CHAPITRE 1

Programmation linéaire

1.1 Introduction

Bien qu'on puisse modéliser des problèmes d'optimisation et utiliser des logiciels sans connaître la théorie qui se cache derrière, quelques notions sont utiles pour démystifier le sujet. Ce chapitre présente donc des bases de programmation linéaire suffisantes pour les chapitres 5 à 14 consacrés à la modélisation. Le § 1.2 donne un exemple de programme linéaire puis définit les programmes linéaires en général, sous leurs deux principales formes. Les programmes linéaires à deux variables peuvent être résolus par une méthode géométrique, présentée au § 1.3, qui permet de saisir physiquement les principes.

Les problèmes à plus de deux variables sont résolus par une méthode algébrique célèbre, *l'algorithme du simplexe*, introduite au § 1.4. La forme tableau de l'algorithme, qui facilite les calculs, est présentée au § 1.5. Quelques situations particulières font l'objet du § 1.6. La dualité et l'analyse de sensibilité sont riches en propriétés qui dépassent le cadre de ce livre. Les paragraphes 1.7 et 1.8 consacrés à ces sujets ne présentent que des rudiments. Quelques évolutions récentes comme les méthodes de points intérieurs sont évoquées au § 1.9. La dernière partie est consacrée aux compléments et aux références pour approfondir le sujet.

1.2 Notion de programme linéaire

1.2.1 Composantes d'un programme linéaire et exemple

Soit une usine qui produit deux ciments rapportant 50 $ et 70 $ la tonne. Pour fabriquer une tonne de ciment 1, il faut 40 min de calcination dans un four et 20 min de broyage. Pour fabriquer une tonne de ciment 2, il faut 30 min de four et 30 min de broyage. Le four et l'atelier de broyage sont disponibles 6 h et 8 h par jour. Combien de ciment de chaque type peut-on produire par jour pour maximiser le bénéfice ? Ce problème se modélise par le *programme linéaire* (PL) suivant, en notant x_1 et x_2 les quantités de ciment à fabriquer.

(1) Max $z = 50x_1 + 70x_2$

(2) $40x_1 + 30x_2 \leq 360$

(3) $20x_1 + 30x_2 \leq 480$

(4) $x_1, x_2 \geq 0$

La ligne (1) représente le profit total z qui est le critère à optimiser, appelé aussi *fonction-objectif* (nom composé), *fonction de coût* ou *fonction économique*. *Max* signifie que ce critère doit être maximisé ; on écrirait *Min* pour minimiser. Les autres lignes désignent des *contraintes*. La contrainte (2) concerne la disponibilité du four : elle stipule que le temps total de calcination requis par les ciments ne doit pas dépasser 360 min ou 6 h. La contrainte (3) décrit de même la disponibilité du broyeur. Les contraintes (4) précisent les domaines des variables.

1.2.2 Forme générale d'un programme linéaire et extensions

Plus généralement, on appelle *programme mathématique* un problème d'optimisation d'une fonction-objectif de plusieurs variables en présence de contraintes. Le programme est dit *linéaire* si la fonction et les contraintes sont toutes des combinaisons linéaires de variables. Il a la forme générique suivante. Il comporte n variables non négatives (3), m contraintes d'égalité ou d'inégalité (2), et la fonction-objectif à optimiser (1). Le coefficient de coût ou de profit de la variable x_j est noté c_j, celui de la variable x_j dans la contrainte i est noté a_{ij}. La contrainte i a un second membre constant b_i. Les contraintes simples de positivité ne sont pas incluses dans les m contraintes, car elles sont gérées à part par les algorithmes.

(1) Max ou Min $z = \sum_{j=1}^{n} c_j x_j$

(2) $\forall i = 1 \ldots m : \sum_{j=1}^{n} a_{ij} x_j \leq, = \text{ ou } \geq b_i$

(3) $\forall j = 1 \ldots n : x_j \geq 0$

Des valeurs de variables qui vérifient toutes les contraintes, comme $x_1 = 4$ et $x_2 = 2$ dans l'exemple des ciments, forment une *solution réalisable* (SR) du PL. Une solution réalisable est *optimale* si aucune autre solution n'a un profit supérieur.

Si les variables sont astreintes à être entières, on obtient un *programme linéaire en nombres entiers* (PLNE). Un *programme linéaire en 0-1* est un cas particulier de PLNE dont les variables ne peuvent prendre que deux valeurs 0-1 ; ces variables sont dites *booléennes*, *binaires* ou *de décision*. Un *PL mixte* comprend à la fois des variables continues et des variables entières. Enfin, à partir du moment où au moins une contrainte ou la fonction-objectif n'est plus une combinaison linéaire de variables, on a affaire à un *programme non linéaire* (PNL). Les PLNE et PL en 0-1, qui font l'objet du chapitre 2, sont plus difficiles à résoudre que les PL ordinaires. Les PNL sont encore plus difficiles, sauf dans quelques cas particuliers comme des fonctions-objectifs convexes. Les algorithmes actuels ne trouvent en général qu'un optimum local et ils sortent du cadre de cet ouvrage.

1.2.3 Formes matricielles classiques et conversions

Notons $x = (x_1, x_2, \ldots, x_n)^T$ le vecteur des variables, $b = (b_1, b_2, \ldots, b_m)^T$ celui des seconds membres des contraintes, $c = (c_1, c_2, \ldots, c_n)$ les coûts ou profits associés aux variables, et A la matrice $m \times n$ des a_{ij}. Dans la suite, nous n'écrirons plus les signes de transposition pour alléger l'écriture. On peut alors écrire un PL sous forme matricielle. Deux formes sont courantes : la *forme canonique* avec des contraintes ≤, utilisée pour la résolution graphique, et la *forme standard* avec égalités, pour la résolution algébrique par des algorithmes. Par convention, la forme standard est souvent exprimée avec des seconds membres positifs.

Forme canonique	*Forme standard*
Max $c.x$	Max $c.x$
$A.x \leq b$	$A.x = b$
$x \geq 0$	$x \geq 0$

Ces formes ne servent qu'à simplifier les présentations théoriques. Dans la réalité, un PL peut présenter des égalités et des inégalités, et les logiciels du marché acceptent heureusement ces mélanges. On peut facilement convertir les formes mixtes en formes classiques. Ainsi, toute contrainte d'égalité peut être remplacée par deux inégalités.

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \Leftrightarrow \begin{cases} \sum_{j=1}^{n} a_{ij}x_j \leq b_i \\ -\sum_{j=1}^{n} a_{ij}x_j \leq -b_i \end{cases}$$

On peut convertir une inégalité en égalité en ajoutant ou soustrayant une *variable d'écart* $e_i \geq 0$, propre à chaque contrainte i. À l'optimum, pour une inégalité ≤ concernant la disponibilité d'une ressource i, cette variable indique la quantité inutilisée de la ressource.

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \text{ et } e_i \geq 0 \Leftrightarrow \sum_{j=1}^{n} a_{ij}x_j + e_i = b_i$$

D'autres conversions sont possibles. Ainsi, on peut passer d'une maximisation à une minimisation, car maximiser z revient à minimiser $-z$. Il ne faut pas oublier alors de multiplier par -1 la valeur de la fonction-objectif trouvée par la minimisation ! L'exigence de variables positives n'est pas restrictive, car une variable x_j non contrainte en signe peut toujours s'écrire comme une différence $x'_j - x''_j$ de deux variables non négatives.

1.2.4 Interprétation économique

Un PL a une *interprétation économique* très large. Soit un acteur économique qui exerce n activités avec des intensités x_j à déterminer. Ces activités utilisent m ressources. On connaît la quantité a_{ij} de ressource i nécessaire pour exercer l'activité j avec une intensité 1. On connaît aussi le profit ou le coût c_j pour une intensité 1 de l'activité j. On veut trouver les intensités des activités, compatibles avec les ressources, pour maximiser le profit ou minimiser le coût. Ce problème est modélisable par un PL sous forme canonique.

La programmation linéaire définit donc une classe très large de modèles, mais dans laquelle on prend deux hypothèses restrictives fondamentales : la *proportionnalité* des coûts et des consommations de ressources aux intensités d'activités, et l'*additivité* des consommations de ressources (pas d'interactions entre activités). Nous verrons en fait que de nombreux problèmes, en apparence non linéaires, peuvent être rendus linéaires.

Par exemple, l'hypothèse de proportionnalité n'est pas respectée quand on produit un bien en grande série. Le prix de vente par unité bénéficie souvent de tarifs dégressifs, grâce aux économies d'échelle. Le prix de vente en fonction de la quantité est alors une fonction linéaire par morceaux qui croît de moins en moins vite. Il n'empêche que ce genre de fonction se linéarise facilement, moyennant des variables supplémentaires. Ainsi, le champ d'application de la programmation linéaire est bien plus vaste qu'il n'y paraît.

1.3 Résolution graphique

Appelée aussi *résolution géométrique*, elle est possible pour un PL sous forme canonique avec deux ou trois variables ($n = 2$ ou $n = 3$). Soit par exemple le PL suivant, qu'on interprétera comme le calcul des quantités à produire x_1 et x_2 de deux produits pour maximiser un profit z.

$$
\begin{array}{lcccccc}
\text{Max } z = & x_1 & + & 2 \cdot x_2 & & \\
& x_1 & + & x_2 & \leq & 6 \\
& & & x_2 & \leq & 3 \\
& x_1, & & x_2 & \geq & 0
\end{array}
$$

Ce PL est bien sous forme canonique. En utilisant les notations matricielles, on peut écrire $m = 2$, $n = 2$, $c = (1, 2)$, $x = (x_1, x_2)^T$, $b = (6, 3)^T$ et :

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

Pour ce PL à deux variables, on trace dans le plan les axes des coordonnées pour les valeurs positives de x_1 et x_2. On finit de délimiter le *domaine des solutions réalisables*, qui forme un *polygone convexe*, en traçant les droites d'équations $x_1 + x_2 = 6$ et $x_2 = 3$ (figure 1.1 page suivante). Pour chaque droite, on indique par des flèches le demi-plan à conserver. On peut aussi hachurer les demi-plans interdits, mais la figure est souvent moins lisible.

Traçons la droite de profit $z = 0$. Son intersection avec le domaine des solutions réalisables est réduite au point O et correspond à la solution triviale où on ne produit rien, ce qui ne rapporte rien. Une droite d'équation $z = k$, où k est une constante entre 0 et 9 (exclus) donne par intersection avec le domaine tout un segment de plans de production possibles, ayant le même profit k. On peut augmenter k jusqu'à 9, ce qui donne la solution correspondant au point J : $x_1 = 3$, $x_2 = 3$. Cette solution est optimale, car si on augmente encore k, on sort du domaine. En remplaçant dans le PL les variables par leurs valeurs, on peut vérifier le coût et le respect des contraintes.

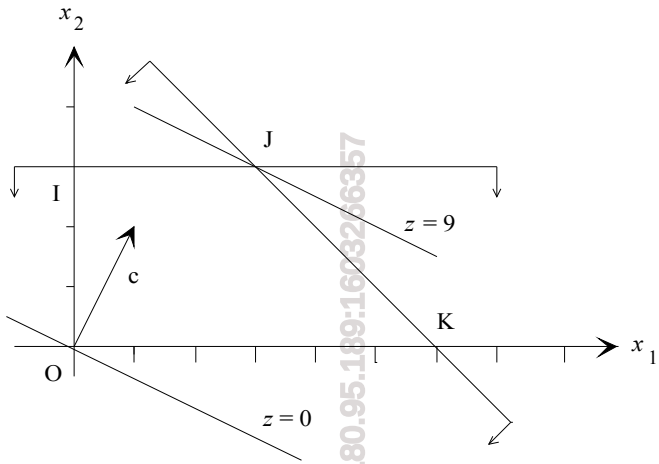

Figure 1.1 – Exemple de résolution graphique

On peut formuler sur la figure des remarques générales. Les contraintes de positivité confinent les solutions dans le quart de plan positif et chaque autre contrainte définit un demi-espace (ici un demi-plan). L'intersection de tous ces espaces forme un *polygone convexe* non vide, ici OIJK. La *convexité* signifie que pour deux points quelconques A et B du polygone, le segment [A, B] est contenu dans le polygone. Pour un nombre quelconque *n* de variables, on parle de *polyèdre convexe*. Pour *k* croissant, les droites d'équation $z = k$ forment une famille de droites parallèles. *z* augmente dans la direction du vecteur *c* de la fonction-objectif. L'optimum x^*, ici unique, est atteint au sommet J = (3, 3) du polyèdre. Le coût optimal correspondant est $z^* = 9$.

Les cas spéciaux suivants pourraient se produire. En supprimant $x_1 + x_2 \leq 6$, le domaine des solutions serait *non borné*, ainsi que l'optimum. L'optimum peut cependant être fini même si le domaine est non borné : ce serait le cas pour $c = (-1, 2)$, qui donnerait le point I. Si on ajoutait la contrainte $x_1 - x_2 \leq -4$, le polyèdre serait *vide* et il n'y aurait aucune solution réalisable. Enfin, si on maximisait $z = x_1 + x_2$, il y aurait *plusieurs optima* (tous les points de l'arête JK). Remarquons cependant que l'ensemble des points optimaux, s'il est non vide, comprend toujours un sommet du polyèdre.

Pour un problème réel comme un plan de production, le domaine n'est normalement pas vide car il y a au moins une solution : le plan de production *actuel* de l'entreprise. De plus, l'optimum est borné en pratique à cause de limites sur les ressources. Dans un problème réel, l'absence de solution indique en général un problème surcontraint, tandis que l'oubli d'une contrainte peut donner un optimum non borné. Une erreur de saisie du modèle dans un logiciel, comme une inversion de signe, peut également produire ces phénomènes. Dans tous ces cas anormaux, il faut revoir soigneusement la formulation.

Si on est bon dessinateur, la résolution graphique est encore possible pour trois variables : les contraintes définissent des demi-espaces dont l'intersection forme un polyèdre à trois dimensions (une sorte de cristal), et les solutions de profit *z* constant forment une famille de plans parallèles. Il est clair qu'il faut utiliser une autre méthode en présence de plus de trois variables. C'est le but de la *résolution algébrique* des sections suivantes.

1.4 Principes de la résolution algébrique

1.4.1 Bases et solutions de base

Cette résolution utilise la forme standard. Rappelons que le système d'égalités linéaires d'un PL en forme standard s'écrit matriciellement $Ax = b$, avec A une matrice $m \times n$ telle que $m \leq n$, x un vecteur $n \times 1$, et b un vecteur $m \times 1$. On suppose aussi qu'il n'y a pas de contrainte redondante, c'est-à-dire combinaison linéaire d'autres contraintes : le *rang* de A vaut m, en termes mathématiques. Heureusement, les logiciels du commerce fonctionnent quand même correctement si ces hypothèses ne sont pas vérifiées !

On appelle *base* de A, ou *matrice de base*, toute sous-matrice carrée inversible B, $m \times m$, de A. Pour une base B, on peut (en réarrangeant les colonnes si nécessaire) partitionner A en (B, N) et x en (x_B, x_N). N est la matrice $m \times (n\text{-}m)$ des colonnes hors base, ou *matrice hors base*. Le vecteur x_B a pour composantes les *m variables de base* (associées aux colonnes de B). x_N a pour composantes les *n-m variables hors base*. Le système de contraintes et la fonction-objectif peuvent alors s'écrire de manière équivalente :

$$A \cdot x = b \Leftrightarrow B \cdot x_B + N \cdot x_N = b \Leftrightarrow x_B = B^{-1} \cdot b - B^{-1} \cdot N \cdot x_N$$
$$z = c \cdot x = c_B \cdot x_B + c_N \cdot x_N = c_B \cdot B^{-1} \cdot b + \left(c_N - c_B \cdot B^{-1} \cdot N\right) \cdot x_N$$

Pour une base choisie B, le système équivalent est simplement une expression des variables de base en fonction des variables hors base. On a une solution évidente en forçant x_N à 0 : on a alors $x_B = B^{-1}.b$. Cette solution du PL est la *solution de base* (SB) associée à la base B. Elle peut violer des contraintes de positivité : on appelle *solution de base réalisable* (SBR) une solution réalisable dont toutes les variables sont positives ou nulles.

1.4.2 Exemple d'énumération des bases et changements de base

Reprenons le PL traité par la résolution géométrique, mais mis sous forme standard en ajoutant une *variable d'écart* x_3 pour la première contrainte, et une autre x_4 pour la seconde.

$$\begin{array}{lrcrcrcrl}
\text{Max } z = & x_1 & + & 2 \cdot x_2 & & & & & \\
 & x_1 & + & x_2 & + & x_3 & & = & 6 \\
 & & & x_2 & & & + & x_4 & = 3 \\
 & x_1, & & x_2, & & x_3, & & x_4 & \geq 0
\end{array}$$

En notant a_i la colonne i de A, la matrice A (2×4) de ce programme linéaire est :

$$(a_1, a_2, a_3, a_4) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Les bases de A sont ses sous-matrices carrées inversibles 2×2, en fait toutes sauf (a_1, a_3). On trouve ainsi les cinq matrices de base de la page suivante, pour lesquelles on a calculé les solutions de base. Toutes ces solutions de base sont réalisables, sauf celle correspondant à B_4.

$$B_1 = (a_1, a_2) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

$$B_2 = (a_1, a_4) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_1 \\ x_4 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \end{pmatrix}$$

$$B_3 = (a_2, a_3) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

$$B_4 = (a_2, a_4) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_2 \\ x_4 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ -3 \end{pmatrix}$$

$$B_5 = (a_3, a_4) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \Rightarrow x_B = \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} = B^{-1}b = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \end{pmatrix}$$

Dans cet exemple, on ne peut pas visualiser le polyèdre en quatre dimensions. Mais le polyèdre utilisé pour la résolution graphique en est une projection à deux dimensions, sans les variables d'écart (figure 1.1). *On constate alors que les SBR correspondent aux sommets J, K, I et O du polyèdre*. La base B_5 est la matrice identité associée aux variables d'écart. La SBR associée correspond au point O de la résolution graphique. Dans un contexte de production, elle consiste à ne rien produire : on met à zéro les variables du PL canonique d'origine, et les variables d'écart sont égales aux quantités de ressources disponibles (non consommées), définies par le vecteur *b*.

Les opérations précédentes ont l'air compliquées sous forme matricielle. Elles sont en fait simples si on détaille les équations en exprimant les variables de base en fonction des variables hors base. Par exemple, pour retrouver la solution de base associée à B_1 (point J de la résolution graphique), on part du système $Ax = b$ de la forme standard.

$$x_1 + x_2 + x_3 = 6$$
$$x_2 + x_4 = 3$$

Il faut transformer ce système pour exprimer les variables de base x_1 et x_2 en fonction des autres variables (hors base) x_3 et x_4. Il se trouve que x_2 est déjà exprimée en fonction de x_4 dans la ligne 2. En substituant son expression ($x_2 = 3 - x_4$) dans la première équation, on obtient le système suivant, traduction non matricielle du système $x_B = B^{-1}.b - B^{-1}.N.x_N$. En fixant les variables hors base x_3 et x_4 à 0, on retrouve effectivement la solution réalisable de base $x_1 = 3$ et $x_2 = 3$ correspondant au point J.

$$x_1 = 3 - x_3 + x_4$$
$$x_2 = 3 - x_4$$

Les bases pour deux sommets adjacents du polyèdre ne diffèrent que par une variable. Si on dispose du système exprimant les variables de base en fonction de celles hors base, on peut déduire le système pour l'autre base, sans inversion de matrice. Par exemple, supposons que l'on parte de la base B_5 des variables d'écart x_3 et x_4, correspondant au point O.

$$x_3 = 6 - x_1 - x_2$$
$$x_4 = 3 - x_2$$

Pour passer à la base B_3 formée des variables x_2 et x_3 et correspondant au point I, x_2 remplace x_4 dans la base. Ceci s'effectue dans la ligne définissant x_4 en déplaçant x_2 dans le membre de gauche et x_4 dans le membre de droite. Il faut ensuite éliminer x_2 dans les membres de droite des autres équations, réservées aux variables hors base. Ce qui s'effectue simplement en remplaçant x_2 par son expression, 3 - x_4. Cette transformation du système est appelée *pivotage* de x_4 vers x_2. On obtient ainsi le système suivant :

$$x_3 = 3 - x_1 + x_4$$
$$x_2 = 3 - x_4$$

1.4.3 Propriétés fondamentales de la programmation linéaire

Les méthodes de résolution algébriques reposent sur deux propriétés fondamentales à admettre, mais vues intuitivement dans les paragraphes précédents. Les précautions de style sont nécessaires à cause des polyèdres non bornés ou vides :

- Si une fonction linéaire atteint son maximum (ou son minimum) sur le polyèdre X défini par les contraintes, alors cet optimum a lieu en un sommet de X.
- Si X est non vide, alors il existe au moins un sommet, et l'ensemble des sommets correspond aux solutions de base réalisables.

Ainsi, bien qu'un polyèdre non vide contienne une infinité de solutions réalisables, il suffit de consulter ses sommets, en nombre fini, pour trouver l'optimum. Le nombre de sommets peut être énorme : pour 100 contraintes et 400 variables, on peut avoir autant de matrices de base que de façons de choisir 100 colonnes parmi 400, soit de l'ordre de 10^{100} sommets ! L'algorithme du simplexe va seulement générer une suite de sommets de profits croissants.

1.4.4 Algorithme du simplexe à la main

Reprenons notre PL favori en forme standard, avec ses deux contraintes d'égalité :

$$\begin{array}{lrcrcrcrl} \text{Max } z = & x_1 & + & 2 \cdot x_2 & & & & & \\ & x_1 & + & x_2 & + & x_3 & & = & 6 \\ & & & x_2 & & & + & x_4 & = 3 \\ & x_1, & & x_2, & & x_3, & & x_4 & \geq 0 \end{array}$$

L'algorithme du simplexe a été publié par Dantzig en 1949 [Dantzig 1949]. Il construit une suite de solutions de base réalisables de profit croissant, jusqu'à ce qu'il n'y ait plus de gain possible. Géométriquement, il visite une suite de sommets adjacents du polyèdre. Le passage d'une base à l'autre s'effectue par des opérations de pivotage (*cf.* § 1.4.2). On part de la base évidente (matrice identité) formée par les variables d'écart x_3 et x_4. La réécriture des contraintes pour obtenir les expressions de x_3 et x_4 en fonction des variables hors base x_1 et x_2 est immédiate. La fonction-objectif est ajustée de la même manière, mais au début il n'y a rien à faire puisque les variables d'écart y ont un coefficient nul. On obtient :

$$x_3 = 6 - x_1 - x_2$$
$$x_4 = 3 - x_2$$
$$z = 0 + x_1 + 2x_2$$

La SBR initiale associée s'obtient en mettant les variables hors base des seconds membres à 0. On trouve $x_1 = 0$, $x_2 = 0$, $x_3 = 6$, $x_4 = 3$ et $z = 0$. Elle correspond au point O de la résolution géométrique. Pour changer de SBR, on imagine que les variables hors base sont nulles dans le système précédent. Une variable hors base, actuellement à 0, va être choisie pour entrer en base et augmenter jusqu'à annuler une variable de base. À la suite de cette opération de pivotage, la variable hors base qu'on augmente, dite *entrante*, remplace celle qui s'annule dans la solution de base, dite *sortante*. Géométriquement, on passe sur un sommet adjacent du polyèdre.

Pour déterminer la variable entrante, on examine les coefficients des variables hors base dans z, appelés *profits marginaux* ou *réduits* (*coûts marginaux* en minimisation). Ils donnent le gain obtenu en augmentant de 1 la variable associée. En fait, l'algorithme converge si on augmente une des variables de profit marginal strictement positif, mais plus rapidement en moyenne si on choisit parmi elles celle *de profit marginal maximal*, ici x_2.

Voyons maintenant comment trouver la variable sortante. D'après la première contrainte, x_2 ne peut pas augmenter au-delà de 6, sinon x_3 deviendrait négative. La seconde contrainte est encore plus limitante, car x_2 peut augmenter jusqu'à 3 seulement. Ainsi, x_4 s'annule. *La variable sortante est donc la première qui s'annule quand on augmente la variable entrante*. Les nouvelles variables de base sont donc x_3 et x_2, et on doit les écrire, ainsi que z, uniquement en fonction des nouvelles variables hors base x_1 et x_4. On obtient :

$$x_2 = 3 - x_4$$
$$x_3 = 6 - x_1 - x_2 = 6 - x_1 - (3 - x_4) = 3 - x_1 + x_4$$
$$z = 0 + x_1 + 2(3 - x_4) = 6 + x_1 - 2x_4$$

La SBR actuelle se lit en plaçant à 0 les variables hors base : $x_2 = 3$, $x_3 = 3$, $x_1 = x_2 = 0$, avec un profit $z = 6$. Elle correspond au point I du polyèdre. Pour augmenter encore le profit, on peut seulement augmenter x_1, car c'est la seule variable hors base de profit unitaire strictement positif. x_1 peut augmenter jusqu'à 3 et remplace dans la base x_3, qui s'annule. On écrit donc x_1, x_2 et z en fonction de x_3 et x_4 :

$$x_1 = 3 - x_3 + x_4$$
$$x_2 = 3 - x_4$$
$$z = 6 + (3 - x_3 + x_4) - 2x_4 = 9 - x_3 - x_4$$

À ce stade la SBR associée est $x = (3,3,0,0)$, de profit 9. Elle correspond au sommet J du polyèdre. On est à l'optimum, car toutes les variables hors base ont un profit unitaire négatif : le profit diminuerait si on augmentait l'une d'entre elles. Finalement, sur les cinq bases possibles pour ce PL, l'algorithme du simplexe n'en a consulté que trois. Pour un PL volumineux, l'économie par rapport à une énumération complète des solutions de base est énorme : des tests numériques montrent qu'en moyenne le nombre de pivotages reste proportionnel au nombre m de contraintes.

Le processus est facilement adaptable au cas d'une *minimisation*. La seule différence est la règle pour la variable entrante : il faut prendre celle ayant le plus petit coût marginal strictement négatif (le plus grand en valeur absolue), le but étant d'obtenir un coût minimal.

1.5 Algorithme du simplexe forme tableau

Dans le § 1.4.4, nous avons manipulé directement le PL en opérant par calcul matriciel sans nous en apercevoir. En effet, à toute itération et pour une base B, le PL était écrit sous la forme équivalente définie à la fin du § 1.4.1 :

$$x_B = B^{-1} \cdot b - B^{-1} \cdot N \cdot x_N = b' - B^{-1} \cdot N \cdot x_N$$
$$z = c_B \cdot B^{-1} \cdot b + \left(c_N - c_B \cdot B^{-1} \cdot N\right) \cdot x_N = z_B + \Delta_N \cdot x_N$$

Pour les contraintes, on reconnaît chaque variable de base $x_B(i)$, égale à un terme constant b'_i (sa valeur actuelle) plus une combinaison linéaire des variables hors base. L'expression de z comporte un terme constant noté z_B (valeur actuelle de la fonction-objectif) plus une combinaison linéaire dans laquelle Δ_N désigne le vecteur des profits marginaux.

La *forme tableau* de l'algorithme du simplexe facilite les calculs précédents et se prête bien à la programmation. Reprenons le programme linéaire vu pour la résolution géométrique, une fois mis sous forme standard.

$$\begin{array}{lrcrcrcl} \text{Max } z = & x_1 & + & 2 \cdot x_2 & & & & \\ & x_1 & + & x_2 & + & x_3 & = & 6 \\ & & & x_2 & & + & x_4 & = & 3 \\ & x_1, & & x_2, & & x_3, & x_4 & \geq & 0 \end{array}$$

Le tableau initial de ce PL est composé de quatre sous-tableaux. Le tableau central T est chargé avec la matrice A du PL. Le tableau-colonne *Base* donne les indices des variables de base actuelles, les variables d'écart x_3 et x_4. Le tableau-colonne b' contient les seconds membres. La ligne Δ est la fonction-objectif sous la forme $0.x_B + \Delta_N.x_N = -z_B$ (les variables de base ont un profit marginal nul). Notez que la case $-z_B$ contient bien la valeur de la fonction-objectif, mais multipliée par -1. Actuellement cette valeur est nulle.

Base	T	1	2	3	4	b'	$b'_i / T_{ie} > 0$
3	1	1	1	1	0	6	6
4	2	0	[1]	0	1	3	$3 \rightarrow s$
	Δ	1	2	0	0	0	
			↑			$-z_B$	
			e				

Effectuons la première itération, tout à fait équivalente à ce que nous avons réalisé en manipulant directement les équations. Elle construit le tableau de la prochaine base.

- La variable hors base x_e entrant en base est x_2, car elle a le coût réduit strictement positif le plus grand sur la ligne Δ. On repère sa colonne (*colonne pivot*) avec un *e*.
- La variable sortant de la base est celle s'annulant en premier quand x_e augmente. C'est celle qui minimise les b'_i / T_{ie} avec $T_{ie} > 0$ ou, comme on suppose des seconds membres positifs dans la forme standard, les $b'_i / T_{ie} > 0$. Ceci se produit à la ligne $s = 2$, dite *ligne pivot*. *Base*[*s*] donne l'indice de la variable de base correspondante, x_4.
- On entoure le *pivot* T_{se}. Pour exprimer x_2 en fonction des variables hors base, il faut écrire dans le tableau suivant la ligne du pivot, divisée par le pivot, pour obtenir un 1 à la place du pivot. Le pivot valant déjà 1, on se contente de recopier la ligne du pivot.
- On fait apparaître ensuite des 0 dans les T_{ie} des lignes $i \neq s$: on multiplie la ligne du pivot (déjà calculée dans le tableau suivant) par T_{ie} puis on la soustrait à la ligne *i*. Les lignes obtenues sont copiées dans le tableau suivant. Ceci élimine x_e des équations autres que la ligne *s*. Ici, le traitement équivaut à soustraire la ligne 2 à la ligne 1.
- On applique la même opération à la ligne Δ, y compris pour $-z_B$: la ligne du pivot est multipliée par Δ_e et soustraite à la ligne Δ pour obtenir la nouvelle ligne Δ du tableau suivant. Ici, ceci revient à multiplier la ligne 2 par 2 et à la soustraire à la ligne Δ.
- On n'oublie pas de mettre à jour l'indice de la variable de base correspondant désormais à la ligne *s* dans *Base* : 2 au lieu de 3. On obtient le tableau suivant :

Base	*T*	1	2	3	4	*b'*	$b'_i / T_{ie} > 0$
3	1	[1]	0	1	-1	3	$3 \rightarrow s$
2	2	0	1	0	1	3	–
	Δ	1	0	0	-2	-6	
		↑ *e*				$-z_B$	

Ce tableau se lit comme suit :

$$x_1 + x_3 - x_4 = 3$$
$$x_2 + x_4 = 3$$
$$x_1 - 2x_4 = -6$$

On retrouve le système obtenu par l'algorithme du § 1.4.4, sauf que les variables de base ne sont plus dans les membres de gauche : c'est le tableau *Base* qui donne leurs colonnes. La solution de base actuelle est donc $x_B = (x_2, x_3) = (3, 3)$, $x_N = (x_1, x_4) = (0, 0)$. La matrice identité des variables de base s'est déplacée dans *T* aux colonnes 3 et 2. Le profit actuel est $z_B = 6$. Le vecteur des profits unitaires des variables hors base est $\Delta_N = (1, -2)$.

L'algorithme n'est pas terminé car on peut encore augmenter le profit en augmentant x_1. On n'a pas le choix car c'est la seule variable hors base de profit unitaire > 0. Le minimum des $b'i / T_{ie} > 0$ est obtenu sur la ligne 1. Le tiret sur la ligne 2 signifie que x_1 peut augmenter autant qu'on veut sans affecter la variable de base x_2. On trouve $e = 1$, $s = 1$, *Base*[*s*] = 1 (x_1 entre en base et x_3 en sort). Le pivotage sur $T_{se} = T_{11}$ revient à soustraire la ligne du pivot à la ligne Δ pour faire apparaître un 0 dans Δ(1). On obtient le tableau suivant.

Base	T	1	2	3	4	b'
1	1	1	0	1	-1	3
2	2	0	1	0	1	3
	Δ	0	0	-1	-1	-9

$-z_B$

C'est la fin de l'algorithme car les profits marginaux sont négatifs ou nuls. On retrouve évidemment la solution optimale de la méthode géométrique et de l'algorithme non matriciel : $x_1^* = 3$, $x_2^* = 3$, les autres variables à 0, et un coût total $z^* = 9$.

1.6 Cas spéciaux pour l'algorithme du simplexe

1.6.1 Optimum non borné

Considérons le programme linéaire suivant sous forme canonique :

$$\begin{aligned} \text{Max } z = \quad & x_1 + 2\cdot x_2 \\ -2\cdot x_1 + \quad & x_2 \le 2 \\ -x_1 + 2\cdot x_2 \le & \ 5 \\ x_1 - 4\cdot x_2 \le & \ 4 \\ x_1, \quad x_2 \ge & \ 0 \end{aligned}$$

La résolution géométrique montre qu'il n'a pas d'optimum borné (figure 1.2).

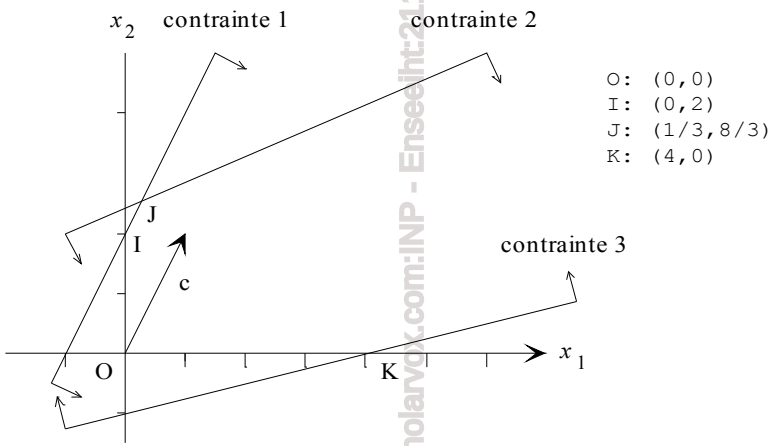

Figure 1.2 – Résolution géométrique d'un cas sans optimum borné

Résolvons-le maintenant par l'algorithme du simplexe. On passe à la forme standard en introduisant trois variables d'écart x_3, x_4, et x_5. Le premier tableau correspond au sommet O du polyèdre de la forme canonique.

Base	T	1	2	3	4	5	b'	$b'_i / T_{ie} > 0$
3	1	-2	[1]	1	0	0	2	$2 \rightarrow s$
4	2	-1	2	0	1	0	5	2.5
5	3	1	-4	0	0	1	4	–
	Δ	1	2	0	0	0	0	
			↑				$-z_B$	
			e					

Le deuxième tableau correspond au point I :

Base	T	1	2	3	4	5	b'	$b'_i / T_{ie} > 0$
2	1	-2	1	1	0	0	2	–
4	2	[3]	0	-2	1	0	1	$1/3 \rightarrow s$
5	3	-7	0	4	0	1	12	–
	Δ	5	0	-2	0	0	-4	
		↑					$-z_B$	
		e						

Le troisième et dernier tableau correspond au point J. On peut ensuite augmenter le coût en faisant entrer x_3 en base. *L'optimum n'est pas borné*, car on peut augmenter x_3 sans annuler une variable hors base : il n'y a pas de $b'_i / T_{ie} > 0$. Cette situation est parfaitement détectée par les logiciels commerciaux, avec un message du genre *Unbounded optimum*.

Base	T	1	2	3	4	5	b'	$b'_i / T_{ie} > 0$
2	1	0	1	-1/3	2/3	0	8/3	–
1	2	1	0	-2/3	1/3	0	1/3	–
5	3	0	0	-2/3	7/3	1	43/3	–
	Δ	0	0	4/3	-5/3	0	-17/3	$-z_B$

1.6.2 Absence de base initiale évidente

Le problème

Soit le programme linéaire suivant sous forme canonique :

$$\begin{aligned} \text{Max } z = \; & x_1 + 2 \cdot x_2 \\ & x_1 + x_2 \leq 6 \\ & x_2 \leq 3 \\ & x_1 + x_2 \geq 1 \\ & x_1 \;,\; x_2 \geq 0 \end{aligned}$$

Pour la forme standard, il faut *soustraire* une variable d'écart à la troisième contrainte. En effet, une variable d'écart doit être non négative, comme toute autre variable :

$$
\begin{array}{lrcrcrcrcrcl}
\text{Max } z = & x_1 & + & 2 \cdot x_2 & & & & & & & & \\
 & x_1 & + & x_2 & + & x_3 & & & & & = & 6 \\
 & & & x_2 & & & + & x_4 & & & = & 3 \\
 & x_1 & + & x_2 & & & & & - & x_5 & = & 1 \\
 & x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 & \geq & 0
\end{array}
$$

On n'a plus la matrice identité habituelle, qui nous fournissait une base initiale évidente. Ceci est confirmé par l'interprétation géométrique (forme canonique) : l'optimum est atteint en J = (3,3), mais l'origine O n'est plus réalisable (figure 1.3).

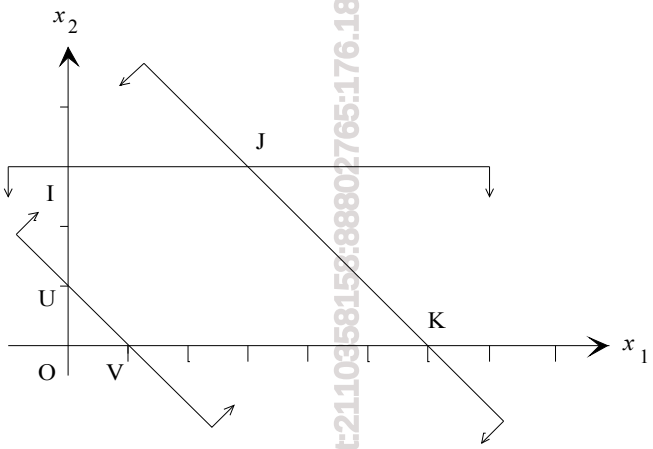

Figure 1.3 – Un cas où le point O n'est pas solution de base

Pour démarrer l'algorithme du simplexe, on pourrait trouver par tâtonnement une sous-matrice *B* inversible, mais l'effort de calcul peut être énorme pour un grand PL. Les logiciels utilisent en pratique deux méthodes basées sur l'emploi de variables artificielles : la méthode des deux phases et la méthode du grand *M*. L'idée est la suivante : on fait apparaître une matrice identité en ajoutant aux contraintes qui en ont besoin une *variable artificielle* (VA), avec un coefficient 1.

Les variables x_1 à x_5 sont dites *légitimes* : elles sont nécessaires à l'obtention de la forme standard. Les variables d'écart x_3 et x_4 nous donnent deux colonnes de matrice identité, les contraintes correspondantes (première et deuxième) n'ont donc pas besoin de VA. On ajoute x_6 à la troisième contrainte pour compléter la base. Une telle variable est réellement artificielle et n'a aucun sens économique : à l'optimum, elle doit être nulle (hors base) sinon la contrainte n'est pas vérifiée avec égalité ! Les deux méthodes des 2 phases et du grand M ont précisément pour but de s'en débarrasser.

$$
\begin{array}{lrcrcrcrcrcl}
\text{Max } z = & x_1 & + & 2 \cdot x_2 & & & & & & & & \\
& x_1 & + & x_2 & + & x_3 & & & & & = & 6 \\
& & & x_2 & & & + & x_4 & & & = & 3 \\
& x_1 & + & x_2 & & & & & - & x_5 + x_6 & = & 1 \\
& x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 , x_6 & \geq & 0
\end{array}
$$

Méthode des deux phases

Dans la *phase 1*, on ignore la vraie fonction-objectif et on cherche à minimiser la somme des p variables artificielles qu'on a dû introduire. On résout donc le PL auxiliaire suivant, où x_A désigne le vecteur des p VA, et e un vecteur avec p composantes à 1. Notez qu'on est en minimisation, le choix de la variable entrante dans l'algorithme du simplexe est modifié : c'est celle de plus petit coût marginal négatif.

$$\text{Min } z = e \cdot x_A = \sum_{j=1}^{p} x_A(j)$$

$$A \cdot x + x_A = b$$

$$x, x_A \geq 0$$

Si tout se passe bien, on parvient à annuler la somme des VA et à trouver une base sans VA. On obtient donc une solution réalisable pour le problème de départ. La *phase 2* consiste à éliminer du tableau les colonnes des VA, devenues inutiles, à remplacer la ligne Δ par la vraie fonction-objectif, et à continuer le simplexe sur le tableau obtenu. C'est en phase 2 qu'on peut détecter si le PL d'origine n'a pas d'optimum borné. En réalité, d'autres cas spéciaux peuvent se produire en fin de phase 1 :

- Si $x_A \neq 0$ (fonction-objectif non nulle), le PL d'origine n'a pas de solution réalisable.
- Si $x_A = 0$ avec des VA dans la base, on peut montrer que les contraintes associées à ces variables sont redondantes. On peut les éliminer et attaquer la phase 2.

À notre connaissance, cette technique de variables artificielles est la seule qui permette de détecter assez simplement les PL sans solution et les contraintes redondantes. Elle peut aussi servir à savoir si un système d'inégalités linéaires a ou non des solutions, en lui ajoutant des variables d'écart et des variables artificielles comme pour un programme linéaire.

Appliquons la méthode des deux phases à l'exemple. La base est formée des colonnes 3, 4 et 6. Mais dans l'algorithme du simplexe, les variables de base doivent avoir des coûts réduits nuls, ce qui n'est pas le cas pour x_6. Transformons Δ en une ligne correcte Δ', en lui soustrayant la ligne 3 du tableau. Ceci fait apparaître le véritable coût de la solution de base actuelle : 1, c'est-à-dire qu'elle contient une VA. Étant en minimisation, il faut faire entrer en base la variable de coût réduit négatif minimal. On a le choix entre $e = 1$ et $e = 2$. Par convention, nous prenons le plus petit indice.

Base	T	1	2	3	4	5	6	b'	$b'_i / T_{ie} > 0$
3	1	1	1	1	0	0	0	6	6
4	2	0	1	0	1	0	0	3	–
6	3	[1]	1	0	0	-1	1	1	$1 \rightarrow s$
	Δ	0	0	0	0	0	1	0	
	Δ'	-1	-1	0	0	1	0	-1	
		↑						$-z_B$	
		e							

Après pivotage sur T_{31}, on obtient le tableau suivant :

Base	T	1	2	3	4	5	6	b'	
3	1	0	0	1	0	1	-1	5	
4	2	0	1	0	1	0	0	3	
1	3	1	1	0	0	-1	1	1	
	Δ'	0	0	0	0	0	1	0	$-z_B$

La phase 1 se termine ici avec succès, puisque tous les coûts marginaux sont positifs ou nuls (on est en minimisation). La solution réalisable trouvée correspond au point V du polyèdre de la forme canonique.

Base	T	1	2	3	4	5	b'	$b'_i / T_{ie} > 0$
3	1	0	0	1	0	[1]	5	$5 \rightarrow s$
4	2	0	1	0	1	0	3	–
1	3	1	1	0	0	-1	1	–
	Δ'	1	2	0	0	0	0	
	Δ''	0	1	0	0	1	-1	
						↑	$-z_B$	
						e		

On peut mener la phase 2 car la VA x_6 a été éjectée de la base. Pour débuter, les colonnes des VA (ici une seule) sont supprimées du tableau de fin de phase 1, ce qui donne le tableau ci-dessus. La fonction-objectif d'origine $x_1 + 2.x_2$ est réintroduite. Elle doit être exprimée en fonction des variables hors base, comme les autres lignes. Pour cela, on soustrait la ligne 3 à la ligne Δ', ce qui donne une ligne correcte Δ''.

Après pivotage sur T_{15} on parvient au point K du polyèdre de la forme canonique (tableau suivant). Notez qu'on a un peu "triché" en faisant entrer en base x_5 au lieu de x_2, pour gagner une itération : on a ainsi emprunté la suite de sommets V, K, J au lieu de V, U, I, J !

Base	T	1	2	3	4	5	b'	$b'_i / T_{ie} > 0$
5	1	0	0	1	0	1	5	–
4	2	0	[1]	0	1	0	3	$3 \rightarrow s$
1	3	1	1	1	0	0	6	6
	Δ''	0	1	-1	0	0	-6	
			↑				$-z_B$	
			e					

Un dernier pivotage nous amène à l'optimum, au point J = (x_1, x_2) = (3,3), de coût 9.

Base	T	1	2	3	4	5	b'	
5	1	0	0	1	0	1	5	
2	2	0	1	0	1	0	3	
1	3	1	0	1	-1	0	3	
	Δ''	0	0	-1	-1	0	-9	$-z_B$

La méthode du grand M

Cette méthode procède en une phase. Elle ajoute les VA à la fonction-objectif du PL d'origine, mais pénalisées par un coût -*M*, *M* étant un grand nombre positif (10^6 par exemple). Avec les mêmes notations que pour les deux phases, on résout en fait le PL :

$$\text{Max } z = c \cdot x - M \cdot e \cdot x_A = \sum_{j=1}^{n} c_j x_j - M \sum_{j=1}^{p} x_A(j)$$

$$A \cdot x + x_A = b$$

$$x, x_A \geq 0$$

Si tout se passe bien, le simplexe se débarrasse des VA dès les premières itérations et ne les fait plus entrer en base à cause de leur coût énorme. On peut ignorer les colonnes des VA dès qu'elles ne sont plus en base. Les cas spéciaux suivants peuvent aussi se produire :

- Le PL modifié n'a pas d'optimum borné et $x_A = 0$: le PL d'origine est aussi non borné.
- Le PL modifié n'a pas d'optimum borné et $x_A \neq 0$: le PL d'origine n'a pas de solution.
- Le PL modifié a un optimum borné, mais $x_A \neq 0$: le PL d'origine n'a pas de solution.

En général, la méthode du grand *M* donne moins d'itérations que celle des deux phases. En revanche, sur ordinateur, la coexistence de *M* avec des petits nombres engendre souvent des problèmes de précision, ce qui fait que les logiciels commerciaux utilisent plutôt la méthode des deux phases.

Voyons la méthode sur l'exemple. Les tableaux *T*, *Base* et *b*' sont identiques à ceux du début de la phase 1 dans la méthode des deux phases. La ligne Δ est chargée avec les coûts des variables légitimes et un coût -*M* pour chaque VA. On rend nuls les coûts marginaux des variables de base en ajoutant la ligne 3, multipliée par *M*, à la ligne Δ, ce qui donne la ligne Δ'.

Base	T	1	2	3	4	5	6	b'	$b'_i / T_{ie} > 0$
3	1	1	1	1	0	0	0	6	6
4	2	0	1	0	1	0	0	3	3
6	3	1	[1]	0	0	-1	1	1	$1 \rightarrow s$
	Δ	1	2	0	0	0	$-M$	0	
	Δ'	M+1	M+2	0	0	$-M$	0	M	
			↑ e					$-z_B$	

Une itération mène à une solution réalisable (point U du polyèdre de la forme canonique).

Base	T	1	2	3	4	5	6	b'	$b'_i / T_{ie} > 0$
3	1	0	0	1	0	1	-1	5	5
4	2	-1	0	0	1	[1]	-1	2	$2 \rightarrow s$
2	3	1	1	0	0	-1	1	1	–
	Δ'	-1	0	0	0	2	$-M$-2	-2	
						↑ e		$-z_B$	

L'algorithme du simplexe se poursuit en ignorant la colonne 6 :

Base	T	1	2	3	4	5	b'	$b'_i / T_{ie} > 0$
3	1	[1]	0	1	-1	0	3	$3 \rightarrow s$
5	2	-1	0	0	1	1	2	–
2	3	0	1	0	1	0	3	–
	Δ'	1	0	0	-2	0	-6	
		↑ e					$-z_B$	

Base	T	1	2	3	4	5	b'	$b'_i / T_{ie} > 0$
1	1	[1]	0	1	-1	0	3	$3 \rightarrow s$
5	2	0	0	1	0	1	5	–
2	3	0	1	0	1	0	3	–
	Δ'	0	0	-1	-1	0	-9	$-z_B$

On retrouve l'optimum calculé par la méthode des deux phases !

1.7 Dualité

1.7.1 Définition et exemple

À tout programme linéaire, appelé par convention *PL primal*, on peut associer un autre PL appelé son *dual*. Voici un PL primal sous forme générale, c'est-à-dire non mis sous forme canonique ou standard, et son dual. On a séparé dans ce primal les variables positives de celles non contraintes en signe, d'une part, et les contraintes d'égalité des contraintes d'inégalité, d'autre part. Ceci partitionne la matrice A en quatre sous-matrices.

Primal	Dual
Max $z = c_1^T x_1 + c_2^T x_2$	Min $w = b_1 y_1 + b_2 y_2$
$A_{1,1}x_1 + A_{1,2}x_2 \le b_1$	$y_1 \ge 0$
$A_{2,1}x_1 + A_{2,2}x_2 = b_2$	y_2 de signe quelconque
$x_1 \ge 0$	$A_{1,1}^T y_1 + A_{2,1}^T y_2 \ge c_1$
x_2 de signe quelconque	$A_{1,2}^T y_1 + A_{2,2}^T y_2 = c_2$

Voici un exemple plus simple, avec un PL primal sous forme canonique :

Max $z = 4x_1 + 7x_2$	Min $w = 6y_1 + 8y_2$
$3x_1 + 5x_2 \le 6$	$3y_1 + y_2 \ge 4$
$x_1 + 2x_2 \le 8$	$5y_1 + 2y_2 \ge 7$
$x_1, x_2 \ge 0$	$y_1, y_2 \ge 0$

Notez comment on est passé mécaniquement au dual pour ce dernier PL canonique :

- Une variable duale $y_i \ge 0$ est associée à chaque contrainte i du primal.
- Dans l'objectif, c est remplacé par b et le sens de l'optimisation est inversé.
- Dans les contraintes, le second membre b est remplacé par c, et $\le$ est remplacé par $\ge$.
- La matrice A est transposée.

La notion de primal et de dual est relative, car la transformation peut avoir lieu dans les deux sens : le dual du dual est le primal.

1.7.2 Interprétation économique

Reprenons l'interprétation économique d'un PL primal en forme canonique, vue au § 1.2.4. Supposons qu'une autre firme B veuille racheter les ressources de l'entreprise A à moindre coût, avec un prix unitaire de y_i pour la ressource i. A n'acceptera de vendre que si, pour chaque activité j, le prix de vente des ressources nécessaires à l'exercice de cette activité avec une intensité 1 est au moins égal au profit unitaire c_j auquel elle renoncera. Les prix y_i optimaux sont précisément solutions du dual suivant, dans lequel la somme dans chaque contrainte j est la somme des prix de vente des ressources pour l'activité j :

$$\text{Min} \sum_{i=1}^{m} y_i b_i$$

$$\forall j = 1 \ldots n : \sum_{i=1}^{m} a_{ij} y_i \geq c_j$$

$$\forall i = 1 \ldots m : y_i \geq 0$$

La dualité est également utilisable par la firme qui possède les biens. Dans le primal, la valeur optimale d'une variable duale y_i correspond à l'amélioration du profit quand on relâche la contrainte i d'une unité (c'est-à-dire quand on augmente de 1 la capacité b_i de la ressource i pour une contrainte ≤, ou quand on la diminue pour ≥). Pour ces raisons, y_i est souvent appelé *coût réduit*, ou *marginal*, de la contrainte i.

Les économistes sont friands de cette interprétation. Les variables duales sont des sortes de *coûts d'opportunité* : augmenter la capacité d'une ressource est une opportunité d'augmenter le profit, et la variable duale est le coût de cette opportunité non réalisée. Les coûts réduits des contraintes sont très utilisés par les décideurs. Heureusement, il n'est pas nécessaire de construire le dual et de le résoudre : les logiciels commerciaux fournissent pour un PL primal un listing de résultats avec le coût réduit de chaque contrainte.

1.7.3 Propriétés du dual

Une présentation complète de la dualité dépasse le cadre de ce livre, aussi citerons-nous seulement quelques propriétés intéressantes. Les deux premières ont déjà été mentionnées dans les deux paragraphes précédents.

- Le dual du dual est le primal.
- La solution du dual est égale aux coûts réduits du primal, et *vice versa*.
- Étant donné un primal et son dual, soit les deux problèmes ont un optimum fini, soit ils sont tous deux sans solution, soit l'un est sans solution et l'autre sans optimum fini.
- Pour un primal en maximisation, le coût d'une solution réalisable est inférieur ou égal au coût d'une solution réalisable du dual. Ces coûts sont égaux si et seulement si les deux solutions sont optimales.
- Pour toute contrainte ≤ du primal, soit la contrainte est *saturée* (vérifiée avec égalité), soit la variable correspondante du dual est nulle. Pour toute contrainte ≥ du dual, soit la contrainte est saturée, soit la variable associée du primal est nulle.

La dernière propriété, connue sous le nom de *théorème des écarts complémentaires* ou de *relations d'exclusion*, est très utilisée. Intuitivement, dans l'interprétation économique, elle signifie qu'une contrainte non saturée (non vérifiée avec égalité à l'optimum) a un coût réduit nul : en effet, la ressource n'étant pas utilisée complètement, augmenter sa capacité ne rapporte rien. Le théorème peut s'écrire matriciellement pour les deux formes classiques d'un PL. Pour un primal canonique et deux solutions optimales x^* et y^* du primal et du dual, on a $y^*.(b - A.x^*) = 0$ et $(y^*.A - c).x^* = 0$. Pour un primal standard, on a toujours $y^*.(b - A.x^*) = 0$ grâce aux égalités, la propriété se réduit donc à $(y^*.A - c).x^* = 0$.

1.7.4 Utilité de la dualité

Outre l'interprétation économique, la dualité est très utilisée en programmation linéaire, notamment pour faciliter les calculs. Par exemple, un PL avec dix variables et deux contraintes ne peut pas être résolu graphiquement, tandis que cela ne pose pas de problème pour son dual à deux variables et dix contraintes. Les propriétés permettent aussi d'accélérer la résolution du PL dual si on dispose déjà de la solution optimale du primal.

Voici un petit exemple, avec un PL primal sous forme canonique et son dual.

Max $z = x_1 + x_2$	Min $w = 24y_1 + 17y_2 + 21y_3 + 15y_4$
(1) $3x_1 + x_2 \le 24$	(5) $3y_1 + 2y_2 + y_3 - y_4 \ge 1$
(2) $2x_1 + x_2 \le 17$	(6) $y_1 + y_2 + 3y_3 + 3y_4 \ge 1$
(3) $x_1 + 3x_2 \le 21$	$y_1, y_2, y_3, y_4 \ge 0$
(4) $-x_1 + 3x_2 \le 15$	
$x_1, x_2 \ge 0$	

L'algorithme du simplexe ou une résolution graphique trouve l'optimum défini par $z = 11$, $x_1 = 6$ et $x_2 = 5$. D'après la propriété 4, on sait déjà que la valeur optimale de l'objectif du dual est $w = 11$. Les contraintes (2) et (3) sont saturées, mais (1) et (4) ne le sont pas. D'après le théorème des écarts complémentaires, y_1 et y_4 sont donc nulles. Ensuite, x_1 et x_2 sont non nulles dans le primal. Les contraintes correspondantes (5) et (6) du dual sont donc saturées. Vérifiées avec égalité, elles donnent les deux équations à deux inconnues (7) et (8) qui fournissent les deux autres variables duales : $y_2 = 2/5$ et $y_3 = 1/5$.

(7) $2y_2 + y_3 = 1$

(8) $y_2 + 3y_3 = 1$

Quand on résout un PL par l'algorithme du simplexe, on peut déduire du tableau final une solution optimale du problème dual. Ceci explique que les logiciels sont capables de fournir la valeur de la variable duale pour chaque contrainte, dans leur listing de résultats. Cette déduction est facile à expliquer si le PL de départ avec n variables et m contraintes est sous forme canonique. Le PL mis en forme standard a une variable d'écart par contrainte, indicées de x_{n+1} à x_{n+m}. Si la i-ème variable d'écart x_{n+i} est en base, alors la variable duale de la contrainte i est nulle : $y_i = 0$. Si elle est hors base, son coût réduit dans la ligne Δ est l'opposé de la valeur de y_i (ou la valeur de y_i en minimisation).

Reprenons par exemple le PL déjà utilisé dans le § 1.3 et le § 1.5, avec la forme canonique à gauche et la forme standard à droite.

Max $z = x_1 + 2x_2$	Max $z = x_1 + 2x_2$
$x_1 + x_2 \le 6$	$x_1 + x_2 + x_3 = 6$
$x_2 \le 3$	$x_2 + x_4 = 3$
$x_1, x_2 \ge 0$	$x_1, x_2, x_3, x_4 \ge 0$

Rappelons le tableau final du simplexe tel qu'obtenu à la fin du § 1.5.

Base	T	1	2	3	4	b'	
1	1	1	0	1	-1	3	
2	2	0	1	0	1	3	
	Δ	0	0	-1	-1	-9	$-z_B$

Les variables d'écart x_3 et x_4 correspondent aux contraintes 1 et 2 de la forme canonique. Elles ne sont pas en base à l'optimum. Leurs coûts réduits sur la ligne Δ du tableau sont donc respectivement égaux aux variables duales, multipliées par -1 car on est en maximisation : $y_1 = 1$ et $y_2 = 1$.

1.8 Analyse de sensibilité

Une question fréquente concerne les conséquences sur la solution si des données changent. On peut modifier les données et relancer l'algorithme du simplexe, mais, heureusement, ce n'est pas toujours nécessaire. *L'analyse de sensibilité* consiste à calculer les intervalles dans lesquels peut varier un coefficient de coût c_j ou un second membre b_i, sans que l'optimum change. Les logiciels ont souvent une commande "*Range*" qui calcule ces intervalles. Les informations fournies ne sont valides que si on fait varier un seul paramètre à la fois.

Changement d'un coefficient de coût

Montrons sur un exemple qu'il n'est pas nécessaire de tout recalculer quand on change un coefficient de coût dans la fonction-objectif. Soit le programme linéaire suivant :

$$\text{Max } z = 3x_1 - 2x_2$$
$$2x_1 - x_2 \le 30$$
$$x_1 - x_2 \le 10$$
$$x_1, x_2 \ge 0$$

Le tableau final du simplexe est le suivant :

Base	T	1	2	3	4	b'	
2	1	0	1	1	-2	10	
1	2	1	0	1	-1	20	
	Δ	0	0	-1	-1	-40	$-z_B$

Dans quel intervalle la solution reste-t-elle optimale si on fait varier le coefficient c_1 de x_1 ? La fonction-objectif initiale, paramétrée par c_1, est $z = c_1.x_1 - 2.x_2$. Le tableau final nous donne les expressions de x_1 et x_2 en fonction des variables hors base :

$$x_1 = 20 - x_3 + x_4$$
$$x_2 = 10 - x_3 + 2x_4$$

La fonction-objectif peut ainsi être réécrite en fonction des variables hors base x_3 et x_4. Ce qui redonne la ligne Δ du tableau, mais paramétrée par c_1 :

$$\begin{aligned} z &= c_1 \cdot x_1 - 2 \cdot x_2 \\ &= c_1 \cdot (20 - x_3 + x_4) - 2 \cdot (10 - x_3 + 2x_4) \\ &= 20 \cdot (c_1 - 1) + (2 - c_1) \cdot x_3 + (c_1 - 4) \cdot x_4 \end{aligned}$$

La solution reste optimale si les coûts réduits des variables hors base sont négatifs ou nuls. C'est le cas si $2 - c_1 \leq 0$ et $c_1 - 4 \leq 0$, c'est-à-dire si $c_1 \in [2,4]$. Une analyse similaire pour le coefficient de coût c_2 de x_2 permet de trouver $c_2 \in [-3,-3/2]$. Ainsi, si on modifie un seul coefficient à la fois en restant dans son intervalle, les valeurs optimales de x_1 et x_2 ne changent pas. Bien entendu, il faut recalculer le coût total.

Ces informations sont précieuses pour les décideurs. Soit par exemple un aliment pour bétail à teneurs garanties en protéines, lipides et glucides, produit à partir de matières premières comme l'orge, le sorgho, le soja, etc. Le mélange de coût minimal est déterminé par les cours du jour de ces matières. L'analyse de sensibilité permet de préciser pour chaque matière première les plages de coûts dans lesquelles le mélange reste optimal.

Changement d'un second membre

Conceptuellement, il suffit de passer au dual : les variations de seconds membres du primal deviennent des variations de coefficients dans la fonction-objectif. Sans aller aussi loin, et sans chercher à optimiser les calculs, montrons sur l'exemple précédent qu'il est possible de trouver sans tâtonner la plage de variation du premier second membre b_1.

La base optimale B est formée des colonnes de x_1 et x_2. On ne touche pas au contenu du tableau T et aux coûts, mais le changement du second membre b_1 peut rendre la solution de base non réalisable. En effet, la solution de base actuelle x_B est de la forme $x_B = B^{-1}.b$, et ses composantes doivent être positives ou nulles (*cf.* § 1.4). D'où :

$$B = \begin{pmatrix} 2 & -1 \\ 1 & -1 \end{pmatrix} \qquad B^{-1} = \begin{pmatrix} 1 & -1 \\ 1 & -2 \end{pmatrix}$$

$$x_B = B^{-1}b = \begin{pmatrix} 1 & -1 \\ 1 & -2 \end{pmatrix} \times \begin{pmatrix} b_1 \\ 10 \end{pmatrix} \geq 0 \Leftrightarrow \begin{cases} b_1 - 10 \geq 0 \\ b_1 - 20 \geq 0 \end{cases} \Rightarrow b_1 \in]20,+\infty]$$

Une analyse de sensibilité similaire donne pour b_2 l'intervalle $]-\infty,15]$.

1.9 Les algorithmes modernes

1.9.1 Raffinements dans l'algorithme du simplexe

L'algorithme du simplexe nécessite de nombreux raffinements pour traiter de façon robuste de grands PL. La version simple que nous avons vue, la forme tableau, a l'inconvénient de prendre trop de place si la matrice est *peu dense* (avec beaucoup de zéros) et d'accumuler

progressivement des erreurs d'arrondi au cours des pivotages. Elle nécessite aussi la forme standard, avec des contraintes d'égalité. Citons six raffinements des logiciels modernes. Les langages de modélisation ne sont pas mentionnés ici ; ils font l'objet du chapitre 3.

Le formatage automatique en forme standard

On peut traiter directement un PL qui mixe des contraintes ≤, ≥ et =, voire des variables non contraintes en signe. Les logiciels modernes font les changements nécessaires pour se ramener à une forme standard avec variables non négatives.

La gestion des cas à problème

Les versions modernes du simplexe détectent sans difficulté les PL à solution non bornée ou sans solution. Quand la base évidente (point O du polyèdre) n'est pas disponible, elles calculent une base initiale par une des techniques de variables artificielles que nous avons exposées (méthodes des deux phases ou du grand M).

Le contrôle de la précision

À une itération n° k du simplexe, on utilise l'inverse $B^{-1}(k)$ de la matrice de base actuelle $B(k)$. À l'itération suivante $k+1$, l'inverse de la nouvelle matrice de base $B(k+1)$ est déduite par pivotage de l'inverse $B^{-1}(k)$ de la base précédente. Ceci est possible car deux matrices de base successives ne diffèrent que d'une colonne. Le prix à payer est une perte progressive de précision au cours des itérations. Cette perte peut être importante et rapide sur certaines matrices très particulières, dites de Hilbert.

La précision peut être améliorée en utilisant des formes plus stables de l'algorithme du simplexe *(algorithme révisé du simplexe)* et en décomposant les matrices en une matrice triangulaire supérieure et une inférieure *(décomposition LU)*. Périodiquement, il faut aussi inverser explicitement la matrice de base actuelle pour purger les erreurs d'arrondi.

La gestion efficace des matrices creuses

Les grands PL rencontrés en pratique sont creux à au moins 90 %, c'est-à-dire que leurs matrices A ont au moins 90 % de zéros. Il existe des techniques pour compresser ces matrices de manière à ne stocker que les éléments non nuls. Par exemple, on crée des listes chaînées d'éléments non nuls dans chaque ligne et chaque colonne. Outre un gain énorme de place qui permet de traiter en mémoire des PL plus grands, les calculs sont accélérés.

Le prétraitement

Cette technique *(preprocessing)* consiste à rechercher des simplifications comme par exemple l'élimination de contraintes redondantes ou des changements de variables. Elle permet de diminuer notablement la taille du PL fourni aux algorithmes de résolution.

Variables duales et analyse de sensibilité

Les bons logiciels donnent d'emblée dans leurs listings de résultats les variables duales associées à chaque contrainte. En général, une commande supplémentaire *Range* fournit pour chaque coefficient de coût et chaque second membre l'intervalle à l'intérieur duquel l'optimum reste inchangé.

1.9.2 Les méthodes de points intérieurs

Généralités

L'algorithme du simplexe emprunte une suite de sommets du polyèdre et il est rapide *en moyenne* : le nombre de pivotages est proportionnel au nombre de contraintes m, et chaque pivotage coûte de l'ordre de mn opérations. Klee et Minty [Klee 1972] ont trouvé des exemples pathologiques où le simplexe visite quasiment tous les sommets, dont le nombre peut être énorme (nombre de façons de choisir m colonnes parmi n dans A, soit C_n^m).

Cette possibilité théorique de rencontrer des PL faisant "dérailler" le simplexe a stimulé des recherches pour un algorithme ayant une complexité polynomiale en m et n, même sur les pires PL. Le russe Khachian a trouvé le premier algorithme de ce type en 1979, la *méthode de l'ellipsoïde* [Khachian 1979]. Bien que plus rapide que le simplexe sur les problèmes de Minty, elle n'a jamais pu s'imposer car, en moyenne, elle est bien plus lente que le simplexe sur des problèmes réels.

Karmarkar, des laboratoires ATT, a trouvé en 1984 un nouvel algorithme [Karmarkar 1984]. Son temps de calcul dans le pire des cas est proportionnel à $n^{3.5}L$, L désignant le nombre de bits nécessaires pour coder en mémoire les tableaux A, b et c définissant le PL. Cet algorithme est trop compliqué pour des calculs à la main, c'est pourquoi nous ne le détaillons pas. Il faut seulement retenir qu'il s'agit d'une *méthode de points intérieurs*, où on plonge dans le polyèdre pour aller plus vite vers le sommet optimal.

Des dérivés de cet algorithme sont aujourd'hui compétitifs avec le simplexe. Certains logiciels incluent déjà, en plus du simplexe, un algorithme de points intérieurs. Sur certains types de PL, les méthodes de points intérieurs peuvent être dix fois plus rapides que le simplexe, mais il existe encore de nombreux cas où le simplexe reste plus rapide. Le simplexe a encore de beaux jours devant lui, car il a bénéficié de nombreuses améliorations depuis la version initiale de Dantzig en 1947.

Un exemple d'algorithme de points intérieurs

L'algorithme suivant est plus simple que la version initiale de Karmarkar. Il a été proposé par Adler, Karmarkar, Resende et Veiga [Adler 1989]. Il travaille sur un PL en forme canonique (P) et en maximisation. Il nécessite aussi un point initial x^0 intérieur au polyèdre. Il calcule une suite de points intérieurs $x^0, x^1, ..., x^k$ telle que $b - A{\cdot}x^k > 0$ et $c{\cdot}x^{k+1} > c{\cdot}x^k$.

```
k := 0
Répéter
   v := b - A.x
   D := diag (v1,v2,...,vm)
   h := (A^T.D^-2.A)^-1.c
   f := -A.h
   Si f ≥ 0 alors stop (PL non borné)
   q := p.min{-vi/fi | fi < 0, i = 1..m}
   y := x
   x := x + q.h
   k := k + 1
Jusqu'à (c.x - c.y) < ε.
```

Le vecteur v correspond aux variables d'écart pour la solution actuelle x. Pour calculer D, on initialise un tableau $m \times m$ à 0, et on copie v sur la diagonale. D^{-2} se déduit de D en remplaçant chaque terme $D_{ii} = v_i$ de la diagonale par le carré de son inverse $(1/v_i)^2$. Le vecteur h est une direction de déplacement. Le réel q est la valeur du déplacement dans cette direction.

Le paramètre p est un facteur de sécurité, nombre réel entre 0 et 1 qui limite le déplacement car, sur des PL volumineux, on peut sortir du polyèdre à cause des erreurs de calcul. Sur de petits PL, on peut prendre sans risque $p = 1$. Le nouveau point est donc $x + q.h$. L'algorithme s'arrête quand la variation de coût devient inférieure à une limite fixée ε (10^{-6} par exemple).

Voici un exemple numérique sur un petit PL à $n = 2$ variables et $m = 4$ contraintes :

$$\text{Max } z = 2x_1 + x_2$$
$$x_1 + x_2 \le 4$$
$$-x_1 + x_2 \le 2$$
$$x_1 \le 2$$
$$0.5x_1 - x_2 \le 0$$
$$x_1, x_2 \ge 0$$

La figure 1.4 illustre le polygone défini par les contraintes. L'optimum est $x^* = (2,2)$, avec un coût $z^* = 6$ (point K).

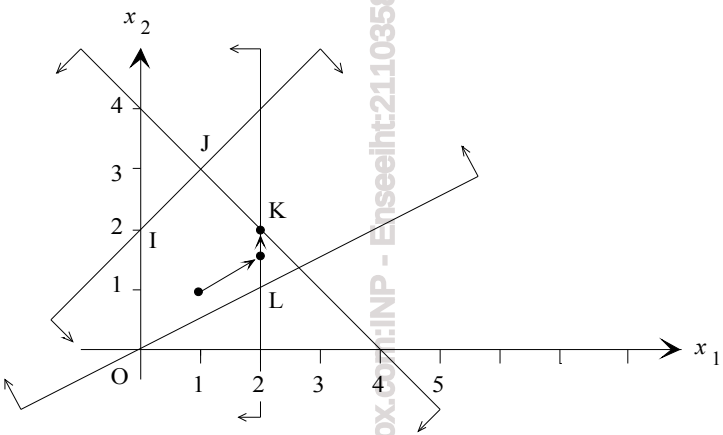

Figure 1.4 – Trajectoire suivie par l'algorithme

En partant du point intérieur $x_0 = (1,1)$ et sans facteur de sécurité ($p = 1$), l'algorithme effectue deux itérations : il atteint d'abord le point $x_1 = (2, 1.591)$, puis x^*. Détaillons les calculs pour la première itération uniquement.

$v = (2,2,1,0.5)$, $D^{-2} = diag\left(\frac{1}{4},\frac{1}{4},1,4\right)$

$A^T.D^{-2}.A = \begin{pmatrix} 2.5 & -2 \\ -2 & 4.5 \end{pmatrix}$, $h = (1.517,0.897)$,

$f = (-2.414,0.621,-1.517,0.138)$

q (sans facteur de sécurité) = 0.659

d'où $x^1 = (2,1.591)$.

En l'absence de point intérieur initial pour le PL de départ (P), on utilise une méthode en deux phases avec une seule variable artificielle x_A. On résout en phase 1 le PL (P') suivant, dans lequel M est un grand réel positif et e un vecteur $m \times 1 = (1, 1, ..., 1)^T$:

$$\text{Max } w = c \cdot x - M \cdot x_A$$
$$A \cdot x - e \cdot x_A \leq b$$
$$x, x_A \geq 0$$

Le vecteur initial x doit être ($\| b \| / \| A.c \|$).c (les doubles barres désignent la norme euclidienne), et la valeur initiale de la variable artificielle x_A est 2.$\| b - A.x \|$. Dans ce cas, le vecteur (x, x_A) est un point intérieur initial pour (P'). L'algorithme de point intérieur est appliqué à (P'), mais avec un test de fin supplémentaire $x_A < 0$. En fin de phase 1, si $x_A < 0$, alors x est un point intérieur pour (P) : on peut passer à la phase 2 qui consiste à appliquer l'algorithme à (P) avec x comme point initial. Si la phase 1 se termine sans avoir $x_A < 0$, alors (P) n'a pas de solution réalisable.

Cet algorithme est assez simple à programmer mais nécessite des calculs pénibles à la main. Il s'agit d'une version pédagogique, bien plus lente que l'algorithme du simplexe et nécessitant de multiples raffinements pour être aussi rapide.

1.10 Références et compléments

La programmation linéaire donne toujours lieu à un ou plusieurs chapitres dans les ouvrages généraux sur la recherche opérationnelle, comme celui d'Alj et Faure en français [Alj 1990] et celui de Winston [Winston 1991] en anglais. Des livres entiers lui sont aussi consacrés. En langue française, citons celui de de Werra [de Werra 1990] et le petit livre de Jacquet-Lagrèze [Jacquet-Lagrèze 1999]. Les titres en langue anglaise sont très nombreux : nous recommandons ceux de Chvàtal [Chvàtal 1983], Bertsimas and Tsisiklis [Bertsimas 1997], Schrijver [Schrijver 1998], Vanderbei [Vanderbei 2001], ainsi que l'ouvrage très fouillé de Bazaraa *et al*. [Bazaraa 2009].

Fleury et Lacomme ont publié récemment [Fleury 2010] un livre sur les concepts avancés de programmation linéaire, incluant la relaxation lagrangienne, la génération de colonnes, la programmation linéaire stochastique et la programmation linéaire multi-objectif. En particulier, les deux derniers sujets sont pour la première fois présentés de manière abordable dans un ouvrage en français.

La programmation non linéaire est souvent traitée avec la programmation linéaire dans des livres généraux sur l'optimisation. Citons en français les livres de Culioli [Culioli 1994] et de Minoux [Minoux 2007], et en anglais les ouvrages de Luenberger [Luenberger 2003] et de Sofer et Nash [Sofer 2008]. Il existe cependant des titres entièrement consacrés à la programmation non linéaire : nous recommandons celui de Bazaraa *et al*. car il contient de nombreux algorithmes et exemples numériques [Bazaraa 2006].

La méthode de points intérieurs du paragraphe 1.9 a été publiée dans un article d'Adler *et al.* [Adler 1989]. La recherche dans ce domaine est très active : la plupart des numéros de la revue *Mathematical Programming* contiennent un article sur le sujet. Il existe un livre très intéressant et abordable d'Arbel [Arbel 1993] sur les méthodes de points intérieurs, auquel est jointe une disquette de logiciels.

CHAPITRE 2

Programmation linéaire en nombres entiers

2.1 Introduction

La programmation linéaire en nombres entiers étudie des programmes linéaires dans lesquels les variables sont astreintes à être entières. En particulier, les variables peuvent être simplement booléennes, c'est-à-dire ne prendre que les valeurs 0 ou 1. De nombreuses contraintes, en apparence non linéaires, peuvent être linéarisées grâce à des variables entières. Ces possibilités étendent énormément le champ d'application de la programmation linéaire. Même si les programmes linéaires obtenus sont souvent difficiles à résoudre, la programmation linéaire en nombres entiers est quand même très utile comme langage de modélisation : elle permet de décrire de façon concise et de communiquer à d'autres personnes des problèmes d'optimisation discrète.

Le § 2.2 introduit la notion de programme linéaire en nombres entiers, les difficultés de résolution, et les grands types de méthodes disponibles dont les plus importantes font l'objet des trois paragraphes suivants. Les programmes linéaires pour lesquels l'algorithme du simplexe trouve automatiquement des solutions optimales entières font l'objet du § 2.3. La méthode de Dakin pour résoudre un programme linéaire quelconque à variables entières est présentée au § 2.4. La méthode additive de Balas est décrite au § 2.7 comme exemple typique d'algorithme pour les variables booléennes. Ces deux méthodes ont été choisies pour leur simplicité. Les méthodes modernes sont beaucoup plus complexes, mais reposent sur les mêmes principes de base.

La modélisation de quelques contraintes courantes grâce à la programmation linéaire en nombres entiers est abordée au § 2.7. En particulier, les variables booléennes permettent de traiter des contraintes très fréquentes comme des coûts fixes, des exclusions, des implications, etc.

2.2 Notions de base

2.2.1 Définitions

Le programme mathématique (P) suivant est un programme linéaire ordinaire (PL) en minimisation, écrit en notation matricielle et en forme standard, c'est-à-dire avec des contraintes d'égalité. m désigne le nombre de contraintes, n le nombre de variables, A est une matrice $m \times n$, c est le vecteur des coûts unitaires associés aux variables, et b est le vecteur des seconds membres des contraintes. Les variables sont des *réels* non négatifs.

$$(P) \quad \text{Min } z = c \cdot x$$
$$A \cdot x = b$$
$$x \geq 0$$

Nous avons vu dans le chapitre 1 que tout PL peut se ramener à cette forme et que l'optimum, quand il existe et qu'il est borné, est atteint en un sommet du polyèdre défini par les contraintes. Les principaux algorithmes de résolution sont les méthodes de type simplexe et les méthodes de points intérieurs. Si, dans (P), on contraint le vecteur x à être entier, on obtient le programme (Q), qui est un *programme linéaire en nombres entiers* (PLNE) :

$$(Q) \quad \text{Min } z = c \cdot x$$
$$A \cdot x = b$$
$$x \in \mathbb{N}^n$$

En général, un tel problème n'a des solutions que si A et b sont aussi entiers, et c'est ce qu'on suppose en pratique. Si les variables valent 0 ou 1 (variables dites *booléennes*, *binaires* ou *de décision*), on obtient un cas particulier de PLNE appelé *PL en 0-1*. Enfin, si seulement certaines variables doivent être entières, il s'agit d'un *PL mixte*. Les principales motivations qui amènent à utiliser la PLNE sont :

- le besoin de variables entières (comme un nombre de camions à acheter) ;
- la modélisation de contraintes et de conditions ingérables par la programmation linéaire en variables continues.

Par exemple, soient deux actions qui peuvent être exécutées ou non. On peut définir deux variables binaires x et y valant 1 seulement si l'action correspondante est effectuée. Si les deux actions sont exclusives, on peut introduire la contrainte $x + y \leq 1$. Il est impossible de modéliser ce genre de situation de manière linéaire sans variables binaires.

2.2.2 Difficultés de la PLNE

Considérons le PLNE suivant à deux variables, qu'on peut résoudre graphiquement sur la figure 2.1. Les + indiquent les points à coordonnées entières dans le domaine des solutions réalisables. Si on relâche la contrainte d'intégrité sur les variables, on obtient un PL ordinaire appelé le *PL relaxé* associé au PLNE. Dans ce cas, l'optimum est atteint en B = (11/4, 9/4) avec un coût de 7,75. *L'optimum entier* est en revanche C = (3,1), de coût 7.

$$\text{Max } z = 2x_1 + x_2$$
$$x_1 + x_2 \leq 5$$
$$-x_1 + x_2 \leq 0$$
$$6x_1 + 2x_2 \leq 21$$
$$x_1, x_2 \in \mathbb{N}$$

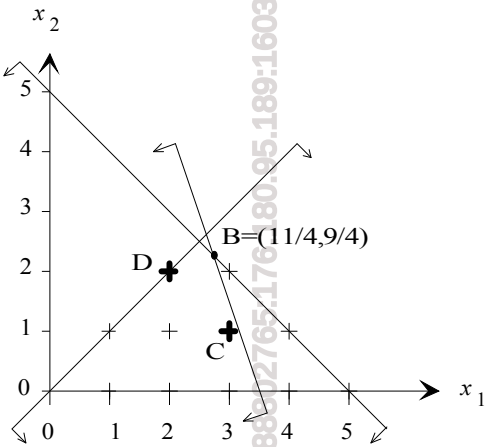

Figure 2.1 – Résolution graphique d'un PLNE

On en déduit les remarques suivantes qui soulignent les difficultés de la PLNE :

- L'optimum entier n'est pas forcément un sommet du polyèdre, il peut être à l'intérieur du polyèdre, comme ici.
- Si le PL relaxé avait un optimum entier, ce serait aussi l'optimum du PLNE.
- Sinon, le coût optimal du PL relaxé donne une borne supérieure (une borne inférieure en minimisation) du coût optimal du PLNE (ici le PL relaxé donne 7,75 au lieu de 7).
- L'arrondi de la solution du PL relaxé (ici D = (2,2)) n'est pas nécessairement optimal pour le PLNE, il pourrait même ne pas être réalisable.
- Le polyèdre peut être non vide, mais n'admettre aucune solution entière.
- On peut construire des cas où les optima du PLNE et du PL relaxé sont aussi éloignés que l'on veut.

Les méthodes de la PL ordinaire ne fonctionnent donc pas pour la PLNE puisqu'elles cherchent un sommet optimal du polyèdre qui, en général, n'a pas de coordonnées entières.

Arrondir la solution du PL relaxé est cependant possible quand les variables représentent des quantités importantes. Par exemple, si on produit des milliers de stylos-billes et que le simplexe donne des solutions non entières, on peut arrondir sans trop de risques : même si une contrainte est légèrement violée après arrondi, il est rare que les capacités de production soient strictes ou connues avec une grande précision pour de telles quantités. Il

n'en est plus de même si le simplexe trouve 0,5 pour une variable de décision qui doit valoir 0 ou 1 : cette solution n'a plus de signification réelle.

Sans hypothèses sur les données, un PLNE est en général un problème difficile, et on ne connaît pas d'algorithme polynomial comme pour les PL ordinaires. Même en ignorant la fonction-objectif, trouver une solution entière à un système d'équations $A.x = b$ reste un problème difficile. Bien sûr, le sommet optimal du PL relaxé peut être entier. Le simplexe résoudra donc optimalement un tel PLNE. C'est hélas rare.

Alors qu'on peut résoudre actuellement des PL à 100 000 variables et plus, les chances de résoudre un grand PLNE dépendent beaucoup de la structure du problème. Typiquement, on parvient à traiter des programmes en 0-1 à quelques centaines de variables et des programmes comportant quelques dizaines de variables entières générales. Bien que des PLNE beaucoup plus volumineux soient facilement résolus dans certaines applications, il est clair qu'il faut prendre des précautions avant de lancer la résolution d'un grand PLNE avec un logiciel, par exemple en effectuant des tests avec des exemples de taille croissante.

Même quand un PLNE est trop gros pour être résolu en un temps raisonnable, la programmation linéaire en nombres entiers reste un puissant langage de modélisation, très appréciée pour exprimer de façon concise un problème d'optimisation discrète, pour le communiquer à des tiers, et pour faire apparaître certaines propriétés de structure.

2.2.3 Méthodes de résolution des PLNE

Les principales méthodes de résolution peuvent se ranger en quatre groupes : résolution comme un programme linéaire ordinaire, méthodes arborescentes, méthodes de coupes et génération de colonnes.

Résolution comme un PL ordinaire

Elle est possible pour les PLNE équivalents à leurs PL relaxés, c'est-à-dire que le sommet optimal du PL relaxé est à coordonnées entières. Dans ce cas, le simplexe n'y voit que du feu en passant de sommet en sommet, et la solution optimale du PL relaxé est aussi celle du PLNE. Ces PLNE existent et certains types importants dans les applications peuvent être détectés par des propriétés présentées au § 2.3.

Méthodes arborescentes

Aussi appelées *méthodes de séparation et d'évaluation (branch-and-bound)*, ce sont les méthodes utilisées par les logiciels du commerce quand on leur soumet un PLNE (ils utilisent le simplexe ou une méthode de points intérieurs s'ils détectent un PL ordinaire). Le principe de ces méthodes est de choisir une variable x et de *séparer* le problème en deux sous-problèmes selon les valeurs de x. Pour un PLNE général, on sépare en considérant un entier p et les deux sous-problèmes $x \leq p$ et $x \geq p+1$. Pour un PL en 0-1, on sépare en considérant les deux cas $x = 0$ et $x = 1$. Les PLNE des sous-problèmes peuvent à leur tour être séparés, ce qui forme progressivement une *arborescence* dont chaque nœud correspond à un sous-problème. Cette arborescence peut être énorme : ainsi, pour un PL en 0-1 à seulement trente variables, on a déjà $2^{30} \approx 10^9$ nœuds possibles.

La majorité des sous-problèmes sont en fait éliminés (on dit que l'arborescence est "élaguée") grâce à une *évaluation*. En chaque nœud P, cette évaluation $ev(P)$ doit être une borne inférieure (en minimisation) ou une borne supérieure (en maximisation) du coût optimal du PLNE. Une évaluation répandue est de résoudre le PL relaxé avec le simplexe.

Voici comment fonctionne l'élimination d'un nœud P : dès que la recherche arborescente a trouvé une première solution entière, ayant un certain coût z, on peut ignorer P si $ev(P) \geq z$ (en minimisation). En effet, toutes les solutions potentielles que peut fournir le nœud auront un coût d'au moins $ev(P)$; elles ne nous intéressent pas car on dispose déjà d'une solution provisoire de coût z.

Pour la PLNE générale, nous exposons au § 2.4 la *méthode de Dakin*, qui utilise le PL relaxé pour évaluer les solutions. Pour les PL en 0-1, nous décrivons au § 2.5 la *méthode de Balas*, plus sophistiquée. Les logiciels modernes utilisent des méthodes analogues, mais avec de très nombreux raffinements qui accélèrent les calculs en diminuant le nombre de sous-cas examinés.

Méthodes de coupes

L'objectif des *méthodes de coupes* ou de troncatures est de tenter de trouver l'enveloppe convexe des solutions entières, c'est-à-dire le plus petit polyèdre contenant toutes les solutions entières du PLNE. On ne sait pas l'obtenir directement. En pratique :

- on résout le PL relaxé ;
- on ajoute des contraintes (appelées *coupes*, *plans sécants* ou *inégalités valides*) qui "rognent" le polyèdre des solutions sans perdre les solutions optimales entières ;
- après un certain nombre de tels ajouts, le PL relaxé a une solution entière, optimum du PLNE.

Un inconvénient est d'obtenir une solution réalisable uniquement à la fin, contrairement aux méthodes arborescentes qui livrent de bonnes solutions provisoires en cours de calcul. Les méthodes de coupes sont surtout utilisées comme techniques d'appoint dans les méthodes arborescentes : elles servent à améliorer au maximum la borne inférieure en chaque nœud de l'arborescence, avant séparation. Les méthodes arborescentes utilisant ces techniques sont appelées *branch-and-cut* en anglais.

Génération de colonnes et branch-and-price

Appelée aussi *programmation linéaire généralisée*, la technique de génération de colonnes est employée pour résoudre des programmes linéaires continus ayant un nombre si élevé de variables qu'il est impossible de générer explicitement toutes les colonnes de la matrice A. Les problèmes de recouvrement (découpe de tôles du chapitre 8, localisation d'émetteurs du chapitre 11) et de partitionnement (découpage électoral du chapitre 14) appartiennent à cette catégorie. Leurs colonnes correspondent à des sous-ensembles d'un ensemble de référence à m éléments. Ils peuvent donc avoir en théorie jusqu'à 2^m colonnes.

Le nombre de colonnes intéressantes (susceptibles de composer de bonnes solutions) est relativement restreint. La génération de colonnes commence donc par sélectionner un nombre très réduit de colonnes prometteuses. Le PL obtenu, appelé *problème-maître restreint* ou PMR, est résolu par l'algorithme du simplexe. Comme il est peu probable que

le PMR donne directement la solution optimale du problème initial, on doit disposer d'un algorithme générateur de colonnes qui, à partir des colonnes du PMR, propose une ou plusieurs colonnes qui améliorent la solution courante. Ces colonnes, qui doivent avoir un coût réduit négatif (en minimisation), sont calculées à partir des variables duales de la solution courante et ajoutées au PMR. Une solution optimale du PMR est obtenue quand le générateur de colonnes ne trouve plus de colonnes améliorantes.

Pour un grand PLNE, la génération de colonnes ne donne que l'optimum du PL relaxé. On doit ensuite utiliser une généralisation des méthodes arborescentes appelée *branch-and-price*. Étant donné que toutes les colonnes du PLNE à résoudre n'apparaissent pas dans le PMR, cette technique génère de nouvelles colonnes en chaque nœud. La génération de colonnes a aussi l'avantage de pouvoir prouver l'optimalité du PL examiné en un nœud de l'arborescence. Les méthodes de branch-and-price sont certainement promises à un bel avenir, mais elles sont encore peu utilisées dans les logiciels d'optimisation commerciaux. Elles sortent donc du cadre de cet ouvrage.

2.3 PLNE équivalents à leur PL relaxé

2.3.1 Définition de la totale unimodularité

Un sommet du polyèdre d'un PLNE peut avoir par coïncidence des coordonnées entières, mais il existe une large classe de PLNE dont tous les sommets sont de ce type. Ces PLNE sont alors résolus optimalement si on exécute l'algorithme du simplexe sur le PL relaxé. Ils sont caractérisés par une matrice A *totalement unimodulaire* (ou TU), c'est-à-dire une matrice dont toute sous-matrice carrée a un déterminant égal à -1, 0, ou +1.

Cette définition s'applique en particulier aux sous-matrices 1×1 : une matrice TU a donc ses éléments dans {-1, 0, +1}. Comme autres propriétés simples, notons que si A est TU, alors sa transposée A^{T} et sa concaténation avec une matrice identité $(A \mid I)$ sont aussi TU. La condition d'appartenance des éléments à {-1, 0, +1} n'est pas suffisante. Ainsi, la matrice A 3×4 ci-dessous a toutes ses sous-matrices d'ordre 1 et 2 avec des déterminants valant -1, 0 ou +1, mais le déterminant de la première sous-matrice 3×3 vaut 2.

$$A = \begin{pmatrix} 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \qquad \begin{vmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & 1 & 1 \end{vmatrix} = 2$$

Tester si une matrice quelconque est unimodulaire n'est pas facile, mais certaines classes de matrices TU peuvent être détectées par des propriétés. Heller et Tomkins ont ainsi proposé une condition suffisante pour qu'une matrice A à éléments dans {-1,0,+1} soit TU (voir par exemple le livre de Papadimitriou et Steiglitz [Papadimitriou 1998]) :

- Chaque colonne ne contient pas plus de deux éléments non nuls.
- On peut partitionner ses lignes en deux sous-ensembles L_1 et L_2 tels que : a) si une colonne a deux éléments non nuls de même signe, l'un est dans L_1, l'autre dans L_2 ; b) si une colonne contient deux éléments non nuls de signe contraire, ils sont tous deux dans un même sous-ensemble L_1 ou L_2.

Par exemple, la matrice A ci-dessous est TU car elle vérifie le théorème de Heller et Tomkins avec $L_1 = \{1,2\}$ et $L_2 = \{3,4\}$. La matrice B est également TU, mais elle ne vérifie pas le théorème car elle n'a pas deux éléments non nuls dans chaque colonne.

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

2.3.2 Totale unimodularité et solutions entières

Soit un programme linéaire continu en forme standard, avec une matrice A $m \times n$, de rang m et à coefficients entiers, et un vecteur b entier. On peut montrer que les trois conditions suivantes sont équivalentes :

- Le déterminant de toute matrice de base B de A vaut +1 ou -1.
- Les sommets du polyèdre $\{x \mid A.x = b,\ x \geq 0\}$ sont à coordonnées entières pour tout vecteur b entier.
- Pour toute base B, la matrice inverse B^{-1} est entière.

Si A est TU, toute sous-matrice B $m \times m$ a un déterminant égal à 0, -1, ou +1. B est une matrice de base si elle est inversible : dét(B) $\neq$ 0. Par conséquent, un PL sous forme standard à matrice TU vérifie la première des trois conditions précédentes. Ses solutions de base sont donc entières et, s'il admet un optimum fini, cet optimum est aussi entier. Ceci s'applique également à la forme canonique $A.x \leq b$ car, après ajout des variables d'écart, on obtient une matrice étendue ($A \mid I$) qui est toujours TU. Le programme dual a également un optimum entier, si les coûts unitaires dans la fonction-objectif du primal sont entiers.

Il faut simplement retenir que si A est TU, un PLNE est équivalent à son PL relaxé. Les contraintes d'intégrité des variables sont alors redondantes, car automatiquement vérifiées. L'algorithme du simplexe permet alors de résoudre efficacement de tels PLNE, même s'ils sont de grande taille. Les trois paragraphes suivants présentent quelques grandes familles de PLNE ayant ces propriétés. Ils sont fréquents dans les applications.

2.3.3 Matrices d'incidence nœuds-arcs et PL de réseaux

Définitions

Un PL est dit *de réseau* si, en ignorant les contraintes simples de bornes inférieures ou supérieures comme $x \leq 3$, chaque colonne de la matrice A a au maximum deux éléments non nuls. S'il y en a deux, ce doit être -1 et +1 ; s'il y en a qu'un, c'est soit -1, soit +1. Une telle matrice A vérifie le théorème de Heller et Tomkins en posant $L_2 = \varnothing$. Elle est donc totalement unimodulaire. À condition que le vecteur b des seconds membres soit entier, les PL de réseau sont résolubles de façon optimale par l'algorithme du simplexe.

On rencontre ces PL dans des problèmes de graphes (ou réseaux). La matrice A correspond alors à un codage du graphe appelé *matrice d'incidence nœuds-arcs*. Soit un graphe G comprenant m nœuds et n arcs reliant chacun un nœud à un autre. La matrice d'incidence

nœuds-arcs a m lignes (une par nœud), et n colonnes, une par arc. Si l'arc en colonne k va du nœud i au nœud j, on a $A(i,k) = 1$, $A(j,k) = -1$ et le reste de la colonne vaut 0. Voici un exemple de graphe, avec des valeurs c_{ij} sur les arcs désignant des distances. L'objectif est de calculer un plus court chemin de a à e.

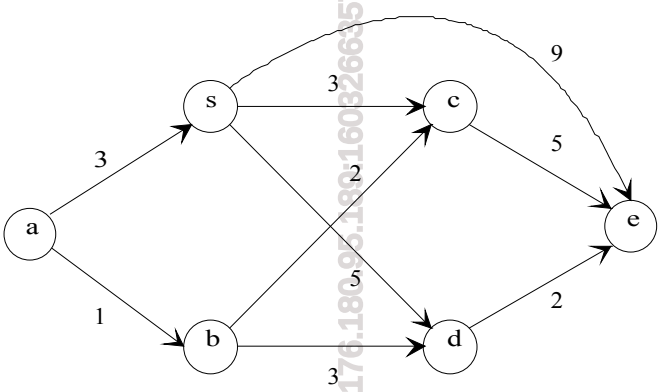

Figure 2.2 – Un problème de plus court chemin

Il est possible de formuler ce problème par un PL en 0-1, bien qu'il existe des algorithmes spécialisés bien plus efficaces travaillant directement sur le graphe. On définit pour chaque arc (i,j) une variable binaire x_{ij} qui vaut 1 seulement si le chemin optimal passe par cet arc. Le programme est écrit dans une police à espacement fixe pour mieux visualiser sa matrice.

```
Min  3x(a,s)+x(a,b)+3x(s,c)+5x(s,d)+9x(s,e)+2x(b,c)+3x(b,d)+5x(c,e)+2x(d,e)

-1 = -x(a,s)-x(a,b)
 1 =                                  x(s,e)                  +x(c,e) +x(d,e)
 0 =  x(a,s)         -x(s,c) -x(s,d) -x(s,e)
 0 =         x(a,b)                          -x(b,c) -x(b,d)
 0 =                 x(s,c)                  +x(b,c)         -x(c,e)
 0 =                         x(s,d)                  +x(b,d)         -x(d,e)
```

La première contrainte signifie qu'un seul arc du chemin part de a, la deuxième qu'un seul arc arrive en e. Les contraintes suivantes concernent tout nœud k autre que a et e : elles signifient que le nombre d'arcs du chemin arrivant en k est égal au nombre d'arcs du chemin qui en partent. Ce nombre vaut 1, mais on n'a pas à le préciser car le chemin est analogue à un flot d'une unité qui se conserve en traversant le réseau. Pour tout nœud k, les contraintes ont été en fait écrites sous la forme de bilans : (nombre d'arcs arrivant en k moins le nombre d'arcs partant de k).

On obtient un beau PL de réseau, avec une matrice A ayant un +1 et un -1 dans chaque colonne. La raison est que chaque variable $x(i,j)$ (et donc l'arc associé) est utilisée dans deux bilans : celui du nœud i, avec un signe -, et celui du nœud j, avec un signe +. Par conséquent, on peut résoudre ce PL en 0-1 directement avec l'algorithme du simplexe : les variables seront automatiquement binaires à l'optimum.

Problèmes de flots

La grande majorité des problèmes à matrices d'incidence nœuds-arcs appartiennent à la grande famille des problèmes de flots. Ces problèmes sont définis sur un graphe (aussi appelé *réseau de transport*) $G = (X, U, C, W, s, t)$, dans lequel un produit va être acheminé. X est un ensemble de n nœuds, U un ensemble de m arcs entre ces nœuds. La capacité d'un arc (i, j) est c_{ij}, le coût de transport par unité de produit traversant l'arc est w_{ij}. Le flot de produit part d'un nœud s appelé *source*, se répartit dans le graphe, et ressort par un nœud t appelé *puits*. Un flot valide doit se conserver en traversant les nœuds internes (lois de Kirchhoff) et respecter les capacités des arcs. Toutes les quantités sont des entiers.

Dans le *problème du flot de coût minimal*, une quantité fixée F de produit doit être transportée de la source au puits en minimisant le coût total de transport. Le problème se modélise comme suit avec des variables Φ_{ij} pour le flux sur chaque arc (i, j). La fonction-objectif (1) cumule les coûts de traversée du produit sur les différents arcs. La contrainte (2) impose un débit total de F à la source. Les fameuses lois de Kirchhoff qui expriment la conservation du flot traversant chaque nœud interne sont traduites par les contraintes (3). Les contraintes (4) précisent que les flux sont entiers (si on considère des flots de véhicules ou de palettes, par exemple) et compatibles avec les capacités.

$$(1) \quad \text{Min} \sum_{i \in X} \sum_{j \text{ succ. de } i} w_{ij} \cdot \Phi_{ij}$$

$$(2) \quad \sum_{j \text{ succ. de s}} \Phi_{sj} = F$$

$$(3) \quad \forall i \neq s,t : \sum_{j \text{ préd. de } i} \Phi_{ji} = \sum_{j \text{ succ. de } i} \Phi_{ij}$$

$$(4) \quad \forall (i, j) \in U : \Phi_{ij} \leq C_{ij} \text{ et } \Phi_{ij} \in I\!N$$

Le *problème du flot maximal* consiste à maximiser le débit total du flot qui part de la source. Par rapport au modèle précédent, F devient une variable, et la fonction-objectif revient simplement à maximiser F. Dans le problème du flot de coût minimal, il est possible que le débit requis F ne puisse pas être atteint à cause des capacités. On peut alors résoudre un problème de flot maximal pour calculer le débit maximal permis F, puis calculer avec le modèle précédent un flot de coût minimal parmi tous les flots maximaux de débit F.

Les autres problèmes de flots ont plusieurs sources, chaque source s ayant une disponibilité a_s, et plusieurs puits, chaque puits t ayant une demande b_t qui doit être satisfaite. On peut se ramener aux cas précédents en convertissant disponibilités et demandes en capacités. Pour cela, une super-source σ est créée et reliée à chaque source s par un arc (σ, s) de capacité a_s. Un super-puits τ est également ajouté ; chaque puits t y est connecté par un arc (t,τ) de capacité b_t.

Si la somme des disponibilités A n'est pas égale à la somme des demandes B, une source ou un puits fictif est ajouté. Par exemple, si $A > B$, on crée un puits fictif p avec une demande $A - B$ et on relie chaque source s à p par un arc de capacité infinie et de coût nul. Après ces transformations sur un problème à sources ou puits multiples, on obtient un problème de flot entre la super-source et le super-puits.

Le *problème de transbordement* est un problème à sources et puits multiples dans lequel les capacités des arcs internes sont infinies. Les seules capacités finies sont celles des arcs traduisant les disponibilités et les demandes. Le *problème de transport* est un problème de transbordement sur un graphe biparti : il n'y a plus de nœuds internes, seuls subsistent un ensemble de sources, un ensemble de puits, et des arcs reliant une source à un puits.

Le *problème d'affectation* est un problème de transport dont les disponibilités et les demandes sont toutes égales à un. Un flux de 1 reliant une source *s* à un puits *t* peut ainsi être interprété comme l'affectation de *s* à *t*.

Le *problème du chemin de coût minimal* est un cas extrême de flot de coût minimal, avec une source unique de disponibilité 1 et un puits unique de demande 1. Le flot de débit 1 va traverser le réseau en suivant un chemin de coût total minimal ! Le tableau 2.1 récapitule la liste de ces problèmes de réseaux classiques, avec les exemples d'applications qui figurent dans le livre.

Tableau 2.1 – Grands types de PL de réseaux

Nom du problème classique	Application dans le livre	Paragraphe
Problème du chemin optimal	Non vu (programmation linéaire peu efficace)	
Problème d'affectation	Correspondance d'avions	10.2
	Affectation de personnes à des postes	13.2
Problème de transport	Production de sucre de canne	5.5
	Location de voitures	9.2
Problème du flot maximal	Adduction d'eau	14.2
Problème du flot de coût minimal	Choix de moyens de transport	9.3

2.3.4 Programmes linéaires MRP

Un PL est dit *MRP* si sa matrice *A* est entière, si chaque contrainte est une égalité, si chaque colonne de *A* a au plus un coefficient non négatif et si c'est un +1, et si les seconds membres (vecteur *b*) sont des entiers non négatifs [Jeroslow 1992]. On admettra que dans ce cas l'optimum est entier. De plus, si les coûts unitaires sont aussi entiers, alors l'optimum du dual est aussi entier. On rencontre ces PL en gestion de production, dans la planification des besoins en composants *(MRP* ou *Material Requirement Planning)*, où les contraintes décrivent la conservation des composants dans les arbres de nomenclature des produits à fabriquer. Le problème de fabrication de jouets du § 7.4 est de ce type.

2.3.5 Transformations en PL de réseau ou MRP

L'algorithme du simplexe trouve encore un optimum entier pour les programmes linéaires transformables en PL de réseau ou MRP par les opérations suivantes :

- multiplication d'une contrainte par une constante non nulle ;
- ajout d'un multiple d'une contrainte à une autre contrainte.

Ces opérations ne changent ni le domaine des solutions réalisables, ni l'ensemble des solutions optimales. Mettre en évidence la transformation suffit pour savoir que le PL initial a un optimum entier : il n'est pas nécessaire d'effectuer réellement la transformation. De même, si on a un PL avec des seconds membres entiers et que son dual est un PL de réseau ou un PL MRP, alors on sait que le PL primal (initial) a un optimum entier.

2.4 Méthode arborescente de Dakin

2.4.1 Exemple à traiter

On prend pour exemple le petit PLNE à deux variables suivant :

$$\text{Max } z = 4x_1 + 3x_2$$
$$3x_1 + 4x_2 \leq 12$$
$$4x_1 + 2x_2 \leq 9$$
$$x_1, x_2 \in \mathbb{N}$$

La taille réduite de deux variables permet la résolution géométrique de la figure 2.3. On trouve pour le PL relaxé l'optimum $x^* = (1.2, 2.1)$ avec un coût $z^* = 11.1$. Le meilleur point entier est en revanche (1, 2), de coût 10.

2.4.2 Principe de la méthode

Détaillons maintenant la méthode arborescente de Dakin. La racine de l'arbre S_0 correspond au PL relaxé. Son domaine de solutions réalisables inclut toutes celles du PLNE. On résout le PL relaxé avec l'algorithme du simplexe. Si, par hasard, le simplexe trouve une solution entière, on s'arrête : on a l'optimum du PLNE. Sinon, le coût maximal rendu par le simplexe donne une évaluation par excès $ev(S_0)$ du PLNE. On initialise alors le coût de la meilleure solution entière déjà trouvée, w, à $-\infty$.

Si le PL relaxé n'a pas de solution entière, on procède à la séparation : une variable non entière x^*_j est choisie dans la solution optimale du PL relaxé. Le plus souvent, on prend la variable dont la valeur est la plus proche d'un entier. Le nœud actuel est séparé en deux fils. L'un reprend le PL relaxé du nœud S_0, avec la contrainte supplémentaire $x_j \leq \text{Int}(x^*_j)$, Int désignant la partie entière. L'autre reprend également le PL relaxé de S_0, mais avec la contrainte additionnelle $x_j \geq \text{Int}(x^*_j) + 1$. Ces contraintes de bornes peuvent être vues comme des coupes simples qui réduisent peu à peu le polyèdre des solutions réalisables.

Les nœuds créés forment une arborescence. Aux itérations suivantes, on choisit le nœud-feuille S_i de plus grande évaluation (le plus prometteur). Il est évalué en résolvant son PL relaxé avec le simplexe. Si $ev(S_i) \leq w$, le nœud est supprimé. Sinon, si la solution est entière, on la conserve comme meilleure solution provisoire pour le PLNE, on met à jour w avec $ev(S_i)$, et on supprime le nœud.

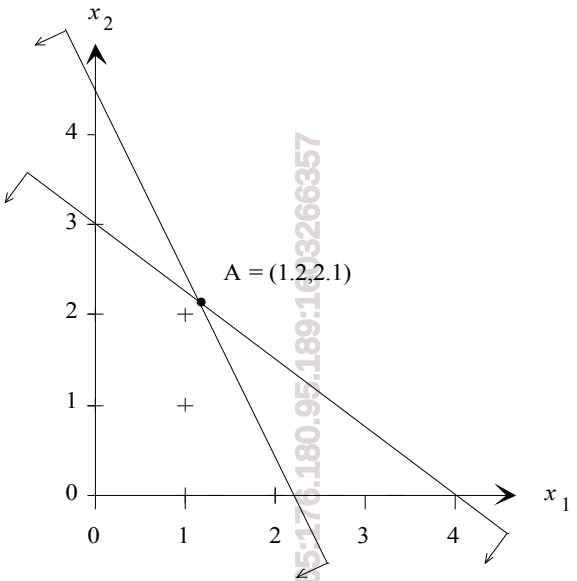

Figure 2.3 – Exemple pour la méthode de Dakin

Si la solution n'est pas entière, le nœud reste dans l'arborescence pour les itérations suivantes. Quand tous les nœuds en attente ont été supprimés, la recherche est terminée. La dernière solution entière provisoire trouvée, s'il y en a une, est la solution optimale du PLNE de départ.

2.4.3 Algorithme général

En ces termes littéraires, la méthode semble compliquée. Elle s'exprime plus simplement avec un algorithme général présenté page suivante.

2.4.4 Application à l'exemple

La figure 2.4 montre l'arborescence parcourue par la méthode de Dakin sur l'exemple. En S_0, aucune variable n'est entière, mais on sait que le coût maximal du PLNE est au plus 11.1. On sépare sur x_2, variable non entière la plus proche d'un entier : on construit deux fils, l'un S_1 avec la contrainte supplémentaire $x_2 \leq 2$, l'autre S_2 avec la contrainte $x_2 \geq 3$. Remarquez que l'union de ces deux cas correspond bien aux solutions du PLNE de départ. On résout immédiatement les PL de ces fils. S_1 a encore une variable fractionnaire x_1 et une évaluation de 11. S_2 fournit une première solution entière (0,3) de coût 9.

S_1 doit être développé, car d'après son évaluation il est possible qu'il fournisse des solutions meilleures, de coût 10 ou 11. On sépare donc S_1 en S_3 (PL de S_1 plus la contrainte $x_1 \leq 1$) et en S_4 (PL de S_1 plus $x_1 \geq 2$). S_3 fournit une meilleure solution entière (1,2) de coût 10. Pour S_4, le simplexe donne la solution (2, 0.5) avec un coût de 9,5 : ce nœud peut être supprimé car cette évaluation par excès des solutions entières est moins bonne que 10. La recherche est terminée, la dernière solution entière trouvée (dans S_3) est donc optimale.

Dans ce petit exemple, peu de nœuds sont construits et les solutions peuvent être calculées graphiquement. Pour un grand PL, la convergence vers l'optimum peut être très lente.

```
Initialiser une liste L avec un nœud S(0) contenant le PL relaxé du PLNE
Résoudre ce PL et garder l'optimum (variables et objectif) dans le nœud
Initialiser w, coût de la meilleure solution entière trouvée, à -∞
Répéter
   Chercher dans L le nœud S(i) d'évaluation maximale (S(0) au début)
   Si ev(S(i)) ≤ w alors
      Supprimer dans L le nœud S(i)
   Sinon
      Si la solution stockée dans S(i) est entière alors
         Conserver cette solution comme meilleure solution entière
         w := ev(S(i))
         Supprimer dans L le nœud S(i)
      Sinon
         Choisir une variable non entière x*(j)
         Créer un fils avec le PL de S(i) et la contrainte x(j)≤Int(x*(j))
         Résoudre le PL du nœud-fils et ranger la solution dans ce nœud
         Ajouter le nœud à L
         Créer l'autre fils avec la contrainte x(j)≥Int(x*(j))+1
         Résoudre le PL du nœud-fils, ranger la solution dans ce nœud
         Ajouter le nœud à L
      FinSi
   FinSi
Jusqu'à ce que L soit vide
Si w = -∞ alors
   Signaler que le PLNE est infaisable
Sinon
   Sortir la meilleure solution trouvée et son coût w
FinSi.
```

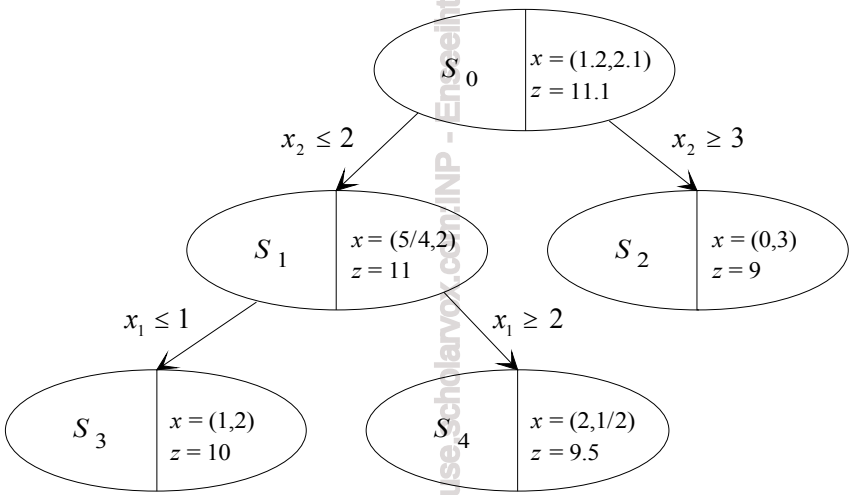

Figure 2.4 – Arborescence de la méthode de Dakin

2.5 Méthode de Balas pour les PL en 0-1

2.5.1 Introduction

La méthode nécessite un PL sous forme canonique, en minimisation et avec $c \geq 0$. On peut toujours se ramener à ce cas en remplaçant la variable x_j par $x'_j = 1 - x_j$ quand $c_j < 0$. Les logiciels commerciaux se chargent de la conversion dans le format approprié.

$$\text{Min } z = cx \quad (c \geq 0)$$
$$Ax \leq b$$
$$x \in \{0,1\}^n$$

La méthode de Balas est un bel exemple de méthode arborescente fixant progressivement à 0 ou à 1 des variables x_j. Comme toute méthode générale en PLNE ou PL-01, elle ne peut traiter que des problèmes pas trop grands ($n = 200$ par exemple), ce qui est loin des dizaines de milliers de variables traitables par le simplexe en PL continue. L'exposé de la méthode est assez compliqué, mais un exemple avec l'arborescence est donné à la fin.

2.5.2 Définition des nœuds et séparation

Un nœud (ou sommet) S_t de l'arborescence correspond à une *solution partielle*, c'est-à-dire qu'un sous-ensemble $F(t)$ de variables a été *fixé* à 0 ou 1, les variables restantes formant l'ensemble $L(t)$ des variables *libres* (non encore fixées). On distingue dans $F(t)$ les variables fixées à 0, $F_0(t)$ et celles fixées à 1, $F_1(t)$. On choisit une variable x_j de $L(t)$ et on la fixe à 0 ou bien à 1, ce qui donne deux fils.

2.5.3 Évaluation des nœuds

En tout nœud S_t nouvellement créé, le PL en 0-1 peut se réécrire comme suit. Dans ce modèle, on constate que les variables fixées à 0 ont été supprimées. En particulier, une contrainte peut aussi disparaître si tous les x_j correspondant à ses a_{ij} non nuls sont fixés à 0. Les variables fixées à 1 font apparaître un terme constant positif (car $c \geq 0$) dans la fonction économique et donnent à chaque contrainte un nouveau second membre s_i.

$$\text{Min } z_t = \sum_{j \in L(t)} c_j . x_j + \sum_{j \in F_1(t)} c_j$$
$$\forall i = 1 \ldots m : \sum_{j \in L(t)} a_{ij} . x_j \leq b_i - \sum_{j \in F_1(t)} a_{ij} = s_i$$
$$\forall j = 1 \ldots L(t) : x_j \in \{0,1\}$$

Rappelons que dans toute recherche arborescente (en minimisation), une *évaluation* d'un nœud doit être une estimation par défaut des coûts des solutions que peut contenir ce nœud. Ici, il est clair qu'une évaluation $ev(S_t)$ du sommet S_t est obtenue immédiatement en fixant x_j à 0 pour tout j de $L(t)$, puisque $c \geq 0$:

$$ev(S_t) = \sum_{j \in F_1(t)} c_j$$

2.5.4 Abandon de l'examen d'un nœud

Après avoir créé un sommet S_t on l'évalue puis on effectue *dans l'ordre* les tests suivants :

Pas de meilleure solution réalisable

Comme dans toute recherche arborescente (en minimisation), on laisse tomber S_t si $ev(S_t)$ est supérieure ou égale à z_{prov}, valeur de la meilleure solution provisoire trouvée jusqu'à présent. En effet, ce nœud ne peut pas fournir de meilleure solution.

Nouvelle meilleure solution

x^t, solution obtenue en posant $x_j = 0$ pour tout j de $L(t)$, est une solution réalisable du programme linéaire courant si $s_i \geq 0$ pour tout i. S_t est alors un nœud terminal (ou *feuille*) de l'arborescence et $ev(S_t) = \text{Min } z_t$: l'évaluation est *exacte* en ce sommet et on a trouvé une nouvelle meilleure solution, qui remplace la précédente.

Sommet sans solutions réalisables

Par exemple, soit la contrainte $3.x_1 - 4.x_2 + 3.x_3 - 5.x_4 \leq -10$. La partie gauche est minimisée en mettant à 0 les variables à coefficients > 0, à 1 celles à coefficients < 0. On obtient alors -9, et la contrainte ne peut être satisfaite. Par conséquent, S_t peut être écarté pour absence de solutions si :

$$\exists i \in \{1,2,\ldots m\} : t_i = \sum_{j \in L(t)} \text{Min}(0, a_{ij}) > s_i$$

La somme des minima t_i est tout simplement la somme des a_{ij} négatifs. Ce test ne concerne que les contraintes telles que $s_i < 0$.

2.5.5 Implications

Pour réduire encore le nombre de nœuds examinés, de nombreux auteurs ont proposé des tests permettant de déduire certaines variables. Citons celui de Geoffrion [Salkin 1975]. Soit par exemple la contrainte $-3.x_1 + 2.x_2 - 5.x_3 + 6.x_4 \leq -4$. La somme des coefficients négatifs (-8) étant inférieure à -4, le nœud n'a pas été éliminé par le test précédent. Pour que cette contrainte soit vérifiée, il faut au moins que $x_3 = 1$ et $x_4 = 0$, car si $x_3 = 0$ ou $x_4 = 1$, il n'y a plus de solution d'après le dernier test du § 2.5.4. Donc, si la somme des a_{ij} négatifs, sauf un certain a_{ip}, excède s_i, x_p doit être mis à 1. Si la somme des a_{ij} négatifs, plus un certain $a_{ip} > 0$, excède s_i, alors x_p doit être mis à 0. D'où le test sous forme mathématique :

$$\text{Si } \exists i \in 1\ldots m, \exists p \in L(t), \sum_{j \in L(t)} Min(0, a_{ij}) + \left| a_{ip} \right| > s_i \text{ alors} \begin{cases} x_p = 0 \text{ si } a_{ip} > 0 \\ x_p = 1 \text{ si } a_{ip} < 0 \end{cases}$$

Ce test se mène rapidement en calculant d'abord la somme des $a_{ij} < 0$, puis en conduisant le test pour chaque a_{ip} lors d'une deuxième passe. Plusieurs variables peuvent être forcées dans chaque contrainte. Il est recommandé de repasser sur toutes les contraintes tant qu'on trouve au moins une variable à forcer dans au moins une d'entre elles.

2.5.6 Parcours de l'arborescence

Il s'agit d'une méthode dite *en profondeur d'abord* : on développe toujours le dernier nœud créé dans l'arborescence. Cette technique est ici la plus adaptée pour trouver rapidement une première solution réalisable. En séparant un nœud sur une variable x_j, on traite d'abord le fils "$x_j = 1$". En effet, si le nœud séparé n'est pas terminal, c'est qu'il faut fixer à 1 au moins une variable pour améliorer la meilleure solution provisoire.

Geoffrion a proposé de stocker en tout nœud l'indice j de la variable fixée, sous la forme $+j$ si x_j a été fixée à 1, $-j$ sinon. Lors d'un retour en arrière dans l'arborescence, un indice négatif indique un fils droit déjà traité ($x_j = 0$) et on peut encore remonter. En revanche, si on trouve un indicateur positif, on le transforme en son opposé et on traite le fils droit.

2.5.7 Choix de la décision de séparation en un nœud

Comme dans toute méthode arborescente, après le choix du prochain sommet S_t à séparer, il faut préciser la décision de séparation à appliquer. Il est certain que fixer une variable libre quelconque à 0 ou 1 convient, mais Balas a proposé une heuristique de choix qui diminue significativement le nombre moyen de nœuds générés par sa méthode.

Soit $Q(t) = \{i \mid s_i < 0\}$ l'ensemble des lignes du PL courant ayant un second membre négatif. Cet ensemble est non vide, sinon x^t serait une solution réalisable et l'exploration de S_t aurait été abandonnée. Notons $R(t)$ l'ensemble des indices de variables libres ayant un coefficient négatif sur au moins une des lignes de $Q(t)$:

$$R(t) = \left\{ j \in L(t), \exists i \in Q(t), a_{ij} < 0 \right\}$$

Par exemple, si les contraintes du programme linéaire courant sont :

$$\begin{array}{lrcrcrcr}
(1) & -7x_1 & + & 2x_2 & + & 4x_3 & \leq & 3 \\
(2) & 2x_1 & - & 3x_2 & - & 5x_3 & \leq & -2 \\
(3) & 8x_1 & - & x_2 & - & 4x_3 & \leq & -3
\end{array}$$

Alors $Q(t) = \{2{,}3\}$ et $R(t) = \{2{,}3\}$. Pour satisfaire (2) et (3), il faut qu'au moins une des variables de $R(t)$ soit fixée à 1. Pour obtenir une solution réalisable, il faut donc choisir dans $R(t)$ une variable et la fixer à 1. Autant donc choisir une de ces variables pour séparer. Constatant qu'on a une solution réalisable quand tous les seconds membres s_i sont non négatifs, Balas a proposé une mesure empirique de la *proximité d'une solution*, $P(t) < 0$:

$$P(t) = \sum_{i \in Q(t)} s_i = \sum_{i=1,m} Min(0, s_i)$$

$P(t)$ est donc la somme des seconds membres négatifs. Elle dépend du nombre de ces seconds membres et de leurs valeurs : à nombre égal de $s_i < 0$, la présence de grands s_i (en valeur absolue) va probablement demander plus de fixations de variables à 1, et donc plus de nœuds à examiner. La proximité d'une solution si on fixe x_j à 1 est alors :

$$P(t, j) = \sum_{i=1,m} Min(0, s_i - a_{ij})$$

Le terme s_i - a_{ij} est la nouvelle valeur du second membre après forçage de x_j à 1. Balas propose de séparer sur la variable j^* qui augmente le plus la proximité d'une solution si on la met à 1, c'est-à-dire la plus prometteuse pour trouver une nouvelle solution rapidement :

$$P(t, j^*) = \underset{j \in R(t)}{Max} \{P(t, j)\}$$

Dans l'exemple précédent, la séparation se fera ainsi sur x_3, car :

$R(t) = \{2,3\}$

$P(t,2) = \min(0,3-2) + \min(0,-2+3) + \min(0,-3+1) = -2$

$P(t,3) = \min(0,3-4) + \min(0,-2+5) + \min(0,-3+4) = -1$

Remarques

- Si $P(t,j^*) = 0$, la solution obtenue en posant $x_j^* = 1$ et $\forall\, j \in L(t) \setminus \{j^*\}$, $x_j = 0$ est une nouvelle solution réalisable, de coût $ev(S_t) + c_j^*$.
- Pour départager des *ex aequo* ayant la même mesure de proximité, on choisit la variable de plus petit coût.

2.5.8 Un exemple

L'exemple suivant va permettre de démystifier la méthode, qui n'est pas si compliquée. Le lecteur va pouvoir vérifier l'arborescence développée grâce à la figure 2.5.

$$\begin{array}{lrrrrrcr} \text{Min} & 5x_1 & +7x_2 & +10x_3 & +3x_4 & +x_5 & & \\ & -x_1 & +3x_2 & -5x_3 & -x_4 & +4x_5 & \leq & -2 \\ & 2x_1 & -6x_2 & +3x_3 & +2x_4 & -2x_5 & \leq & 0 \\ & & x_2 & -2x_3 & +x_4 & +x_5 & \leq & -1 \end{array}$$

$x \in \{0,1\}^5$

Traitement du nœud-racine S_0

Au nœud-racine S_0, on a $L(0) = 1\ldots5$, $Q(0) = \{1,3\}$ et $R(0) = \{1,3,4\}$. On a l'évaluation triviale $ev(S_0) = 0$! Le tableau 2.2 donne une présentation bien commode pour visualiser les PL résiduels et calculer les proximités.

Tableau 2.2 – Détails du traitement du nœud-racine S_0

Coefficients et seconds membres du PL courant						Second membre s_i - a_{ij} si		
x_1	x_2	x_3	x_4	x_5	s_0	$x_1 = 1$	$x_3 = 1$	$x_4 = 1$
-1	3	-5	-1	4	-2	-1	3	-1
2	-6	3	2	-2	0	-2	-3	-2
0	1	-2	1	1	-1	-1	1	-2
					$P(0) \rightarrow$	-4	-3	-5

Les colonnes de droite donnent les nouvelles valeurs des seconds membres si on met à 1 la variable correspondante x_j de $R(t)$. La proximité $P(t,j)$ est alors la somme des termes *négatifs* de la colonne j. Ici, on sépare sur $x_3 = 1$ ou 0, ce qui donne d'abord le nœud S_1 pour $x_3 = 1$. Le nœud S_4 avec $x_3 = 0$ sera construit plus tard.

Traitement du nœud S_1

On a $x_3 = 1$, $L(1) = \{1,2,4,5\}$, $F_1(1) = \{3\}$, $Q(1) = \{2\}$, $R(1) = \{2,5\}$, $ev(S_1) = 10$. Le tableau 2.3 résume la situation avec les calculs des proximités. On sépare sur x_2. Le nœud S_2 obtenu pour $x_2 = 1$ est en fait une première solution réalisable (*cf.* remarques à la fin du § 2.5.7). Cette solution est $x = (0,1,1,0,0)$, de coût 17. C'est la meilleure solution provisoire.

Tableau 2.3 – Détails du traitement du premier fils S_1 de S_0

Coefficients et seconds membres du PL courant					s_i - a_{ij} si	
x_1	x_2	x_4	x_5	s_1	$x_2 = 1$	$x_5 = 1$
-1	3	-1	4	3	0	-1
2	-6	2	-2	-3	3	-1
0	1	1	1	1	0	0
				$P(1) \rightarrow$	0	-2

Traitement du nœud S_3

Il reste à traiter le fils droit S_3 de S_1, correspondant à $x_3 = 1$ et $x_2 = 0$. On a $L_3 = \{1,4,5\}$, $F_1(3) = \{3\}$, $F_0(3) = \{2\}$, $Q(3) = \{2\}$, $R(3) = \{5\}$. En construisant le tableau suivant, on s'aperçoit que dans la deuxième contrainte, t_2 (somme des $a_{2j} < 0$, *cf* 2.5.4) = -2 > -3. On laisse donc tomber le nœud S_3 car il n'a pas de solution réalisable.

Tableau 2.4 – Traitement du fils droit S_3 de S_1

x_1	x_4	x_5	s_3
-1	-1	4	3
2	2	-2	-3
0	1	1	1

Traitement de S_4

On remonte sur le nœud-racine S_0, puis on construit le nœud S_4 correspondant à $x_3 = 0$. L'ensemble des variables libres est alors $L(4) = \{1,2,4,5\}$. Le tableau suivant montre que la somme des coefficients négatifs dans la troisième contrainte est $t_3 = 0 > -1$; il n'y a pas de solution. C'est la fin de la recherche arborescente.

Tableau 2.5 – Traitement du fils droit S_4 du nœud-racine S_0

x_1	x_2	x_4	x_5	s_4
-1	3	-1	4	-2
2	-6	2	-2	0
0	1	1	1	-1

Conclusion

La solution réalisable découverte en premier (la seule !) est donc optimale. Remarquez qu'une seule solution complète (vecteur binaire) a été construite, par rapport à une énumération complète qui aurait dû inspecter $2^5 = 32$ vecteurs-solutions. Le gain par rapport à une énumération exhaustive est modeste sur ce petit exemple, il peut être énorme pour un PL plus volumineux.

La figure 2.5 résume l'arborescence développée. Les valeurs dans les nœuds, à droite, sont les évaluations. Dans des cas plus compliqués, le nombre de nœuds pourrait être beaucoup plus grand, et des élagages se produiraient grâce à l'évaluation. Supposons par exemple qu'en S_3 il y ait des solutions réalisables, mais avec une évaluation de 18 : on laisserait tomber ce nœud car on disposerait déjà d'une solution de coût 17.

Plusieurs solutions provisoires, de coût décroissant, peuvent être découvertes pendant l'exploration de l'arbre. Cette propriété est intéressante quand la recherche est trop longue : on peut stopper l'algorithme et se contenter de la dernière solution trouvée, même si elle n'est pas optimale. Sur un PL en 0-1 très contraint, il peut aussi arriver de terminer la recherche sans avoir trouvé une seule solution : ceci prouve que le PL est irréalisable.

Les logiciels modernes incluent de nombreux tests permettant de forcer des variables à 0 ou 1, comme celui de Geoffrion. Ces tests sont si efficaces que la plupart des petits PL peuvent être résolus à la racine, sans aucune séparation. Sur de grand PL, les tests permettent typiquement de diviser par deux le nombre de variables et le nombre de contraintes.

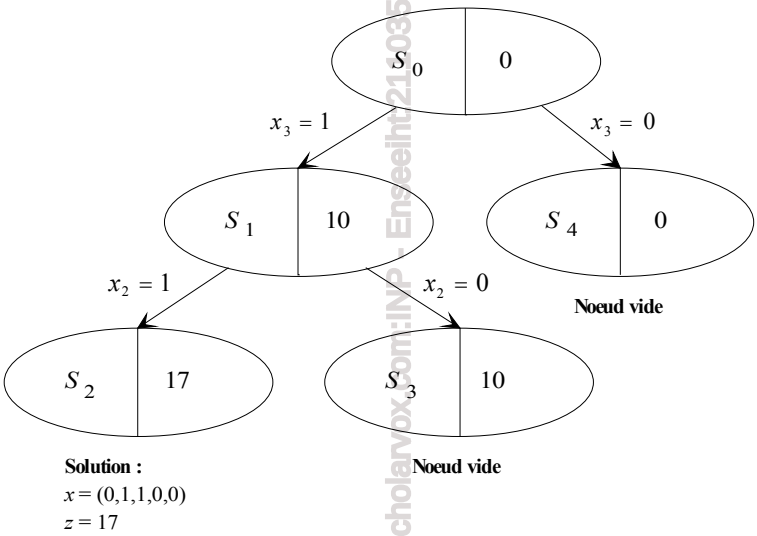

Figure 2.5 – Arborescence de la méthode de Balas

2.6 Techniques de modélisation en PLNE

2.6.1 Généralités

La PLNE permet de modéliser de nombreuses contraintes complexes, intraitables sans variables entières. Les techniques utilisées sont très astucieuses et reposent souvent sur des variables binaires. Les paragraphes suivants donnent quelques exemples de contraintes fréquentes. Les problèmes d'application du livre en contiennent beaucoup d'autres.

2.6.2 Contraintes de sac à dos

Elles sont utilisées pour exprimer des contraintes de capacité. Soit par exemple un camion avec une charge utile de b tonnes et n colis de poids a_i, avec un poids total dépassant b. Les sous-ensembles de colis qu'on peut charger peuvent être définis par des variables binaires x_i valant 1 si et seulement on prend le colis i, et par la contrainte suivante :

$$\sum_{i=1,n} a_i.x_i \le b$$

2.6.3 Contraintes d'affectation

Soit n objets à affecter à n places. Chaque objet i doit être affecté à une place j et chaque place j doit recevoir un objet i. Si $x_i \in \{1, 2, \ldots, n\}$ est le n° de place où est affecté l'objet i, les x_i doivent être tous différents. Il est impossible d'exprimer cette contrainte sous forme linéaire. La solution est d'utiliser des variables binaires x_{ij} qui valent 1 si et seulement si l'objet i est affecté à la place j :

$$\forall i : \sum_{j=1,n} x_{ij} = 1 \;\; et \;\; \forall j : \sum_{i=1,n} x_{ij} = 1$$

2.6.4 Variables discrètes générales

En modélisation, il arrive souvent qu'une variable x ne puisse prendre qu'un ensemble fini de k valeurs $\{v_1,v_2,\ldots,v_k\}$. Cette situation est traitable en introduisant k variables booléennes y_i, $i = 1\ldots k$. Il suffit ensuite d'effectuer le changement de variable :

$$x = \sum_{i=1,k} v_i.y_i$$

Et de poser la contrainte additionnelle :

$$\sum_{i=1,k} y_i = 1 .$$

Cette technique permet en théorie de transformer tout PLNE en PL-01, à condition que le domaine $P = \{x \in \mathbb{R}^{n+}, A.x = b\}$ soit borné (c'est alors un polyèdre convexe). Cette condition permet d'associer à toute variable x_j des bornes de variation : $\alpha_j \le x_j \le \beta_j$.

Ces bornes peuvent se calculer par programmation linéaire. Pour calculer β_j par exemple, il suffit de résoudre le programme linéaire suivant :

$$\begin{aligned} &\text{Max } x_j \\ &A \cdot x = b \\ &x \geq 0 \end{aligned}$$

Ainsi, on peut se ramener au cas où toute variable x_j ne prend qu'un ensemble fini de valeurs. Si une variable x_j varie entre 0 et k, elle peut par exemple être remplacée par son expression en base 2 : $x_j = y_0 + 2.y_1 + ... + 2^p.y_p$, p étant le plus petit entier tel que $k \leq 2^{p+1}$. L'avantage de la base 2 est d'introduire de simples variables binaires. Bien entendu, la transformation n'est rentable que si les plages de variation des variables sont assez petites.

2.6.5 Variables nulles ou bien bornées inférieurement

Soit par exemple un produit j qui n'est pas fabriqué, ou bien fabriqué en quantité au moins égale à Q_j, par exemple pour des questions de rentabilité : on a $x_j = 0$ ou bien $x_j \geq Q_j$. Soit une variable-indicateur $y_j \in \{0,1\}$, valant 1 si le produit est fabriqué. La contrainte peut alors s'exprimer comme suit, M désignant un entier assez grand. Si $y_j = 0$, alors $x_j = 0$, et si $y_j = 1$, alors $x_j \geq Q_j$. N'importe quelle borne supérieure de x_j convient pour M, mais, pour éviter des problèmes numériques, il vaut mieux prendre le plus petit M qui convient.

$$\begin{cases} x_j \geq Q_j.y_j \\ x_j \leq M.y_j \\ y_j \in \{0,1\} \end{cases}$$

Souvent, un des sens de l'équivalence est assuré par l'optimisation de la fonction-objectif, ce qui permet d'économiser une contrainte. Par exemple, un coût fixe c_j peut être induit par la décision de fabriquer le produit. Un terme $c_j.y_j$ va donc apparaître dans la fonction-objectif si le but est de minimiser le coût total. La contrainte $x_j \leq M.y_j$ peut alors être supprimée : il sera possible d'avoir $x_j = 0$ et $y_j = 1$, mais la minimisation du coût va réduire y_j à 0 dans toute solution optimale.

Cette technique d'indicateur peut aussi servir pour traduire des implications. Soit par exemple une variable booléenne y qui doit valoir 1 si des variables x_1, x_2 ou x_3 sont non nulles. Elle sera positionnée correctement par la contrainte : $x_1 + x_2 + x_3 \leq M.y_i$.

2.6.6 Contraintes disjonctives en ordonnancement

En ordonnancement, notamment dans les emplois du temps, il est fréquent que deux tâches i et j ne puissent pas s'exécuter en même temps. Ces deux tâches, de durées p_i et p_j, sont à ordonnancer à des instants t_i et t_j à déterminer. Il n'y aura pas de chevauchement si i se termine avant le début de j, ou *vice versa*, c'est-à-dire : $t_i + p_i \leq t_j$ ou $t_j + p_j \leq t_i$. On introduit une variable binaire y_{ij} valant 1 si et seulement si i est avant j, puis les deux contraintes suivantes dans lesquelles M est une constante positive assez grande. Par exemple, si $y_{ij} = 1$, la première contrainte devient active et la seconde est redondante car trivialement vérifiée. Ce genre de contraintes est utilisé au chapitre 6 dans le problème du § 6.4.

$$\begin{cases} t_i + p_i \le t_j + M.(1 - y_{ij}) \\ t_j + p_j \le t_i + M.y_{ij} \end{cases}$$

2.6.7 Respect d'un sous-ensemble de contraintes

Parfois, il est impossible de satisfaire toutes les contraintes d'un programme linéaire, par exemple quand il s'agit de simples préférences dans la confection d'un emploi du temps. Une solution sera acceptable si elle vérifie au moins k contraintes parmi les m contraintes du PLNE. Pour un PLNE sous forme canonique, ces m contraintes sont de la forme :

$$\forall i = 1 \ldots m : \sum_{j=1,n} a_{ij}.x_j \le b_i$$

Elles sont remplacées par les contraintes suivantes, dans lesquelles m - k variables binaires y_i vont être égales à 1. Une contrainte de la première ligne avec $y_i = 1$ pourra être violée, et k contraintes avec $y_i = 0$ devront être respectées.

$$\begin{cases} \forall i = 1 \ldots m : \sum_{j=1,n} a_{ij}.x_j \le b_i + M.y_i \\ \sum_{i=1,m} y_i = m - k \\ y \in \{0,1\}^m \end{cases}$$

2.6.8 Expressions logiques

Les variables binaires permettent de traduire simplement des expressions logiques. Il suffit d'associer une variable binaire à chaque proposition, en convenant que la variable vaut 1 si et seulement si la proposition est vraie. Le tableau 2.6 montre comment les opérations booléennes classiques peuvent alors se traduire par des contraintes linéaires.

Tableau 2.6 – Traduction d'expressions logiques sous forme linéaire

Expression logique	Traduction sous forme de contrainte
non *P*	$p = 0$ (ou $1 - p = 1$)
P ou *Q* (ou inclusif)	$p + q \ge 1$
P ou bien *Q* (ou exclusif)	$p + q = 1$
P et *Q*	$p = 1$ et $q = 1$ (ou $p + q = 2$)
$P \Rightarrow Q$ (*P* implique *Q*)	$p \le q$
$P \Leftrightarrow Q$ (*P* équivalent à *Q*)	$p = q$

2.6.9 Valeurs absolues

La fonction valeur absolue $|x|$ est non seulement non linéaire, mais elle est aussi non dérivable, avec un point anguleux en $x = 0$. On peut pourtant linéariser les contraintes contenant des valeurs absolues. Une contrainte de type $|x| \le k$, avec une expression linéaire x et une constante positive k est évidemment équivalente à $-k \le x \le k$.