictly decrease the objective function. In the case of IPS method, the positive edge criterion can be used to construct the reduced problem since it identifies the compatible variables.

As part of a simplex method, if $wA_j = 0$, x_j is incompatible and x_j is the entering variable, the algorithm will perform a degenerate pivot unless $(Q^z A_j)_i < 0$ for all i. Note that the positive edge criterion does not identify the non degenerate pivot of a variable x_j with $(Q^z A_j)_i \leq 0$ for all i. The positive edge rule identifies compatible variables similar to Pan's concept.

In the context of IPS method, the reduced problem may contain (1 time on 2^{144}) an incompatible variable j such that $wA_j = 0$. In this situation, if the variable x_j is in the basis A_B and we must reduce the problem, the basis A_B will be infeasible for (LP). In this case, we must find a new basis by starting with A_B and by solving (LP) until we obtain a feasible basic solution. Then we construct the reduced problem with the

new feasible basis. Another situation may happen : if the variable x_j is in the optimal basis solution of (RP), this basis will be infeasible for (LP). In this case, since IPS terminates by solving (LP) and by starting with the optimal solution of (RP), the method will find the optimal solution anyway. The only inconvenient is that IPS may take more computational time than usual. However, to reduce the probability of this situation, more than one test of the positive edge criterion may be computed.

We will explain in §5.4 that our algorithm eliminates incompatible variables but does not eliminate degenerate rows. In this case, the consequence of identifying wrongfully an incompatible variable is the same as in the context of a simplex method.

5.3.2 Complexity Discussion

In this section, we discuss the complexity of our criterion and of Pan's method.

Recall that λ_a is the density of the constraint matrix, let λ_b be the percentage of the non zero coefficients in the compatibility matrix Q^z and l be the number of basic computer cycles needed for a multiplication and an addition. Recall that m is the number of constraints, n is the number of variables and $m < n$ as is encountered most of the time.

We compute the complexity of the positive edge criterion for n variables. First, we need to multiply a vector of $m - p$ non zero elements by a $(m - p) \times m$ matrix (Q^z), that is, $lm\lambda_b(m - p)$ basic cycles. We multiply this resulting vector by the constraint matrix A. We obtain a total complexity of

$$[lm\lambda_b(m - p)] + [ln\lambda_a m] = O(mn).$$

To compute the Pan's reduction, we need to multiply the $m \times m$ inverse basic matrix (Q) by the $m \times n$ constraints matrix (A). For each coefficient of the resulting matrix, we must multiply a row of Q by a column of A. This uses $l\lambda_a m$ basic computer cycles. For computing the mn coefficients, we get

$$lmn\lambda_a m = O(m^2 n).$$

We compare the time needed to identify the compatible variables with Pan's method and with the positive edge rule. On large-scale instances ($m \approx 100\ 000$, $n \approx 450\ 000$) Pan's method needs 2500 seconds. On the same problem, our positive edge rule needs only 0.5 seconds. On the tested instances, the percentage of the variables that are compatible is 10%.

5.4 Applications

This section proposes algorithms using the positive edge criterion in an external loop to identify variables allowing non degenerate pivots. That is because we do not have access of the internal working of ILOG to fully integrate the positive edge in CPLEX.

We present, first, our basic algorithm and its possible improvements. Then, we present computational experiments that we obtained on *fleet assignment* (FA) problems, *manpower planning* (MPP) problems and benchmark instances for commercial LP solvers. The definition and the characteristics of these instances are also presented.

5.4.1 Partially Reduced Algorithm

We are now in a position to state a basic algorithm using the positive edge criterion. We qualify this algorithm basic since it does not work with a fully reduced problem such as the one described by Pan (1998). Actually, the reduced problem in our basic algorithm contains only the compatible variables but it contains all rows of A. We call it the *partially reduced problem* (PRP). This is because we want to keep up to date the dual variables for all rows of A. Indeed, the use of a fully reduced problem would need to add a procedure computing the dual variables associated with the eliminated rows. Thus, when we work with PRP, the reduced cost of the incompatible variables are easy to obtain but the basis is larger and the cost per iteration is higher.

Algorithm 5.1 Partially reduced algorithm

1: Start with an initial basis.
2: Identify the compatible variables with respect to the current basis.
3: Construct PRP by deleting the incompatible variables.
4: Solve PRP until σ pivots are performed or optimality is reached.
5: Let k be the number of negative reduced cost incompatible variables.
6: If $k = 0$ and PRP is optimal, STOP, the current solution of PRP is optimal for (LP).
7: If $k = 0$ and PRP is not optimal, go to Step 4.
8: Construct APRP by adding the k negative reduced cost incompatible variables to PRP.
9: Solve APRP until β pivots are performed or optimality is reached. Go to Step 2

The *partially reduced algorithm* (PRA) is stated as Algorithm 5.1. We call the *augmented partially reduced problem* (APRP) the PRP that contains also incompatible variables having a negative reduced cost. In Step 1, the algorithm starts with a feasible basis as in El Hallaoui *et al.* (2007). Then, the method uses the positive edge criterion to identify in Step 2 the compatible variables. By removing the incompatible variables, we form the PRP at Step 3. We solve at Step 4 the PRP problem until optimality is reached or σm pivots are performed. At Step 5, we compute the reduced cost of

all incompatible variables using the dual variables of PRP. The number of negative reduced cost incompatible variables is denoted k. The optimality criterion is at Step 6: when $k = 0$ and PRP is optimal, thus no variables have a negative reduced cost, so the current solution is optimal for (LP). If $k = 0$ and PRP is not optimal (Step 7), return to Step 4. If $k > 0$, we add the k incompatible variables to PRP to form APRP (Step 8). At Step 9 the APRP problem is solved to optimality or until βm pivots are performed. Then the method recreates the PRP problem by returning to Step 2.

For all tests in this section, the value of σ is obtained by a piecewise linear function. More precisely, $\sigma = 0.2$ if $m \leq 10\ 000$, $\sigma = 0.1$ if $10\ 000 < m \leq 25\ 000$, $\sigma = 0.05$ if $25\ 000 < m \leq 75\ 000$, $\sigma = 0.02$ if $75\ 000 < m \leq 400\ 000$ and $\sigma = 0.01$ if $m > 400\ 000$. The value of β is 2σ.

5.4.2 Improvements

It would be possible to work with a *fully reduced problem* if we could have access to the information of the $PLUQ$ decomposition computed by CPLEX. In fact, the computation of the inverse basis by CPLEX is a *black box*. By having the L^{-1} and the Q matrices of the $PLUQ$ decomposition, it would be possible to reduce completely the problem since we would know the row association and permutations between the A and the B^{-1} matrices, which means that we could solve (RP). Then we could compute the dual variables by adding the rows eliminated and the basic variables associated with each of these rows.

In §5.4.3, we present the results of PRA and an estimation of the result that we would obtain if we could know the information above, i.e., the *fully reduced algorithm* (FRA).

Moreover, it would be interesting to integrate the positive edge pivot rule into a simplex method such as CPLEX. Thus, the efficiency of the positive edge criterion could be greater with partial pricing and the other internal improvement procedures of CPLEX. Furthermore, as we mentioned in §5.3.2, we could combine the positive edge criterion with other pricing rules such as the steepest edge.

5.4.3 Preliminary Results

This section presents the first results obtained with our basic algorithm. We begin with the presentation of the medium-sized FA instance characteristics, followed by the results of that type of problem. Then we present instance characteristics and the results on large-scale problems (MPP instances). We complete the tests with some benchmark instances for commercial LP solvers.

All tests were performed on a 2.8GHz PC running Linux and all comparisons are made with CPLEX 9.1.3. All time results are in seconds.

FA Instances

Fleet assignment problems consist in maximizing the profits of assigning a type of aircraft to each flight segment over a horizon of one week. The paths of the aircrafts must respect maintenance conditions and availability. Our problem instances are generated from real data with 2505 flight segments and four types of aircraft. The variables are flight sequences between maintenance bases. Those problems have one set partitioning constraint per flight segment, one availability constraint per aircraft type, and one flow conservation constraint between flight sequences at maintenance base. Some variables are also bounded above. Those problems are not so large

but there are some typical master problems that need to be solved at each iteration imbedded in a *branch & bound* procedure and in a column generation algorithm. These instances were used in Raymond *et al.* (2009a, b).

Table 5.2 gives the number of constraints and variables of each instance along with the average proportion of degenerate variables encountered with IPS.

Table 5.2: Characteristics of FA instances.

instance	constraints	variables	degeneracy	instance	constraints	variables	degeneracy
FA6	5067	17594	68%	FA13	5159	25746	65%
FA7	5159	20434	59%	FA14	5159	22641	71%
FA8	5159	21437	65%	FA15	5182	23650	63%
FA9	5159	23258	66%	FA16	5182	23990	64%
FA10	5159	24492	66%	FA17	5182	24282	65%
FA11	5159	24812	66%	FA18	5182	24517	65%
FA12	5159	24645	66%	FA19	5182	24875	65%

For the FA instances, the partially reduced algorithm finds an optimal solution in 142 seconds on average compared to an average of 578 seconds for CPLEX. The average reduction factor (CPLEX / PRA) of the total computing time is 4.01. This saving becomes significant when more than thousand such problems need to be solved during the complete *branch & bound* and column generation algorithm.

We note that we start both algorithms with the initial basis obtained by a phase I computed by CPLEX. Table 5.3 presents the results for the FA instances. The second and the third columns are the time in seconds of the algorithms and the reduction factor column is the ratio of CPLEX time to PRA time.

Table 5.3: Computational results for FA instances.

instance	CPLEX	PRA	reduction factor	instance	CPLEX	PRA	reduction factor
FA6	269	84	3.20	FA13	696	149	4.67
FA7	289	90	3.21	FA14	662	150	4.41
FA8	419	110	3.81	FA15	577	178	3.24
FA9	498	128	3.89	FA16	947	168	5.63
FA10	654	158	4.13	FA17	612	155	4.15
FA11	546	150	3.64	FA18	665	160	4.16
FA12	617	142	4.35	FA19	641	167	3.83
				AVERAGE	578	142	4.01

MPP Instances

Manpower planning problems are two instances that we obtained from the compagny AD OPT. The instances have approximately 99 000 constraints and around 450 000 variables. The average proportion of degenerate variables encountered with PRAis 80% for both.

Here, we present the results that we obtained with our partially reduced algorithm for the MPP instances. Moreover, we estimate the results that we would obtain if we were able to know the $PLUQ$ decomposition that CPLEX used (see §5.4.2). That is, we estimate the results that we would obtain if we had worked with a fully reduced problem instead of a partially reduced problem. To estimate the time of FRA, we construct and solve the first fully reduced problem encountered and we compare its solution time with the solution time of the first partially reduced problem. Then, we extrapolate the total time of FRA by using the times ratio computed previously.

Furthermore, we present our results when we start with an initial basis computed by phase I of CPLEX and when we start without a basis. When the methods start

with an initial basis, as in a column generation algorithm, we do not include the time for finding it in the total computing time. Conversely, when the algorithms start without a basis, then the total computing time includes the phase I time. Moreover, when CPLEX starts without an initial basis, it uses its *presolve* procedure. This situation is the worst case analysis. Thus, even if both algorithms start from scratch and the CPLEX algorithm used its well-tuned presolve procedure, the computational experiments show that the positive edge criterion is efficient and has great potential.

Table 5.4: Results for MPP instances starting without basis.

instance	CPLEX (presolve)	PRA	FRA	reduction factor
MPP1	908	365	329	2.74
MPP2	863	371	334	2.58
AVG	883	368	332	2.66

Table 5.4 presents solution time in seconds of the partially reduced algorithm, the fully reduced algorithm and the CPLEX method starting without an initial basis. Table 5.5 presents the solution time when the three algorithms start with a phase I initial basis. Note that the reduction factor of total computing time given in Tables 5.4 and 5.5 is the ratio of the CPLEX time on the FRA time.

Table 5.5: Results for MPP instances starting with phase I basis.

instance	CPLEX (no presolve)	PRA	FRA	reduction factor
MPP1	1500	345	309	4.85
MPP2	1454	350	313	4.65
AVG	1477	348	311	4.75

We see that, when the algorithms started without a basis, CPLEX needed 883 seconds on average to reach optimality compared to 368 for our partially reduced algorithm and 332 for the fully reduced algorithm. When started with a phase I initial basis, CPLEX needed 1477 seconds whereas our partially reduced algorithm needed 348 seconds and the fully reduced algorithm needed 311 seconds on average to find the optimal solution. The reduction factor of total computing time is 2.66 in the worst case, that is, when the algorithms start without a basis and 4.75 when they start with a phase I initial basis.

Other Instances

The instances of this section are given on Mittelmann's web page[1] and they are benchmark instances for commercial LP solvers. We choose these large-scale problems because there are highly degenerate. Table 5.6 gives the number of constraints and variables of each instance along with the average proportion of degenerate variables encountered with PRA.

Table 5.6: Characteristics of Mittelmann instances.

instance	constraints	variables	degeneracy	instance	constraints	variables	degeneracy
PDS-20	33 874	108 175	82%	PDS-80	129 181	434 580	84%
PDS-30	49 944	158 489	82%	PDS-90	142 823	475 448	84%
PDS-40	66 844	217 531	81%	PDS-100	156 243	514 577	84%
PDS-50	83 060	274 814	82%	Fome12	24 285	48 920	54%
PDS-60	99 431	336 421	83%	Fome13	48 569	97 840	46%
PDS-70	114 944	390 005	83%	Storm	528 186	1 259 121	57%

Several algorithm performance times are given on Mittelmann's web page. All of these algorithms start without an initial basis. To follow the methodology of Mittelmann

[1] http://plato.asu.edu/ftp/lpcom.html

and to compare to the other algorithms, we do not give an initial basic feasible solution to our algorithm. Thus, phase I time is included in the total computing time of PRA, FRA and CPLEX. We note that the latter use the *presolve* procedure since no basis is given.

Table 5.7 gives the time results of PRA, FRA and CPLEX on the Mittelmann instances. The reduction factor presented is the times ratio of CPLEX and FRA.

Table 5.7: Results for Mittelmann instances starting without a basis.

instance	CPLEX (presolve)	PRA	FRA	reduction factor	instance	CPLEX (presolve)	PRA	FRA	reduction factor
PDS-20	12	15	14	0.86	PDS-80	1835	861	659	2.78
PDS-30	28	41	37	0.76	PDS-90	2102	1032	772	2.72
PDS-40	142	128	101	1.41	PDS-100	1901	1017	780	2.44
PDS-50	395	240	203	1.95	Fome12	704	2342	2239	0.31
PDS-60	844	439	349	2.42	Fome13	2310	13370	12900	0.17
PDS-70	1134	701	550	2.06	Storm	6751	2156	1957	3.45

The average reduction factor for the PDS instances is 1.93, where the reduction factor on the five larger instances is 2.48. Moreover, we observe that better results are obtained on the largest sizes of the PDS instances where the CPLEX solution times are higher. For the Fome problems, our algorithm did not perform well. We will discuss the bad behavior on these problems in the next section. Finally, the Storm problem is solved 3.45 times faster than the CPLEX algorithm.

5.4.4 Limitations of the Algorithm

As we mentioned previously, the algorithm that we used to show the potential of the positive edge criterion is preliminary. On Fome and on some PDS instances, our

algorithm does not improve the performance time of CPLEX. In this section, we investigate the bad performances of our algorithm on some models and we propose improvements.

From Table 5.6, we observe that the percentage of degeneracy of the Fome problems is less than the FA, the MPP and the PDS models. As our algorithm is made for high degenerate problems, a part of the bad performances can be explained by that.

In Table 5.8, we give the density of the constraint matrix and the percentage of the average of number of variables (RP var.) encountered in the reduced problems for the MPP, the PDS, the Fome and the Storm instances.

Table 5.8: Density of constraint matrix and proportion of variables in (RP).

instance	density	RP var.	instance	density	RP var.
PDS-20	6.3e-5	32%	PDS-90	1.5e-5	31%
PDS-30	4.3e-5	32%	PPDS-100	1.4e-5	31%
PDS-40	3.2e-5	32%	Fome12	1.1e-4	55%
PDS-50	2.6e-5	32%	Fome13	6.0e-5	55%
PDS-60	2.1e-5	31%	Storm	5.0e-6	57%
PDS-70	1.9e-5	31%	MPP1	2.6e-5	26%
PDS-80	1.7e-5	31%	MPP2	2.6e-5	27%

We observe that the Fome problems have a higher density than the Storm, the MPP and the PDS models. Moreover, we observe that the performance of our algorithm on PDS problems is better on problems that have a lower density. Higher density can increase the number of pivots required to solve a linear program. Indeed, entering a variable with a high density has more chance to result in a small decrease of the objective function value. It results in a long tail where the preliminary algorithm solves the reduced problem and enters a small number of negative reduced cost incompatible variables at each iteration. It is well known that algorithms that use an external loop to add variables (like column generation) need more iterations when

the density of the constraint matrix is high. The positive edge rule identifies pivots that are non null but does not give information on the size of the step of the pivot. A combination of the positive edge criterion and the steepest edge criterion, as we discuss in §5.3.2, could increase the performance of our pricing criterion on high density instances.

Moreover, the average percentage of the number of variables in the reduced problem of the Fome problems is relatively high. It means that around 55% of the variables are compatible, that is, are in the reduced problem. This is caused by a lower degeneracy compared to the MPP and the PDS models.

The Storm instance has almost the same percentage of degeneracy than the Fome problems but its density is less than all other models. Furthermore, the Storm problem is a very large-scale instance. We observe that the best performance on the PDS instances are on the larger size instances.

Important improvements could be done on our algorithm. One of the bad behavior is that a negative reduced cost incompatible variable entering in the reduced problem cannot decrease the objective value function by itself since it is incompatible (entering a incompatible variable cause a degenerate pivot). We see in the paper of El Hallaoui *et al.* (2007) that entering a combination of incompatible variables that solve a *complementary problem* results in a strict decrease of the objective function value. A complementary problem could be created from original coefficients and solved to select better incompatible variables. Finally, a fully integration of the positive edge criterion in a commercial simplex method would eliminate the external loops.

5.5 Conclusion

We propose a new pricing criterion to identify the columns allowing non degenerate pivots. We called our new pricing rule the *positive edge* criterion. We develop a basic algorithm that uses the IPS scheme (see El Hallaoui *et al.* (2007)) and the positive edge criterion. The preliminary results show the potential of our pricing rule. We tested our basic algorithm on medium-sized instances of the fleet assignment problem and on large-scale instances of MPP. On the first group of instances, we obtained a reduction factor of more than 4. For large-scale problems, the reduction factor is 4.75 for the MPP instances. Other large-scale instances have been tested and the reduction factor is up to 3.45.

For future research, we shall try to develop an algorithm that works with a fully reduced problem. Moreover, it would be more efficient to solve a subproblem to identify a better set of incompatible variables to enter in the reduced problem as in El Hallaoui *et al.* (2007). Lastly, it would be interesting to integrate the positive edge criterion in a commercial simplex method.

Discussion Générale et Conclusion

Nous nous sommes intéressés à la dégénérescence dans les programmes linéaires. Pour augmenter la performance de la résolution par la méthode du simplexe, Pan (1998) propose de solutionner itérativement des problèmes réduits en variables et en contraintes. Villeneuve (1999) et El Hallaoui *et al.* (2005, 2008b, a) ont développé la méthode de Pan (1998) pour les problèmes de partitionnement. Pour les problèmes linéaires généraux, El Hallaoui *et al.* (2007) proposent de résoudre un problème complémentaire pour identifier les variables à entrer dans le problème réduit.

De ces faits, aucun algorithme utilisant la méthode du problème réduit n'avait été implémenté de façon efficace pour les problèmes linéaires généraux. De plus, la théorie n'était pas généralisée pour les problèmes comportant des bornes sur les variables. Ensuite, la théorie actuelle impose de débuter l'algorithme qu'avec une base initiale complète. Enfin, la théorie de Pan (1998) nécessite la multiplication de la matrice des contraintes par l'inverse de la base. Cette multiplication ne permet pas de solutionner des problèmes de très grande taille.

À partir de ces limites, nous avons exposé nos objectifs : implémenter l'algorithme de El Hallaoui *et al.* (2007) de manière à résoudre efficacement chaque sous-procédure et l'ensemble de celles-ci ; généraliser la théorie et implémenter de l'algorithme ; résoudre des problèmes dégénérés de très grande taille.

Le premier objectif a été atteint. Effectivement, nous avons présenté des résultats numériques qui démontrent l'efficacité de notre implémentation et de nos recherches. Un facteur de diminution du temps de résolution de 12 a été observé lors de la comparaison de notre algorithme à CPLEX sur des problèmes de répartition de flottes

d'avions. Ainsi, avec le premier article, nous avons présenté un algorithme efficace pour les problèmes linéaires dégénérés de taille considérable ($m = 5000$, $n = 20\ 000$).

Notre deuxième objectif a aussi été atteint. Nous avons présenté dans le chapitre 4 l'article qui traite de cet objectif. En effet, nous avons généralisé la théorie et l'implémentation de IPS pour obtenir un algorithme qui n'utilise qu'un logiciel (CPLEX), qui débute avec une solution initiale plutôt qu'une base initiale et qui résout des problèmes comportant des bornes sur les variables. De plus, nous avons démontré par nos résultats numériques que notre algorithme est plus performant sur les problèmes où les variables sont bornées. L'explication est que les variables qui sont à leurs bornes supérieures sont traitées comme des variables nulles, ce qui augmente la dégénérescence et, par conséquent, la performance de notre algorithme.

Le troisième objectif a non seulement été atteint mais dépassé. Nous avons proposé et développé un nouveau critère d'évaluation des variables. Ce critère permet de construire le problème réduit et donc de manière générale, d'identifier si un pivot sera dégénéré. Notre critère que l'on nomme *positive edge* a une complexité de calcul identique à celle du calcul du coût réduit. Avec la règle du *positive edge*, il est possible de résoudre des problèmes de très grande taille ($m = 100\ 000$, $n = 450\ 000$) avec la méthode du problème réduit. Bien que notre algorithme pour la résolution de tels problèmes soit de base, nous obtenons un facteur de diminution du temps de résolution de plus de 4 par rapport à CPLEX. L'arrivée de ce nouveau critère dans le domaine de la programmation linéaire permettra de résoudre des problèmes de plus en plus gros. C'est une avancée importante pour la recherche opérationnelle.

Bien que nos objectifs aient été atteints, certaines questions de recherche s'ajoutent. En effet, il serait intéressant d'intégrer la règle du *positive edge* dans un logiciel qui utilise la méthode du simplexe telle CPLEX. Ainsi, à chaque pivot le critère du *positive edge* pourrait être calculé, ce qui améliorait la performance de l'algorithme. Enfin, la

résolution d'un problème complémentaire pourrait être ajoutée à notre algorithme pour résoudre les problèmes de très grande taille.

Bibliographie

BARAHONA, F. ET ANBIL, R. (2000). The volume algorithm : Producing primal solutions with a subgradient method. *Mathematical Programming*, 87, 385-399.

BLAND, R.G. (1977). New finite pivoting rules for the simplex method. *Mathematical of Operations Research*, 2, 103-107.

BOUBAKER, K., DESAULNIERS, G. ET ELHALLAOUI, I. (2008). Bidline scheduling with equity by heuristic dynamic constraint aggregation. *Les cahiers du GERAD*, Montreal, Canada. G-2008-43.

BROOKS, R. ET GEOFFRION, A.M. (1966). Finding Everett's multipliers by linear programming. *Operations Research*, 14(6), 1149-1153.

CHARNES, A. (1952). Optimality and degeneracy in linear programming. *Econometrica*, 20, 160-170.

CHVÁTAL, V. (1983). *Linear programming.* New-York : W.H. Freeman.

DANTZIG, G.B. (1949). Programming in a linear structure. *Econometrica*, 17, 73-74.

DANTZIG, G.B. ET WOLFE, P. (1960). Decomposition principle for linear programs. *Operations Research*, 9, 101-111.

DAVIS, T.A. ET DUFF, I.S. (1997). An unsymmetric-pattern multifrontal method for sparse LU factorization. *Journal on Matrix Analysis and Applications*, 18(01), 140-158.

DU MERLE, O., VILLENEUVE, D., DESROSIERS, J. ET HANSEN, P. (1999). Stabilized column generation. *Discrete Mathematics*, 194, 229-237.

ELHALLAOUI, I., DESAULNIERS, G., METRANE, A. ET SOUMIS, F. (2008a). Bi-dynamic constraint aggregation and subproblem reduction. *Computers and Operations Research*, doi : 10.1016/j.cor.2006.10.007.

ELHALLAOUI, I., METRANE, A., DESAULNIERS, G. ET SOUMIS, F. (2007). An improved primal simplex algorithm for degenerate linear programs. *Les cahiers du GERAD*, Montreal, Canada. G-2007-66.

ELHALLAOUI, I., METRANE, A., SOUMIS, F. ET DESAULNIERS, G. (2008b). Multi-phase dynamic constraint aggregation for set partitioning type problems. *Mathematical Programming*, doi : 10.1007/s10107-008-0254-5.

ELHALLAOUI, I., VILLENEUVE, D., SOUMIS, F. ET DESAULNIERS, G. (2005). Dynamic aggregation of set partitioning constraints in column generation. *Operations Research*, 53, 632-645.

FILIZ, B., CSIZMADIA, Z. ET ILLAS, T. (2001). Anstreicher-Terlaky type monotonic simplex algorithms for linear feasibility problems. *Operations Research*, 35, 286-303.

FISHER, M.L. (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27, 1-18.

FRELING, R., WALGELMANS, A.P.M. ET PAIXAO, J.M.P. (1999). An overview of models and techniques for integrating vehicle and crew scheduling. Dans N.H.M. Wilson (éd.), *Computer-Aided Transit Scheduling*, vol. 471, *Lectures Notes in Economics and Mathematical Systems* (pp. 441-460). Heidelberg : Springer-Verlag.

FUKUDA, K. (1982). *Oriented matroid programming.* Thèse de doctorat. University of Waterloo, Canada.

GAMACHE, M., SOUMIS, F., MARQUIS, G. ET DESROSIERS, J. (1998). A column generation approach for large scale aircrew rostering problems. *Operations Research*, 47, 247-263.

GASS, S.I. (1993). Encounters with degeneracy : A personal view. *Annals of Operations Research*, 47, 335-342.

GEOFFRION, A.M. (1968). *Relaxation and the dual method in mathematical programming.* Working paper 135, Western Management Science Institute, University of California at L.A.

GEOFFRION, A.M. (1974). Lagrangian relaxation for integer programming. *Mathematical Progamming Study*, 2, 82-114.

GEORGE, K. ET OSBORNE, M.R. (1993). On degeneracy in linear programming and related problems. *Annals of Operations Research*, 47, 345-359.

GILMORE, P.C. ET GOMORY, R.E. (1961). A linear programming approach to the cutting stock problem. *Operations Research*, 9, 849-859.

GILMORE, P.C. ET GOMORY, R.E. (1963). A linear programming approach to the cutting stock problem - Part II. *Operations Research*, 11, 863-888.

GLOVER, F. (1968). Surrogate constraints. *Operations Research*, 16, 741-749.

GOFFIN, J.-L. ET VIAL J.-P. (1990). Cutting plane and column generation techniques with the projective algorithm. *Journal of Optimization Theory and Applications*, 65, 409-429.

GOFFIN, J.-L. ET VIAL J.-P. (1999). Convex nondifferentiable optimization : A survey focussed on the analytic center cutting plane method. *Les cahiers du GERAD*, Montreal, Canada. G-99-17.

HAASE, K., DESAULNIERS, G ET DESROSIERS, J. (2001). Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35, 286-303.

HELD, M. ET KARP, R.M. (1971). The travelling salesman problem and minimum spanning trees : Part II. *Mathematical Programming*, 1, 6-25.

HIRIART-URRUTY, J. ET LEMARÉCHAL, C. (1991). *Convex Analysis and Minimization Algorithms II : Advanced Theory and Bundle Methods.* A Series of Comprehensive Studies in Mathematics. New-York : Springer.

KARWAN, M.L. ET RARDIN, R.L. (1979). Some relationships between Lagrangean and surrogate duality in integer programming. *Mathematical Programming*, 17, 320-334.

KELLEY, J.E. (1960). The cutting-plane method for solving convex programs. *Journal of SIAM*, 8, 703-712.

MENDELSSOHN, R. (1982). An iterative aggregation procedure for Markov decision process. *Operations Research*, 30, 62-73.

NASH, S.G. ET SOFER, A. J. (1996). *Linear and Non-Linear Programming.* New-York : McGraw Hill.

OMER, J. (2006). *Méthode de Réduction Dynamique de Contraintes pour un Programme Linéaire.* Thèse de maître, École Polytechnique de Montréal, Qc, Canada.

OVERTON, L.M. (2001). *Numerical Computing with IEEE Floating Point Arithmetic.* Philadelphia : SIAM.

PAN, P.-Q. (1998). A basis deficiency-allowing variation of the simplex method for linear programming. *Computers and Mathematics with Applications*, 36(3), 33-53.

RAYMOND, V., SOUMIS, F. ET METRANE, A. (2009a). Improved primal simplex method version 3 : Cold start, generalization for variable bounded problems, new implementation. *soumis à European Journal of Operational Research.*

RAYMOND, V., SOUMIS, F. ET ORBAN, D. (2009b). A new version of the improved primal simplex for degenerate linear programs. *Computer and Operational Research*, doi : 10.1016/j.cor.2009.03.020.

ROGERS, D.F., PLANTE, R.D., WONG, R.T. ET EVANS J.R. (1991). Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39, 553-582.

RYAN, D.M. ET OSBORNE, M. (1988). On the solution to 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 and Operations Research*, 41, 385-392.

TERLAKY, T. ET SUSHONG, Z. (1993). Pivot rules for linear programming : A survey on recent theoretical developments. *Annals of Operations Research*, 46, 203-233.

VILLENEUVE, D. (1999). *Logiciel de Génération de Colonnes.* Thèse de doctorat. École Polytechnique de Montréal, Qc, Canada.

WOLFE, P. (1963). A technique for resolving degeneracy in LP. *SIAM Journal*, 2, 205-211.