Université Pierre et Marie Curie

Optimisation Combinatoire : Programmation Linéaire et Algorithmes

Pierre Fouilhoux
pierre.fouilhoux@lip6.fr

29 septembre 2015

Avant-propos

Table des matières

Première partie

Modéliser

Chapitre 1

Programmation mathématique

1.1 Définitions

Un *Programme Mathématique* (mathematical program), noté PM, est un problème d'optimisation sous contrainte ($\mathcal{P}$) qui peut s'écrire de la façon suivante :

$$\begin{array}{ll} \text{Maximiser } f(x) & \\ \quad \text{sous les contraintes} & \\ \quad g_i(x) \leq 0 & i = 1, ..., m \\ \quad x \in S. & \end{array}$$

où
• S est une partie de $\mathbb{R}^n$ et x est un vecteur appelé *variable*, ces n composantes sont dites les *inconnues* du problème,
• la fonction $f : S \to \mathbb{R}$ est appelée *fonction objective* ou *objectif* (objective function),
• les fonctions $g_i : S \to \mathbb{R}$, $i = 1, ..., m$, forment des inégalités qui sont appelées les *contraintes* (constraint) du problème.

On peut remarquer qu'un PM peut être une maximisation ou une minimisation (il suffit de poser la fonction $f' = -f$.
On appelle *inégalités* une contrainte $g_i(x) \leq 0$ ou $g_i(x) \geq 0$: en cas de présences des deux contraints $g_i(x) \leq 0$ et $g_i(x) \geq 0$, on parle alors d'égalité $g_i(x) = 0$.

Un vecteur $\bar{x}$ vérifiant les contraintes d'un PM est dit *solution* ou *solution réalisable* du PM. L'ensemble des solutions d'un PM forme un *domaine de définition*. Le domaine de définition d'un PM peut être : vide (dans ce cas, le problème n'admet pas de solutions), dans le cas contraire, le PM admet des solutions. Sous certaines conditions, il peut exister des solutons x^* dites *optimales*, c'est-à-dire qui maximisent la fonction

$f(x)$ sur toutes les solutions du PM.

Plusieurs cas de PM sont à mettre en évidence :
- si l'ensemble S est continu, on parle de *programme mathématique continu* (continuous) ,
- si l'ensemble S est discret (c'est-à-dire isomorphe à $\mathbb{N}^n$), on parle de *programme mathématique discret* (discrete) que l'on notera ici (PMD) ; on le dit également *entier* (integer program) si $S \subset \mathbb{N}^n$ ou même *binaire* si $S \subset \{0,1\}^n$. En fait tout PMD peut se ramener au cas d'un programme entier, voir même d'un programme binaire.
- si certaines composantes du vecteur x solution prennent leurs valeurs dans un ensemble discret et les autres dans un ensemble continu, on le dit *programme mathématique mixte* (Mixed Integer Program ou MIP).

Dans le cas des programmes entiers (donc discrets également), on peut noter alors un (PMD) de la façon suivante :

$$\begin{array}{ll} \text{Maximiser } f(x) & \\ \quad \text{sous les contraintes} & \\ \quad g_i(x) \leq 0 & i = 1, ..., m \\ \quad x \in \mathbb{Z}^n. & \end{array}$$

On désigne alors $x \in \mathbb{Z}^n$ comme étant la *contrainte d'intégrité* (ou d'entiéreté en Belgique ou d'intégralité au Québec) (integrity or integrality constraint).

On appelle *relaxation* le fait de "relâcher", c'est-à-dire supprimer une contrainte du problème. Ainsi, un programme relaxé désignera un programme où l'on aura supprimé une ou plusieurs contraintes.
On appelle *relaxation continue* le fait de "relâcher" les contraintes d'intégrité du problème. Par abus de langage, on appelle aussi souvent *relaxation continue* le fait de résoudre le programme que où l'on a relâché les contraintes d'intégrité (l'expression désigne même parfois la solution optimale obtenue).

1.2 Difficulté de la PMD

On appelle *programmation mathématique discrète* l'étude des programmes mathématiques discrets. Il s'agit d'un domaine plutôt récent qui n'est pas constitué d'une seule approche théorique. Il existe beaucoup de cas particuliers, de cadres théoriques ou expérimentaux qui ont apporté des réussites parfois surprenantes : c'est le cadre de l'approche polyédrale pour le problème du voyageur de commerce qui a permis de résoudre à ce jour les plus grandes instances de ce problème-référence.

En fait, on sait parfois bien résoudre des PMD particuliers, mais on ne sait pas résoudre un PMD en général. Nous verrons ici certains cas que l'on sait résoudre efficacement. et, dans les cas plus difficiles, comment se ramener à ces cas simples.

La première remarque qui peut sauter aux yeux est d'imaginer que résoudre un (PMD) revient à "arrondir" la solution de sa relaxation continue. L'exemple suivant témoigne de l'insuffisance de cette remarque :

Prenons un PMD à deux variables et une seule contrainte où toutes les fonctions sont linéaires, ce qui constitue le cas le plus simple que l'on puisse imaginer.

$$\begin{aligned} &\text{Maximiser } 10x_1 + 11x_2 \\ &10x_1 + 12x_2 \leq 59 \\ &x_1 \text{ et } x_2 \geq 0 \\ &x_1, x_2 \text{ entiers.} \end{aligned}$$

En dessinant le domaine de définition, on obtient la figure 1.1 suivante : On remarque

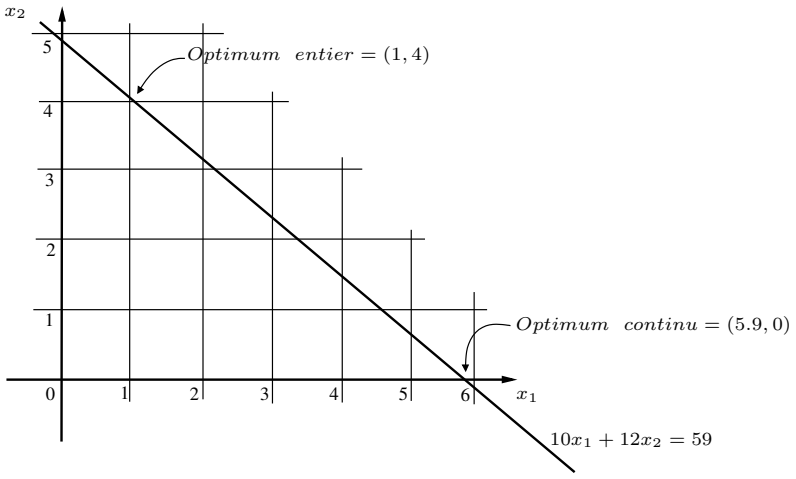


FIGURE 1.1 Ecart entre optimum entier et continu

alors que l'optimum de la relaxation continue a une valeur objective de 59 et celui de l'optimum entier est de 54 seulement. Mais surtout, on peut noter l'écart complet de structure et de position des deux points optimum (qui sont ici chacun solution optimale unique).

En fait, résoudre un (PMD) donné est au moins aussi difficile que résoudre sa relaxation continue. En général, il l'est bien davantage : certains problèmes, comme c'est le cas lorsque toutes les fonctions sont linéaires, ont des relaxations continues polynomiales et des versions discrètes NP-difficiles. Inversement, si l'on ne sait pas résoudre

efficacement la version continue d'un programme, il est quasi impensable de résoudre sa version discrète.

1.3 Programme linéaire

Si la fonction objective f et les fonctions contraintes g sont toutes linéaires : on parle de *Programme linéaire discret* (ou entier ou mixte) (on notera PLNE). Ce cas étant largement le plus utilisé et le plus étudié, on l'appelle parfois même simplement *Programme entier (ou mixte)* (Integer Program ou Mixed Integer Program) (MIP).

$$\begin{aligned} &\text{Maximiser } c_1^T x_1 + c_2^T x_2 \\ &\quad \text{sous les contraintes} \\ &\quad A_1 x_1 + A_2 x_2 \leq b \\ &\quad x_1 \in \mathbb{R}^{n_1} \\ &\quad x_2 \in \mathbb{Z}^{n_2}. \end{aligned}$$

où $= c_1, c_2$ sont des vecteurs et A_1 et A_2 des matrices avec x_1 partie continue de la solution et x_2 partie entière de la solution.

Un problème linéaire continu peut être résolu en temps polynomial (Khachiyan 1979). Il existe des algorithmes polynomiaux efficaces pour résoudre un programme linéaire comme ceux dits de *points intérieurs* initiés par Karmarkar (1984). Néanmoins l'algorithme du *simplexe* (Dantzig 1947) est le plus célèbre (et le plus efficace dans le cas général) des algorithmes de résolution, bien qu'il ne soit pas polynomial !

L'algorithme du simplexe repose sur le fait qu'une solution optimale d'un programme linéaire peut être prise parmi les sommets du polyèdre de $\mathbb{R}^n$ déterminé par $Ax \leq b$.

En revanche, la PLNE est un problème NP-difficile. Il est facile de montrer que la PLNE est un problème NP-difficile car de nombreux problèmes NP-difficiles peuvent être exprimés comme des PLNE.

Il existe de nombreux solveurs de PL : des solveurs commerciaux Cplex (IBM), Xpress, Gurobi (microsoft), et même Matlab ou Excel... ; des solveurs académiques Lp de COIN-OR, Soplex de la ZIB ; et des solveurs libres comme Glpk (gnu). Les meilleurs d'entre eux peuvent résoudre des PL jusqu'à 200000 variables et 200000 contraintes en quelques secondes.
En revanche, les solveurs entiers performants sont beaucoup moins performants : ils sont en général liés aux solveurs PL : Glpk par exemple ne dépassent pas quelques 100 aine de variables et contraintes ; les solveurs commerciaux Cplex ou Gurobi sont les plus performants (Xpress est un peu en-dessous) pouvant réussir parfois quelques milliers de variables/contraintes ; un solveur "universitaire" les rattrape : SCIP de la

ZIB. Un des objectifs de ce cours est de comprendre comment et dans quels cas ces solveurs atteignent de telles capacités.

1.4 Programme non-linéaire

- Programme convexe

Soit un ensemble $S \subset \mathbb{R}^n$ convexe (un ensemble de points tels que tout segment entre deux points est entièrment dans l'ensemble convexe). Une fonction $f : S \to \mathbb{R}$ est dite *fonction convexe* si elle vérifie

$$\forall x_1, x_2 \in S, \forall \lambda \in [0,1], \quad f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2).$$

Une fonction est dite *concave* si la fonction $-f$ est convexe.

Considérons alors le (PM) suivant

$$\begin{array}{l} \text{Minimiser } f(x) \\ x \in S. \end{array}$$

Si la fonction objective f d'un (PMD) est convexe (ou concave) et que l'ensemble S est un convexe fermé non-discret, on parle de *programme convexe*. Pour un programme convexe, tout optimum local est global. Les cas non-convexe (ou non-concave pour une maximisation), n'ont pas toujours d'optimum global.

Pour le cas des programmes convexes, suivant les propriétés de la fonction f : (continuité, différentiabilité,...), il existe des algorithmes plus ou moins efficace pour déterminer le minimum d'une fonction (méthode de Newton, méthode de sous-gradient,...).

Considérons à présent le (PM) suivant

$$\begin{array}{ll} \text{Minimiser } f(x) & \\ \quad \text{sous les contraintes} & \\ \quad g_i(x) \leq 0 & i = 1, ..., m \\ \quad x \in S. & \end{array}$$

Si f est convexe et S convexe non discret, on parle de *programme convexe avec contrainte*. On sait déterminer par les conditions de Karush-Kuhn-Tucker dans quel cas ce programme possède ou non un optimum. Il existe également des dérivés des méthodes citées précédemment pour résoudre le problème.

La résolution de ces programmes est souvent appelée *Programmation non-linéaire* et constitue un domaine de recherche à part entière. Il existe des solveurs continues efficaces capables de résoudre un grand nombre de cas de ces programmes. Citons par exemple le freeware SolvOpt http ://www.kfunigraz.ac.at/imawww/kuntsevich/solvopt.

On appelle aujourd'hui *programmation non-linéaire discrète* l'étude générale des programmes convexes sont contraintes le plus souvent linéaire. Il s'agit d'un domaine de recherche plutôt récent que nous évoquerons ici que succinctement.

• Programme quadratique

Si la fonction f est quadratique et les fonctions g sont linéaires : on parle de *programme quadratique discret* (PQD). Si f est quadratique et convexe et les fonctions g linéaires, on dit que le (PMD) est un *programme quadratique convexe discret*. On peut l'écrire sous la forme :

$$\begin{aligned} &\text{Maximiser } x^T Qx + Lx = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j + \sum_{i=1}^{n} l_i x_i \\ &\quad \text{sous les contraintes} \\ &\quad g(x) \leq 0 \\ &\quad x_{\in} \mathbb{Z}^n. \end{aligned}$$

Si les fonctions f et g sont quadratiques, on parle parfois également de programme quadratique, on le nommera ici *programme quadratique à contraintes quadratique*.

Une matrice est dite *positive* (resp. *semi-définies positives*) si, pour tout $x \in R^n$, $x^T Ax = \sum_{i=1}^{n} \sum_{j=1} A_{ij} x_i x_j > 0$ (resp $x^T Ax \geq 0$). Dans le cas où la matrice Q d'un PQ est définie positive, la fonction f est convexe et il existe un optimum local.

Résoudre la relaxation continue d'un PQD est un problème NP-difficile en général. Il existe de très performants algorithmes pour résoudre la relaxation continue des PQD avec contraintes linéaires. Ces algorithmes sont en fait inspirés des méthodes pour la PL en utilisant des principes proches de l'algorithme du simplexe ou des points intérieurs (appelé parfois dans ce cas Barrier algorithm). Les implémentations dans Cplex permettent de résoudre également de grands programmes. Des astuces que nous présenterons plus loin permettent également de ramener ces programmes à des programmes linéaires : cela permet aujourd'hui de résoudre également des PQD à contraintes quadratiques (Cplex). Des solveurs libres existent OpenOpt par exemple.
La version discrète de ces problèmes est difficile et les tailles solvables sont assez réduites (Cplex atteint quelques centaines de contraintes/variables).

Si Q est définie positive, c'est-à-dire de la *programmation quadratique convexe à contrainte quadratique*, la résolution est polynomiale : on sait la résoudre efficacement. On se ramène alors au cas d'un programme semi-défini.

• Programme semi-défini

Un autre type de programme continu, qui a priori sort un peu du cadre défini ici de la (PM) classique est celui du programme semi-défini.On définit alors un *programme semi-défini* comme étant celui des deux problèmes ci-dessous

Maximiser $Trace(A_0^T X)$	Minimiser $c^t y$
sous les contraintes	sous les contraintes
$A_i X \leq c_i, i = 1, ..., m$	$\sum_{i=1,...,m} A_i y_i - A_0$ est semi-définie positive
X matrice semi-définie positive.	$y \in I\!R^m$

où A_i, $i = 0, ..., m$, sont des matrices semi-définies positives et $c \in I\!R^m$.
En fait, ces deux programmes sont dit duaux l'un de l'autre et ont la même solution optimale. Le programme dual écrit ci-dessus se rapproche de l'écriture d'un programme mathématiques.
Il est facile de montrer que tout programme quadratique convexe peut se ramener à un programme semi-défini et que la programmation semi-définie est une généralisation de la programmation linéaire.

Des extensions des méthodes de points intérieurs permettent de résoudre efficacement la SDP : on trouvera des références à des techniques et des solveurs à http ://www-user.tu-chemnitz.de/ helmberg/semidef.html.
Il ne semble pas aujourd'hui pertinent de considérer des SDP discrets. En revanche, les SDP jouent un grand rôle pour obtenir de bonnes relaxations de problèmes NP-difficiles.

1.5 Un puissant outil de modélisation

Tout au long de ce cours, nous verrons à quel point l'écriture sous forme d'un PMD est un puissant outil de modélisation. En fait, on peut le voir souvent comme l'écriture naturelle algébrique d'un problème.

Inversement, comme nous l'avons cité, il n'existe pas de méthodes génériques efficaces pour résoudre un PMD. Le fait de ramener aussi facilement un problème à un PMD est donc parfois dangereux : on voit souvent des chercheurs ou des ingénieurs R&D conclure un travail de modélisation en affirmant que "comme on a ramené notre

problème à un PMD que les solveurs n'arrivent pas à résoudre, notre problème est difficile et nous allons utiliser des méthodes approchées". Cet état d'esprit, fort répandu malheureusement, est doublement faux. Tout d'abord, ramener un problème à un PMD ne prouve en rien la difficulté d'un problème : ce n'est pas une preuve de complexité, il peut ainsi exister d'autres pistes théoriques ou algorithmiques pour résoudre le problème d'origine. D'autre part, il peut exister plusieurs PMD modélisant le problème et différentes techniques pour le résoudre : c'est justement cette étude que nous allons mener dans ce cours.

Chapitre 2

Convexification et linéarisation

Comme nous l'avons remarqué auparavant, les cas de PM non discrets pour lesquels nous avons des outils de résolutions efficaces sont les cas de PL et de PQ avec matrice définie positive. Cette partie présente quelques exemples où l'on ramène un PM à ces cas particuliers en modifiant son écriture.

2.1 Convexification

On appelle *convexification* l'ensemble des astuces permettant de rendre convexe ou d'améliorer la convexité d'un PM ou d'un PMD.
Par exemple, si la matrice de la fonction objective d'un PMD n'est pas définie positive, il est possible de la rendre définie positive en la réécrivant différement (ou en lui ajoutant des variables fictives bien choisie).

Considérons l'exemple suivant :
La fonction

$$q(x_1, x_2) = Cx_1x_2 \quad \forall (x_1, x_2) \in \{0, 1\}^2$$

où C est une constante positive. Cette fonction n'est pas convexe car, pour tout x_1, x_2 pris dans $\mathbb{R}$, elle n'est pas toujours positive ou nulle. En revanche, la fonction

$$\tilde{q}(x_1, x_2) = \frac{1}{2}C(x_1 + x_2)^2 - \frac{1}{2}C(x_1 + x_2) \quad \forall (x_1, x_2) \in \{0, 1\}^2$$

est convexe. En effet, la partie quadratique de $\tilde{q}$ est positive ou nulle pour toute valeur des variables dans $\mathbb{R}$.
Or on peut remarquer que pour tout $(x_1, x_2) \in \{0, 1\}^2$, on a

$$\tilde{q}(x_1, x_2) = \frac{1}{2}C(x_1^2 + x_2^2 - 2x_1x_2) - \frac{1}{2}C(x_1 + x_2) = q(x_1, x_2)$$

car $x_1^2 = x_1$ et $x_2^2 = x_2$ pour des variables binaires !

On peut en fait généraliser l'astuce précédente à toute fonction quadratique : on peut ainsi ramener une fonction quadratique non convexe à un cas convexe (matrice définie positive). Pour en apprendre davantage, consulter [1].

2.2 Linéarisation

On appelle *linérisation* l'ensemble des astuces permettant de transformer en un PL ou un PLNE un PMD qui n'est pas linéaire à l'origine. Généralement, ces transformations demandent l'ajout de nombreuses variables supplémentaires.

Cette section comporte quelques astuces pour effectuer une telle linéarisation à partir d'une forme quadratique De manière plus générale, dans le cadre des problèmes d'optimisation combinatoire, la section suivante tente de montrer comment modéliser directement un problème en utilisant un PLNE.

• *Une variable à valeurs dans un espace discret*

Soit x une variable prenant ses valeurs parmi les n possibilités $v_1, ..., v_n \in \mathbb{R}$. On peut alors poser n variables de décisions binaires $y_1, ..., y_n$ telle que $y_i = 1$ si $x = v_i$ et 0 sinon.

Ce cas peut alors se modéliser par les deux contraintes $x = \sum_{i=1}^{n} v_i y_i$ et $\sum_{i=1}^{n} y_i = 1$.

• *Ecriture en variables binaires*

Si l'on peut déterminer des bornes sur les variables, on peut utiliser l'idée du cas précédent pour écrire un PLNE en variables entières comme un PLNE à variables binaires. En effet, considérons une variable x à valeurs entières entre 0 et u, $u \in \mathbb{N}$. Soit n tel que $2^n \leq u < 2^{n+1}$. On remplace alors x par sa représentation binaire : $x = \sum_{i=1}^{n} 2^i y^i$ où $y_i \in \{0, 1\}$ pour $i = 1, ..., n$.

• Le "ou" numérique

On veut représenter une variable x devant prendre des valeurs soit 0, soit être plus grande que L où L et x sont bornées par une valeur M.
On ajoute une variable $y \in \{0, 1\}$ et on utilise les contraintes

$$x \geq Ly \qquad \text{et} \qquad x \leq My.$$

• Carré d'une variable binaire
Soit $x \in \{0,1\}$. Alors la variable x^2 est équivalente à la variable x.

• Produit de deux variables binaires :
Soit $x \in \{0,1\}$ et $y \in \{0,1\}$, on veut obtenir une variable e ayant la valeur $e = xy$.
En fait, on a le résultat suivant :

$$e = xy \Leftrightarrow \begin{cases} e \leq x \\ e \leq y \\ e \geq x + y - 1 \\ e \geq 0 \\ e \in I\!R \end{cases}$$

On peut généraliser ce résultat au produit d'une variable binaire par une variable entière (bornée), au produit de plusieurs variables binaires, au carré d'une variables binaires,...

Ainsi, au total de ces trois remarques, on peut remarquer que toute forme quadratique peut se ramener à un PLNE binaire ! Mais cela se fait au prix fort, en ajoutant de nombreuses variables et contraintes.

2.3 Modélisation par un PLNE

Cette section donne des pistes d'idées pour modéliser certains liens logiques entre des variables ou des contraintes du problème.

• *Cas simples*
Soient a ,b et c des évènements correspondant aux variables de décisions binaires x_a, x_b et x_c.

— Si a et b ne peuvent pas se produire tous les deux : $x_a + x_b \leq 1$.
— Si au moins des évènement parmi a et b doit se produire : $x_a + x_b \geq 1$.
— Si a se produit alors b doit se produire : $x_a \leq x_b$.
— On peut noter que l'inégalité précédente modélise heureusement la contraposée de la proposition logique correspondante : si b ne se produit pas, a ne doit pas se produire.
— Si a se produit, alors b et/ou c doivent se produire : $x_a \leq x_b + x_c$.
— Si a ne se produit pas, alors une quantité y doit être nulle, sinon y est libre dans $I\!R$. On doit fixer une quantité M telle que y ne peut jamais être supérieure à M lorsqu'on atteint l'optimum du problème. Une telle constante M existe, car sinon le problème serait non borné : $y \leq Mx_a$ (contrainte dite de "big M").

• *Maximiser la valeur minimale d'un ensemble de fonctions linéaires*
Si on veut maximiser la plus petite valeur prise par un ensemble de fonctions linéaires $a^i x$, $i = 1, ...m$, il suffit d'ajouter une variable z et les contraintes $z \leq a^i x$, $i = 1, ...m$: la fonction objective devient alors Max z.

• *k contraintes parmi n*
On dispose d'un lot de n contraintes $a^1 x \leq b^1, a^2 x \leq b^2, ..., a^n x \leq b^n$ parmi lesquelles k au moins doivent être satisfaites (c'est-à-dire que les autres sont libres d'être satisfaites ou non). Pour chacune des contraintes $a^i x \leq b^i$, $i = 1, ..., n$, on détermine une valeur M_i suffisamment grande pour que $a^i x \leq b^i + M_i$ soit satisfaite quelque soit x.
On pose alors $y_1, ..., y_n$ des variables binaires et ce cas de figure se modélise alors de la façon suivante.

$$\begin{array}{lcl} a^1 x \leq b^1 & & a^1 x \leq b^1 + M_1 y_1 \\ a^2 x \leq b^2 & & a^2 x \leq b^2 + M_2 y_2 \\ ... & \Rightarrow & ... \\ a^n x \leq b^n & & a^n x \leq b^n + M_n y_n \\ & & \sum_{i=1}^{n} y_i = n - k \end{array}$$

Remarquons que si une seule contrainte doit être satisfaite parmi deux (c'est-à-dire si $k = 1$ et $n = 2$), on peut utiliser une seule variable binaire y en posant $y_1 = y$ et $y_2 = 1 - y$.

• *Implication entre contraintes*
Soit $a^1 x \leq b^1$ et $a^2 x \leq b^2$ deux contraintes telles que si $a^1 x < b^1$, alors $a^2 x \leq b^2$ doit être satisfaite, mais que, par contre, si $a^1 x \geq b^1$, alors $a^2 x \leq b^2$ peut ou non être satisfaite.
On prend M tel que $-a^1 x \leq -b^1 + M$ et $a^2 x \leq b^2 + M$ soient vérifiées pour toute valeurs de x. On peut utiliser une variable de décision binaire y et écrire alors :

$$-a^1 x \leq -b^1 + M(1 - y) \quad (2.1)$$
$$a^2 x \leq b^2 + My \quad (2.2)$$

En effet, si $a^1 < b$, alors la contrainte (2.1) implique que $y = 0$, et ainsi la contrainte (2.2) est équivalente à $a^2 x \leq b^2$ qui doit donc être satisfaite. Pour le cas contraire (i.e. si $a^1 x \geq b^1$), alors y peut prendre la valeur 0 ou 1, c'est-à-dire que $a^2 x \leq b^2$ peut être satisfaite ou non.

Chapitre 3

Optimisation Combinatoire et modélisations

La programmation linéaire en nombres entiers est un outil puissant de modélisation : en effet, de nombreux problèmes d'optimisation combinatoire peuvent être formulés comme des PLNE particuliers. Ainsi, un algorithme permettant de résoudre un PLNE permet de résoudre beaucoup de problèmes d'optimisation combinatoire. Il existe d'ailleurs de nombreux logiciels, appelés solveurs entiers, utilisés pour la résolution de problèmes d'optimisation combinatoire pratiques (industrie, transport,...). Malheureusement, les seuls algorithmes (B&B) connus pour résoudre tous les PLNE sont exponentiels car ils consistent à énumérer un grand nombre de solutions . Nous verrons dans cette partie, comment l'algorithme B&B peut-être enrichi par un algorithme de coupes.

D'autre part, il est souvent difficile de trouver une formulation PLNE efficace pour traiter problème d'optimisation combinatoire. Nous verrons qu'il est possible d'utiliser des formulation *non compactes*, c'est-à-dire possédant un nombre exponentiel de contraintes (Résolution par algorithme de coupes ou B&C) ou de variables (Résolution par Génération de colonnes ou B&C).

3.1 Problèmes d'Optimisation Combinatoire

Un problème d'optimisation combinatoire (OC) consiste à déterminer un plus grand (petit) élément dans un ensemble fini valué. En d'autres termes, étant donnés une famille $\mathcal{F}$ de sous-ensembles d'un ensemble fini $E = \{e_1, ...e_n\}$ et un système de poids $w = (w(e_1), ..., w(e_n))$ associé aux éléments de E, un problème d'optimisation consiste à trouver un ensemble $F \in \mathcal{F}$ de poids $w(F) = \sum_{e \in F} w(e)$ maximum (ou minimum),

i.e.

$$\max \text{ ou } \min \{w(F) \mid F \in \mathcal{F}\}.$$

La famille $\mathcal{F}$ représente donc les solutions du problème. Elle peut correspondre à un ensemble de très grande taille que l'on ne connaît que par des descriptions ou des propriétés théoriques qui ne permettent pas facilement son énumération.

Une première formulation PLNE "naïve" est alors possible : il suffit de prendre une variable 0-1 associée à chaque éléments de F et de poser le PLNE associé à ces variables. Le seul algorithme adaptée est alors l'énumération une à une des solutions. Comme ceci n'est pas envisageable, on abandonnera cette formulation. Cette idée néanmoins prouve que tout problème d'OC peut être formulé comme un PLNE.
A l'opposée de cette idée, on peut vouloir associer une variable 0-1 à chaque élément E : pour pouvoir alors définir une formulation associée, il faut pouvoir définir par des inégalités le fait que les solutions décrites par la variable doivent être dans l'ensemble $\mathcal{F}$. On dit dans ce cas que la formulation associée est *naturelle*. Même si elle existe toujours théoriquemen, il n'est pas toujours simple de déterminer une telle formulation.

Nous allons voir que cette description peut mener à plusieurs formulations PLNE bien différentes qui méneront à des résolutions différentes d'un même problème.

3.2 Notations de théorie des graphes

Les graphes que nous considérons ici sont finis et sans boucle ni arête multiple. Nous notons un graphe non-orienté par $G = (V, E)$ où V est l'*ensemble des sommets* et E l'*ensemble des arêtes* de G. Si $e \in E$ est une arête d'extrémités u et v, l'arête e peut être également notée uv.

Soit $G = (V, E)$ un graphe. Un *sous-graphe* $H = (W, F)$ de G est un graphe tel que $W \subseteq V$ et $F \subseteq E$. On dit qu'un sous-graphe $H = (W, F)$ de G *couvre* les sommets de G si $W = V$. On appelle *sous-graphe partiel* de G un sous-graphe $G' = (V, E')$ avec $E' \subset E$. Si $W \subset V$, alors $E(W)$ désigne l'ensemble de toutes les arêtes ayant leurs deux extrémités dans W. Le graphe $H = (W, E(W))$ est le sous-graphe de G *induit* par W.

Si $F \subset E$, alors $V(F)$ désigne l'ensemble des sommets de V qui sont extrémités des arêtes de F. Si $W \subset V$, alors $\delta(W)$ est l'ensemble des arêtes avec une extrémité dans W et l'autre extrémité dans $V \setminus W$. L'ensemble $\delta(W)$ est appelé une *coupe*. On note $\delta(v)$ à la place de $\delta(\{v\})$ pour $v \in V$ et on appelle $\delta(v)$ l'*étoile* de v. Pour

$v \in V$, on désigne par $N(v)$ l'ensemble des sommets adjacents à v. Si $W \subset V$, on pose $N(W) = \left(\bigcup_{v \in W} N(v)\right) \setminus W$, et on nomme $N(W)$ l'ensemble des *voisins* de W. Soit F un ensemble d'arêtes. On dénote par $G \setminus F$ le graphe obtenu à partir de G en supprimant les arêtes de F.

Une *chaîne* P dans $G = (V, E)$ est une séquence d'arêtes $e_1, e_2, ..., e_k$ telles que $e_1 = v_0v_1, e_2 = v_1v_2, ..., e_k = v_{k-1}v_k$. P est dite *chaîne élémentaire* si elle passe au plus une fois par le même sommet. Les sommets v_0 et v_k sont les extrémités de P et on dit que P relie v_0 à v_k. Le nombre k d'arêtes de P est appelé la *taille* de P. Si $P = e_1, e_2, ..., e_k$ est une chaîne (resp. chaîne élémentaire) reliant v_0 à v_k et $e_{k+1} = v_0v_k \in E$, alors la séquence $e_1, e_2, ..., e_k, e_{k+1}$ est dite un *cycle* (resp. *cycle élémentaire*) de taille $k + 1$. Lorsqu'il n'y a pas ambiguïté, on utilisera chaîne pour désigner une chaîne élémentaire. Un cycle (chaîne) est dit *impair* si la taille est impaire, autrement, il est dit *pair*. Si P est un cycle ou une chaîne et uv une arête de $E \setminus P$ avec $u, v \in V(P)$, alors uv est appelé une *corde* de P. Un *trou* de G est un cycle sans corde. Un graphe est dit *connexe* si, pour tout couple de sommets u et v, il existe une chaîne joignant u et v.

Un chemin (resp. cycle) élémentaire qui passe par tous les sommets d'un graphe est dit *hamiltonien*. Un chemin (resp. cycle) élémentaire qui passe une fois par toutes les arêtes d'un graphe est dit *eulérien*.

Un graphe $G = (V, E)$ est dit *complet* si toute paire de sommets de V est jointe par une arête de E. On note par K_q le graphe complet sur q sommets. On appelle *clique* de G un sous-graphe complet de G. Un *stable* d'un graphe est un ensemble de sommets deux à deux non adjacents.

Un graphe est appelé *biparti* si son ensemble de sommets peut être partitionné en deux ensembles de sommets non vides V_1 et V_2 tels qu'aucun couple de sommets de V_1 (respectivement de V_2) ne soit adjacent. Il est bien connu qu'un graphe est biparti si et seulement s'il ne contient pas de cycle impair.

On note un graphe orienté $G = (V, A)$ où A est l'ensemble des *arcs*. Certaines définitions vues précédemment sur les graphes non-orientés sont alors à adapter au cas orienté. Un *chemin* est une séquences d'arcs et un chemin qui revient sur son sommet d'origine est un *circuit*. On note $N^+(v)$ (resp. $N^-(v)$) l'ensemble des sommets successeurs (resp. prédécesseurs) de v. On note $\delta^+(v)$ (resp. $\delta^-(v)$ l'ensemble des arcs sortant de (resp. entrant en) v

3.3 Problèmes classiques en optimisation combinatoire

3.3.1 Le problème du sac-à-dos

Considérons n objets, notés $i = 1, ..., n$, apportant chacun un bénéfice c_i mais possédant un poids a_i. On veut ranger ces objets dans un "sac" que l'on veut au maximum de poids b. Le problème de *sac-à-dos* (*knapsack*) consiste à choisir les objets à prendre parmi les n objets de manière à avoir un bénéfice maximal et respecter la contrainte du poids à ne pas dépasser. Chaque objet i, $i \in \{1, ..., n\}$, doit être sélectionné au moins p_i fois et au plus q_i fois.

Ce problème se rencontre bien entendu dès que l'on part en randonnée en voulant emmener le plus possible d'objets utiles (nourriture, boissons,...). Mais ce problème est plus fréquemment utilisé pour remplir les camions de transport, les avions ou bateaux de fret et même pour gérer la mémoire d'un microprocesseur.

La formulation PLNE du problème de sac-à-dos est très simple. On utilise pour chaque objet $i \in \{1, ..., n\}$, une variable entière x_i correspondant au nombre de fois où l'objet i est choisi. Le problème du sac-à-dos est donc équivalent au programme en nombres entiers suivant.

$$\begin{aligned} \text{Max } & \sum_{i=1}^{n} c_i x_i \\ & \sum_{i=1}^{n} a_i x_i \leq b, \qquad (3.1) \\ & p_i \leq x_i \leq q_i, \text{ pour } i = 1, ..., n, \\ & x_i \in I\!N, \text{ pour } i = 1, ..., n. \end{aligned}$$

La contrainte (3.1) est dite *contrainte de sac-à-dos*. Elle est l'unique contrainte de ce problème qui est pourtant NP-complet.

3.3.2 Recouvrement, pavage et partition

Soit $E = \{1, ..., n\}$ un ensemble fini d'éléments. Soit $E_1, ..., E_m$ des sous-ensembles de E. A chaque ensemble E_j on associe un poids c_j, $j = 1, ..., m$.

Une famille $F \subseteq \{E_1, ..., E_m\}$ est dite
- un *recouvrement* de E si $\cup_{E_j \in F} E_j = E$, pour tout $j \in \{1, ..., m\}$,
- un *pavage* de E si $E_j \cap E_k = \emptyset$, pour tout $j \neq k \in \{1, ..., m\}$,
- une *partition* de E si F est à la fois un recouvrement et un pavage.

Prenons par exemple 5 éléments $E = \{1, 2, 3, 4, 5\}$ et 4 sous-ensembles $E_1 = \{1, 2\}$, $E_2 = \{1, 3, 4, 5\}$, $E_3 = \{3, 4\}$ et $E_4 = \{3, 4, 5\}$. On peut remarquer facilement sur la

figure 12.1 que $\{E_1, E_2\}$ est un recouvrement, que $\{E_1, E_3\}$ est un pavage et $\{E_1, E_4\}$ est une partition.

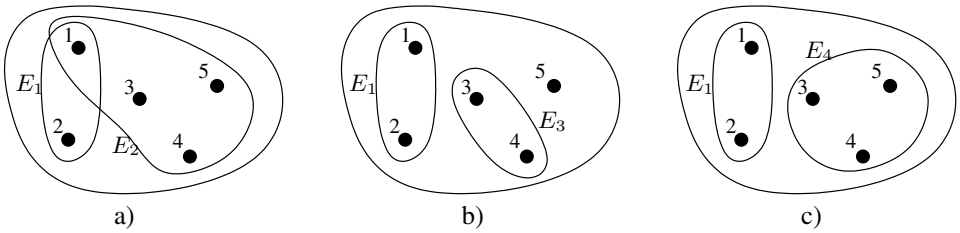


FIGURE 3.1 Illustration de a) Recouvrement, b) Pavage et c) Partition

Le *problème de recouvrement* (resp. *pavage*, *partition*) consiste à déterminer un recouvrement (resp. pavage, partition) dont la somme des poids des ensembles qui le forment est de poids minimum (resp. maximum, minimum/maximum).

Ces problèmes peuvent se modéliser en utilisant des variables binaires $x_1, ..., x_m$ associées aux sous-ensembles $E_1, ..., E_m$. Pour cela, on considère A la matrice en 0-1 dont les lignes correspondent aux éléments $1, ..., n$ et les colonnes aux sous-ensembles $E_1, ..., E_m$ et dont les coefficients sont $A_{ij} = 1$ si $i \in E_j$ et 0 sinon. Ainsi les trois problèmes peuvent s'écrire.

Recouvrement	Pavage	Partition
$\text{Min} \sum_{i=1}^{m} c_j x_j$	$\text{Max} \sum_{i=1}^{m} c_j x_j$	$\text{Max (ou Min)} \sum_{i=1}^{m} c_j x_j$
$Ax \geq \mathbb{1}$	$Ax \leq \mathbb{1}$	$Ax = \mathbb{1}$
$x \in \{0,1\}^m$	$x \in \{0,1\}^m$	$x \in \{0,1\}^m$

où $\mathbb{1}$ est un vecteur dont chaque composante est 1.

Dans l'exemple associée à la figure 12.1, la matrice A est alors

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

On peut interpréter les contraintes du problème de recouvrement (resp. pavage, partition) comme le fait qu'un élément de E doit être pris au moins une fois (resp. au plus une fois, exactement une fois).

Ces problèmes ont de multiples applications. Par exemple, considérons une région où l'on désire implanter des casernes de pompiers afin de couvrir toutes les villes. Pour

chaque caserne potentielle, on détermine les communes desservies et le coût d'installation de la caserne. Comme l'on veut couvrir toutes les communes, il s'agit clairement d'un problème de recouvrement.

3.3.3 Le problème du stable

Soient $G = (V, E)$ un graphe non-orienté et c une fonction poids qui associe à tout sommet $v \in V$ un poids $c(v)$. Un *stable* de G est un sous-ensemble S de sommets de V tel qu'il n'existe aucune arête de E entre 2 sommets de S. Le *problème du stable de poids maximum* consiste à déterminer un stable S de G tel que $c(S) = \sum_{v \in S} c(v)$ soit maximum.

La figure 3.2 donne par exemple un graphe où tous les sommets sont valués de poids 1. On peut noter que tous les sommets isolés sont des stables de poids 1 et qu'un stable a un poids au maximum 2, ce qui est obtenu par exemple par le stable $\{v_1, v_5\}$.

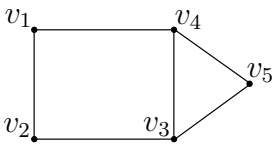


FIGURE 3.2 Illustration du problème du stable

Notons χ^S le vecteur d'incidence d'un stable S dans V, c'est-à-dire un vecteur indexé sur les sommets de V tel que $\chi^S(u) = 1$ si $u \in S$ et 0 sinon. Un tel vecteur doit nécessairement vérifier, pour toute arête uv de E, qu'au plus une seule des deux extrémités parmi u et v peut être prise dans S. Ce que l'on peut décrire par l'inégalité linéaire $x(u) + x(v) \leq 1$, appelée *inégalité aux arêtes*.

Inversement, considérons un vecteur binaire y, indexé sur les sommets de V, qui vérifie toutes les inégalités associées aux arêtes du graphe G. Alors y est un vecteur d'incidence d'un stable de G. En effet, si l'on pose W l'ensemble des sommets $v \in V$ tel que $y(v) = 1$, alors clairement W ne contient pas deux sommets adjacents.

Ainsi le problème du stable est équivalent au PLNE suivant :

$$\begin{aligned} \text{Max} \sum_{u \in V} & c(u)x(u) \\ x(u) + x(v) &\leq 1, \quad \text{pour tout } uv \in E, \\ x(u) &\in \{0, 1\}, \quad \text{pour tout } u \in V. \end{aligned} \tag{3.2}$$

On nomme *compacte* une formulation qui contient un nombre polynomial de variables et de contraintes. On peut noter que cette formulation est compacte car elle contient un nombre polynomial de contraintes et de variables en fonction de la taille de l'entrée

du problème. En effet, il y a une variable par sommet du graphe et une inégalité par arête. On appelle d'ailleurs cette formulation du problème du stable "formulation aux arêtes".

Le problème du stable est donc équivalent à un PLNE qui possède un nombre réduit de contraintes et de variables. Le problème est pourtant NP-complet et difficile à résoudre.

D'autres formulations de ce problème sont possibles. Par exemple, on peut remplacer les contraintes (3.2) par les contraintes dites *de cliques*

$$\sum_{u \in K} x(u) \leq 1, \text{ pour toute clique } K \text{ de } G. \tag{3.3}$$

Les contraintes d'arêtes (3.2) sont contenues dans la famille des contraintes de cliques (3.3). On peut constater que la nouvelle formulation est aussi équivalent au problème du stable mais elle contient d'avantage de contraintes (en fait un nombre exponentiel dans le pire des cas). En revanche, la solution de la relaxation linéaire de la formulation par les cliques sera plus proche de la solution optimale du problème du stable que la relaxation de la formulation par les arêtes. Or, les algorithmes de branchement et évaluation nous disent que plus la relaxation est bonne, plus l'algorithme est rapide.

On peut se poser plusieurs questions : Comment choisir une bonne formulation ? Comment décider si une contrainte est utile ou non à la relaxation ?

3.3.4 Le problème du voyageur de commerce

Soit $G = (V, E)$ un graphe non-orienté où les sommets V sont appelées des villes et les arêtes entre deux villes des liaisons. Soit $c(e)$ un coût associé à la liaison e, pour toute $e \in E$. Le *problème du voyageur de commerce* (TSP, Traveller Salesman Problem) consiste à déterminer un cycle passant par chaque ville une et une seule fois. Un tel cycle est dit *hamiltonien*, on l'appelle aussi souvent *tour* ou *tournée*. Ce problème classique a trouvé de nombreuses applications dans des domaines pourtant aussi éloignés que la production de lait ou la construction de circuits intégrés.

Regardons le problème assymétrique : c'est-à-dire que les coûts des arcs $c_{ij} \neq c_{ji}$ pour tout arête $ij \in e$. On considère le graphe complet $D = (V, A)$, avec des coûts sur les arcs c_{ij}. On recherche le cycle orienté, appelé *tour* qui contient les n villes et qui soit de taille minimale. On définit les variables $x_{ij} = 1$ si l'arc (i, j) est dans le tour et 0 sinon. Même avec ce formalisme déjà très précis, il existe plusieurs formulations possibles.

Regardons le PLNE suivant.

$$\text{Min} \sum_{i,j} c_{ij} x_{ij}$$

$$\begin{aligned}
\textstyle\sum_{j \in V} x_{ij} = 1 \quad & \text{pour tout } i \in V, && (3.4)\\
\textstyle\sum_{i \in V} x_{ij} = 1 \quad & \text{pour tout } j \in V, && (3.5)\\
x_{ij} \geq 0 \quad & \text{pour tout } (i,j) \in V \times V,\\
x_{ij} \in I\!N \quad & \text{pour tout } (i,j) \in V \times V.
\end{aligned}$$

La relaxation linéaire de ce PLNE est souvent entière (par exemple pour certains types de graphes comme les graphes bipartis). Malheureusement, les solutions peuvent contenir plusieurs cycles orientées, que l'on appelle *sous-tours*. Les contraintes 3.4 sont appelées *inégalités d'affectation*.

• *"Elimination des sous-tours" par la formulation MTZ*

Pour éliminer les sous-tours, on peut aussi utiliser la formulation de Miller-Tucker-Zemlin (MTZ). On ajoute de nouvelles variables réelles u_i, $i = 1, ..., n$, associées aux villes et les contraintes :

$$\begin{aligned}
& u_1 = 1, && (3.6)\\
& 2 \leq u_i \leq n \text{ pour tout } i \neq 1, && (3.7)\\
& u_i - u_j + 1 \leq n(1 - x_{ij}) \text{ pour tout } i \neq 1, j \neq 1. && (3.8)
\end{aligned}$$

On apelle ces dernière inégalités *inégalités MTZ*. Elles permettent d'éliminer les sous-tours :
- car, pour tout arc (i, j) où $x_{ij} = 1$, elles forcent $u_j \geq u_i + 1$,
- si une solution du PLNE formé pas les inégalités (3.4) contient plus d'un sous-tour, alors l'un deux au moins ne contient pas le sommet 1 et, sur ce sous-tour, les variables u_i s'incrémentent à l'infini.

On peut par contre remarquer que, de surcroit, dans le cas d'une solution réalisable pour le PLNE formé des inégalités (3.4) et des 3 inégalités MTZ, les variables u_i, $i = 1, ..., n$ indiquent la position de la ville i dans le tour. Ainsi, si on veut indiquer qu'une ville i doit être positionnée avant une autre, ou proche d'une autre ville j, on peut ajouter des contraintes sur concernant u_i et u_j.

Cette formulation possède n variables de plus, mais elle a l'énorme avantage d'être compacte. On peut donc directement utiliser une procédure de branchement et séparation (Branch& Bound) et donc un solveur entier. Cette formulation est plus faible que

la formulation par les coupes et elle n'a pas d'intérêt autre qu'être compacte.

•*"Elimination des sous-tours" en brisant les sous-tours*

Pour éliminer les sous-tours, on peut ajouter les contraintes suivantes, dites contraintes de sous-tours :

$$\sum_{i \in S, j \in S} x_{ij} \leq |S| - 1, \text{ pour tout } S \subset V, |S| > 1, S \neq V \tag{3.9}$$

Ces contraintes empèchent les solutions d'avoir des sous-tours. Néanmoins, ces contraintes sont en nombres exponentielles. Pour pouvoir résoudre une telle formulation, on utilise les algorithmes de coupes, qui mettent en pratique les résultats obtenus lors d'une étude polyédrale du problème. Le principe consiste à ajouter progressivement les inégalités dans la formulation. Ici l'ajout se fait par un algorithme, dit algorithme de séparation, qui revient à détecter des cycles dans un graphe.

•*"Elimination des sous-tours" par la connexité par les coupes*

Une autre façon d'interdire les sous-tours est de remarquer qu'une solution du TSP est un ensemble d'arcss C qui forme un sous-graphe partiel connexe, c'est-dire qu'il y a au moins un arc sortant de chaque ensemble de sommets, c'est-à-dire au moins un sommet par coupes du graphe.

$$\begin{aligned} &\sum_{e \in \delta^+(W)} x(e) \geq 1, \text{ pour tout } W \subsetneq V et W \neq \emptyset, \\ &\sum_{e \in \delta^-(W)} x(e) \geq 1, \text{ pour tout } W \subsetneq V et W \neq \emptyset, \end{aligned} \tag{3.10}$$

Les contraintes *de coupes* (9.2) quant à elles, obligent l'ensemble d'arêtes $C = \{e \in E \mid x(e) = 1\}$ à former un graphe connexe. En effet, si ce graphe n'était pas connexe, alors il existerait une coupe dans le graphe qui n'intersecterait pas C.
On peut montrer qu'en fait les contraintes (3.9) et (9.2) sont équivalentes (cela se montre en utilisant les contraintes de degré).
Il y a à nouveau un nombre exponentiel de contraintres de connexité. On utilisera alors un algorithme de coupes et branchement. C'est actuellement le principe qui a permis d'obtenir les meilleures solutions.

•*"Cas symétrique"*

Pour le cas symétrique du TSP, on peut simplifier les formulations précédentes, on obtient alors :

Soit x une variable associée aux liaisons de E telle que $x(e) = 1$ si e prise dans la solution et 0 sinon. Le problème du voyageur de commerce est équivalent au PLNE suivant.

$$Min \sum_{e \in E} c(e)x(e)$$

$$\sum_{e \in \delta(v)} x(e) = 2, \text{ pour tout } u \in V, \tag{3.11}$$

$$\sum_{e \in \delta(W)} x(e) \geq 2, \text{ pour tout } W \subsetneq V \text{ et } W \neq \emptyset, \tag{3.12}$$

$$x(e) \in \{0, 1\}, \text{ pour tout } e \in E.$$

En fait, les contraintes (9.1) dites *de degré* forcent à 2 le nombre d'arêtes incidentes à un même sommet. Les contraintes *de coupes* (3.12) quant à elles, obligent l'ensemble d'arêtes $C = \{e \in E \mid x(e) = 1\}$ à former un graphe connexe. En effet, si ce graphe n'était pas connexe, alors il existerait une coupe dans le graphe qui n'intersecterait pas l'ensemble C or ceci est impossible car cette contrainte est exigée pour chaque coupe.

On peut remarquer qu'il y a un nombre exponentielle de contraintes de connexité. De plus, leur énumération est loin d'être évidente. On peut donc se poser la question : Comment résoudre un programme linéaire possédant un nombre exponentiel de contraintes ? Comment produire les contraintes d'un tel programme linéaire ?

3.3.5 Le problème de coloration

Soit $S \subseteq I\!N$. Une *coloration* des sommets d'un graphe $G = (V, E)$ est une fonction $r : V \to I\!N$ telle que $r(u) \neq r(v)$ pour tout couple de sommets adjacents u, v. Les entiers utilisés par la fonction r sont appelés les *couleurs* utilisées pour la coloration. Une *k-coloration* est une coloration $c : V \to \{1, .., k\}$. Un graphe est dit *k-coloriable* s'il possède une k-coloration.

Tester si un graphe est k-coloriable est un problème NP-complet si $k \geq 3$, et il est polynomial si $k = 2$. En effet, si $k = 2$, il suffit de tester si le graphe est biparti, c'est-à-dire s'il ne contient pas des cycle impair. Ce qui peut se faire par un simple parcours de graphe.

Soit $G = (V, E)$ un graphe non-orienté. Le *problème de coloration* consiste à déterminer le plus petit k tel que G soit k-coloriable.

Soit un graphe $G = (V, E)$. Une formulation en variables entières peut être donnée de la façon suivante. On associe à chaque sommet u de V un vecteur binaire à K dimensions $x_u = (x_u^1, ..., x_u^K)$, où K est une borne supérieure sur la coloration de G (au maximum $K = |V|$). Comme l'on désire minimiser le nombre de couleur, on ajoute une variable binaire w_l par couleur $l = 1, ..., K$ indiquant si cette couleur a été utilisée ou non. Le problème est donc équivalent au programme

$$\text{Min} \sum_{l=1}^{K} w_l$$

$$\sum_{l=1}^{K} x_u^l = 1, \quad \text{pour tout } u \in V, \tag{3.13}$$

$$x_u^l + x_v^l \leq w_l, \quad \text{pour tout } e = uv \in E \text{ et } 1 \leq l \leq K, \tag{3.14}$$

$$x_u^l \in \{0, 1\}, \quad \text{pour tout } u \in V \text{ et } 1 \leq l \leq K.$$

La contrainte (3.13) oblige chaque sommet à être affecté à une et une seule couleur. La contrainte (3.14) oblige que deux sommets adjacents n'aient pas la même couleur. Dans cette formulation, il y a dans le pire des cas $n^2 + n$ variables binaires et $n * m$ contraintes. Malgré sa taille polynomiale raisonnable, cette formulation s'avère être en pratique très difficile à résoudre. (En fait, on peut remarquer que ce PLNE peut être vu comme un programme linéaire modélisant le problème du stable dans un graphe particulier constitué de k copies du graphes d'origine où chaque copie d'un sommet est reliée à toutes les autres copies de ce même sommet).

Cette formulation, en pratique, s'avère très difficile à résoudre par les algorithmes de branchements. Cependant, d'autres formulations ont été proposées, plus pour donner des bornes inférieures au problème qu'un réel résultat exact optimal. La formulation suivante s'est avérée par exemple très performante pour donner des bornes inférieures. Elle repose sur le fait qu'une coloration est en fait une couverture d'un graphe par un nombre minimum de stables.

Soit $\mathcal{S}$ l'ensemble des stables non vides de G. On associe à chaque stable $S \in \mathcal{S}$ une variable binaire t_S. Le problème de coloration est alors équivalent au programme en nombres entiers suivant (Mehrotra et Trick 1995).

$$\text{Min} \sum_{S \in \mathcal{S}} t_S$$

$$\sum_{S \in \mathcal{S} \mid u \in S} t_S = 1, \quad \text{pour tout } u \in V, \tag{3.15}$$

$$t_S \in \{0, 1\}, \quad \text{pour tout } S \in \mathcal{S}. \tag{3.16}$$

En fait cette formulation possède la particularité de contenir un nombre exponentiel de variables et un nombre très réduit de contraintes. L'algorithme nécessaire pour résoudre une telle formulation s'appelle *algorithme de génération de colonnes* car ils consistent à générer des variables que l'on ajoute itérativement au problème. Ce procédé peut être vu comme le procédé dual des méthodes de coupes auxquelles ce cours s'intéresse.

3.3.6 Le problème du flot max

Soit $G = (V, A)$ un graphe orienté où l'on associe une capacité $c(a) \in \mathbb{N}$ associé à chaque arc $a \in A$. On suppose que G contient un sommet s, appelé source, sans prédecesseur à partir duquel on peut atteindre tout sommet de G et un sommet t, appelé puits, qui est accessible depuis tout sommet de G.
Un *s-t-flot* est un vecteur positif $x \in \mathbb{R}^m$ s'il vérifie la contrainte de "conservation de flot"

$$\sum_{a \in \delta^+(u)} x(a) - \sum_{a \in \delta^-(u)} x(a) = 0 \quad \forall u \in v \setminus \{s, t\}.$$

Un flot est dit *réalisable* si $x(a) \leq c(a)$ pour tout arc $a \in A$.

Le *problème du flot de capacité maximale* ou *problème du flot max* consiste à rechercher un flot tel que la valeur $\sum_{a \in \delta^+(s)} x(a)$ soit maximale. On peut remarquer que, par conservation des flots, cette valeur est exactement aussi $\sum_{a \in \delta^-(s)} x(a)$.
On sait depuis le théorème de Ford-Fulkerson que si c est entier, alors il existe une flot optimal entier et on sait le déterminer en temps largement polynomial.

Considérons le PL suivant qui a en fait toujours des solutions entières quelque soit le vecteur coût c (c'est donc en quelque sorte un '"PLNE").

$$\begin{aligned}
\text{Max} \quad & v \\
& \sum_{a \in \delta^+(u)} x(a) - \sum_{a \in \delta^+(u)} x(a) = 0 \quad \forall u \in V \setminus \{s, t\} && (3.17)\\
& \sum_{a \in \delta^+(s)} x(a) - v = 0, && (3.18)\\
& \sum_{a \in \delta^-(t)} x(a) + v = 0, && (3.19)\\
& x(a) \leq c(a) \quad \forall a \in A, && (3.20)\\
& v \geq 0, && (3.21)\\
& x(a) \geq 0 \quad \forall a \in A. && (3.22)
\end{aligned}$$

Le résultat de Ford-Fulkerson indique ainsi que ce problème possède une solution optimale entière quelque soit le vecteur de capacités entier ! On verra dans la section 8 que ce PL a en fait toujours une solution optimale entière.

3.3.7 Le problème du couplage biparti

Soit un graphe biparti complet $G = (V_1 \cup V_2, E)$ associé à un poids $c \in I\!N^m$ associé aux arêtes de E. On appelle *couplage* de G un ensemble d'arêtes deux à deux non incidentes.
Le *problème du couplage biparti* consiste à déterminer un couplage qui maximise la somme des poids de ses arêtes.

Il existe plusieurs algorithmes polynomiaux efficaces pour résoudre ce problème classique (dont le célèbre algorithme hongrois). De plus, il a été montré qu'une affectation est entière dès que le vecteur poids c est entier.
Considérons le PL suivant.

$$\begin{aligned}
&\text{Max} \sum_{e \in E} c(e)x(e) \\
&\sum_{e \in \delta(u)} x(e) \leq 1 \quad \forall u \in V_1 && (3.23)\\
&\sum_{e \in \delta(u)} x(e) \leq 1 \quad \forall u \in V_2 && (3.24)\\
&x(e) \geq 0 \quad \forall e \in E. && (3.25)
\end{aligned}$$

Ce PL a des solutions entières quelque soit le vecteur de poids entier ! En effet, la matrice de ce PL est TU : on verra cela dans la section 8.

Chapitre 4

Exercices

4.1 Modélisation de petits problèmes

4.1.1 Problème de production (tiré d'un livre de 1ereS)

Un fabriquant de yaourt produit 2 types de yaourts à la fraise A et B à partir de fraise, de lait et de sucre. Chaque yaourt doit respecter les proportions suivantes de matières premières.

	A	B
Fraise	2	1
Lait	1	2
Sucre	0	1

Les matières premières sont en quantité limitée : 800 kilos de fraises, 700 kilos de lait et 300 kilos de sucre. La vente des yaourts A rapportent 4 € par kilo et les yaourts B 5 € .

a) Donner une formulation du problème sous la forme d'un programme linéaire.

b) Donner la représentation graphique du problème.

c) Donner la solution optimale.

4.1.2 Problème de transport

Une entreprise de construction d'automobiles possède 3 usines à Paris, Strasbourg et Lyon. Elle a besoin d'acheminer les métaux nécessaires à partir du Havre ou de Marseille. Chaque usine nécessite hebdomadairement 400 tonnes à Paris, 300 tonnes à Strasbourg et 200 tonnes à Lyon. Les ports du Havre et de Marseille peuvent fournir respectivement 550 tonnes et 350 tonnes.

Les coûts de transport entre ces villes sont donnés en kilo-€ par tonne dans le tableau suivant.

	Paris	Strasbourg	Lyon
Le Havre	5	6	3
Marseille	3	5	4

Proposer une modélisation de ce problème de transport de manière à satisfaire la demande, à partir des quantités disponibles et en minimisant les coûts de transport.

4.1.3 Combinaison optimale en vitamines (Diet Problem)

Une personne soucieuse de sa forme physique souhaite absorber chaque jour 36 unités de vitamine A, 28 unités de vitamine C et 32 unités de vitamine D. Deux marques sont succeptibles de forunir ces apports. La marque 1 coûte 3 euros et procure 2 unités de vitamine C et 8 unités de vitamine D. La marque 2 coûte 4 euros et procure 3 unités de vitamine A, 2 unités de vitamine C et 2 unités de vitamine D.
Il s'agit de trouver la combinaison respectant les exigences d'absorption quotidiennes au moindre coût.
- Définir les variables de décision de ce problème.
- Formuler la fonction objectif.
- Ecrire les contraintes.

4.1.4 Constitution d'un portefeuille optimal

On suppose qu'un investisseur boursier désire choisir son portefeuille en minimisant le risque pris. Il dispose de n titres boursiers $T_1, ...T_n$ et qu'il désire en choisir k parmi eux. Le rendement de chaque titre est noté r_i, $i = 1, ..., n$. Ce rendement est une variable aléatoire dont on connaît l'espérance $E(r_i)$ et la variance notée $V(r_i)$. Le problème consiste à choisir k titres et la proportion d'investissement dans chaque tout en minimisant les risques. On supposera pour simplifier ici que l'on peut prendre un pourcentage réel quelconque d'un titre boursier ; on suppose aussi que les variables aléatoires sont indépendantes, i.e. le rendement des titres sont indépendants les uns des autres. On exige cependant que, si un titre est choisi, sa proportion est alors supérieur à un pourcentage ϵ_i donné.
Si l'on note x_i la proportion choisie du titre i, $i = 1, ..., n$, on estime que l'espérance du rendement global procuré par un portefeuille est alors $\sum_{i=1}^{n} E(r_i)x_i$. De plus, la valeur du rendement global mesure le risque pris par l'investisseur, on peut la calculer par la formule $\sum_{i=1}^{n} V(r_i)x_i^2$.

Proposer un PMD modélisant ce problème.

4.1.5 Centres de loisir

Une région est divisée en six zones (zones 1,...,6). La commune souhaite construire des centres de loisir dans certaines de ces zones. Et elle désire monter un nombre minimum de centres de telle manières que, pour chaque zone, il existe au moins un centre qui se trouve à au plus 15 minutes (en voiture) de cette zone. Le temps nécessaire pour aller d'une zone à l'autre est donné dans la table suivante :

de	zone 1	zone 2	zone 3	zone 4	zone 5	zone 6
zone 1	0	10	20	30	30	20
zone 2	10	0	25	35	20	10
zone 3	20	25	0	15	30	20
zone 4	30	35	15	0	15	25
zone 5	30	20	30	15	0	14
zone 6	20	10	20	25	14	0

1) Formuler le problème qui consiste à déterminer le nombre minimum de centres à construire ainsi que les zones où ceux-ci doivent être construits comme un programme linéaire en nombres entiers.
2) Modifier le programme pour qu'il corresponde à la contrainte suivante : si un centre est construit dans la zone 1, alors un centre doit être construit dans la zone 4.
3) Quelle inégalité permet de modéliser la contrainte suivante : une zone au moins parmi les zones 1, 2 et 3 doit avoir au moins un centre à au plus 15 minutes. Est-il nécessaire de l'ajouter au programme ?

4.1.6 Diététique avare

Le cuisinier de l'hôtel doit préparer le petit-déjeuner de La Castafiore. La cantatrice doit suivre un régime auquel son avarice donne raison. Le cuisinier doit donc composer un menu comportant au plus 1000 calories et qui soit le moins cher possible, à partir des ingrédients suivants :

	Poids unitaire en grammes	Calories par gramme	Prix unitaire en euros
toast	40	3,4	1,50
miel (par toast)	15	2,8	0,50
confiture (par toast)	10	5	0,40
thé (une tasse)	100	0,2	1,00
lait (un verre)	100	0,5	0,70
beurre (par toast)	10	8	0,30
oeuf (sur le plat)	60	1	0,50
bacon (la tranche)	20	8	1,00
jus d'orange	30	4	1,20
sucre (morceau)	5	4	0,10

4.1.7 Problème d'affectation quadratique

On considère n unités de calcul (processeurs) que l'on doit positionner sur n emplacements prédéterminés de façon à affecter une seule unité par emplacement (et inversement). Ces unité vont communiquer entre elles lors d'échange de données : entre une unité i placée sur un emplacement k et une unité j placée sur un emplacement l, se passe une interaction de coût c_{ijkl}. Ce coût est en fait le produit d'un flux prédéterminée f_{ij} entre i et j multiplié par la distance d_{kl} de k à l. On désire minimiser le coût d'implantation des n unités sur les n emplacements.

Proposer une formulation quadratique de ce problème.

4.2 Exercices

4.2.1 Linéarisation de la différence entre variables

Soit x et y deux variables entières positives. On désire modéliser linéairement le fait que x et y doivent prendre des valeurs différentes. On suppose qu'il existe une grande valeur M telle que x et y soit toujours plus petite strictement que M.
Donner une écriture linéaire (en nombres entiers) de cette différence entre variables.

4.2.2 Linéarisation d'un quotient

On désire linéariser le rapport de deux fonctions linéaires de variables bivalentes, c'est-à-dire donner un programme linéaire en nombres entiers équivalent à la formula-

tion non-linéaire suivante :

$$(D)\begin{cases} Max & \dfrac{a_0+\sum_{i=1}^{n} a_i x_i}{b_0+\sum_{i=1}^{n} b_i x_i} \\ & Tx \leq \alpha \\ & x_i \in \{0,1\} \qquad \forall i \in \{1,...,n\}. \end{cases}$$

où $Tx \leq \alpha$ est un ensemble de contraintes linéaires.

On suppose que
— $a_i \geq 0$ pour $i = 0, 1, ..., n$
— $b_i \geq 0$ pour $i = 1, ..., n$ et $b_0 > 0$

1) Montrer que la fonction objective est toujours bornée par une valeur M.
2) Ajouter à la formulation une variable réelle z égale à la valeur de la fonction objective. Réécrivez alors (D) comme une formulation contenant entre autres les termes quadratiques $(zx_i)_{i\in\{1,...,n\}}$, mais où le quotient a disparu.
3) On désire ajouter des variables réelles $(t_i)_{i\in\{1,...,n\}}$. Proposer 3 lots de n inégalités linéaires impliquant que t_i ait la valeur du produit zx_i pour $i = 1, ..., n$.
4) En déduire une formulation PLNE équivalente à (D).

4.2.3 Décision en production industrielle

Une entreprise produisant des billes de plastique veut s'implanter sur une nouvelle zone géographique. Ces billes de plastique sont la matière première de nombreux objets industriels (sièges, manches d'outils, bidons,...). L'entreprise a démarché n clients et prévoit de vendre, sur un horizon de 5 années à venir, d_i tonnes de billes de plastique à chaque client $i \in \{1, ..., n\}$. L'entreprise dispose de m sites potentiels $s_1, ..., s_m$ pour installer ses usines. On a évalué à c_j euros le coût d'installation d'une usine sur le site s_j, $j = 1, ..., m$. Les usines prévues ne sont pas toutes de même capacité de production : un site s_j aura une capacité de M_j tonnes de billes sur les 5 années à venir, $j = 1, ..., m$. On suppose que le coût de production est indépendante du lieu de production. Enfin, on connaît les coûts de transports par tonne c_{ij} entre un client i et un site s_j, pour $i = 1, ..., n$ et $j = 1, ..., m$.

L'entreprise souhaite déterminer les sites sur lesquels établir ses usines pour pouvoir satisfaire la demande de ses clients tout en minimisant le coût total (installation, production et livraison) sur les 5 prochaines années.

Remarquons tout d'abord que, comme le coût de production ne dépend pas du lieu de production, ce coût est fixe quelque soient les choix d'implantation des usines. Les variables du problème se limitent donc à :

- y_j valant 1 si l'on décide d'implanter une usine sur le site s_j et 0 sinon, $j = 1, ..., m$.
- x_{ij} la quantité en tonnes de billes de plastique à transporter du site s_j au client i, $i = 1, ..., n$ et $j = 1, ..., m$.

Le problème est alors équivalent au PLNE suivant :

$$\text{Min} \sum_{j=1}^{m} c_j y_j + \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\sum_{j=1}^{m} x_{ij} = d_i, \qquad \text{pour tout } i \in \{1, ..., n\}, \tag{4.1}$$

$$\sum_{i=1}^{n} x_{ij} \leq M_j y_j, \qquad \text{pour tout } j \in \{1, ..., m\}, \tag{4.2}$$

$$y_j \in \{0, 1\}, \qquad \text{pour tout } j \in \{1, ..., m\},$$

$$x_{ij} \geq 0, \qquad \text{pour tout } i \in \{1, ..., n\} \text{ et } j \in \{1, ..., m\}.$$

Les contraintes (4.1) modélisent le fait qu'il faut couvrir la demande de chacun des clients. On peut noter que ces contraintes pourraient être de manière équivalente écrites $\sum_{j=1}^{m} x_{ij} \geq d_i$. Les variables de décision y_j, $j = 1, ..., n$ désignent donc le fait qu'une usine est implantée ou non. Ainsi les contraintes (4.2) indiquent que si le site s_j, $j = 1..., m$ n'est pas choisi pour l'implantation d'une usine, alors aucune production n'en sortira vers aucun des clients. Dans le cas contraire, cette production devra être limitée à M_j pour l'usine j.
On peut noter qu'il s'agit ici d'un problème mixte mêlant des variables continues et des variables de décision binaires.

4.3 Un peu plus dur

4.3.1 Verre de spin et Max-Cut

Modélisation d'un verre de spin

On utilisera dans cet exercice le problème de coupe de poids maximum qui peut se définir de la façon suivante. Soit $G = (V, E)$ un graphe non orienté et une fonction poids $w(e), e \in E$. Pour un sous-ensemble non vide de sommets $W \subset V$, on note $\delta(W)$ l'ensemble des arêtes avec exactement une extrémité dans W et on appelle cet ensemble une *coupe*. Le problème de coupe de poids maximum (max-cut) consiste à déterminer un ensemble d'arêtes $E' \subset E$ tel que E' soit une coupe et que $w(E') = \sum_{e \in E'} w(e)$ soit

maximum.

En physique statistique, on appelle *verre de spin* un état de la matière, caractérisé à l'échelle microscopique par une aimantation (moment magnétique ou spin) de chaque atome dans une direction particulière. A l'état de verre de spin, le système a des propriétés de supra-conducteur, comme par exemple d'être un puissant éléctro-aimant. On mesure la faculté d'un matériau à s'aimanter sous l'action d'un champ magnétique, par sa *susceptibilité magnétique*. En fait, l'état de verre de spin s'obtient à une température qui minimise la sucseptibilité magnétique du système.

On sait par exemple obtenir l'état de verre de spin pour des système magnétiques obtenusen diluant des atomes d'un matériau magnétique (Fer) dans un matériau non magnétique (Or) et en plaçant le système à la bonne température. Dans un tel système, la susceptibilité magnétique peut être définie comme la somme des énergie d'interaction magnétique entre les atomes de fers. Entre deux atomes de fer i, j, il existe une énergie d'interaction

$$H_{ij} = -J(d)S_iS_j$$

où S_i (resp. S_j) est le moment magnétique (spin) de l'atome i (resp. j) et $J(d)$ une fonction qui dépend de la distance d entre les deux atomes. Ce phénomène intrigue énormémement les physiciens et ils ont proposé diverses modèles pour comprendre le lien entre cette fonction de suceptibilité et l'état de verre de spin. Un des plus célèbres modèles est de considérer que les valeurs possibles pour leurs spins sont uniquement $+1$ ou -1 et que les interactions entre atomes n'ont lieu qu'entre les plus proches voisin. Ainsi le problème de minimisation de la fonction susceptibilité dépend uniquement de la configuration S des spins, c'est-à-dire une affection de $+1$ ou de -1 à chaque atome) et vaut alors

$$H(S) = \sum_{ij \in L} -J_{ij}S_iS_j$$

où L est l'ensemble des couples d'atomes ayant une interaction et J_{ij} un valeur constante (dépendant uniquement de la distance entre i et j).

Soit $G = (V, E)$ un graphe où les sommets correspondent aux atomes de fer et les arêtes aux liaisons de L. On associe à chaque arête ij le poids $w_{ij} = -J_{ij}$.
a) Montrer que le problème de minimisation de la fonction H pour un verre de spin revient à minimiser une fonction quadratique à valeur bivalente prise dans $\{-1, +1\}$.
b) Pour une affectation S de valeurs -1 ou $+1$ aux sommets de V, on définit les ensembles $V_+ = \{i \in V \mid S_i = +1\}$ et $V_- = \{i \in V \mid S_i = -1\}$. Montrer que ce problème se ramène au problème de la coupe maximum dans le graphe G. (Indication : $\sum_{ij \in E} w_{ij}$ est une quantité constante).

Formulation en nombres entiers

Soit $G = (V, E)$ un graphe non orienté et une fonction poids $c(e), e \in E$. On admet le résultat suivant :
Lemme 1 : Un ensemble E' d'arête de G est une coupe si et seulement si E' intersecte tout cycle de G en un nombre pair d'arête.

1) Soit E' une coupe de G. Montrer que pour tout cycle C de G et pour tout $F \subseteq C$ avec $|F|$ impair, $E' \cap C \neq F$.

2) Soit $E' \subseteq E$ un ensemble quelconque d'arêtes. On pose $\chi^{E'}(e) = \begin{cases} 1 \text{ si } e \in E' \\ 0 \text{ sinon.} \end{cases}$
Déduire que le vecteur $\chi^{E'}$ associé à une coupe E' de G vérifie l'inégalité suivante :

$$x(F) - x(C \setminus F) \leq |F| - 1 \quad \text{pour tout cycle } C, F \subseteq C, |F| \text{ impair.} \quad (1)$$

3) Montrer que déterminer une coupe de poids maximum dans G revient à résoudre le programme linéaire en nombres entiers suivant

$$(P) \begin{cases} \text{Maximiser } \displaystyle\sum_{e \in E} c(e)x(e) & & \\ \quad x(F) - x(C \setminus F) \leq |F| - 1, & \text{pour tout cycle } C,\ F \subseteq C,\ |F| \text{ impair}, & (1) \\ \quad 0 \leq x(e) \leq 1, & \text{pour toute arête } e \in E, & (2) \\ \quad x(e) \text{ entier}, & \text{pour toute arête } e \in E. & (3) \end{cases}$$

On appelle les contraintes de type (1) des *contraintes de cycles*, de type (2) des *contraintes triviales* et de type (3) des *contraintes d'intégrité*.

Deuxième partie

Résoudre

Chapitre 5

Résolution approchée ou exacte

5.1 Enumération

Pour les petits programmes, le fait d'avoir à déterminer une solution x^* qui soit supérieure à toute solution d'un ensemble discret fini pourrait laisser penser qu'il suffit d'énumérer les solutions entières du domaine de définition.

5.1.1 L'explosion combinatoire

Malheureusement, en fonction de la taille n d'un problème, il existe bien souvent $n!$ solutions. Le tableau suivant donne une idée des temps de calcul nécessaires pour différentes valeurs de n. Dans ce tableau, on suppose que traiter une solution prend le temps de $100n$ opérations élémentaires sur un des plus puissants calculateurs connus, c'est-à-dire Blue Gene construit par IBM en partenariat avec les Etats-Unis d'Amérique. La vitesse de calcul de cet ordinateur est évaluée en 2008 à 367 teraflops où 1 teraflop représente le traitement de mille milliards d'opérations par seconde. Ainsi pour $n = 10$, cet ordinateur peut traiter 367 milliards de solution du voyageur de commerce par seconde.

n	n !	Blue Gene	Ordinateur du futur
10	3628800	0.000098 sec	0,000000...1 sec
15	$1,3.10^{12}$	5,4 sec	0,00000...1 sec
20	$2,4.10^{18}$ 2,4 milliards de milliards	150 jours	
1000	10^{301}	...	...

Pour $n = 10$ ou 20, Blue Gene traite les solutions instantanément. Mais à chaque cran, $n!$ augmente très rapidement. Pour $n = 50$, Blue Gene met quelques minutes.

Et pour n=100, Blue Gene mettrait 14,3 milliards d'années, c'est-à-dire plus de la durée de vie du soleil ! Des calculs d'autant plus impossible, que $n = 100$ est une petite valeur pour ce genre de problèmes où l'on désire traiter des milliers, voir des millions de sommets. La dernière colonne du tableau représente une prospective sur les ordinateurs du futur. Imaginez que les ordinateurs connaissent prochainement une évolution aussi importante que celle qu'ils en ont connue depuis leur début, alors des problèmes avec $n = 50$ seront très rapidement traitées, mais le gain pour de plus grandes valeurs comme $n = 100$ ne sera pas suffisant pour changer l'ordre de grandeurs des durées de calcul. On appelle ce phénomène l'*explosion combinatoire*.

Elle est due à la complexité exponentielle des méthodes d'énumération, il existe heureusement de nombreux problèmes que l'on peut résoudre efficacement sans avoir recours à de telles méthodes énumératives !

La difficulté de résolution d'un problème à $n!$ solutions n'est donc pas un simple problème technique que des ordinateurs très puissants pourraient balayer. Il est nécessaire de contourner l'explosion combinatoire par des outils mathématiques et algorithmiques.

5.1.2 Cas polynomiaux

Il ne faudrait néanmoins pas confondre explosion combinatoire et difficulté/complexité d'un programme discret ! En effet, ce n'est pas parce qu'un PMD aurait un nombre exponentiel de solutions admissibles qu'il est difficile.
Il existe ainsi des cas de PMD que l'on sait résoudre en temps polynomial ! Il faut donc ainsi étudier de près leur complexité avant de se lancer dans des algorithmes d'énumération ou de branchement.

C'est le cas par exemple des problèmes de plus courts chemins (même non-linéaires) dans des graphes ou de flots : il existe un nombre exponentiel de solutions mais des algorithmes très efficaces et largement polynomiaux existent pour les résoudre : utiliser un modèle PMD et un solveur générique n'aurait ici que peu de sens.

5.2 Solutions approchées et garantie

Dans le but d'obtenir des algorithmes de branchement performants, il est nécessaire de rechercher de bonnes solutions pour le problème. Evidemment, cette démarche est également utile d'une manière générale.

5.2.1 Trouver des solutions réalisables

On dit souvent que tous les moyens sont bons pour découvrir une bonne solution d'un problème complexe comme les PMD. On peut diviser ces "moyens" en plusieurs catégories :

• heuristique spécifiques : suivant les problèmes une idée heuristique conduit parfois à de bonnes solutions (par exemple reproduire l'expérience d'un spécialiste)

• méta-heuristiques (Voir Annexe A pour davantage de détail) : il s'agit du cadre d'heuristiques qui reproduisent des principes généraux d'amélioration de solutions réalisables. Elles sont souvent utilisables pour des instances de très grandes tailles (là où les méthodes exactes échouent). On les divise en général entre :

- heuristique gloutonne : c'est le cadre des heuristiques où l'on ne remet jamais en question une décision prise précédemment. En général, pour les problèmes difficiles (NP-difficiles), ces heuristiques sont peu performantes. Il faut néanmoins noter qu'elles sont souvent les seules que l'on peut utiliser dans les cas où l'on doit traiter des données en temps réel (cas online). D'autre part, ces heuristiques sont parfois optimales (par exemple l'algorithme de Prim pour la recherche d'arbre couvrant).
- méthodes itératives qui tentent de reproduire le mécanisme des méthodes d'optimisation continue. Elles reposent sur la définition de voisinage : c'est-à-dire de déterminer comment passer d'une solution réalisable à une autre solution réalisable "voisine" dans le but d'en déterminer une meilleure. Les plus célèbres sont la descente stochastique, le recuit simulé et la méthode Taboo.
- méthodes évolutives Elles reproduisent des phénomènes biologiques ou physiques où l'on voit apparaître des solutions noouvelles en "mélangeant" des informations provenant d'un lot de solutions réalisables. A chaque itération de cette méthode, on obtient ainsi un nouveau lot de solutions réalisables construits à partir du lot précédent. Les plus célèbres : algorithme génétique, les colonies de fourmis, les essaims de particules...

• heuristique primale : dans le cadre des PMD, une relaxation continue fournie une solution fractionnaire (que l'on appelle solution primale) qui peut-être "proche" d'une solution entière réalisable. On peut ainsi imaginer en quelque sorte d'arrondir ces solutions fractionnaires. (Par exemple, dans le cas du problème de sac-à-dos : l'arrondi de la solution franctionnaire à l'entier inférieur donne une solution réalisable.) Ces méthodes s'appellent ainsi parfois *méthode d'arrondi* (rouding). Suivant le problème et l'arrondi utilisé, cette méthode ne fournit pas toujours une solution réalisable : il faut alors soit la corriger, soit la refuser. Il est aussi possible de "tirer au sort" un arrondi (random rounding).

On peut noter également que les moteurs d'inférences de la programmation par contrainte permettent parfois de déduire des solutions réalisables de manière générique et efficace. Dans ces cas, on peut construire des algorithmes de branchement efficace (appelés parfois méthode Dive&Fix).

5.2.2 Solution à garantie théorique

Les méthodes précédentes ont été décrites dans le but de donner des idées pour produire des solutions réalisables dites "approchées". Il est bien plus intéressant de produire des solutions pour lesquelles on peut indiquer son éloignement à l'optimum. On parle alors de *solutions garanties* (provably good solutions).

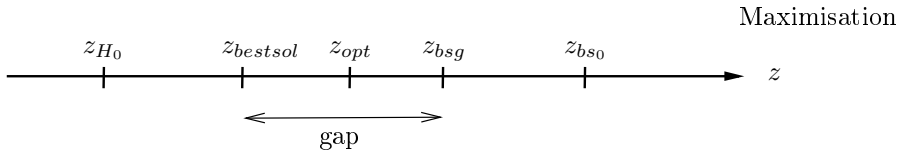
• *Relation Min-Max* : on appelle une relation Min-Max un encadrement de la solution optimale d'un problème par une borne min et une borne max obtenue par un même algorithme. Cet algorithme est le plus souvent obtenu en utilisant la théorie de la dualité ou des propriétés combinatoires particulière.

• *Algorithme d'approximation* : on appelle algorithme d'α-approximation un algorithme (polynomial ou à la rigueur efficace) donnant une solution qui est au pire à $\alpha\%$ de l'optimum. Dans le cas d'une maximisation, cela siginifie que la valeur z_{approx} est telle que $\alpha z_{opt} \leq z_{approx} \leq z_{opt}$.
Ces algorithmes sont le plus souvent très spécifiques et sont basés sur des principes algorithmes parfois très fins. Il est également possible que des technique d'arrondis ou des heuristiques primales puissent fournir des algorithmes d'approximation intéressant. Ces techniques demandent des preuves théoriques poussées et fournissent ainsi des algorithmes efficaces à garantie théorique. En revanche, ils ne donnent que rarement des solutions intéressantes.

5.2.3 Solution approchée avec garantie expérimentale

On parle de *garantie expérimentale* dans le cas où un algorithme permet de donner à la fois une solution approchée et une borne sur son écart à l'optimum. C'est le cas pour les algorithmes de branchement-évaluation. En effet, dans le cas maximisation, si l'on dispose d'une solution réalisable (borne inférieure) produite par une méthode heuristique initiale z_{H_0} et d'une évaluation du problème z_{bs_0}, on obtient un encadrement de la solution optimale z_{opt} : $z_{H_0} \leq z_{opt} \leq z_{bs_0}$. Au cours de l'algorithme, on découvre de nouvelles solutions réalisables et de meilleure borne (supérieure). En fait, à tout moment, on peut considérer la meilleure solution réalisable rencontrée $z_{bestsol}$ et la plus petite valeur de borne supérieure globale rencontrée z_{bsg}. On peut donc considérer tout

au long du déroulement de l'algorithme l'augmentation de la borne inférieure et la réduction de la borne supérieure : cela reduit l'écart.



On mesure cet écart en tant qu'écart relatif appelé souvent *gap* :

$$gap = \frac{z_{bsg} - z_{bestsol}}{z_{bestsol}}.$$

Ainsi, dans le pire des cas, la borne supérieure est égale à z_{opt}, ce qui signifie que la meilleure solution réalisable trouvée est au plus loin de l'optimum : on aurait ainsi $(1 + gap)z_{bestsol} = z_{opt}$ ce qui signifie qu'au pire $z_{bestsol}$ est à $gap\%$ de l'optimum.
On peut ainsi obtenir à tout moment une garantie expérimentale sur les solution approchées. Si le gap est peut important, par exemple inférieur à 5%, on peut souvent considérer que l'on a obtenu une très bonne solution (éventuellement même optimale !) : industriellement, les valeurs que l'on manipulent en entrée du problème sont parfois imprécises ou soumises à de fortes variations, il n'est pas systématiquement nécessaire d'obtenir une meilleure solution ! Inversement, si le gap est mauvais (supérieur à 5%), cela peut-être dû au fait que la solution approchée trouvée est mauvaise... mais cela peut aussi provenir d'une mauvaise évaluation.

5.3 Résolution exacte

Le contenu de ce module est clairement de présenter des outils permettant la résolution exacte des PMD. En revanche, nous présenterons également comment les tecniques exactes permettent également d'obtenir des valeurs approchées avec garantie lorsque les dimensions des problèmes deviennent trop importantes.

Chapitre 6

Algorithme de Branchement-Evaluation

Plutôt qu'énumérer, on peut tenter d'utiliser le paradigme informatique classique du "diviser pour régner" (divide&conqueer) consistant à diviser le problème en sous-problème plus simple à résoudre. Comme nous cherchons ici à optimiser une fonction, nous pourrons bénéficier de plusieurs moyens permettant d'éviter d'explorer tous les sous-problèmes.

6.1 Illustration par le problème du voyageur de commerce

Considérons le problème du voyageur de commerce afin d'introduire la méthode de Branch&Bound.

Considérons n villes $v_0, v_1.., v_n$ et les distances inter-villes d_{ij}, $i, j \in \{0, .., n\}$. Le problème du voyageur de commerce consiste à donner une tournée allant et revenant à la ville v_0 en passant une fois par chacune des villes. Une solution de ce problème s'écrit donc sous la forme d'une liste de villes données dans l'ordre de la tournée, que l'on peut noter $\sigma(1), ..., \sigma(n)$ où σ est une permutation des n villes. On a donc clairement $n!$ solutions possibles. Nous avons vu que l'explosion combinatoire d'une énumération la rend difficilement envisageable dans tous les cas. Néanmoins, dans cette section, nous nous intéressons aux méthodes énumératives, c'est-à-dire des méthodes qui explorent l'espace des solutions pour trouver une solution de meilleur coût.

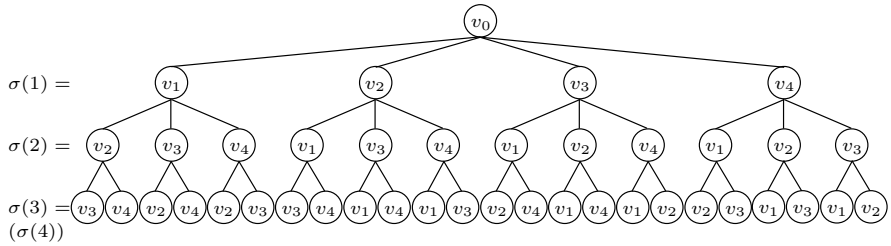
Les méthodes de Branch&Bound essayent d'éviter d'explorer entièrement l'espace des solutions en utilisant les caractéristiques des solutions optimales et des estimations

du coûts des solutions. Malheureusement, ces améliorations ne changent pas la complexité de ces méthodes qui restent de complexité exponentielle. C'est pourquoi il est souvent nécessaire de chercher d'autres techniques de résolution. Ce que nous verrons dans les chapitres suivants.

Prenons dans cet exemple 5 villes v_0, v_1, v_2, v_3 et v_4 et les distances inter-villes données par le tableau suivant :

	v_0	v_1	v_2	v_3	v_4
v_0	-	8	4	2	3
v_1	9	-	7	1	6
v_2	3	7	-	6	6
v_3	2	1	6	-	4
v_4	7	6	6	2	-

Pour construire une méthode énumérative, il faut choisir une façon d'énumérer les solutions de façon à n'en oublier aucune et en évitant de traiter plusieurs fois la même solution. Dans notre exemple, on peut considérer l'arborescence d'énumération suivante.



A chaque niveau de l'arborescence en partant de la racine, on fixe quelle ville sera visitée à la $i^{\text{ème}}$ place de la tournée ($i = 0$, puis $i = 1$, puis $i = 2$,...). Ainsi, chaque nœud correspond à une tournée partielle (et valide) des villes. Par exemple, le $4^{\text{ème}}$ nœud en partant de la gauche du niveau où l'on affecte la $2^{\text{ème}}$ place, correspond à la tournée partielle $\{v_0, v_2, *, *\}$. Au rang le plus bas, chaque feuille correspond à une tournée complète, c'est-à-dire à une solution du problème.
Notre but est donc de décider si l'exploration d'une branche est utile avant d'en énumérer toutes les solutions. Pour cela, on va donner une *évaluation* des solutions que peut porter une branche, c'est-à-dire donner une borne inférieure des coûts des solutions de la branche. Si cette évaluation est plus grande qu'une solution que l'on connaît déjà, on peut *élaguer* cette branche, c'est-à-dire éviter de l'explorer.

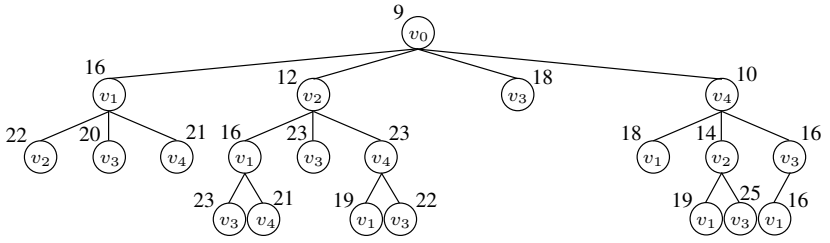
Dans notre exemple du voyageur de commerce, avant de commencer l'énumération, tentons de trouver une bonne solution. Par exemple, on peut prendre une heuristique

gloutonne qui consiste à prendre en premier la ville la plus près de v_0, puis en deuxième une ville (parmi les villes restantes) qui soit la plus près de la première, et ainsi de suite. On obtient ainsi une solution réalisable $s_1 = \{v_0, v_3, v_1, v_4, v_2\}$ de coût $c_1 = 18$. Une solution réalisable nous donne une bonne supérieure pour notre problème, en effet, nous savons que la solution optimale du problème est au plus de coût 18.

Il nous serait à présent utile de trouver une évaluation du coût de la solution par une borne inférieure. En effet, si l'on pouvait par exemple déterminer que la solution optimale est au moins de coût 18, nous aurions déjà résolu notre problème. Une idée possible ici est de dire que l'on passe par toutes les villes et que l'on doit donc sortir de chacune des villes pour aller vers une autre ville. En d'autre terme, le trajet pour aller d'une ville v_i à une autre ville sera au moins égale à la plus petite distance issue de v_i. Ainsi, en sommant les coûts minimaux de chaque ligne du tableau, on obtient une valeur qui sera toujours inférieure au coût d'une solution du problème. Par exemple, ici on obtient une évaluation de 2+1+3+2+2=9. Une solution de ce problème sera donc au moins de coût 9. Malheureusement, la meilleure solution que nous connaissons est de coût 18, nous devons donc commencer l'énumération.

La première branche à gauche issue de la racine de l'arborescence contient toutes les solutions où v_1 est affectée à la place 1. Si l'on applique notre idée d'évaluation à la sous-matrice obtenue en ôtant la ligne correspondant à v_0 (on connaît déjà la ville qui suit v_0 et la colonne correspondant à v_1 (la ville allant en v_1 est déjà fixée, c'est v_0), on obtient alors 8 pour la somme des coûts minimaux des lignes. En ajoutant le coût $c_{v_0v_1} = 8$, on sait alors que toute solution de cette branche a au moins une coût de 16. On explore alors le noeud de l'arborescence commençant par $\{v_0, v_1, v_2\}$. Cela correspond à une matrice où l'on a ôté les lignes correspondant à v_0 et v_1 et les colonnes correspondant à v_1 et v_2. La somme des coûts minimaux des lignes restantes est alors de 7. En ajoutant alors le coût $c_{v_0v_1} + c_{v_1v_2} = 15$, on sait que toute solution de cette branche est au moins de 22. Comme nous connaissons déjà une solution de coût 18, on peut donc éviter d'explorer (élaguer) cette branche.

La figure suivante résume l'exploration complète de notre arborescence par un parcours en profondeur. Les valeurs qui accompagnent les nœuds sont les évaluations des branches et celles qui accompagnent les feuilles sont les coûts des solutions associées.



Lors de cette exploration, on a pu élaguer totalement la 3$^{\text{ème}}$ branche. C'est la

dernière branche qui nous a conduit à une feuille correspondant à une solution $s_2 = \{v_0, v_4, v_3, v_1, v_2\}$ de coût 16. Or, la dernière branche ne peut pas mener à une solution de meilleur coût que 16, on a pas besoin d'explorer cette dernière branche. En conclusion, la meilleure solution pour le problème est de coût 16 et nous connaissons s_2 une des solutions optimales possibles.

6.2 Définitions et algorithme de Branch&Bound

Nous considérons ici les algorithmes de Branch&Bound pour un PMD où l'on recherche une solution maximale optimale x dans $\{1, ..., k\}^n$. Pour un minimisation, il suffit de remplacer tous les maxima par des minima et inversement.

6.2.1 Sous-problèmes, évaluation et stérilité

La résolution d'un problème complexe peut exiger de découper l'espace de ses solutions de manière à remplacer ce problème par plusieurs *sous-problèmes* identiques au premier mais ayant des espaces de solutions réduits. Ces sous-problèmes pouvant eux-mêmes se diviser en d'autres sous-problèmes, on parle alors d'une arborescence de sous-problèmes.

On appelle *branchement* le fait de diviser un problème en un ensemble de sous-problèmes tels que l'union de leurs espaces de solutions forme l'espace des solutions du problème-père. On appelle les sous-problèmes obtenus des *problèmes-fils*. On appelle parfois le branchement une *séparation* mais ce nom est dangereux car confondu avec la séparation des algorithmes de coupes.

Le branchement le plus courant consiste à choisir l'une des variables x_i puis à définir un problème-fils pour chaque valeur de x_i prise dans $\{1, ..., k\}$. Il est souvent utile de considérer des branchements plus généraux où l'on branche sur des sous-problèmes plus larges ($x_1 <= 3\,;\ x_1 > 3$) par exemple ou même sur des contraintes plus complexes ($x_1 + x_2 <= 5\,;\ x_1 + x_2 > 5$).

Considérons à présent un sous-problème (qui peut-être tout le programme), on peut alors lui donner plusieurs caractéristiques :
- on appelle *évaluation* une borne supérieure de la valeur optimale de ce sous-problème. Il est important de remarquer qu'alors cette évaluation est également une évaluation de tous les sous-problèmes issus de ce sous-problème. On étudiera dans une section suivante comment obtenir de bonne évaluation.
- un sous-problème est dit *stérile* dans deux cas :
  - soit il ne contient aucune solution (il est infaisable).

- s'il est faisable et qu'on en connaît la meilleur solution (on parle alors de la *solution associée* ou *optimum local* de ce sous-problème). A force de brancher, il devient facile de déterminer une solution à un sous-problème (par exemple s'il est une feuille de l'arborescence).

6.2.2 Efficacité d'un algorithme de branchement

Toute l'efficacité d'un algorithme de branchement repose en fait sur l'utilisation d'une comparaison entre des évaluations et minimums locaux et globaux. En effet :
- la meilleure des solutions valides rencontrées au cours de l'exploration constitue une borne inférieure pour le PMD.
- dans l'arborescence, si l'on considère la plus grande des évaluations (c'est-à-dire des bornes supérieures) d'un niveau, on obtient alors une borne supérieure pour le PMD que l'on appelle alors *borne supérieure globale.*

Ainsi, on peut agir sur l'exploration des sous-problèmes en *élaguant* certains sous-problèmes, c'est-à-dire que l'on explore pas ou que l'on stoppe l'exploration de la branche issue de ce problème car lorsque l'évaluation de ce sous-problème est inférieure à la meilleure solution connue pour le PMD entier.

L'autre aspect qui permet d'explorer moins de sous-problèmes est de permettre l'apparition de nombreux sous-problèmes stériles, ou plus précisemment de sous-problème pour lesquelles on connaît l'optimum local. Ceci peut s'obtenir par exemple en ayant une méthode d'heuristique pour produire ce minimum qui correspond à la borne supérieure locale. Pour le cas de la PLNE, on peut aussi avoir des PL donnant directement la solution entière : il faut pour cela avoir un PL entrant dans l'un des cas polynomiaux (voir sections suivantes).

Un autre point important de l'efficacité d'un algorithme est de pouvoir contourner le problème des symétries. Deux solutions sont symétriques si elles s'obtiennet l'une de l'autre en échangeant des valeurs des variables et si elles ont même coût. Un algorithme de branchement va devoir explorer toutes les solutions symétriques proches de l'optimum. Plusieurs techniques récentes permettent de contourner cette exploration : pertubations, branchement orbital,...

6.2.3 Schéma de l'algorithme

On appelle *algorithme de Branch&Bound* ou *séparation-évaluation* (B&B) les algorithmes de branchement utilisant les notions de stérilité, borne inf et borne sup afin

d'éviter l'exploration systématique de toute l'arborescence.

Macro-algorithme de B&B

On note ici L la structure de données des sous-problèmes à explorer

(1) L← Problème correspondant au PMD
(2) Tant que L non vide faire
(3) Extraire un sous-problème P de L.
(4) Si évaluation de P est $>$ à la borne inf alors
(5) Créer les problèmes-fils P_i de P
(6) Pour chaque P_i faisables dont l'évaluation est $>$ à la borne inf faire
(7) Si on dispose d'une solution associée à P_i alors mise à jour borne inf.
(8) Sinon ajout de P_i dans L.
(9) Fin Pour.
(10) Fin Si.
(11) Fin Tant que.

Ce macro-algorithme ne décrit pas tout le fonctionnement d'un B&B, il faut préciser :
• la nature de la structure de données L : Il s'agit d'une simple liste de sous-problème. Par contre, le codage de chaque sous-problème est important : les sous-problème étant liés entre eux par une arborescence, on peut remarque que beaucoup des informations d'un sous-problème sont redondantes par rapport aux autres : la meilleure façon de stocker ces informations est une structure d'arbre. Ainsi L est une liste des noeuds d'un arbre. Lors de l'élagage (ligne 4), il peut aussi vouloir simultanément vider partiellement L de tous les problèmes d'évaluation strictement supérieure à la borne inf : cela demande de mettre à jour la structure de l'arbre. Il existe de nombreuses astuces pour réduire l'espace mémoire nécessaire et accélérer les opérations sur les structures de données.
• La création de sous-problème (ligne 5) est souvent appelé "le choix du branchement" : il s'agit dans le cas le plus simple de choisir quelle variable sera utilisée pour brancher (dite alors variable de séparation).
• L'ordre dans lequel sont traités les sous-problèmes stockés dans la structure L a une influence sur le fonctionnement de l'algorithme. On parle alors de *stratégie de branchements*.

La stratégie de branchement est cruciale pour une bonne exploration. Les stratégies classiques consistent soit à parcourir l'arborescence des sous-problèmes en largeur (width first search), en profondeur (deep first search) ou en recherchant le sous-problème non traité de meilleur évaluation (best first search).

Lorsque l'on dispose d'une fonction d'évaluation, il peut sembler judicieux d'opter pour la politique d'exploration du meilleur élément (en fait, statistiquement, elle n'est pas forcément meilleure que les autres). On utilise alors une structure de tas pour coder L. Les solveurs utilisent en général une stratégie consistant à effectuer d'abord quelques descentes en profondeur dans l'arbre afin d'obtenir des bornes inf, puis une exploration par meilleure évaluation.
On peut noter que l'exploration en profondeur ne nécessite pratiquement pas de mémoire : en effet, l'exploration des branches les unes après les autres n'exigent pas de retenir tout l'arbre. On appelle alors l'algorithme de B&B l'algorithme *d'énumération implicite*.

Pour finir notons qu'il est très utile de posséder une bonne heuristique pour le problème P avant de lancer l'algorithme de branchement. En effet, une heuristique fournit une solution réalisable pour P et par conséquent une borne inf qui permet de stériliser un plus grand nombre de problèmes. On appelle ces méthodes des heuristiques. Il est également possible d'utiliser une solution fractionnaire pour déterminer une solution réalisation : on parle alors d'*arrondi* ou d'*heuristique primale*.

6.3 Programmation dynamique et dominance

La structure d'un problème permet fréquemment d'éviter une exploration systématique comme l'effectue un algorithme de B&B. On peut considérer deux cas.

Lorsque les valeurs des solutions sont liées entre elles par une formule de récurrence basée sur le principe suivant. Supposons qu'une solution d'un problème est déterminé par une suite de décisions $D_1, D_2, ...D_p$: l'hypothèse dite de la *programmation dynamique* est qu'une prise de décisions optimales $D_1, ...D_p$ est telle que, pour tout entier $k = 1, ...p-1$, les décisions $D_{k+1}, ...D_p$ doivent être optimales. Cette hypothèse est très forte car elle a pour conséquence de donner un ordre à l'exploration des solutions en éliminant l'exploration de certaines. De plus, sous certaines conditions à prouver, elle permet même de montrer qu'il n'y a qu'un nombre polynomial de sous-problèmes (appelés alors *états* à explorer.
On peut citer en exemple l'algorithme de Dijkstra pour les plus courts chemins ou la résolution des cas polynomiaux du problème du voyageur de commerce.

Lorsque les solutions ont des structures fortement liées entre elles, on peut détecter des "dominances" entre solutions : si dans une arborescence, on peut montrer que tout sous-problème descendant d'un sous-problème a est au moins aussi bon que tout descendant d'un sous-problème b : on dit que a domine b. Dans ce cas, il suffit d'explorer

a : on dit que a tue b (cela revient à stériliser b dès que a est explorer par exemple). Ces dominances se rencontrent dans les problèmes à forte structure, par exemple des problèmes d'ordonnancement ou d'optimisation sur les graphes, où les solutions ont des points ou des parties en communs.

6.4 Branchement pour les PMD et pré-traitement

Un algorithme de branchements pour les PMD demande de mettre en pratique les pricipes décrits précédemment : il faut étudier précisément quelles méthodes d'évaluation, d'heuristique, de stratégie de branchement, de choix de branchement etc.

On peut néanmoins vouloir construire un algorithmes de branchement générique pour les PMD, ces algorithmes respectent alors les choix suivants :
- l'évaluation est obtenue par relaxation continue du PMD (cela limite donc ce genre d'algorithme aux PLNE et aux Programme quadratique).
- la stratégie de branchement consiste à effectuer un peu au hasard des explorations en profondeur de l'arbre afin de générer des solutions réalisables qui produiront des bornes inf (en maximisation). Ensuite, l'algorithme réalise une exploration par "meilleur noeud d'abord".
- le branchement est effectué sur les variables, en choisissant la variables la "plus fractionnaire". Par exemple, pour des variables binaire, on choisit la variable de valeur la plus proche de 0,5. (En pratique, les solveurs branche plutôt sur la première variable fractionnaire rencontrée).

Il est important de noter que les différences de capacité entre les algorithmes génériques sur les PMD (c'est-à-dire les capacités des produit Cplex, Scip ou Gurobi) proviennent de bonnes implémentations des algorithmes de branchements, mais pas seulement.
Une bonne part de leurs capacités proviennent de techniques de *prétraitement* sur les variables et les contraintes. Ces prétraitements sont des algorithmes qui explorent la structure du problèmes pour rechercher par exemple des inférences numériques (dominances par exemple) ou logique (si les variables sont binaires) : ces algorithmes sont basées sur des moteurs de programmation par contraintes. Ces prétraitements permettent ainsi de réduire le nombre de variables et de contraintes : dans les problèmes "réels" tirés d'applications industrielles, ces prétraitement sont même parfois capables de réduire la taille des PMD de manière étonnante.
Enfin, une dernière avancée théorique récente sur la gestion des symétries dans les branchements de PMD ont permis d'obtenir des solutions pour des PMD "non-linéaires".

Néanmoins l'augmentation des capacités récentes des solveurs de PLNE provient

surtout de méthodes théoriques que nous étudieront dans les parties suivantes.

6.5 Exercices

Les exercices suivants sur le branchement peuvent être réalisés “à la main”.

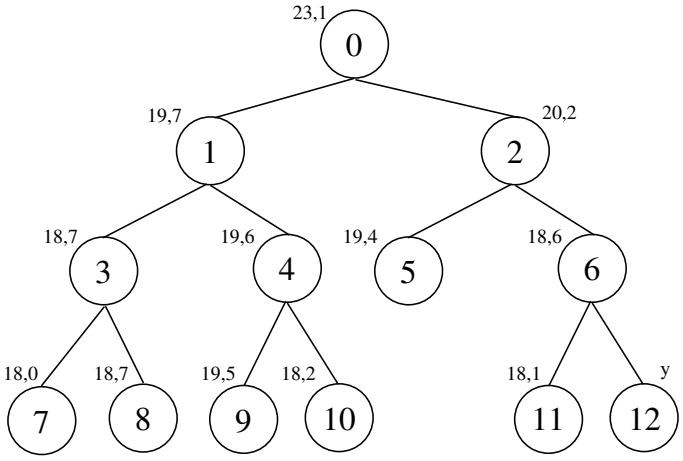
6.5.1 Représentation graphique

Résoudre le PLNE suivant en utilisant une représentation graphique des solutions :

$$\begin{aligned} \text{Min } & 4x_1 + 5x_2 \\ & x_1 + 4x_2 \geq 5 \\ & 3x_1 + 2x_2 \geq 7 \\ & x_1 \geq 0, x_2 \geq 0 \\ & x_1, x_2 \in I\!N. \end{aligned}$$

6.5.2 Arborescence à compléter

Dans l'exploration d'un arbre de branchements d'un problème en nombres entiers où l'on désire maximiser une fonction objectif, on se retrouve, à un instant donné, à avoir exploré les sous-problèmes donnés par l'arbre ci-dessous.



Les cercles représentent des sous-problèmes traités. Leur numéro est là uniquement pour leur donner un nom. La valeur indiquée au-dessus d'un sous-problème est l'évaluation de ce sous-problème.

1) Si, après évaluation du sommet 12, son évaluation y est inférieure ou égale à 18,1, quelle sera alors la valeur de la meilleure borne supérieure pour le problème ? Justifiez votre réponse.
2) Cette exploration de l'arbre de branchement est effectuée "au meilleur sous-problème d'abord". Quel est le sous-problème qui sera traité après le sous-problème 12 ?
3) On dispose en fait d'une solution entière valide pour le problème qui a pour valeur 19. Que peut-on en déduire pour la suite de l'exploration ?
4) Pour le sous-problème 7, l'évaluation 18 a été obtenue par une solution entière du problème, que peut-on déduire pour la suite de l'exploration ?

6.5.3 Sac-à-dos binaire

On peut résoudre le PLNE à une seule contrainte (dit "de sac-à-dos entier") en utilisant le fait que la relaxation linéaire d'un tel PLNE peut-être faite facilement. En effet, considérons un PLNE respectant la forme suivante :

$$\begin{aligned} \text{Max } & c_1x_1 + c_2x_2 + c_3x_3 + ... + c_nx_n \\ & a_1x_1 + a_2x_2 + a_3x_3 + ... + a_nx_n \leq b \\ & x_i \in \{0,1\} \quad \forall i \in \{1,...,n\}. \end{aligned}$$

Supposons que les conditions suivantes soient respectées :

$$b > 0\,;\; c_i, a_i > 0 \text{ pour } i = 1,...,n\,;\; \sum_{i=1}^{n} a_i > b$$
$$\text{et } \frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \frac{c_3}{a_3} \geq ... \geq \frac{c_n}{a_n}.$$

On peut alors définir l'entier $k \in \{1,...,n-1\}$ tel que $\sum_{i=1}^{k} a_i \leq b$ et $\sum_{i=1}^{k+1} a_i > b$.
La solution suivante est optimale pour la relaxation continue du PMD :
$x_i^* = 1$ pour $i = 1,...,k$
$x_k^* = \frac{b - \sum_{i=1}^{k} a_i}{a_{k+1}}$
$x_i^* = 0$ pour $i = k+2,...,n$.
Utiliser cette relaxation pour résoudre l'instance suivante :

$$\begin{aligned} \text{Max } & 16x_1 + 22x_2 + 8x_3 + 12x_4 \\ & 5x_1 + 7x_2 + 3x_3 + 4x4 \leq 14 \\ & x_i \in \{0,1\} \quad i = 1,...,4. \end{aligned}$$

Chapitre 7

Evaluation par relaxation

Les bonnes performances de la PMD reposent essentiellement sur sa capacité à fournir des bornes intéressantes (borne inf en maximisation et borne sup en minimisation) : ces bornes sont obtenues par *relaxation* : on appelle relaxation d'un PMD :
- soit le fait d'élargir l'ensemble des solutions de manière à optimiser sur un espace plus large (et plus facile à optimiser) : cela comporte la relaxation de contraintes (et en particulier la relaxation continue).
- soit le fait de remplacer la fonction objective par une autre fonction ayant la même valeur optimale (par exemple la relaxation lagrangienne).

7.1 Relaxation continue

Pour un PMD, la relaxation continue est une relaxation naturelle qui est, pour la PLNE (par l'algorithme du simplexe, des points intérieurs ou du volume) et la programmation quadratique (avec matrice définie positive) très efficace. Néanmoins, nous verrons que la relaxation continue n'est pas forcemment la meilleure (voir chapitre suivant sur la relaxation lagrangienne).

Il est également possible d'améliorer la borne obtenue par relaxation continue en utilisant des idées spécifiques au problème.

En ce qui concerne plus spécifiquement la PLNE, on peut améliorer la valeur de relaxation linéaire en ajoutant des contraintes, le *renforcement* ou des variables, la *reformulation*. Nous verrons cela dans les parties suivantes.

7.2 Relaxation lagrangienne

Un cadre particulièrement intéressant pour la relaxation (mais aussi pour la décomposition voir partie suivante) est celui donné par la dualité lagrangienne. Son principe consiste à ajouter des termes d'une inégalité dans la fonction objective en ajoutant un multiplicateur visant à pénaliser la fonction objective si la solution trouvée ne vérifie pas cette inégalité. Cette relaxation fournit fréquemment de très bonne valeur de relaxation, bien souvent meilleure que la relaxaton continue.

Un cadre particulièrement intéressant pour la relaxation (mais aussi pour la décomposition voir partie suivante) est celui donné par la dualité lagrangienne.

7.2.1 Définitions et résultats généraux

On considère ici un PM (qui peut être un PMD) écrit de la façon suivante :

$$\begin{array}{ll} \text{Min } f(x) & \\ g_i(x) \leq 0 & i \in I \\ x \in S \subset I\!R^n. & \end{array}$$

On associe à chaque contrainte $i \in I$ une valeur réelle $\lambda_i \geq 0$ que l'on appelle *multiplicateur de Lagrange*. On considère également le vecteur réel $\lambda = (\lambda_i, i \in I)$.
On pose alors la *fonction de Lagrange*

$$L(x, \lambda) = f(x) + \sum_{i \in I} \lambda_i g_i(x)$$

qui est une fonction qui a pour paramètres les deux vecteurs x et λ et prend ses valeurs dans $I\!R$. On suppose ici que le PM est tel que le problème $Min_{x \in S}\{L(x, \lambda)\}$ a une solution optimale pour tout $\lambda \geq 0$.

Définition 7.2.1 Pour des vecteurs $\bar{x}$ et $\bar{\lambda} \geq 0$ donnés, on dit que le couple $(\bar{x}, \bar{\lambda})$ est un *point-selle* de la fonction de Lagrange si

$$\begin{array}{rcccll} L(\bar{x}, \lambda) & \leq & L(\bar{x}, \bar{\lambda}) & & & \forall \lambda \geq 0 \\ & & L(\bar{x}, \bar{\lambda}) & \leq & L(x, \bar{\lambda}) & \forall x \in S. \end{array}$$

En d'autre terme, un point-selle $(\bar{x}, \bar{\lambda})$ est tel que la fonction L pour x fixé à $\bar{x}$ atteind un maximum en $\bar{\lambda}$ et que la fonction L pour λ fixé à $\bar{\lambda}$ atteind un minimum en $\bar{x}$. On a alors le résultat suivant pour caractériser les points-selles :

Théorème 7.2.2 *Etant donnés des vecteurs $\bar{x}$ et $\bar{\lambda} \geq 0$, le couple $(\bar{x}, \bar{\lambda})$ est un point-selle de la fonction de Lagrange si et seulement si*

i) $L(\bar{x}, \bar{\lambda}) = Min_{x \in S}\{L(x, \bar{\lambda})\}$
ii) $g_i(\bar{x}) \leq 0$, $\forall i \in I$,
iii) $\bar{\lambda}_i g_i(\bar{x}) = 0$, $\forall i \in I$.

Preuve : ($\Rightarrow$) La condition i) se déduit par construction d'un point-selle.
Par définition, on sait que $\forall \lambda \geq 0$, $L(\bar{x}, \lambda) \leq L(\bar{x}, \bar{\lambda})$, ce qui peut s'écrire également

$$\sum_{i \in I} \lambda_i g_i(\bar{x}) \leq \sum_{i \in I} \bar{\lambda}_i g_i(\bar{x}) \quad \forall \lambda \geq 0. \quad (*)$$

Supposons que ii) soit faux et qu'il existe $i_0 \in I$ tel que $g_{i_0}(\bar{x}) > 0$, dans ce cas, en prenant λ_{i_0} très grand, on ne vérifierait pas (*), une contradiction. Donc ii) est vérifié.
En appliquant (*) avec $\lambda = 0$, on obtient $\sum_{i \in I} \bar{\lambda}_i g_i(\bar{x}) \geq 0$, or comme par définition $\bar{\lambda}_i \geq 0$ et $g_i(\bar{x}) \leq 0$ pour tout $i \in I$, on obtient donc iii).
($\Leftarrow$) Par la propriété i), on sait que pour tout $x \in S$, $L(\bar{x}, \bar{\lambda}) \leq L(x, \bar{\lambda})$, c'est-à-dire une partie de la définition d'un point-selle.
Par ii), pour tout $\lambda \geq 0$, $L(\bar{x}, \lambda) = f(\bar{x}) + \sum_{i \in I} \lambda_i g_i(\bar{x}) \leq f(\bar{x})$. De plus, par iii), comme $\bar{\lambda}_i g_i(\bar{x}) = 0$, $\forall i \in I$, on obtient que $L(\bar{x}, \bar{\lambda}) = f(\bar{x})$.
Par conséquent, $L(\bar{x}, \lambda) \leq f(\bar{x}) = L(\bar{x}, \bar{\lambda})$. Donc $(\bar{x}, \bar{\lambda})$ est un point-selle pour L. □

Le théorème suivant permet de faire le lien entre point-selle et optimum du PMD.

Théorème 7.2.3 *Etant donné un point-selle* $(\bar{x}, \bar{\lambda})$ *pour* L *défini par les vecteurs* $\bar{x}$ *et* $\bar{\lambda} \geq 0$, *alors* $\bar{x}$ *est un optimum global pour le PMD.*

Preuve : En appliquant le théorème précédent, on sait par i) que pour tout $x \in S$

$$f(\bar{x}) + \sum_{i \in I} \bar{\lambda}_i g_i(\bar{x}) \leq f(x) + \sum_{i \in I} \bar{\lambda}_i g_i(x)$$

Or par iii), on obtient

$$f(\bar{x}) \leq f(x) + \sum_{i \in I} \bar{\lambda}_i g_i(x)$$

et par ii), comme de plus $\lambda \geq 0$, on obtient finalement que $f(\bar{x}) \leq f(x)$ pour tout $x \in S$. □

Ces théorèmes permettent ainsi d'étudier la fonction de Lagrange pour résoudre le PM. Cette relaxation ramène parfois à des problèmes d'optimisation continue en ôtant des contraintes difficiles.
Néanmoins, cette étude n'est possible que pour les PM oú il existe un point-selle pour L : or il n'existe pas de point-selle pour les PMD ! On va néanmoins pouvoir utiliser la fonction de Lagrange dans le contexte de la dualité lagrangienne.

7.2.2 Dualité lagrangienne

On considère la fonction, dite *duale lagrangienne*, suivante :

$$w(\lambda) = \inf_{x \in S} \{L(x, \lambda)\} \quad \forall \lambda \geq 0.$$

Dans le cas qui nous intéresse ici, c'est-à-dire des ensembles S discrets finis, cette fonction peut s'écrire plus simplement

$$w(\lambda) = \min_{x \in S} \{L(x, \lambda)\} \quad \forall \lambda \geq 0.$$

On définit alors le *problème dual (lagrangien)* du PM :

$$\begin{array}{l} \text{Max } w(\lambda) (= \text{Max}_\lambda \{\text{Min}_{x \in S} L(x, \lambda)\}) \\ \lambda \subset I\!R^m_+. \end{array}$$

En fait, ce problème dual correspond au fait de rechercher un point-selle pour L. On peut néanmoins noter que ce problème est défini même s'il n'existe pas de point-selle (comme c'est le cas pour un PMD).

On a alors les théorèmes suivants de la dualité.

Théorème 7.2.4 *(Théorème de la dualité faible)*
Si l'on note $f(x^*)$ l'optimum global de f et $w(\lambda^*)$ l'optimum global du problème dual, alors
Pour tout $\lambda \geq 0$, $w(\lambda) \leq w(\lambda^*) \leq f(x^*)$.

Preuve : Soit $\lambda \geq 0$. Par définition de la fonction duale, on a $w(\lambda) \leq f(x) + \sum_{i \in I} \lambda_i g_i(x)$, pour tout $x \in S$. Comme $\lambda g(x) \leq 0$ pour tout $x \in S$, $w(\lambda^*) \leq f(x)$.
En particulier, pout x^* et λ^* solutions optimales, on a donc $w(\lambda) \leq w(\lambda) \leq f(x^*)$. □

Théorème 7.2.5 *(Concavité de la fonction duale)*
La fonction duale $\lambda \mapsto w(\lambda)$ est une fonction concave de λ.

Preuve : Soit λ^1 et λ^2 deux vecteurs réels positifs et soit $t \in [0, 1]$. Posons alors $\lambda = t\lambda^1 + (1-t)\lambda^2$. Pour ce λ donné, on sait par hypothèse qu'il existe un $\bar{x}$ tel que

$$w(\lambda) = f(x) + \sum_{i \in I} \lambda_i g_i(x).$$

D'autre part, par définition de w appliquée à λ^1 et $lambda^2$, on a

$$w(\lambda^1) \leq f(\bar{x}) + \sum_{i \in I} \lambda^1_i g_i(\bar{x}) \quad \text{et} \quad w(\lambda^2) \leq f(\bar{x}) + \sum_{i \in I} \lambda^2_i g_i(\bar{x})$$

Par combinaison linéaire des deux inégalités, on obtient donc

$$tw(\lambda^1) + (1-t)w(\lambda^2) \leq f(\bar{x}) + \sum_{i \in I} \left(t\lambda_i^1 + (1-t)\lambda_i^2\right) g_i(\bar{x})$$

Par conséquent,

$$tw(\lambda^1) + (1-t)w(\lambda^2) \leq f(\bar{x}) + \sum_{i \in I} \lambda_i g_i(x) = w(\lambda).$$

□

Ainsi la fonction w est concave, ce qui permet d'utiliser des outils efficaces de l'optimisation continue. Il est important de noter qu'aucune hypothèse n'a été faite sur la nature des fonctions f et g_i qui peuvent être quelconque, ni sur la convexité de l'ensemble S.

Dans le cas des ensembles S discrets, la fonction f n'est pas différentiable en tout point mais la concavité de w permet cependant de montrer que tout optimum local de w est global : ce qui fait que le problème dual est en général plus facile à résoudre que le problème dual.

Enfin, ce dernier théorème indique que, si le PM admet un point-selle, optimiser w revient à optimiser f.

Théorème 7.2.6 *(Théorème de la dualité)*
Si le problème PM admet un point-selle (x^*, λ^*) alors $w(\lambda^*) = f(x^*)$.
Inversement, s'il existe x^* et $\lambda^* \geq 0$ tels que $w(\lambda^*) = f(x^*)$, alors le PM admet un point-selle et (x^*, λ^*) est un tel point-selle.

En fait, cette théorie de la dualité permet de ramener, lorsqu'il y a des points-selles, le PMD a un la minimisation d'une fonction concave. De plus, on sait facilement utiliser des méthodes de sous-gradient pour cette fonction.

Remarque : la dualité lagrangienne appliquée à un programme linéaire est en fait équivalente à la dualité "classique" de la PL.

Malheureusement, comme un PMD ne possède pas en général de point-selle, ce dernier théorème n'est cependant pas applicable pour S discret, on verra dans la section suivante que l'écart $f(x^*) - w(\lambda^*)$ est souvent non nul.

7.2.3 Saut de dualité

On appelle *saut de dualité* le fait que la valeur $f(x^*) - w(\lambda^*)$ soit non nul. C'est le cas lorsqu'un PM ne possède de point-selle pour sa fonction duale associée. C'est le cas

par exemple de la plupart des PMD.

Prenons par exemple le PLNE suivant où $S = \{0,1\}^3$

$$\begin{aligned} \text{Min } & f(x) = 10 - 3x_1 - 2x_2 - x_3 \\ & 2x_1 + 3x_2 + 4x_3 \leq 4 \\ & x = (x_1, x_2, x_3) \in \{0,1\}^3. \end{aligned}$$

L'ensemble des points de S est $y^1 = (0,0,0)$, $y^2 = (1,0,0)$, $y^3 = (0,1,0)$, $y^4 = (1,1,0)$, $y^5 = (0,0,1)$, $y^6 = (1,0,1)$, $y^7 = (0,1,1)$ et $y^8 = (1,1,1)$. Parmi elles, on trouve une solution optimale au PLNE pour $y^2 = (1,0,0)$ qui vaut $f(y^2) = 7$.

En posant $g(x) = 2x_1 + 3x_2 + 4x_3 - 4$, on définit la fonction de Lagrange pour chaque point y^i de S comme étant

$$L(y^i, \lambda) = f(y^i) + \lambda g(y^i) = 10 - 3y_1^i - 2y_2^i - y_3^i + (2y_1^i + 3y_2^i + 4y_3^i - 4)\lambda \quad \forall \lambda \in I\!R_+,$$

c'est-à-dire

$$\begin{array}{ll} L(y^1,\lambda) = 10 - 4\lambda \quad \forall \lambda \in I\!R_+, & L(y^5,\lambda) = 9 \quad \forall \lambda \in I\!R_+, \\ L(y^2,\lambda) = 7 - 2\lambda \quad \forall \lambda \in I\!R_+, & L(y^6,\lambda) = 6 + 2\lambda \quad \forall \lambda \in I\!R_+, \\ L(y^3,\lambda) = 8 - \lambda \quad \forall \lambda \in I\!R_+, & L(y^7,\lambda) = 7 + 3\lambda \quad \forall \lambda \in I\!R_+, \\ L(y^4,\lambda) = 5 + \lambda \quad \forall \lambda \in I\!R_+, & L(y^8,\lambda) = 4 + 5\lambda \quad \forall \lambda \in I\!R_+. \end{array}$$

La fonction duale du PMD est $w(\lambda) = Min_{i \in \{1,\ldots,8\}}\{L(y^i, \lambda)\} \quad \forall \lambda \geq 0$ qui est facile à calculer pour un λ donné. On obtient par exemple $w(0) = 4$, $w(0,5) = 5,5$ et $w(1) = 5$. On veut en fait résoudre le programme dual qui est alors $Max_{\lambda \geq 0} w(\lambda)$. Or la fonction w est concave ! On sait donc par ces trois valeurs (en 0, 0,5 et 1) que son optimum sera atteint sur l'intervalle $[0,1]$. On peut ainsi, par exemple par recherche dichotomique, on obtient alors un maximum pour que l'on peut visualiser comme étant le point le plus haut de l'enveloppe des droites $L(y^i, \lambda)$, c'est-à-dire, après calcul simple, $\lambda^* = \frac{2}{3}$ et $w(\lambda^*) = \frac{17}{3}$.
(On peut noter que comme la relaxation linéaire de ce problème est égale à la relaxation lagrangienne, on obtient également un optimum fractionnaire à $x_R = (1, \frac{2}{3}, 0)$ qui donne $f(x_R) = \frac{17}{3}$.)

Ainsi, pour cet exemple, on obtient un saut de dualité de $f(y^2) - w(\lambda^*) = \frac{4}{3}$.

7.2.4 La relaxation lagrangienne

La théorie de la dualité lagrangienne trouve son application la plus fréquente dans le cadre de la *relaxation lagrangienne* c'est-à-dire le fait d'appliquer la fonction de Lagrange pour un sous-ensemble bien choisi de contraintes.

Considérons le PM suivant :

$$\begin{array}{l} \text{Min } f(x) \\ \quad g_i(x) \leq 0 \qquad i \in I \\ \quad x \in S \subset I\!R^n. \end{array}$$

où S est un ensemble non triviale pouvant être lui-même défini par des contraintes.
On choisira d'appliquer la relaxation lagrangienne aux contraintes g_i dans le cas où :
- le problème de minimisation de la fonction lagrangienne obtenue sur S doit pouvoir être fait de manière efficace et exacte.
- l'ensemble des contraintes g_i relaxée n'est pas trop important et bien choisie pour limiter le nombre de multiplicateur à manipuler.
L'objectif de cette relaxation est de donner de bonne borne d'évaluation du problème. Cependant il n'est pas rare que l'on obtienne également des solutions primales intéressantes.

Un exemple très fréquent en pratique se situe dans le cadre des PLNE où certaines contraintes sont jugées "difficiles" à traiter. On peut alors écrire le PLNE de la façon suivante :

$$(P) \quad \begin{array}{l} \text{Min } c^T x \\ \quad Ax \leq b \\ \quad x \leq d \\ \quad x \in I\!N^n. \end{array}$$

où les contraintes $Ax \leq b$ sont des contraintes que l'on voudrait ôter du programme : c'est principalement le cas si elles sont "couplantes" (voir section suivante).

Pour mesurer l'efficacité de la relaxation lagrangienne, on va tenter de comparer la valeur d'évaluation obtenue par la relaxation linéaire du PLNE et celle obtenue par la relaxation lagrangienne L des contraintes $Ax \leq B$, c'est-à-dire les valeurs de

$$(P_R) \quad \begin{array}{l} \text{Min } c^T x \\ \quad Ax \leq b \\ \quad Cx \leq d \\ \quad x \in I\!R^n \end{array} \qquad \text{et} \qquad (D_L) \quad \begin{array}{l} Max_\lambda \ \ Min_{x \in S} \ \ \{L(x, \lambda)\} \\ \text{avec } S = \{x \in I\!N^n \mid Cx \leq d\}. \end{array}$$

En fait, on peut prouver le résultat suivant

Lemme 7.2.7 *La valeur optimale du dual lagrangien (D_L) est égale à la valeur optimale du problème suivant :*

$$(P') \quad \begin{array}{l} Min\ c^T x \\ \quad Ax \leq b \\ \quad x \in conv(S). \end{array}$$

Preuve : On se limite ici au cas où S contient un nombre fini de points que l'on note $S = \{x^1, x^2, ..., x^K\}$. Dans ce cas, la fonction de Lagrange est alors

$$w(\lambda) = Min_{k=1}^{K} \left(c^T x^k + \lambda^T (Ax^k - b) \right).$$

On peut voir que le dual lagrangien (D) peut se réécrire alors comme un PL en variables linéaires $(\lambda, z) \in I\!R^{m+1}$ suivant

$$(PL) \quad \begin{array}{l} \text{Max } z \\ \quad z \leq c^T x^k + \lambda^T (Ax^k - b) \quad \forall k \in \{1, ..., K\} \\ \quad \lambda \in I\!R_+^m \\ \quad z \in I\!R. \end{array}$$

En associant aux contraintes du PL précédent les variables duales de la PL $u_k \geq 0$, on va prouver que l'on obtient le programme dual de la PL du programme linéaire précédent :

$$(DPL) \quad \begin{array}{l} \text{Min } \sum_{i=1}^{K} (c^T x^k) u_k \\ \quad \sum_{i=1}^{K} (Ax^k) u_k \leq b \\ \quad \sum_{i=1}^{k} u_k = 1 \\ \quad u_k \in I\!R_+ \quad \forall k \in \{1, ..., K\}. \end{array}$$

En fait, le premier lot de contraintes correspond en fait aux variables λ et la dernière contrainte à la variable z. Le lot de contraintes associé aux variables λ n'est pas immédiat : pour l'obtenir, il faut remarquer que les contraintes de (PL) peuvent chacune s'écrire

$$z - \sum_{i=1}^{m} \lambda_i (A^i x^k - b_i) \leq c^T x^k$$

où A^i désigne le vecteur ligne correspondant à la ligne i de A. Donc la contrainte associée à la variable λ_i dans (DL) est alors

$$-\sum_{k=1}^{K} (A^i x^k - b_i) u_k \geq 0$$

$$i.e. \sum_{k=1}^{K} A^i x^k u_k \leq b_i \sum_{k=1}^{K} u_k.$$

Par la dernière contrainte $\sum_{i=1}^{k} u_k = 1$, on obtient alors la contrainte correspondante dans (DL).

Considérons à présent un point x quelconque de $conv(S)$, par définition il existe un vecteur $t \in I\!R_+^K$ tel que $\sum_{i=1}^{k} t_k = 1$ et $x = \sum_{i=1}^{K} t_k x^k$. On remarque donc que le programme (DPL) revient en fait à rechercher un point $x = \sum_{i=1}^{K} u_k x^k$ dans $conv(S)$.

Par conséquent (DPL) est exactement équivalent au programme donné dans l'énoncé du lemme. □

Ainsi dans le cas où $conv(S) \subsetneq \{x \in I\!R^n \mid Cx \leq d\}$, ce qui est le cas plus fréquent en PLNE, on voit que la relaxation lagrangienne donne une meilleure évaluation que la relaxation linéaire.

7.2.5 Application au problème du voyageur de commerce

On illustre le principe de la relaxation lagrangienne à l'exemple célèbre du voyageur de commerce.

Soit $G = (V, E)$ un graphe pour lequel on recherche un circuit hamiltonien de plus petite longueur où $c(e), e \in E$, est la longueur associée à une arête. On choisit un sommet particulier que l'on nomme sommet 1 et on nomme $G \setminus 1$ le graphe obtenu à partir de G en supprimant dans G le sommet 1 et ses arêtes incidentes. On appelle un *1-arbre* de G un arbre-couvrant de $G \setminus 1$ auquel on ajoute deux arêtes quelconque incidente à 1. On notera par la suite $\mathcal{T}$ l'ensemble des 1-arbre de G.
La suite de cet exercice repose sur la remarque simple suivante : "un cycle hamiltonien est un 1-arbre pour lequel chaque sommet a exactement un degré 2".

Pour $T \in \mathcal{T}$ un 1-arbre de G, on pose le *vecteur d'incidence* χ^T tel que pour toute arête $e \in E$, $\chi^T(e) = 1$ si $e \in T$ et 0 sinon. On note également $\chi(\mathcal{T}) = \{\chi^T \mid T \in \mathcal{T}\}$, c'est-à-dire l'ensemble des vecteurs d'incidence des 1-arbres de G.

On pose alors les variables binaires suivantes : $x(e) = 1$ si on choisit l'arête $e \in e$ dans la solution et 0 sinon et on formule le problème selon le PMD suivant :

$$\begin{array}{l} \text{Min } \sum_{e \in E} c(e)x(e) \\ \quad \sum_{e \in \delta(u)} x(e) = 2 \quad \forall u \in V \\ \quad x \in \chi(\mathcal{T}). \end{array}$$

On remarque que le vecteur x solution est nécessairement un 1-arbre tel que le degré de chacun de ses sommets est exactement 2.

Etant donné un poids w positif associé aux arêtes de G, on sait résoudre efficacement le problème suivant : optimiser une fonction linéaire $\sum_{e \in E} \bar{c}(e)x(e)$ sur l'ensemble $\chi(\mathcal{T})$. Ceci se fait en recherchant un arbre couvrant de poids minimum au sens du poids w dans $G \setminus 1$ et en lui ajoutant les deux arêtes incidentes à 1 de plus faibles valeurs selon w.

Pour ramener le problème du voyageur de commerce à la résolution du problème précédent, on va réaliser une relaxation lagrangienne des contraintes $\sum_{e \in \delta(u)} x(e) =$

2 $\forall u \in V$. Pour chaque sommet $u \in v$, on pose les contraintes $g_u(x) = \sum_{e \in \delta(u)} x(e) - 2$ pour tout $x \in \chi(\mathcal{T})$.
La fonction de Lagrange est alors :

$$L(x, \lambda) = c^T x + \sum_{u \in V} \lambda_u g_u(x) \quad \forall \lambda \in I\!R^n_+.$$

On remarque que cette fonction est la somme, pour chaque arête $e = uv \in E$, de la variable $x(e)$ multipliée par la somme de $c(e)$ et de $\lambda_u + \lambda_v$. Ainsi

$$L(x, \lambda) = \sum_{uv \in E} (c(uv) + \lambda_u + \lambda_v) x(uv) - 2 \sum_{u \in V} \lambda_u.$$

On remarque alors que le dernier terme de L est la constante $K = -2 \sum_{u \in V} \lambda_u$ lorsque λ est fixé. De plus, en posant $\bar{c}(uv) = c(uv) + \lambda_u + \lambda_v$ pour toute $uv \in E$, on a $L(x, \lambda) = \sum_{uv \in E} \bar{c}(uv) x(uv) - K$ qui est alors une fonction linéaire. On sait donc calculer pour tout vecteur $\lambda \geq 0$, la valeur de la fonction duale

$$w(\lambda) = Min_{x \in \chi(\mathcal{T})} L(x, \lambda).$$

On peut facilement appliquer un algorithme de sous-gradient pour calculer le maximum de la fonction w sur λ (en fait, il existe même des formules automatique pour calculer ici le gradient). Les expérimentations de cette technique permettent de résoudre le problème de voyageur de commerce pour des tailles intéressantes : en effet, on dispose d'algorithme permettant de découvrir une solution x correspondant aux bonnes valeurs de λ. En utilisant simplement un algorithme pour déterminer une bonne valeur de λ, on donne également une très bonne évaluation du coût d'une tournée hamiltonienne.

Chapitre 8

Points extrêmes et cas polynomiaux

Dans cette section, nous donnons quelques cas particuliers où la relaxation linéaire d'un PLNE peut être réalisée de manière à fournir directement, par l'algorithme du simplexe par exemple, une solution entière.

8.1 Polyèdre et points extrêmes

Un polyèdre est tout simplement une figure géométrique définie comme la partie délimitée par des "plans". Dans un espace à une dimension, les "plans" sont des points et les polyèdres sont des intervalles connexes. Dans un espace à deux dimensions, les "plans" sont des droites et les polyèdres sont des carrés, des rectangles,... Dans un espace de dimension 3, les "plans" sont des plans et les polyèdres sont des cubes, des dodécaèdres,...

En fait, dans $\mathbb{R}^n$, on appelle *hyperplan* H un sous-espace de $\mathbb{R}^n$ défini comme l'ensemble des points vérifiant une équation linéaire, c'est-à-dire qu'il existe $\alpha, a_0, a_1, ...a_n \in R$ tels que $H = \{x \in \mathbb{R}^n \mid a_0 + a_1x_1 + ... + a_nx_n = b\}$.

Définition 8.1.1 Un *polyèdre* $P \subseteq \mathbb{R}^n$ est l'ensemble des solutions d'un système fini d'inégalités linéaires, i.e.,

$$P = \{x \in \mathbb{R}^n \mid Ax \leq \alpha\},$$

où A est une matrice $m \times n$ (m et n entiers positifs) et $b \in \mathbb{R}^m$.

Nous dirons alors que le système $Ax \leq b$ définit (ou caractérise) le polyèdre P. Dans ce cours, nous considérons des polyèdres uniquement rationnels, c'est-à-dire pour lesquels les coefficients du système $Ax \leq b$ sont tous rationnels. Si A est une matrice $m \times n$, on désigne par A_i (resp. A^j) la $i^{\text{ème}}$ ligne (resp. $j^{\text{ème}}$ colonne) de A pour $i = 1, ..., m$ (resp.$j = 1, ..., n$).

Un *point* d'un polyèdre est donc défini par des coordonnées $\tilde{x} \in \mathbb{R}^n$ tel que $A\tilde{x} \leq b$. Un *polytope* est un polyèdre borné, c'est-à-dire qu'un polyèdre $P \subseteq \mathbb{R}^n$ est un polytope s'il existe $x^1, x^2 \in \mathbb{R}^n$ tel que $x^1 \leq x \leq x^2$, pour tout $x \in P$.

Définition 8.1.2 Un point x d'un polyèdre P est dit *point extrême* ou *sommet* (vertex) de P s'il n'existe pas deux solutions x^1 et x^2 de P, $x^1 \neq x^2$, telles que $x = \frac{1}{2}x^1 + \frac{1}{2}x^2$.

En d'autre terme, un point extrême de P est un point de P qui n'est pas le milieu d'un segment contenu dans P.

Théorème 8.1.3 *Soit $P = \{Ax \leq b\}$ un polyèdre de $\mathbb{R}^n$. Soit $\tilde{x} \in \mathbb{R}^n$ un point. On note alors $\tilde{A}x \leq \tilde{b}$ la sous-matrice des contraintes de $Ax \leq b$ formée par les inégalités vérifiées à l'égalité par $\tilde{x}$.*
Alors $\tilde{x}$ est un point extrême de P si et seulement l'ensemble $\tilde{A}x \leq \tilde{b}$ est de rang n.

Autrement dit, un point $\tilde{x}$ est extrême si on peut produire n inégalités vérifiées par $\tilde{x}$ à l'égalité qui sont linéairement indépendantes.

8.2 Points extrêmes et polyèdres entiers

Rappelons deux résultats de la programmation linéaire qui sont issues de l'algorithme du simplexe.
- Tous les solutions optimales d'un PL se situe sur l'un des hyperplans définissant le polyèdre des solutions.
- L'ensemble des points extrêmes du polyèdre des solutions contient au moins une solution optimale. On peut donc limiter la recherche des solutions optimales.

Il est donc intéressant de regarder ce qu'il se passe pour un PLNE en le surposant à la "grille" des points de coordonnées entières.

Un polyèdre est dit *pointu* s'il contient au moins un point extrême. (Par exemple, le polyèdre $\{(x_1, x_2) \in \mathbb{R}^2 : x_1 \geq 0\}$ ne contient pas de points extrêmes. Un polyèdre est dit *rationnel* s'il peut être défini par un système où toutes les inégalités ont des coefficients rationnels.
On étudiera par la suite des polyèdres rationnels pointus (ce qui n'est pas très restrictif).

On dit qu'un point de $\mathbb{R}^n$ est *entier* si ces coordonnées sont entières. Un polyèdre est dit *entier* si tous ses points extrêmes sont entiers. On a alors le théorème suivant.

Théorème 8.2.1 *Un polytope rationnel P est entier si et seulement si, pour tout vecteur entier c, la valeur optimale de $Max\{c^T x \mid x \in P\}$ est entière.*

Preuve : Le sens direct est immédiat.
Inversement, considérons $v = (v_1, ..., v_n)^T$ un point extrême de P (il en existe un car il est pointu). Admettons que l'on puisse prouver qu'il existe un vecteur entier w tel que v soit l'unique solution optimale de $max\{w^T x \mid x \in P\}$ (admettons-le). Prenons $\lambda \in \mathbb{Z}$ tel que $\lambda w^T v \geq \lambda w^T u + u_1 - v_1$ pour tout u point extrême de P. On peut noter que v est encore l'unique solution optimale de $max\{\lambda w^T x \mid x \in P\}$. Ainsi, en posant le poids $\bar{w} = (\lambda w_1 + 1, \lambda w_2, ..., \lambda w_n)^T$, on voit que v est aussi solution optimale de $max\{\bar{w}^T x \mid x \in P\}$ car $\lambda w^t v > x$ pour tout $x \in P$. Or par construction $\bar{w}^T v = \lambda w^T v + v_1$ et comme, par hypothèse, $\bar{w}^T v$ et $\lambda w^T v$ sont entiers, alors v_1 est entier. On peut reproduire cela pour toutes les composantes donc v est entier. □

Il est possible d'étendre ce résultat aux polyèdres non bornés.
Ce théorème indique ainsi les polyédres dont les sommets coîncident avec la "grille" des entiers. Ce sont de polyèdres particuliers qui représentent un cas particulier très intéressant pour la résolution des PLNE !

8.3 Totale unimodularité et cas polynomial

Il serait très intéressant de pouvoir caractériser les matrices correspondant à des polyèdres entiers.

Une matrice carrée A est dite *unimodulaire* si A est entière et si son déterminant vaut $+1$ ou -1.

Lemme 8.3.1 *Soit A une matrice carrée de taille m entière, inversible. Alors $A^{-1}b$ est un vecteur entier pour tout vecteur entier b de taille m si et seulement si A est unimodulaire.*

Preuve : Par un résultat classique d'algèbre linéaire, on sait que $A^{-1} = \frac{A^{adj}}{det(A)}$ où A^{adj} est la matrice adjointe à A, c'est-à-dire la matrice obtenue en transposant la matrice des cofacteurs $C_{ij} = (-1)^{i+j} M_{ij}$ et où M_{ij} est le déterminant de la sous-matrice obtenue depuis A en supprimant la ligne i et la colonne j. Donc si A est entière, A^{adj} est aussi entière. Donc si A est unimodulaire, A^{-1} est entière et donc $A^{-1}b$ est un vecteur entier.
Inversement, si $A^{-1}b$ est un vecteur entier pour tout vecteur entier b de taille m, alors en particulier $A^{-1}e_i$ est entier pour e_i le i$^{\text{ème}}$ vecteur unité pour tout $i = 1, ..., m$. Donc A^{-1} est entière et donc $det(A)$ et $det(A^{-1})$ sont tous deux entiers. Comme

$det(A).det(A^{-1}) = 1$, on a donc $det(A) = 1$ ou -1. □

Ce lemme mène alors à la définition suivante. Une matrice A $m \times n$ avec $n \geq m$ de rang m est dite *unimodulaire* si A est entière et si la matrice associée à chacune de ses bases a un déterminant $+1$ ou -1. (On rappelle qu'une base de A est un ensemble de m vecteurs colonnes linéairement indépendants et la matrice associée à une base est donc une sous-matrice carrée $m \times m$ inversible).
On peut alors prouver le résulat suivant.

Théorème 8.3.2 *[Veinott et Dantzig]*
Soit A une matrice $m \times n$ entière de plein rang. Le polyèdre défini par $Ax = b, x \geq 0$ est entier pour tout vecteur b entier si et seulement si A est unimodulaire.

On appelle une matrice *totalement unimodulaire* (TU) une matrice telle que toutes ses sous-matrices carrées ont un déterminant valant 0, 1 ou -1.
Ainsi, pour une matrice unimodulaire, les coefficients sont donc uniquement 0, 1 et -1.
On peut remarquer qu'en fait une matrice A $m \times n$ est TU si et seulement si la matrice $[AI]$ de taille $m \times (m + n)$, qui est obtenue en collant une matrice identité à A, est unimodulaire.

Une conséquence du théorème précédent donne alors le théorème important suivant.

Théorème 8.3.3 *[Hoffman-Kruskall]*
Soit A une matrice $m \times n$ TU. Alors le polyèdre défini par $Ax \leq b, x \geq 0$ est est entier pour tout vecteur b entier.

On peut noter que ce théorème n'est pas une caractérisation des polyèdres entiers. De plus, il n'est pas évident de détecter si une matrice est TU. Un résultat essentiel (et complexe) de Seymour (1980) prouve en fait que ces matrices peuvent être construites selon un schéma particulier.

Si ce résultat est assez complexe, il existe des cas particulier de matrice TU très connus.

Théorème 8.3.4 *[Poincaré 1900]*
Soit A une matrice dont les coefficients sont 0, 1 ou -1 et telle que chaque colonne contient au plus une fois le coefficient 1 et au plus une fois le coefficient -1. Alors A est TU.

Et ce théorème plus général.

Théorème 8.3.5 *Soit A une matrice dont les coefficients sont 0, 1 ou -1 et telle que*
- chaque colonne contient au plus deux coefficients non nuls.
- les lignes de A puissent être partionnées entre deux ensembles I_1 et I_2 tels que si une colonne a deux coefficients de signes différents (resp. de même signe) alors leurs lignes sont dans le même ensemble I_1 ou I_2 (resp. l'une dans I_1 et l'autre dans I_2).
Alors A est TU.

Ce résultat prouve ainsi que les matrices de flots (voir section 3) sont TU : on le prouve par le théorème 8.3.4 de Poincaré ou en utilisant le théorème 8.3.5 en se rendant compte que dans ce cas $I_2 = \emptyset$.
De même les matrices du problème du couplage biparti qui sont TU (3.24) par le théorème 8.3.5 en utilisant $I_1 = V_1$ et $I_2 = V_2$.

8.4 Totale duale intégralité et min-max

Un autre cas intéressant pour des matrices particulières est donné par la dualité forte en PL qui dit que les problèmes dual et primal coïncident

$$(P_R) \quad max\{c^T x \mid Ax \leq b\} = min\{y^t b \mid y^T A = c, y \geq 0\} \quad (D_R).$$

Si l'on considère que le programme (P_R) est la relaxation linéaire d'un PLNE (P) modélisant un problème d'Optimisation Combinatoire, la dualité nous guide alors vers d'éventuel encadrements min-max pour le problème, c'est-à-dire un moyen d'obtenir facilement un encadrement de la solution optimale du problème d'OC.

En fait, le cas qui nous intéresse est lorsque (P_R) correspond à un polyèdre entier, dans ce cas, il est possible que le programme (D_R) ait des propriétés similaires. En fait, on a le résultat simple suivant.

On appelle un système $Ax \leq b$ *totalement dual intégral* (TDI) si, pour tout vecteur entier c tel qu'il existe une solution optimale de (P_R), alors cette solution peut être obtenue par un vecteur y entier dans (D_R).

Théorème 8.4.1 *Soit $Ax \leq b$ un système TDI avec $P = \{x \mid Ax \leq b\}$ rationnel et b entier. Alors P est un polytope entier.*

Preuve : Par la propriété TDI, pour tout solution optimale x du problème, il existe un vecteur y solution du dual qui soit entier et qui réalise cet optimum, c'est à dire tel que $cx = y^T b$. Or comme b est entier, si y est entier, cx est donc entier. Donc par le théorème 8.2.1, x est entier. Donc P est entier. □

Ce résultat nous dit donc que s'il existe un algorithme approché résolvant le problème d'OC, il est alors utile de regarder le problème combinatoire défini dualement : cela donnera des résultat min-max.

On pourrait aussi s'intéresser à la caractérisation de systèmes TDI. Or cela n'a pas vraiment de sens par rapport à la résolution d'un problème d'OC : en effet, pour tout système rationnel $Ax \leq b$, il existe un entier positif t tel que $\frac{1}{t}Ax \leq \frac{1}{t}b$ soit TDI. L'existence d'un tel système ne dit donc rien sur la structure du polyèdre P associé. En fait, on a le résultat suivant.

Théorème 8.4.2 *[Giles and Pulleyblank]*
Soit P un polyèdre rationnel. Alors il existe un système TDI $Ax \leq b$ avec A entier tel que $P = \{Ax \leq b\}$. De plus, si P est entier, alors b peut-être choisi entier.

Ce résultat nous dit donc qu'il existe toujours un système TDI pour tout polyèdre P entier et que par conséquent, il existe un système ayant toujours des solutions entières. Ce théorème nous donne donc un moyen de prouver l'intégrité d'un polyèdre : déterminer un système TDI.

C'est le cas, par exemple, du système correspondant à la dualité des problèmes flot-max/coupe-min.

8.5 Exercices

8.5.1 Points fractionnaires du problème du sac-à-dos

1) Considérons le problème de sac-à-dos suivant :

$$\begin{aligned} \text{Max} \quad & c_1x_1 + c_2x_2 + c_3x_3 \\ & 2x_1 + 2x_2 + 5x_3 \leq 8 \\ & x_i \leq 1 \quad i = 1, ..., 3, \\ & x_i \geq 0 \quad i = 1, ..., 3, \\ & x_i \in I\!N \quad i = 1, ..., 3. \end{aligned}$$

Montrer que ce problème admet des points extrêmes fractionnaires. Donner un cas où il est optimal.

Réponse : Considérons le point $\tilde{x} = (1, 1, \frac{4}{5})$. On peut remarquer que ce point vérifie à l'égalité 3 inégalités de la relaxation linéaire du PLNE : $x_1 = 1$, $x_2 = 1$ et l'inégalité principale. De plus, ces 3 inégalités sont clairement linéairement indépendante. Donc

ce point est un point extrême fractionnaire du problème. Par exemple, pour le poids $\tilde{c} = (10, 10, 1)$ ce point est clairement optimal.

2) Considérons le problème de sac-à-dos suivant où b est entier.

$$\begin{aligned}
\text{Max} \quad & \sum_{i=1}^{n} c_i x_i \\
& \sum_{i=1}^{n} x_i \leq b \\
& x_i \leq 1 \quad i = 1, ..., n, && (8.1)\\
& x_i \geq 0 \quad i = 1, ..., n, && (8.2)\\
& x_i \in I\!N \quad i = 1, ..., n. && (8.3)
\end{aligned}$$

Montrer que ce système est entier.

Réponse : Un point extrême de ce système doit vérifier à l'égalité (au moins) $n-1$ inégalités linéairement indépendantes de ce système. Or il y a peu d'inégalités ici qui peuvent être vérifiées à l'égalité ! Il y a deux cas :
- soit l'inégalité principale n'est pas vérifiée à l'égalité : dans ce cas, seules les inégalités triviales sont serrées et le point est entier.
- soit l'inégalité principale est vérifiée et alors il y a $n-1$ inégalités triviales serrées. Donc le point est composée de $n-1$ composantes entières 0 ou 1. Notons alors N^+ les composantes à 1. La dernière composante inconnue est donc de valeur $b - |N^+|$: notons que cette quantité est nécessairement positive, entière et vaut au plus 1. Donc cette dernière composante est elle aussi entière.

8.5.2 Un petit exemple non TDI

Considérons le problème (très simple) suivant. On appelle *multi-ensemble*, un ensemble d'éléments où chaque élément peut-être représenté plusieurs fois. Etant donné complet à 4 sommets $K_4 = (V, E)$ dont les arêtes sont munies du poids $w(e)$, $e \in E$, déterminer un multi-ensemble d'arêtes de poids maximal de K_4 tel que, pour chaque sommet du graphe, il y ait au plus 2 arêtes incidentes à ce sommet (on peut appeler ce problème le "2-couplage" maximal).

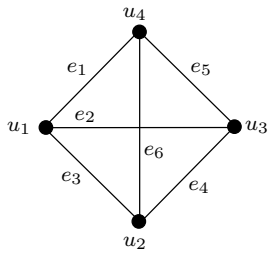


FIGURE 8.1 Le graphe K_4

On peut remarquer que ce problème se modélise par le PLNE P suivant :

$$\begin{aligned}
&\text{Max} \sum_{e \in E} w(e)x(e) \\
&\sum_{e \in \delta(u)} x(e) \leq 2 \quad \forall u \in V, && (8.4)\\
&x(e) \geq 0 \quad \forall e \in E, && (8.5)\\
&x(e) \in I\!N \quad \forall e \in E. && (8.6)
\end{aligned}$$

Par exemple, pour $w(e) = 1$ pour $e \in E$, la solution optimale de ce problème est clairement 4 : en effet, soit une arête, sans perte de généralité prenons e_1, est prise 2 fois et dans ce cas seule e_4 peut être ajoutée au plus 2 fois ; soit chaque arête est prise au plus 1 fois, et une solution optimale est alors clairement de prendre 4 arêtes formant un carré.

1) Montrer que le le système $\widetilde{P}$ formé par les inégalités (8.4-8.5) n'est pas TDI.

Réponse : Pour cela, considérons le dual de ce système basé sur des variables y associée aux inégalités (8.4), c'est-à-dire aux sommets du graphe :

$$\begin{aligned}
&\text{Min} \sum_{v \in V} 2y(v) \\
&y(u) + y(v) \geq w_e \quad \forall e = uv \in E, && (8.7)\\
&y(v) \geq 0 \quad \forall v \in V. && (8.8)
\end{aligned}$$

On peut remarquer que, pour le poids $w(e) = 1, e \in E$, il est impossible qu'il existe une solution duale entière. En effet, toute solution entière de ce dual doit avoir au moins 3 des variables y égales à au moins 1 (sinon une des inégalités ne seraient pas satisfaites pour une des arêtes) : donc toute solution entière de ce dual est au moins de valeur 6. Il n'existe donc pas de solutions entières du dual correspondant à une solution optimale du primal de valeur 4 : le système n'est donc pas TDI.

8.5.3 La dualité Flot Max / Coupe Min

En reprenant l'énoncé du problème du flot max de la section 3, on peut définir un deuxième problème appelé *problème de la coupe minimum* ou encore *coupe min*. On appelle *s-t-coupe orientée* ou même *s-t-coupe* quand il n'y a pas équivoque un ensemble d'arcs C sortant d'un ensemble de sommets W tel que $s \in W$ et $t \notin W$, *i.e.* $C = \delta^+(W)$. Remarquer que de même $C = \delta^-(\bar{W})$ où $\bar{W} = V \setminus W$. On appelle alors capacité d'une coupe la somme des capacités des arcs de la coupe.
D'après le théorème de Ford-Fulkerson concernant les flots maximum, on sait que pour tout flot maximal, on peut déterminer une coupe de capacité minimum associée tel que la valeur du flot soit égale à la valeur de la coupe

Montrer que le PL du flot max est associée à un PL dual formant un système TDI. Indication : les solutions du problème de min-cut sont solutions du dual.

Réponse : Ecrivons le dual de la façon suivante : on note $\pi(u)$ les variables duales associées aux 3 premières inégalités pour $u \in V$ et on note $\gamma(u,v)$ les variables duales associées aux arcs $a = (u,v)$ des contraintes de capacités.

$$\begin{aligned}
\text{Min} \sum_{a=(u,v)\in A} & c(a)\gamma(u,v) \\
& \pi(u) - \pi(v) + \gamma(u,v) \geq 0 \quad \forall a = (u,v) \in A, && (8.9)\\
& -\pi(s) + \pi(t) \geq 1, && (8.10)\\
& \pi(u) \lesseqgtr 0 \quad \forall u \in V, && (8.11)\\
& \gamma(u,v) \geq 0 \quad \forall a = (u,v) \in A. && (8.12)
\end{aligned}$$

Etant donnée un flot maximal, c'est-à-dire une solution du primal, on sait donc déterminer une s-t-coupe de capacité minimale. On peut voir que cette coupe correspond à une solution entière du duale qui correspond à cette même valeur : ce qui prouve que le système est TDI.
Cette solution est donnée par $\tilde{\gamma}(u,v) = 1$ si $u \in W$ et 0 sinon et $\tilde{\pi}(u) = 1$ si $u \notin W$ et 0 sinon. En effet, on peut prouver que cette solution $(\tilde{\gamma}, \tilde{\pi})$ est solution duale en considérant les 4 cas possibles de situation d'une arête dans le graphe par rapport à la coupe (soit $u \in W, v \notin W$; $u \in W, v \in W$, $u \notin W, v \in W$ and $u \notin W, v \notin W$).

Chapitre 9

Algorithmes de coupes

Dans la partie suivante, nous étudierons le cadre théorique générale de l'ajout de contraintes à un PLNE qui se nomme "approches polyédrales". Néanmoins, nous avons déjà rencontré deux cas où on considère un grand nombre de contraintes :
- les formulations PLNE contenant un grand nombre de contraintes (potentiellement exponentiel) que l'on ne peut donc pas énumérer.
- des exemples où l'ajout de contraintes supplémentaires permet d'obtenir une meilleure valeur de relaxation : on appelle cela parfois le "renforcement" de formulation.

Nous tenterons de répondre plus tardivement aux question : quelle contrainte ajoutée pour améliorer une relaxation ? quelle formulation considérer etc ?

Dans ce chapitre, nous allons étudier comment gérer algorithmiquement l'ajout d'un grand nombre de contraintes dans un PLNE.

9.1 Coupes et séparation

Nous considérons ici un programme linéaire

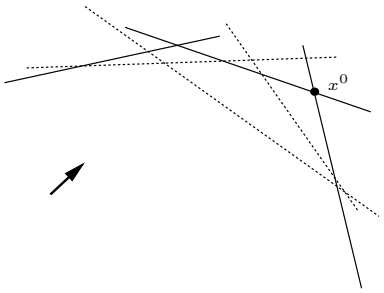
$$(P) \quad Max\{c^T x \mid Ax \leq b\}$$

comportant n variables mais où A contient un nombre important, par exemple exponentiel par rapport à n de contraintes (on dit que le programme est non compact sur les contraintes.
Un tel programme ne peut donc pas être introduit comme instance d'un algorithme de B&B et donc ne peut pas être utilisé directement dans les solveurs entiers. Nous allons pourtant voir qu'il existe un cadre algorithme efficace, les algorithmes de coupes pour ces programmes.

Choisissons tout d'abord un sous-ensemble de $A_0x \leq b_0$ de contraintes de $Ax \leq b$ tel qu'il existe une solution finie x^0 au problème $(P_0) \quad Max\{c^Tx \mid A_0x \leq b_0\}$. La solution x_0 est donc un point extrême du polyèdre défini par le système $A_0x \leq b_0$. On notera alors $(A \setminus A_0)x \leq (b \setminus b_0)$ les inégalités de $Ax \leq b$ qui ne sont pas dans $A_0x \leq b_0$.

La figure suivante représente en 2 dimensions, les contraintes de $A_0x \leq b_0$ en traits pleins et celles de $(A \setminus A_0)x \leq (b \setminus b_0)$ sont en pointillés. On notera alors $(A \setminus A_0)x \leq (b \setminus b_0)$ ces inégalités. La flèche indique la "direction d'optimisation" c'est-à-dire la direction orthogonale à toute droite d'équation $z = c^Tx$.

$$(P_0) \begin{cases} \text{Max } c^Tx \\ A_0x \leq b_0 \end{cases}$$



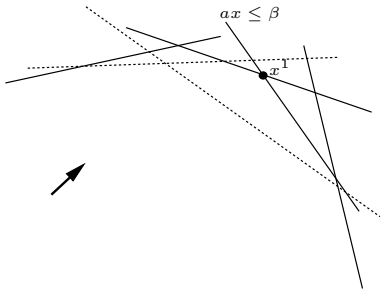
On peut remarquer que x^0 n'est pas pas forcément un point extrême de $Ax \leq b$: cela se produit s'il existe une inégalité de $(A \setminus A_0)x \leq (b \setminus b_0)$ qui n'est pas satisfaite par x^0 : on dit que x^0 *viole* la contrainte. Pourtant x^0 peut être un point extrême de $Ax \leq b$: dans ce cas, x^0 serait alors une solution optimale de (P) ! Cette idée donne lieu au problème de séparation.

Problème de séparation Etant donné un point $x \in I\!R^n$, le *problème de séparation* associé à $Ax \leq b$ et x consiste à déterminer si x satisfait toutes les inégalités de $Ax \leq b$ et sinon à trouver une inégalité de $Ax \leq b$ violée par x.

On peut répondre à ce problème par un algorithme que l'on appelle *algorithme de séparation*. Si on connaît un tel algorithme de séparation, on sait donc déterminer une inégalité $ax \leq \beta$ qui est violée par le point extrême x^0. On ajoute alors cette contrainte aux inégalités du système courant : on appelle alors une telle inégalité une *coupe* car elle "coupe" le point indésirable de l'espace des solutions possibles.

Sur la figure suivante, la contrainte $\alpha x \leq \beta$ qui était violée par x^0 a été ajoutée au système courant : on voit qu'elle a "séparé" une partie de l'espace indésirable. En posant alors $A_1x \leq b_1 = (A_0 \leq b_0) \cup (ax \leq \beta)$ et $(P_1) \quad Max\{c^Tx \mid A_1x \leq b_1\}$, on peut reproduire le même algorithme de séparation pour le nouveau point extrême x^1 obtenue par la résolution optimale du PL $(P1)$.

$$(P_1)\begin{cases} \text{Max } c^T x \\ A_0 x \leq b_0 \\ \alpha x \leq \beta \end{cases}$$



En réitérant l'algorithme de séparation autant tant que l'on peut ajouter des contraintes violées, on obtient alors une algorithme appelé *méthode de coupes* (cutting plane algorithm). A la fin d'une méthode de coupes, on obtient donc nécessairement un point extrême x^* qui est point extrême du système courant et qui n'est pas violé par une contrainte de $Ax \leq b$: on obtient donc une solution optimale de (P).

9.2 Equivalence Séparation et Optimisation

La question importante est donc alors de connaître la terminaison et la complexité d'une méthode de coupes. Ceci nous est donné par le résultat extrêmement puissant suivant :

Théorème 9.2.1 [Grötschel, Lovász et Schrijver [4] (1981)]
Une méthode de coupes sur un système $Ax \leq b$ de contraintes est polynomial si et seulement si l'algorithme de séparation associé à $Ax \leq b$ est polynomial.

Ce résultat fondamental permet ainsi de manipuler des formulations exponentielles pour la PLNE ! Il indique ainsi qu'optimiser est équivalent à séparer.

- *Taille du PL manipulé*

Une autre conséquence concerne la taille du système linéaire utilisé : en fait, comme on ajoute une inégalité un nombre polynomial de fois : le système courant a au plus une taille polynomiale ! Ainsi, au lieu de manipuler un système exponentiel, un simple système polynomial est suffisant.

Cette remarque n'est pas surprenante, en effet, pour définir un point extrême (solution) il suffit de n contraintes (linéairement indépendantes) satisfaites à l'égalité : le reste peut être donc être "écarté" du PL correspondant : la question est alors : comment trouver ces n contraintes ? En effet, si on les a, une seule utilisation de l'algorithme de séparation permettrait de tester si le point est une solution de $Ax \leq b$.

- *Preuve de polynomialité et de NP-difficulté*

Cette équivalence entre Séparation et Optimisation permet de prouver que certains

problèmes sont polynomiaux : il faut pour cela avoir prouvé que votre problème se ramène à résoudre un PL à nombre exponentielle d'inégalités mais pour lequel vous connaissez une méthode de coupes polynomiale.

Inversement, si vous prouvez que, pour toute fonction poids, la méthode de coupes associées à votre PL est NP-difficile, alors le problème équivalent au PL sera à son tour NP-difficile.

9.3 Exemples de séparation de contraintes

Dans cette section, nous aborderons des exemples d'algorithmes de séparation de contraintes célèbres.

9.3.1 Contraintes de coupes pour le TSP

Considérons la formulation suivante pour le voyageur de commerce symétrique que nous avons vu dans le début de cette partie.

$$
\begin{aligned}
& Min \sum_{e \in E} c(e)x(e) \\
& \sum_{e \in \delta(v)} x(e) = 2, \text{ pour tout } u \in V, && (9.1) \\
& \sum_{e \in \delta(W)} x(e) \geq 2, \text{ pour tout } W \subsetneq V \text{ et } W \neq \emptyset, && (9.2) \\
& x(e) \geq 0, \text{ pour tout } e \in E \\
& x(e) \in \{0,1\}, \text{ pour tout } e \in E.
\end{aligned}
$$

Les contraintes (9.2) sont en nombre exponentiel. On peut construire un algorithme de coupes en prenant comme ensemble initial les contraintes de degré et les contraintes triviales.
Le but ici est de résoudre la relaxation linéaire de cette formulation PLNE. Notons x^* une solution optimale du PL limité aux contraintes initiales. Le problème de séparation peut alors s'écrire de la façon suivante : déterminer une contrainte (9.2) violée par x^* si elle existe et dire sinon qu'il n'en n'existe pas.
Ce problème se ramène en fait à déterminer une coupe min dans le graphe G en utilisant pour capacité le vecteur x^* associée aux arêtes de G. En effet, si l'on dispose d'une coupe $\delta(W^*)$ avec $W^* \subsetneq V$ et $W^* \neq \emptyset$ et $x^*(\delta(W^*)) = \sum_{e \in \delta(W)} x^*(e)$ minimum, on a deux cas :

- soit $x^*(\delta(W^*)) < 2$ dans ce cas, on a déterminé une contrainte $\sum_{e\in\delta(W^*)} x(e) \geq 2$ violée par x^*
- soit $x^*(\delta(W)) \geq 2$ dans ce cas, toutes les contraintes (9.2) sont non violées.

Comme on sait déterminer une coupe-min avec des capacités positives (et x^* est un vecteur positif) en temps polynomial, on sait donc résoudre en temps polynomial cette formulation exponentielle.

9.3.2 Contraintes de clique pour le stable

On a vu qu'une formulation possible pour le problème du stable est d'utiliser les contraintes de cliques.

$$\begin{aligned} \text{Max } & \sum_{u\in V} c(u)x(u) \\ & \sum_{u\in K} x(u) \leq 1, \text{ pour toute clique } K \text{ de } G. \\ & x(u) \in \{0,1\}, \text{ pour tout } u \in V. \end{aligned} \tag{9.3}$$

Cette formulation contient un nombre exponentiel de cliques. On peut prouver que le problème de séparation associé à ces contraintes est équivalent à recherche une clique de plus grand poids positifs associés aux sommets dans le graphe G : or ce problème est NP-complet. Ce qui signifie que l'on ne peut pas construire un algorithme de coupes pour cette formulation.

En revanche, on peut considérer la formulation suivante

$$\begin{aligned} \text{Max } & \sum_{u\in V} c(u)x(u) \\ & x(u) + x(v) \leq 1, \text{ pour tout } uv \in E, \\ & \sum_{u\in K} x(u) \leq 1, \text{ pour toute clique } K \text{ de } G, |K| \geq 3. \\ & x(u) \in \{0,1\}, \text{ pour tout } u \in V. \end{aligned} \tag{9.4}$$

En effet, seules les contraintes d'arêtes sont nécessaires à la formulation : elles peuvent donc être énumérées et utilisées tout au long de l'algorithme de coupes. Les contraintes associées à des cliques de tailles au moins 3 seront elles utilisées dans un algorithme de coupes heuristiques : c'est-à-dire qu'au lieu de rechercher s'il existe ou non une contrainte de cliques violées, on recherche heuristiquement une contrainte de

cliques violée.
Cela peut être fait en utilisant par exemple un algorithme glouton : pour une solution x^* du problème relaxée, on recherche un sommet de grand poids, puis on essaye d'ajouter un sommet de grand poids formant une clique avec le sommet précédent et on réitère l'idée. Cet algorithme glouton n'est évidemment pas exacte mais permet d'obtenir des contraintes de cliques en temps polynomial : on peut donc le réitérer dans un processus de génération de contraintes.
L'algorithme présenté précedemment ne permet de résoudre la relaxation linéaire du PLNE du stable donné précédemment : toutefois, on obtient à la fin une bien meilleure valeur de relaxation linéaire que la formulation limitée aux contraintes dites "aux arêtes".

9.3.3 Contraintes de cycles impairs pour le stable

On peut remarquer que, lorsqu'un graphe est limité à un cycle impair, la formulation aux arêtes associées possède un point extrême fractionnaire en prenant $\frac{1}{2}$ pour chaque sommet du cycle.

L'inégalité suivante est valide car, pour un cycle impair, on ne peut prendre "qu'un sommet sur deux".

$$\sum_{u \text{ dans } C} x(u) \leq \left\lfloor \frac{|C|}{2} \right\rfloor \quad \text{pour tout cycle impair } C, \tag{9.5}$$

(9.6)

De plus, cette inégalité est séparable en temps polynomial si les inégalité d'arêtes sont toutes vérifiées.

En effet, considérons un point $\tilde{x}$ à séparer qui vérifient toutes les inégalités d'arêtes. On pose alors

$$w_{uv} = 1 - \tilde{x}_u - \tilde{x}_v$$

qui est bien positif ou nul.
L'inégalité est alors équivalente à

$$\sum_{e \in C} w_{uv} \geq 1$$

Le problème de séparation revient à déterminer un plus petit cycle impair par rapport au poids w dans le graphe. Si ce poids est strictement plus petit que 1, cela signifie qu'on a déterminé une inégalité violée. Si ce poids est plus grand que 1, cela signifie que l'inégalité avec le plus petit écart avec la violation, n'est justement pas violée :

donc il n'y a pas d'inégalités violées.
Pour déterminer un cycle impair de plus petits poids par rapport à w, on crée un graphe biparit G' où chaque sommet i de G est dupliquée en i' et i'' et où chaque arête ij de G devient deux arêtes $i'j''$ et $i''j'$ portant chacune le poids w_{ij}. Dans ce cycle, tout chemin (élementaire) de i' à i'' va correspondre à un cycle impair. En effectuant une recherche d'un plus court chemin de i' à i'' dans G' pour tout $i \in V$, on obtient un cycle impair de plus petit poids dans G.

9.4 Algorithmes de coupes et branchements

Ainsi, dans le cas (rare) où l'on connaît un système linéaire définissant un polyèdre entier ET que l'on connait un algorithme de séparation polynomial pour chacune de ces contraintes, on obtient alors une méthode de coupes qui permet de résoudre un PLNE en temps polynomial.

Cependant, il y a peu d'espoir de connaître un tel système pour un problème d'Optimisation Combinatoire quelconque (surtout NP-complet). D'autre part, le problème de séparation sur certaines classes d'inégalités valides peut être lui-même NP-difficile. Dans ce cas, on ne peut disposer que de techniques de séparation approchées.
Ainsi, une méthode de coupes seule peut ne fournir que des solutions fractionnaires (non optimales). Dans ce cas, on exécute une étape de branchement qui peut consister à choisir une variable fractionnaire x_i dans la solution et à considérer deux sous-problèmes du problème courant en fixant x_i à 0 pour l'un et à 1 pour l'autre. On applique alors la méthode de coupes pour les deux sous-problèmes. La solution optimale du problème sera donc la meilleure entre les deux solutions entières obtenues pour les deux sous-problèmes. Cette phase de branchement est répétée de manière récursive jusqu'à l'obtention d'une solution entière optimale. Cette combinaison de la méthode de branchements et d'une méthode de coupes au niveau de chaque noeud de l'arbre de branchement est appelée *méthode de coupes et branchements* (*Branch and Cut method*). Cette méthode s'est révélée très efficace pour la résolution de problèmes d'optimisation combinatoire réputés pour être difficiles tels que le problème du voyageur de commerce ou celui de la coupe maximum.

La mise en oeuvre d'un algorithme de B&C est donc assez complexe : elle nécessite de bien gérer un algo de B&B et plusieurs algo de séparation. Il faut également gérer les contraintes en très grand nombre et un solveur linéaire... Il est également possible, afin d'éviter de gérer trop de contraintes, de ne pas toute les considérer simultanément. Il existe des "framework" souvent en langage C++ qui permettent de donner le cadre global d'un algorithme de B&C. Ces frameworks gèrent ainsi l'arbre de branchement,

l'interfaçage avec un solveur linéaire, la gestion des contraintes, etc. Les solveurs sont principalement des outils libres, à part Concert Technology, produit lié à Cplex d'Ilog (mais qui n'est pas utilisable dans tous les cas, principalement s'il y a un grand nombre de contraintes) : on peut citer Abacus de la Zib (plus maintenu), BCP de Coin-Or et SCIP de la Zib.

9.5 Exercices

9.5.1 Séparation des inégalités de cycles pour Max-Cut

On a prouvé dans la section 3 que le problème de Max-Cut peut se modéliser en utilisant les inégalités suivantes, dites de cycles :

$$x(F) - x(C \setminus F) \leq |F| - 1, \text{pour tout cycle } C,\ F \subseteq C,\ |F| \text{ impair},$$

On veut créer algorithme de coupes associé aux contraintes de cycles pour un graphe $G = (V, E)$.
Soit $(P*)$ un programme linéaire contenant un nombre polynomial d'inégalité de cycles et d'inégalités triviales. On note x^* la valeur de la relaxation linéaire de $(P*)$. On rappelle alors qu'un algorithme de séparation consiste à déterminer si une solution courante x^*, obtenue par relaxation linéaire, viole une contrainte et dans ce cas, l'algorithme produit une contrainte.

1) Dans le cas où G est un graphe complet, proposer un algorithme permettant de répondre au problème de séparation des inégalités de cycles.
2) On désire à présent produire un algorithme de séparation pour les inégalités de cycles dans le cas d'un graphe quelconque.
a) Montrer qu'une inégalité de cycle peut se réécrire ainsi :

$$\sum_{e \in C \setminus F} x(e) + \sum_{e \in F} (1 - x(e)) \geq 1,\ \text{pour tout cycle } C,\ F \subseteq C,\ |F| \text{ impair},$$

b) Soit (P_i^*) un programme linéaire composée d'une quantité polynomial d'inégalités de cycle et d'inégalités triviales. Soit x^* une solution optimale de (P_i^*). Montrer qu'une inégalité (1) violée par x^* revient à déterminer un plus petit cycle où certaines arêtes sont de poids $x^*(.)$ et d'autres de poids $1 - x^*(.)$.
c) On définit un nouveau graphe G' à partir de G avec deux noeuds i' et i'' pour chaque noeud i de G. Pour toute arête ij de G, on place deux arêtes $i'j'$ et $i''j''$ de poids $x^*(ij)$ et deux arêtes $i'j''$ et $i''j'$ de poids $1 - x^*(ij)$. On calcule ensuite, pour chaque noeud i de G, un plus court chemin de i' à i'' (Un tel calcul peut se faire en temps polynomial par l'algorithme de Dijkstra car tous les poids sont positifs).

Montrer que la valeur d'un plus court chemin parmi tous les chemins obtenus est le poids du cycle recherché à la question précédente.
d) Montrer que si ce poids est supérieur ou égale à 1, x^* ne viole plus d'inégalités de cycles. Indiquer comment, dans le cas contraire, on obtient une contrainte de cycle violée par x^*.
e) Expliquer pourquoi il serait intéressant de rechercher parmi les plus courts chemins, un chemin possédant un nombre minimum d'arêtes.
d) Déduire le schéma d'un algorithme de coupes utilisant les inégalités de cycles pour un graphe quelconque G. Quel est la complexité d'une de ses itérations ? Quelle est sa complexité ?

Chapitre 10

Inégalités valides et renforcement

Comme il est mentionné dans la section précédente, il est utile de déterminer des classes d'inégalités valides (et non redondantes) pour un PLNE. Ces classes sont choisies de manière à couper des points extrêmes fractionnaires qui apparaissent dans la relaxation étudiée du polyèdre. On les détermine soit par intuition, soit par extrapolation de cas simples, soit en les dérivant par des procédés tels que celui de Chvàtal-Gomory,... L'ensemble de ces idées algorithmiques permettent aujourd'hui de résoudre efficacement des PLNE de bonnes tailles (quelques milliers de lignes/contraintes). Surtout dans les cas où le PLNE a une structure particulière. On nomme souvent cet aspect du domaine de recherche MIP pour Mixed Integer Programming.

10.0.2 Contraintes valides et renforcement

Pour un PLNE donné, $(P) \quad Max\{c^T x \mid Ax \leq b, x \in\in I\!N^n\}$ avec un nombre compact de contraintes et de variables, on veut ajouter des contraintes supplémentaires. On considère pour cela le polyèdre $P = \{x \in I\!R^n \mid Ax \leq b\}$.

Définition 10.0.1 Une inégalité (ou *contrainte*) $ax \leq \alpha$ est dite *valide* pour le polyèdre P si elle est vérifiée par tous les points de P, i.e. $P \subseteq \{x \in I\!R^n \mid ax \leq \alpha\}$.

Nous dirons que l'inégalité $ax \leq \alpha$ est *serrée* (resp. *violée*) par x^* si $ax^* = \alpha$ (resp. $ax^* > \alpha$).

Ainsi ajouter une inégalité valide violée à un PLNE permet de couper un point extrême indésirable de l'espace des solutions optimales. Si cet ajout est réalisé dans le cadre d'un algorithme de coupes, cela permet même d'écarter des points optimaux qui ne sont pas des solutions du PLNE.
De plus, ces contraintes permettent d'obtenir une relaxation plus intéressantes (même si parfois c'est numériquement très faible). On parle alors de *renforcement* de la formulation.

10.0.3 Somme de Chvàtal : exemple du problème du stable max

Une technique célèbre pour obtenir de telles inégalités valides de manière automatique est la somme de Chvátal-Gomory.

Elle consiste à additionner plusieurs contraintes et à diviser les coefficients de la contrainte obtenue. En utilisant le fait que certaines sommes de termes sont entières, on peut en déduire une nouvelle contrainte.

Prenons l'exemple du polyèdre associé à la relaxation linéaire de la formulation aux arêtes du stable.
Soit $G = (V, E)$ un graphe non orienté. Soit A la matrice d'incidence aux arêtes de G. On peut montré que le polyèdre

$$\mathcal{D} = \{x \mid xA \leq \mathbb{1}, x \geq 0\}$$

possède toujours des sommets à coordonnées non entières si G n'est pas biparti. En effet, Si G n'est pas biparti, alors G contient un cycle impair C. Soient $(u_1, ..., u_k)$ les sommets du cycle C, alors le vecteur z de $\mathbb{R}^n$ dont les composantes sont nulles si $u \notin \{u_1, ..., u_k\}$ et $\frac{1}{2}$ sinon vérifie bien à l'égalité les k contraintes associées aux arêtes de C et les $n-k$ contraintes triviales associées aux sommets de $V \setminus \{u_1, ..., u_k\}$. Comme les vecteurs des coefficients de ces contraintes sont linéairement indépendantes. On dit alors que e vecteur est un point extrême fractionnaire de $P(G)$.

On veut produire des contraintes qui coupe un tel point.
Supposons que G contienne un cycle impair de G (c'est-à-dire que G n'est pas biparti). On somme les k contraintes associées aux k arêtes du cycle. Dans cette somme, chaque variable associée à un sommet apparaît deux fois, on obtient ainsi :

$$\begin{array}{rcrcrcl}
x(u_1) & + & x(u_2) & & & \leq & 1 \\
 & & x(u_2) & + & x(u_3) & \leq & 1 \\
 & & & & .... & & \\
x(u_1) & & & & + \; x(u_k) & \leq & 1 \\
\hline
\end{array}$$

$$2 \sum_{u \in V(C)} x(u) \leq k$$

$$\sum_{u \in V(C)} x(u) \leq \frac{k}{2}$$

Comme le côté gauche de l'inégalité est une somme d'entier, le côté droit peut lui aussi être ramené à l'entier immédiatement inférieur. D'où, comme k est impair, on obtient

finalement

$$\sum_{u \in V(C)} x(u) \leq \frac{k-1}{2}.$$

Lorsque k est impair, on obtient aini les contraintes dites de cycles impairs pour le problème du stable :

$$\sum_{v \in V(C)} x(v) \leq \frac{|C|-1}{2} \quad \forall C \text{ cycle impair} \tag{10.1}$$

On remarque qu'alors ces contraintes coupent le point fractionnaire z.

10.0.4 Méthode de Chvátal-Gomory et Lovász-Schrivjer.

On peut utiliser la somme de Chvátal-Gomory pour créer un algorithme de coupes génériques et adaptables à tout PLNE : la métode duale fractionnaire. Cette méthode converge vers une solution optimale mais exessivement lentement. En revanche, les inégalités produites par cette méthode permettent d'accélérer fortement les algorithmes de coupes et branchements génériques.

Il existe une méthode similaire appelée méthode de Lovász-Schrivjer.

10.0.5 Autres méthodes

La méthode duale fractionnaire donnée par la somme de Chvátal-Gomory est malheureusement peu efficace si elle est utilisée seule. Le principe a été étendu et permet de construire d'autres "renforcement" d'un PLNE quelconque :
- l'utilisation des structures sous-jacentes de la matrice par exemple des structures de stable (voir exo)
- l'utilisation des structures particulières de certaines contraintes : par exemple de knapsack (voir exo).
- la construction de Lovasz-Shrivjer : qui permet de déduire des contraints en les "multipliant" entre elles. Cela mène à la méthode de Lift-and-Project initiée par Balas.
- les méthode disjonctives qui génèrent aujourd'hui les coupes les plus efficaces (projet Bonemine de Cornuejols, Margot, Bonami).

Ce domaine d'étude très riche et vivant a permis des améliorations substancielles des solveurs PLNE génériques : en revanche, elle ne permet pas de résoudre des problèmes d'Optimisation Combinatoire difficile pour lesquels une étude dédiée est nécessaire.

Chapitre 11

Reformulation et génération de colonnes

On a pu noter qu'il existe plusieurs formulations PLNE utilisant différents espaces de variables pour un même problème. Lorsque l'on change l'espace des variables d'un problème, on parle souvent de *reformulation*.
Très fréquemment une reformulation mène à des formulations PLNE ayant un nombre exponentiel de variables, on doit utiliser alors un algorithme capable de gérer cela : un algorithme de génération de colonnes.

11.1 Comment obtenir une bonne reformulation ?

Cette question est assez complexe. Elle demande tout d'abord de savoir ce qu'est une bonne reformulation : cela se juge en fonction des possibilités efficaces de branchement mais aussi des valeurs de relaxation obtenue.
De plus, il faut pouvoir résoudre une telle reformulation et, dans le cas où il y a un nombre exponentiel de variables, les solveurs PLNE ne peuvent pas directement les résoudre, à l'instar des formulations à nombre exponentiel de contraintes.

Les techniques pour obtenir une reformulation sont :
- les techniques de décomposition de Dantzig-Wolfe (pas vu dans ce module)
- ou découvrir soit même un espace nouveau de variables !

Plusieurs facteurs permettent de juger d'une formulation :
- la valeur de relaxation : plus elle est proche de l'optimum entier, plus la formulation a des chances de bien se comporter dans un algorithme de branchement.
- éviter qu'il existe trop de solutions *symétriques* c'est-à-dire qui ont un même coût et/ou des structures proches : par exemple deux ensembles de sommets indentiques à

un sommet près. Plus il y a de symétries, plus l'algorithme de branchement va devoir explorer de solutions.

11.2 Un exemple de reformulation : le problème de découpes

On s'intéresse ici au problème de découpe optimale de barres de fer, appelé en anglais *cutting stock problem*. Une entreprise possède une quantité importante $\{1, ..., N\}$ de barres de fer de taille identique b qui ont été produites en série. Sur une période donnée, l'entreprise reçoit n commandes $\{1, ..., n\}$: chaque commande $i \in \{1, ..., n\}$ correspond à la demande de n_i barres de fer de la même longueur $b_i \leq b$. On suppose que chaque commande correspond à une taille distincte des autres commandes. L'entreprise désire minimiser le nombre de barres initiales qui doivent être découpées pour produire les barres demandées.

On suppose dans tout l'exercice que $n \leq N$. On peut remarquer qu'alors le problème possède toujours une solution. En fait, comme il est possible de découper chaque demande dans une barre disctincte, cette solution consomme alors n barres.

11.2.1 Première formulation

On propose tout d'abord une première formulation linéaire utilisant des variables x_i^j correspondant au nombre de barres de tailles $b_i, i \in \{1, ..., n\}$ qui seront découpées dans la barre $j \in \{1, ..., N\}$. On utilise également des variables z_j qui prennet la valeur à 1 s'il y a une découpe réalisée dans la barre $j \in \{1, ..., N\}$.

On obtient alors la formulation suivante qui est clairement équivalente au problème.

$$
\begin{aligned}
&\text{Min } \textstyle\sum_{j=1}^{N} z_j \\
&\sum_{j=1}^{N} x_i^j = n_i \quad \forall i \in \{1, ..., n\}, \\
&\sum_{i=1}^{n} b_i x_j^i \leq b z_j \quad \forall j \in \{1, ..., N\}, \\
&x_j^i \leq z_j \quad \forall i \in \{1, ..., n\}, \forall j \in \{1, ..., N\}, \\
&x_i^j \in \{0, 1\} \quad \forall i \in \{1, ..., n\}, \forall j \in \{1, ..., N\}, \\
&z_j \in \{0, 1\} \quad \forall j \in \{1, ..., N\}.
\end{aligned}
$$

On peut noter que cette formulation contient un grand nombre de solutions symétriques : chaque barre est indifférenciée, ce qui implique qu'une même solution du

problème correspond à plusieurs solution du PLNE. Il est possible d'éviter une partie des symétries en ajoutant des inégalités : $\sum_{i=1}^{n} x_i^1 \geq \sum_{i=1}^{n} x_i^2 \geq ... \geq \sum_{i=1}^{n} x_i^N$ qui permettent de dire que la barre j est utilisée seulement si la barre $j-1$ l'a été. Néanmoins ces techniques sont limitées : on lui préfère une reformulation en changeant l'espace des variables.

11.2.2 Seconde formulation contenant un nombre exponentiel de variables

On appelle *découpe* d d'une barre de longueur b un ensemble de longueurs non-nulles $\{l_1, l_2, ..., l_k\}$ telles que $\sum_{j=1,...,k} l_k = b$. On note alors $dec(d) = \{l_1, l_2, ..., l_k\}$. On définit l'ensemble D de toutes les découpes possibles d'une barre de longueur b. On considère les variables t_d indiquant le nombre de barres initiales que l'on va découper selon la découpe $d \in D$. On peut montrer que la formulation suivante est une formulation F du problème de découpe :

$$\begin{aligned} \text{Min} \sum_{d \in D} & t_d \\ \sum_{d \in D \mid b_i \in dec(d)} & t_d \geq n_i \quad \forall i \in \{1, ..., n\}, \\ & t_d \in I\!N \quad \forall d \in D. \end{aligned}$$

Si l'on considère une solution du problème, c'est-à-dire un ensemble de barres où les commandes peuvent être découpées à l'intérieur, on peut facilement compter le nombre de barres initiales découpées selon la même découpe et on construit alors une solution du programme en fixant les variables correspondantes à ces valeurs. Cette solution vérifie alors naturellement les contraintes.
Inversement, les variables donnent directement le nombre de barres qui doivent être découpés selon une découpe donnée. Par les contraintes du programme, cette solution est telle que l'ensemble des barres obtenues contient bien les barres demandées.

11.3 Algorithme de génération de colonnes

On peut voir que le dual de la relaxation linéaire $\tilde{F}$ de F est le programme linéaire suivant

$$\begin{aligned} \text{Max} \sum_{i \in \{1,...,n\}} & n_i \lambda_i \\ \sum_{i \in dec(d)} & \lambda_i \leq 1 \quad \forall d \in D, \\ & \lambda_i \geq 0 \quad \forall i \in \{1, ..., n\}. \end{aligned}$$

Comme on sait que l'on peut produire une solution initiale, c'est-à-dire un ensemble de découpes D_0 dans lequel on peut découper toutes les commande, considérons alors la solution optimale t^0 de $\tilde{F}$ limitée aux variables correspondant à D_0. On appelle de même λ^0 la solution optimale du dual limité aux contraintes correspondant à D_0. On pose également la solution t^* de $\tilde{F}$ obtenue en fixant à 0 toutes les colonnes n'apparaissant pas dans D_0.

On obtient alors une condition d'optimalité : on peut remarquer que par le principe de la dualité forte, le couple de solution t^* et λ^0 est optimal pour $\tilde{F}$ et son dual si et seulement si λ^0 vérifie les contraintes du dual, c'est-à-dire si

$$\sum_{i \in dec(d)} \lambda_i^0 \leq 1 \quad \forall d \in D \setminus \{D_0\}.$$

Si la condition est vérifiée : on a pu résoudre le PL primal. Si ce n'est pas le cas, on peut remarquer que si l'on ajoutait des variables bien choisie dans le primal, cela permetttrait de vérifier la condition ! On appelle le fait d'ajouter des variables la **génération d'une colonne**.

Le problème de génération d'une colonne pour le programme $\tilde{F}$ est-il utile pour répondre à la condition ? Pour répondre à cette question, on veut tester si toutes les contraintes données à la question précédente sont satisfaites par λ^0. On peut remarquer que cette question revient à résoudre le PLNE suivant où l'on utilisee des variables entières y_i associées à la quantité de barres de longueurs b_i que l'on veut découper dans la barre correspond à la nouvelle variable

$$\begin{array}{l} \text{Max } \sum_{i=1}^n \lambda_i^0 y_i \\ \quad \sum_{i=1}^n b_i y_i \leq b \\ \quad y_i \in I\!N \quad \forall i \in \{1, ..., n\}. \end{array}$$

Ce PLNE correspond en fait au problème du sac-à-dos binaire qui est NP-difficile... Néanmoins, on sait assez bien le résoudre en pratique.

Algorithme de génération de colonnes : Ainsi, on peut déterminer s'il la condition est vérifiée et sinon on sait ajouter une variable. On sait que si l'on ajoutait toutes les variables possibles, on obtiendrait à la fin le PL avec toutes ses variables : donc l'algorithme converge en théorie vers l'optimum.

Cette méthode de génération de colonnes pourrait mener dans le pire des cas à un nombre exponentiel d'ajout de variables... Mais il s'agit en fait exactement de l'algorithme du simplexe (ajout de variables en fonction des coûts réduits) ! ! ! On sait que

l'algorithme du simplexe ne nécessite que rarement un grand nombre d'ajouts de variables : comme pour l'algorithme du simplexe, en pratique il est rarement nécessaire d'ajouter toutes les variables dans un alo de génération de colonnes. Mais il est parfois difficile de faire converger l'algorithme (cas de dégénérescence des itérations de l'algorithme du simplexe).

11.4 Algorithme de génération de colonnes et branchement

Le chapitre précédent propose un algorithme de génération de colonnes pour résoure un PL possédant un nombre exponentiel de variables. Cette technique est facilement reproductible pour d'autres problèmes, avec succès. Dans le cadre d'un PLNE, il est nécessaire de coupler cette technique avec un algorithme de branchements : ces algorithmes dit de *génération de colonnes et branchements* ou *Branch-and-Price* sont de plus en plus utilisés, même si les difficultés de convergence citées plus haut restent un écueil fréquent.

Si l'on désire résoudre un PL avec un nombre exponentiel de variables, il existe des framework efficace comme Concert technology de Cplex-IBM ou autres. Ils sont parfois très spécialisés sur certains modèles.
Pour résoudre un PLNE avec un nombre exponentiel de variables, les frameworks sont plus rares : abacus, SCIP et BCP sont utilisables ; il existe aussi Symphony (universitaire) et un petit nouveau BapCode.

On peut avoir besoin de gérer des formulations avec un nombre exponentiel de variables et de contraintes ! Il faut donc un algorithme de *Branch-and-cut-and-Price (BCP)* : il n'existe là que des outils universitaires (déjà cités).

11.5 Exercices

11.5.1 Exercice : le problème de multiflot continu

Le problème de multiflot (ou multicommodity flow) se définit ainsi.
Soit $G = (V, A)$ un graphe orienté où l'on associe une capacité $c(a) \in I\!N$ associé à chaque arc $a \in A$. Soit k commodités, c'est-à-dire des paires de sommets $(s_1, t_1), ..., (s_k, t_k)$. Pour chaque paire $i \in \{1, ..., k\}$, on désire trouver des (s_i, t_i)-flots entiers tels que la somme des capacités utilisés par les flots sur un même arc respecte la capacité d'un

arc. L'objectif est de maximiser la quantité total de flot circulant sur le graphe.

On peut facilement construire une formulation PLNE utilisant de variables $x^i(a)$ indiquant la quantité de flot passant par l'arc a pour la commodité i. Cette formulation, lorsque $k \geq 2$ n'est pas TU et sa résolution n'est pas efficace vue le nombre de variables.

Une reformulation peut être facilement obtenue en posant d'autres variables. soit $\mathcal{P}_{st}$ l'ensemble des chemins (élémentaires) allant du sommet s au sommet t. Posons $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}_{s_i t_i}$ et enfin pour tout arc a, $\mathcal{P}_a = \{\mu \in \mathcal{P} \mid a \in \mu\}$.
On associe une variable f_μ associée à chaque chemin de $\mathcal{P}_{s_i t_i}$ pour tout $i = 1, ..., k$.
On voit que la formulation PL suivante modélise le problème de multiflot continue.

$$(MCF) \qquad \begin{aligned} &\text{Max} \sum_{\mu \in \mathcal{P}} f_\mu \\ &\sum_{\mu \in \mathcal{P}_a} f_\mu \leq c(a) \quad \forall a \in A \\ &f_\mu \geq 0 \quad \forall \mu \in \mathcal{P}. \end{aligned}$$

Cette formulation PL contient un nombre exponentiel de variables mais un nombre très petit de contraintes. Pour déterminer un cadre de génération de colonnes, il faut donc pouvoir déterminer un nombre réduit de variables.
Posons $\mathcal{P}'$ un sous-ensemble de chemin de $\mathcal{P}$ tel que le programme

$$(MCF') \qquad \begin{aligned} &\text{Max} \sum_{\mu \in \mathcal{P}} f_\mu \\ &\sum_{\mu \in \mathcal{P'}_a} f_\mu \leq c(a) \quad \forall a \in A \\ &f_\mu \geq 0 \quad \forall \mu \in \mathcal{P}'. \end{aligned}$$

ait une solution réalisable $(f'_\mu)_{\mu \in \mathcal{P}'}$. On peut noter qu'alors la solution $(f_\mu)_{\mu \in \mathcal{P}}$ obtenu depuis f' en ajoutant des variables nulles pour tous les chemins absents de $\mathcal{P}'$ est alors solution de (MCF). Donc la solution de (MCF') est inférieure à celle de (MCF).
On veut savoir si f obtenue depuis f' est optimale pour (MCF). Utilisons le dual $(D-MCF)$ de (MCF), on pose pour cela λ_a les variables duales associées aux contraintes, c'est-à-dire les arcs de A. Regardons en fait le dual $(D - MCF')$ de (MCF') :

$$(D - MCF') \qquad \begin{aligned} &\text{Min} \sum_{a \in A} c(a)\lambda_a \\ &\sum_{a \in \mu} \lambda_a \geq 1 \quad \forall \mu \in \mathcal{P}' \\ &\lambda_a \geq 0 \quad \forall a \in A. \end{aligned}$$

On remarque que dans le cas du dual, la solution λ' optimal de $(D-MCF')$ est solution de $(D - MCF)$ si et seulement si λ' vérifie toutes les contraintes de $(D - MCF)$ or il

y en a une par chemin de $\mathcal{P}$, c'est-à-dire si

$$\sum_{a \in \mu} \lambda_a \geq 1 \quad \forall \mu \in \mathcal{P}.$$

Dans ce cas, λ' est solution de $(D - MCF)$ et donc solution optimal. Ainsi la solution f obtenue depuis f' correspond à la même valeur objective que la solution duale : par le corollaire de la dualité forte, cela implique que f est optimale pour (MCF).

Cette logique induit un principe de tests polynomials pour l'optimalité : s'il n'existe pas contrainte du dual (correspondant à un chemin) qui soit violée par λ' alors f est optimal, sinon on connait un chemin correspondant à une contrainte violée et on peut l'ajouter dans (MCF) : c'est le *problème de génération d'une colonne* (ou pricing).
La réitération de ce principe donne une *méthode de génération de colonnes* duale de la génération de contraintes vues précédemment : ainsi on a le même résultat théorique : une méthode de génération de colonnes est polynomial si et seulement si l'algorithme de génération d'une colonne est polynomial.

En revanche, cette étape d'ajout de colonnes n'est pas aussi simple en pratique : en effet la convergence de cette méthode est similaire au pire des cas de l'ajout de colonnes dans l'algorithme du simplexe : c'est-à-dire la dégénérescence. Il faut donc utiliser des procédés assez complexes pour permettre une convergence rapide.

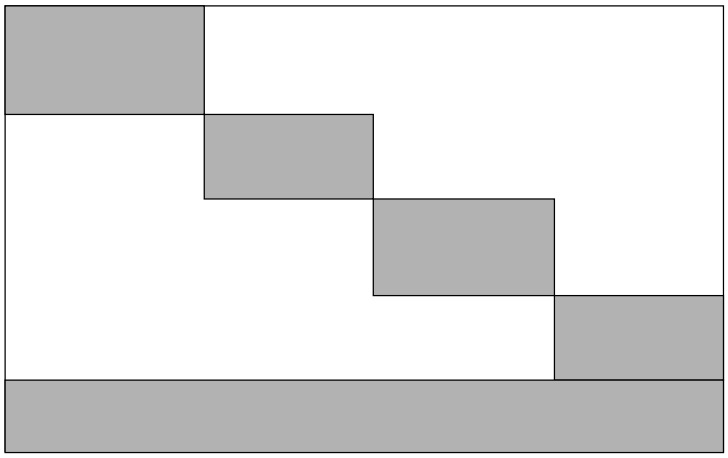
Chapitre 12

Décomposition

12.1 Introduction

On appelle *décomposition* d'un PMD le fait de diviser un PMD en plusieurs sous-problèmes en s'appuyant sur la structure de la matrice des contraintes.

On appelle *matrice des contraintes* la matrice M ayant autant de colonnes que de variables du PMD et de lignes que de contraintes du PMD et telle que $M_{ij} = 1$ si la variable numéro i est utilisée dans la contrainte j et 0 sinon. Il n'est pas rare de voir des structures de matrices très particulières. Après rangement (parfois peu évident) des lignes et des colonnes, on peut voir apparaître la structure suivante où seuls les emplacements gris correspondent à des valeurs non nulles.



On peut voir que seul le bloc du bas concernent toutes les variables du problèmes : on appelle les contraintes de ce bloc *contraintes couplantes*. Les autres blocs diagonaux sont eux plus petits et sont indépendants les uns des autres. Considérons pour chacun des blocs diagonaux, le PMD obtenu à partir du PMD complet en limitant les variables

et les contraintes à ceux du bloc : ces problèmes (plus petis que le PMD complet) sont alors appelés *problème satellites* ou parfois *esclaves*. Résoudre indépendemment ces sous-problèmes n'a pas de sesn par rapport à l'optimisation globale du programme. On peut par contre guider la résolution de chaque sous-problème en les reliant entre eux : en fait, on réalise ce lien en modifiant chaque sous-problème à partir de données obtenues à partir d'un *programme maître* qui contient les contraintes couplantes. L'ensemble du programme maître et des programmes satellites peut alors s'intégrer dans un processus algorithmique qui converge vers la solution optimale du problème.

Suivant la manière selon laquelle on décompose le problème et la technique mathématique utilisée, on obtient des décompositions diverses pour lesquelles plusieurs techniques algorithmiques ont été mises au point. On peut citer principalement :
- la décomposition lagrangienne : on utilise la relaxation lagrangienne des contraintes couplantes pour obtenir des sous-problèmes indépendants. On rassemble les différents résultats en utilisant la dualité lagrangienne.
- la décomposition de Dantzig-Wolfe : on utilise un problème maître associé aux contraintes couplantes et des problèmes-satellites (ou esclave) associé à chaque bloc de la matrice. On rassemble les différents résultats dans un processus itératif utilisant les coûts duaux du problème maître dans les problèmes-satellites : ce processus est proche de l'algorithme de génération de colonnes que l'on verra dans la partie suivante.

On peut noter que ces décompositions mènent parfois à la même décomposition. La mise en oeuvre d'une décomposition demande à chaque fois une étude poussée et dédiée. Ces techniques de décomposition ont menè à de grandes réussites sur des problèmes industriels de très grandes tailles.

12.2 Décomposition de Dantzig-Wolfe

La décomposition de Dantzig-Wolfe permet d'obtenir une meilleure formulation d'un problème.

12.2.1 Formulation de base

Soit $c \in \mathbb{R}^n$ un vecteur-colonne, $A \in \mathbb{R}^{n \times m}$, $a \in \mathbb{R}^m$, $B \in \mathbb{R}^{n \times m'}$ et $b \in \mathbb{R}^{m'}$. On considère le PLNE (P) suivant

$$(P) \begin{cases} \text{Min } c^T x \\ \quad Ax \geq a \\ \quad Bx \geq b \\ \quad x \in \mathbb{N}^n \end{cases}$$

où :
- les contraintes $Ax \geq a$ sont "couplantes"
- les $Bx \geq b$ sont "décomposables", c'est-à-dire que le problème

$$(SP) \begin{cases} \text{Min } d^T x \\ \quad Bx \geq b \\ \quad x \in I\!N^n \end{cases}$$

où d est un vecteur poids quelconque (de signe quelconque...). (SP) est en fait la somme de problèmes sur des espaces de variables indépendantes. On appelle (SP) le sous-problème de (P).

12.2.2 Réécriture et relaxation de (SP)

Le problème (SP) peut être réécrit différemment en utilisant une décomposition dite parfois "par les solutions" dans la logique de la décomposition de Dantzig-Wolfe.
On considère pour cela l'ensemble Q des solutions de (SP). Notons que cet ensemble est en général exponentiel sur la taille du problème. Notons alors $Q = \{\chi^1, ..., \chi^{|Q|}\}$ ces solutions qui sont donc des vecteurs entiers.
Déterminer une solution de (SP) revient donc à choisir un vecteur χ dans Q, en d'autre terme cela revient à réécrire (SP) de la façon suivante :

$$(SP) \begin{cases} \text{Min } d^T x \\ \quad x = \sum_{q=1}^{|Q|} \chi^q \lambda_q \\ \quad \sum_{q=1}^{|Q|} \lambda_q = 1 \\ \quad \lambda_q \in \{0,1\}, \text{ pour tout } 1 \leq q \leq |Q|. \end{cases}$$

On peut noter que les variables λ sont ici des valeurs permettant de définir quels vecteurs choisir dans Q.
On peut alors l'écrire aussi

$$(SP) \begin{cases} \text{Min } d^T x \\ \quad x = \sum_{q=1}^{|Q|} \chi^q \lambda_q \\ \quad \sum_{q=1}^{|Q|} \lambda_q = 1 \\ \quad \lambda_q \geq 0, \text{ pour tout } 1 \leq q \leq |Q|. \end{cases}$$

En effet, on peut noter que toute solution de la 1ère écriture est solution de la 2ème. Inversement, une solution optimale de la 2ème peut être prise parmi les points extrêmes du polyèdre formé par les points entiers de $Bx \leq b$, c'est-à-dire Q.

12.2.3 Décomposition de Dantzig-Wolfe et génération de colonnes

Compte-tenu de la réécriture de (SP), on peut réécrire (P) ainsi

$$(P)\begin{cases} \text{Min } c^T x \\ \quad Ax \geq a \\ \quad x = \sum_{q=1}^{|Q|} \chi^q \lambda_q \\ \quad \sum_{q=1}^{|Q|} \lambda_q = 1 \\ \quad \lambda_q \in \{0,1\}, \text{ pour tout } 1 \leq q \leq |Q|. \end{cases}$$

ou encore en faisant disparaître x.

$$(P)\begin{cases} \text{Min } \sum_{q=1}^{|Q|} (c^T \chi^q) \lambda_q \\ \quad \sum_{q=1}^{|Q|} (A\chi^q) \lambda_q \geq a \\ \quad \sum_{q=1}^{|Q|} \lambda_q = 1 \\ \quad \lambda_q \in \{0,1\}, \text{ pour tout } 1 \leq q \leq |Q|. \end{cases}$$

On considère alors une relaxation "linéraire" de (MP) de (P) (qui n'est pas la relaxation linéaire de la première écriture (P))) ainsi définie :

$$(MP)\begin{cases} \text{Min } \sum_{q=1}^{|Q|} (c^T \chi^q) \lambda_q \\ \quad \sum_{q=1}^{|Q|} (A\chi^q) \lambda_q \geq a \\ \quad \sum_{q=1}^{|Q|} \lambda_q = 1 \\ \quad \lambda_q \geq 0, \text{ pour tout } 1 \leq q \leq |Q|. \end{cases}$$

Il est important de remarquer que (MP), appelé *problème maître* est un PL où le caractère entier des solutions de (SP) est pris en compte.

Afin de résoudre (MP), on peut mettre en place un algorithme de génération de colonnes. Pour cela, on considère le PL (MP_t) qui est une restriction de (MP) à un sous-ensemble $Q^t = \{\chi^{q_1}, ...\chi^{q_t}\}$ de vecteurs de Q tel que (MP) possède une solution.

$$(MP_t)\begin{cases} \text{Min } \sum_{i=1}^{t} (c^T \chi^{q_i}) \lambda_{q_i} \\ \quad \sum_{i=1}^{t} (A\chi^{q_i}) \lambda_{q_i} \geq a \\ \quad \sum_{i=1}^{t} \lambda_{q_i} = 1 \\ \quad \lambda_{q_i} \geq 0, \text{ pour tout } 1 \leq i \leq t. \end{cases}$$

Ecrivons alors le dual de ce PL en posant π le vecteur dual des inégalités provenant de

$Ax \geq a$ et ρ la valeur duale provenant de la contrainte d'égalité.

$$(DMP_t) \begin{cases} \text{Max } \rho + \pi^T a \\ \quad \pi^T A\chi^{q_i} + \rho \leq c^T \chi^{q_i}, \text{ pour tout } 1 \leq i \leq t, \\ \quad \pi \in I\!R_+^m, \\ \quad \rho \in I\!R. \end{cases}$$

On peut voir que la solution optimale de (MP_t) est la solution optimale de (MP) lorsque toutes les solutions $\chi^q \in Q \setminus Q^t$ ont un coût réduit

$$(c - \pi^T A)\chi^q - \rho \geq 0.$$

Ceci donne lieu à un algorithme de génération de colonnes où à chaque itération, on recherche une solution de (SP) selon le poids $d = (c - \pi^T A)$: si ce poids est négatif strictement, on ajoute une colonne $\chi^{q_{t+1}}$ à Q. Et on considère alors (MP_{t+1}) jusqu'à ce que (SP) retourne que la solution optimale a un coût réduit positif, ce qui signifie que la solution du master restreint est une solution optimale de (MP).

12.3 Relaxation Lagrangienne

La relaxation lagrangienne d'une formulation est une relaxation des contraintes du problème dans l'objectif. On fait passer une partie de l'ensemble des contraintes du problème jugées difficiles dans l'objectif. La relaxation lagrangienne de (P) est

$$(LRP) \begin{cases} \text{Min } c^T x - \gamma^T (Ax - a) \\ \quad x = \sum_{q=1}^{|Q|} \chi^q \lambda_q \\ \quad \sum_{q=1}^{|Q|} \lambda_q = 1 \\ \quad \lambda_q \in \{0, 1\}, \text{ pour tout } 1 \leq q \leq |Q|. \end{cases}$$

avec γ le vecteur colonne optimal des multiplicateurs de Lagrange que l'on suppose connu.
On définit le Lagrangien dual de la manière suivante

$$(LDP) \begin{cases} \text{Max Min } c^T x - \gamma^T (Ax - a) \\ \quad \gamma \in R_+^{|a|} \end{cases}$$

où on suppose qu'on a x et qu'on cherche γ. Une fois le min linéarisé on obtient

$$(LDP) \begin{cases} \text{Max } \eta \\ \quad \eta \leq c^T \chi^q - \gamma^T (A\chi^q - a), \text{ pour tout } 1 \leq q \leq |Q| \\ \quad \gamma \in R_+^{|a|} \\ \quad \eta \in R \end{cases}$$

En notant λ_q les variables duales associées aux contraintes du lagrangien dual, on peut écrire son dual

$$(DLDP)\begin{cases} \text{Min } \sum_{q=1}^{|Q|}(c^T\chi^q)\lambda_q \\ \quad \sum_{q=1}^{|Q|}(A\chi^q)\lambda_q \geq a \\ \quad \sum_{q=1}^{|Q|}\lambda_q = 1 \\ \quad \lambda_q \geq 0, \text{ pour tout } 1 \leq q \leq |Q|. \end{cases}$$

On obtient la relaxation linéaire de notre problème dans l'espace de variable correspondant à la décomposition de Dantzig-Wolfe. Donc le dual de la formulation par décomposition de Dantzig-Wolfe correspond au lagrangien dual. En utilisant une formulation de Dantzig-Wolfe, on résout le lagrangien dual en résolvant le dual de la formulation.
Au final, lorsqu'on utilise une formulation obtenue par décomposition de Dantzig-Wolfe, on résout la relaxation linéaire de la formulation qui est la relaxation lagrangienne du problème.

12.4 Décomposition de Dantzig-Wolfe appliquée à la coloration

La formulation par couverture d'ensembles a été obtenue à partir de la relation entre une coloration de cardinalité minimum et une couverture des sommets de cardinalité minimum par des stables du graphe, mais on peut également la déduire d'une décomposition de Dantzig-Wolfe.

12.4.1 Formulation de base

On associe à chaque sommet u de V un vecteur binaire à K dimensions $x_u = (x_u^1, ..., x_u^K)$, où K est une borne supérieure sur la coloration de G (au maximum $K = |V|$). Comme l'on désire minimiser le nombre de couleur, on ajoute une variable binaire w_l par couleur $l = 1, ..., K$ indiquant si cette couleur a été utilisée ou non. Le problème est donc équivalent au programme

$$(P_C)\begin{cases} \text{Min } \sum_{l=1}^{K} w_l \\ \quad \sum_{l=1}^{K} x_u^l = 1, \text{ pour tout } u \in V, \\ \quad x_u^l + x_v^l \leq w_l, \text{ pour tout } e = uv \in E \text{ et } 1 \leq l \leq K, \\ \quad x_u^l \in \{0,1\}, \text{ pour tout } u \in V \text{ et } 1 \leq l \leq K, \\ \quad w_l \in \{0,1\}, \text{ pour tout } 1 \leq l \leq K. \end{cases}$$

Les premières contraintes sont celles couplante ($Ax \leq a$) et les suivantes celles décomposables par "couleur l" ($Bx \geq b$).
Ainsi on a

$$(SP_C)\begin{cases} \text{Min } \sum_{l=1}^{K}(d_{w_l}w_l + d_{xl}x^l) \\ \quad x_u^l + x_v^l \leq w_l, \text{ pour tout } e = uv \in E \text{ et } 1 \leq l \leq K, \\ \quad x_u^l \in \{0,1\}, \text{ pour tout } u \in V \text{ et } 1 \leq l \leq K. \\ \quad w_l \in \{0,1\}, \text{ pour tout } 1 \leq l \leq K. \end{cases}$$

qui est en fait la somme des PLNE pour chaque $l \in \{1, ..., K\}$

$$(SP_C^l)\begin{cases} \text{Min } d_{w_l}w_l + d_{x^l}x^l \\ \quad x_u^l + x_v^l \leq w_l, \text{ pour tout } e = uv \in E \\ \quad x_u^l \in \{0,1\}, \text{ pour tout } u \in V, \\ \quad w_l \in \{0,1\}. \end{cases}$$

Ce problème contient donc K sous-problèmes (SP_C) qui seront ou non utilisées dans le problème suivant la valeur de w_l

12.4.2 Réécriture et relaxation de (SP)

Le problème (SP_C) consiste donc à déterminer K stables. Un vecteur de Q est donc un vecteur représentant K stables et pour chaque stable l un entier w_l indiquant si ce stable l est ou non l'ensemble vide.
Posons alors $\mathscr{S}$ l'ensemble des stables G et $\chi^{\mathscr{S}}$ l'ensemble des vecteurs d'incidence de ces stables. Alors Q peut être défini comme un vecteur composé de toutes les combinaisons de K vecteurs $\chi^S \in \chi^{\mathscr{S}}$ chapeauté chacun de 0 si $S = \emptyset$ et 1 sinon. Ainsi, on peut réécrire (SP_C) comme ci-dessus.
Comme (SP_C) est la somme de K (SP_C^l), on peut écrire plus simplement

$$(SP_C^l)\begin{cases} \text{Min } d_{w_l}w_l + d_{x^l}x^l \\ \quad x_u^l + x_v^l \leq w_l, \text{ pour tout } e = uv \in E \\ \quad x^l = \sum_{s\in\mathscr{S}} \chi^S \lambda_s^l \\ \quad \sum_{s\in\mathscr{S}} \lambda_s^l = 1 \\ \quad \lambda_s^l \in \{0,1\}, \text{ pour tout } s \in \mathscr{S}. \end{cases}$$

On peut remarquer que si $d_{w_l} > 0$ et pour une solution (entière) de (SP_C^l), on a en fait $w_l = \sum_{s\in\mathscr{S}} \lambda_s^l$. En effet, la contrainte sur les arêtes fait que seul un des terme de la

somme est non nul. Ainsi, on peut prouver que les deux valeurs sont identiques.
On peut alors écrire

$$(SP_C^l)\begin{cases} \text{Min } d_{w_l}\sum_{s\in\mathscr{S}}\lambda_s^l + d_{x^l}x^l \\ \quad x_u^l + x_v^l \leq w_l, \text{ pour tout } e=uv\in E \\ \quad x^l = \sum_{s\in\mathscr{S}}\chi^S\lambda_s^l \\ \quad \sum_{s\in\mathscr{S}}\lambda_s^l = 1 \\ \quad \lambda_s^l \in \{0,1\}, \text{ pour tout } s\in\mathscr{S}. \end{cases}$$

12.4.3 Décomposition de Dantzig-Wolfe et génération de colonnes

On peut réécrire (P_C) de la manière suivante

$$(P_C)\begin{cases} \text{Min } \sum_{l=1}^{K}\sum_{s\in\mathscr{S}}\lambda_s^l \\ \quad \sum_{l=1}^{K} x_u^l = 1, \text{ pour tout } u\in V, \\ \quad x^l = \sum_{s\in\mathscr{S}}\chi^S\lambda_s^l \text{ et } 1\leq l\leq K, \\ \quad \sum_{s\in\mathscr{S}}\lambda_s^l = 1 \text{ et } 1\leq l\leq K, \\ \quad \lambda_s^l\in\{0,1\}, \text{ pour tout } s\in\mathscr{S} \text{ et } 1\leq l\leq K \end{cases}$$

On peut noter ici que toutes les variables et vecteurs ne dépendent pas de l... On peut donc réécrire de la façon suivante

$$(P_C)\begin{cases} \text{Min } \sum_{s\in\mathscr{S}}\lambda_s \\ \quad \sum_{s\in\mathscr{S}}\chi_u^s\lambda_s = 1, \text{ pour tout } u\in V, \\ \quad \lambda_s\in\{0,1\}, \text{ pour tout } s\in\mathscr{S} \end{cases}$$

On peut alors considérer la relaxation linéaire (MP_C) de ce programme

$$(MP_C)\begin{cases} \text{Min } \sum_{s\in\mathscr{S}}\lambda_s \\ \quad \sum_{s\in\mathscr{S}}\chi_u^s\lambda_s = 1, \text{ pour tout } u\in V, \\ \quad \lambda_s\geq 0, \text{ pour tout } s\in\mathscr{S} \end{cases}$$

Notons (MP_C^t) la relaxation linéaire de (P_C) restreinte à un sous-ensemble $\mathscr{S}^t$ de stables.

$$(MP_C^t)\begin{cases} \text{Min } \sum_{s\in\mathscr{S}^t}\lambda_s \\ \quad \sum_{s\in\mathscr{S}^t}\chi_u^s\lambda_s = 1, \text{ pour tout } u\in V, \\ \quad \lambda_s\geq 0, \text{ pour tout } s\in\mathscr{S}^t \end{cases}$$

Le dual de ce PL est

$$(DMP_C^t)\begin{cases} \text{Max } \sum_{v\in V}\pi_v \\ \quad \sum_{v\in s}\chi_v^s\pi_v \leq 1, \text{ pour tout } s\in\mathscr{S}, \\ \quad \pi_v \in R, \text{ pour tout } v\in V \end{cases}$$

La solution optimale de ce programme dual est la solution du programme dual non restreint, c'est-à-dire avec toutes les variables, lorsque toutes les variables λ_s sont telles que :

$$(1-\sum_{v\in V}\pi_v\chi_v^s)\geq 0$$

12.5 Relaxation Lagrangienne pour le problème de coloration

On peut considérer la relaxation Lagrangienne suivante à partir de (P_C)

$$(LRC)\begin{cases} \text{Min } \sum_{l=1}^K w_l - \sum\limits_{u\in V}\gamma_u(\sum_{l=1}^K x_u^l - 1) \\ \quad x_u^l + x_v^l \leq w_l, \text{ pour tout } e=uv\in E \text{ et } 1\leq l\leq K, \\ \quad x_u^l \in\{0,1\}, \text{ pour tout } u\in V \text{ et } 1\leq l\leq K, \\ \quad w_l\in\{0,1\}, \text{ pour tout } 1\leq l\leq K. \end{cases}$$

avec γ le vecteur colonne optimal des multiplicateurs de Lagrange que l'on suppose connu.
On définit le Lagrangien dual de la manière suivante :

$$(LDC)\begin{cases} \text{Max Min } \sum_{l=1}^K w_l - \sum\limits_{u\in V}\gamma_u(\sum_{l=1}^K x_u^l - 1) \\ \quad \gamma\in R_+^{|V|} \end{cases}$$

en supposant qu'on a les w_l et les x_u^l optimaux. On linéarise le Min, et on obtient :

$$(LDC)\begin{cases} \text{Max } \eta \\ \quad \eta \leq (\sum_{l=1}^K w_l)\chi^s - \sum\limits_{u\in V}\gamma_u(\chi_u^s - 1) \\ \quad \gamma\in R_+^{|V|} \\ \quad \eta\in R \end{cases}$$

En notant λ les variables associées aux contraintes du lagrangien dual, on peut écrire

son dual :

$$(DLDC)\begin{cases} \text{Min } \sum_{s\in\mathscr{S}} \lambda_s \\ \quad \sum_{s\in\mathscr{S}} \chi_u^s \lambda_s = 1, \text{ pour tout } u \in V, \\ \quad \lambda_s \in \{0,1\}, \text{ pour tout } s \in \mathscr{S} \end{cases}$$

Troisième partie

Approches Polyédrales pour l'Optimisation Combinatoire

Chapitre 13

Définitions et résultats fondamentaux

Ce chapitre donne toutes les définitions et les résultats théoriques nécessaires pour comprendre et utiliser les approches polyédrales pour l'optimisation combinatoire. Dans ce cours, nous nous intéresserons plus particulièrement aux polyèdree utilisés pour définir des problèmes d'optimisation combinatoire : les polyèdres combinatoires.

13.1 Présentation de l'approche polyèdrale sur un exemple dans $\mathbb{R}^2$

Cette section d'introduction a pour but de donner une vue d'ensemble des algorithmes de coupes. A cette fin, nous allons considérer le cas très simple d'un PLNE à 2 variables.

Prenons le programme (P) suivant :

$$\begin{aligned} \text{Max } z = 2x_1 + x_2 & \\ x_1 - 4x_2 &\leq 0, \\ 3x_1 + 4x_2 &\leq 15, \\ x_1 &\geq 0, \\ x_2 &\geq 0, \\ x_1, x_2 &\in \mathbb{N}. \end{aligned}$$

On note (P^*) sa relaxation linéaire, c'est-à-dire le fait d'enlever la contrainte d'intégrité. En deux dimensions, on peut visualiser le polyèdre donné par les 4 contraintes du programme. Ce polyèdre est dessiné en gris sur la figure 13.1.

La solution optimale de (P^*) est naturellement atteinte sur un des sommets de ce polyèdre est a une valeur fractionnaire x^*_{opt} associée à la valeur $z^*_{opt} = 8 + \frac{7}{16}$.

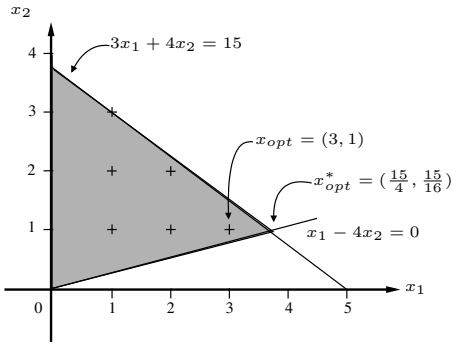


FIGURE 13.1 Domaine de définition de la relaxation linéaire

Les points entiers contenus dans le polyèdre gris forment donc l'ensemble des solutions possibles pour (P). La solution optimale est ici unique, c'est le point x^{opt} de valeur $z_{opt} = 7$. En fait, les solutions de (P) sont aussi prises parmi les sommets d'un polyèdre qui est l'enveloppe convexe des points entiers contenus dans le domaine. On peut aisément comprendre la notion d'enveloppe convexe de points dans un plan par analogie avec un élastique entourant ces points. Ce polyèdre est indiqué également en gris dans la figure 13.2.

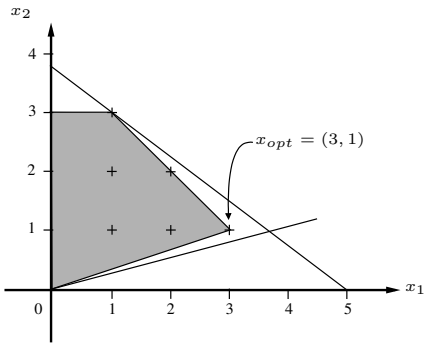


FIGURE 13.2 Enveloppe convexe des solutions entières du problème

On peut remarquer que le polyèdre convexe des points entiers est bien entendu contenu dans le polyèdre des contraintes du programme. Comme il s'agit d'une maximisation on a naturellement $z_{opt} \leq z^*_{opt}$. On peut remarquer que la solution de la relaxation peut éventuellement être entière, dans ce cas elle coïncide avec la solution de (P). C'est en fait cette situation que l'on veut atteindre.

Ces remarques sont toujours vraies quelque soit la dimension du problème. En revanche, si en petites dimensions le calcul de l'enveloppe convexe des points entiers est réalisables, cela devient un problème très difficile en grande dimension car on ne sait le résoudre dans le cas général que par des algorithmes exponentiels.

L'enveloppe convexe est inconnue au départ. Dans le schéma suivant, les droites à l'extérieur schématisent les contraintes connues dans le programme (P) et les points noirs les solutions entières (inconnues a priori) avec leur enveloppe convexe (inconnue de même). Une relaxation linéaire (P) donne pour solution optimale un des sommets de ce polyèdre, x^1 par exemple, qui n'est pas entier (voir illustration 13.3).

$$(P)\begin{cases} \text{Max } cx \\ Ax \leq B \\ x \text{ entier} \end{cases}$$

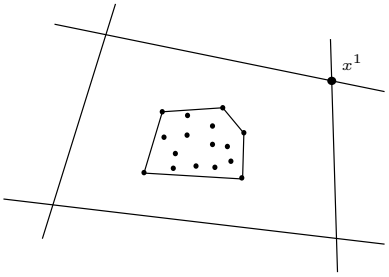


FIGURE 13.3 Solution fractionnaire de la relaxation d'origine

Un algorithme de coupes permet ainsi d'isoler progressivement cette enveloppe convexe en ajoutant des contraintes. Elles "découpent" l'espace entre le polyèdre des contraintes connues et l'enveloppe convexe des solutions.

Sur la figure 13.4 suivante, la contrainte $\alpha x \leq \beta$ indiquée en pointillé permet de séparer un bout inutile du polyèdre des contraintes d'origine.

$$(P_2)\begin{cases} \text{Max } cx \\ Ax \leq B \\ \alpha x \leq \beta \\ x \text{ entier} \end{cases}$$

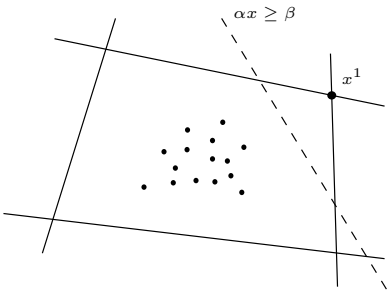


FIGURE 13.4 Solution fractionnaire de la relaxation enrichie d'une contrainte

En réitérant plusieurs fois l'ajout de telles coupes valides dans le cadre d'un algorithme de coupes, on a vu que l'on améliorait la valeur de la relaxation. On peut noter qu'il serait intéressant d'ajoute directement des inégalités du polyèdre de l'enveloppe convexe.

Toute la problèmatique d'une approche polyédrale pour les problèmes d'optimisation combinatoire est de savoir quelles sont les meilleures contraintes à ajouter : c'est-à-dire de connaître au mieux l'enveloppe convexe des solutions du problème.

13.2 Enveloppe convexe

Un ensemble $S \subset I\!R^n$ est *convexe* si, pour deux points quelconques x_1 et x_2 pris dans S, le segment $[x_1, x_2]$ est tout entier contenu dans S, i.e.

$$\forall x_1 \in S, \forall x_2 \in S, \forall \lambda \in [0,1], \text{ alors } \lambda x_1 + (1-\lambda)x_2 \in S.$$

Soit $x \in I\!R^n$. On dit que x est une *combinaison convexe* des points $x^1, ..., x^p \in I\!R^n$ s'il existe p scalaires $\lambda_1, \lambda_2, \ldots, \lambda_p \in I\!R$ tels que

$$x = \sum_{i=1}^{p} \lambda_i x^i \text{ avec } \sum_{i=1}^{p} \lambda_i = 1 \text{ et } \lambda_i \geq 0 \quad \forall i \in \{1, ..., p\}.$$

Soit S un ensemble non vide de points de $I\!R^n$. L'*enveloppe convexe* des points de S, notée *conv(S)*, est l'ensemble de tous les points de $I\!R^n$ qui peuvent s'écrire comme combinaison convexe de points de S. L'enveloppe convexe d'un ensemble de points S peut être vue également comme le plus petit ensemble convexe contenant S.

Etant donné un ensemble de points $S = \{x^1, ...x^p\}$, on peut avoir besoin de tester si un point donné x^* appartient ou non à $conv(S)$. Ce problème se ramène à vérifier si x^* est combinaison convexe des points $x^1, ..., x^p$, c'est-à-dire, s'il existe $\lambda_1, ..., \lambda_p$ tels que

$$\begin{array}{l} \sum_{i=1}^{p} \lambda_i = 1, \\ \sum_{i=1}^{p} \lambda_i x^i = x^*, \\ \lambda_i \geq 0, \text{pour } i = 1, ..., p. \end{array}$$

En fait, un ensemble $S \subset I\!R^n$ est convexe si et seulement si tout point, pouvant s'écrire comme une combinaison convexe des points de S, est dans S. En d'autres termes, S est convexe si et seulement si $S = conv(S)$.
On peut noter que toute intersection d'un nombre fini d'ensemble convexe est encore convexe.

13.3 Polytope des solutions

Les résultats donnés dans ce chapitre s'appliquent pour un polyèdre quelconque. Mais l'objectif de ce document est d'étudier plus particulièrement les polytopes définis à partir d'un problème d'optimisation combinatoire.

Soient $\mathcal{P}$ un problème d'optimisation combinatoire et $\mathcal{S}$ l'ensemble de ses solutions. Le problème $\mathcal{P}$ s'écrit alors

$$\max \{cx \mid x \in \mathcal{S}\}, \tag{13.1}$$

où c est une fonction coût associée aux variables du problème.

• Une enveloppe convexe est un polyèdre

Considérons l'enveloppe convexe $conv(\mathcal{S})$ des solutions de $\mathcal{P}$. Le problème $\mathcal{P}$ a même solution optimale que le problème

$$\max \{cx \mid x \in conv(\mathcal{S})\}. \tag{13.2}$$

En fait, on a le résultat suivant.

Théorème 13.3.1 *Un ensemble (non vide) de points $P \subseteq \mathbb{R}^n$ est un polytope si et seulement s'il existe un ensemble de points S tel que $P = conv(S)$.*

Ce résultat permet de prouver que $conv(S)$ est un polytope (ce que nous avions déjà constaté sans le prouver). Par conséquent, $conv(S)$ peut être décrit par un système fini d'inégalités linéaires et donc le problème 13.2 est un programme linéaire.
Par la programmation linéaire, on sait que 13.2 prend ses solutions parmi les points extrêmes de $conv(S)$, c'est-à-dire précisément les solutions de S. Les problèmes 13.1 et 13.2 sont donc équivalents.
Par conséquent, si nous caractérisons le polyèdre $conv(\mathcal{S})$ par un système d'inégalités linéaires, alors nous ramenons le problème $\mathcal{P}$ à la résolution d'un programme linéaire. Cette enveloppe convexe est appelée *le polytope associé au problème* ou encore *polytope des solutions du problème.*

• Optimiser sur $conv(S)$

La proposition suivante établit la relation entre l'optimisation sur S et celle sur l'enveloppe convexe de S.

Proposition 13.3.2 Soient $S \subseteq \mathbb{R}^n$ un ensemble de points et w un vecteur de $\mathbb{R}^n$. Alors

$$\max\{wx \mid x \in S\} = \max\{wx \mid x \in conv(S)\}.$$

Preuve. Soient $\bar{x} \in S$ tel que $w\bar{x} = \max\{wx \mid x \in S\}$ et $x^* \in conv(S)$ tel que $wx^* = \max\{wx \mid x \in conv(S)\}$. Comme $\bar{x} \in S \subseteq conv(S)$, alors $w\bar{x} \leq wx^*$. De plus, comme x^* est un point optimal sur un polyèdre , x^* est un point extrême de $conv(S)$. Par conséquent $x^* \in conv(S)$. Ce qui implique que $wx^* \leq w\bar{x}$, et alors $wx^* = w\bar{x}$. □

La proposition 13.3.2 établit le lien entre optimiser sur un ensemble de points et optimiser sur un ensemble convexe. En fait, on voit ici le lien entre résoudre un problème d'optimisation combinatoire et résoudre un programme linéaire induit par l'enveloppe convexe des solutions de ce problème. On voit clairement que l'on peut se limiter à étudier les points extrêmes d'un polyèdre lorsque l'on veut optimiser une fonction linéaire sur ce domaine.

• Pas de théorème de Carathéodory

Pour exploiter ce résultat, on peut se demander combien de points sont nécessaires pour définir par combinaison tous les points d'une enveloppe convexe. On a le résultat suivant connu sous le nom de théorème de Carathéodory pour les ensembles convexes.

Théorème 13.3.3 *Soient $S \subseteq \mathbb{R}^n$ un ensemble de points et $x \in conv(S)$. Alors x peut être écrit comme une combinaison convexe de $n' \leq n+1$ points affinement indépendants dans $conv(S)$.*

Le théorème de Carathéodory peut se lire comme le fait que tout point $x \in \mathbb{R}^n$, dans l'enveloppe convexe d'un ensemble S de points, peut s'écrire comme la combinaison convexe d'au plus $n+1$ points affinement indépendants de S. Malheureusement, ceci ne signifie aucunement qu'il existe une sorte de "base convexe" (c'est-à-dire trouver un ensemble générateur). Il faut rester, dans le cas général, dans le cadre des espaces affines (i.e. de la géométrie classique).

13.4 Point intérieur et dimension

Le but de cette section est d'étudier la dimension d'un polyèdre, pris en tant qu'objet dans un espace affin.

• Indépendance linéaire et affine

Des points $x^1, ..., x^k \in \mathbb{R}^n$ sont dits *linéairement indépendants* (resp. *affinement indépendants*) si le système

$$\sum_{i=1}^{k} \lambda_i x^i = 0 \qquad \left(\text{resp. } \sum_{i=1}^{k} \lambda_i x^i = 0 \text{ et } \sum_{i=1}^{k} \lambda_i = 0 \right)$$

admet une solution unique, $\lambda_i = 0$ pour $i = 1, ..., k$.

Si l'on considère la matrice M dont les lignes (ou colonnes) sont les coordonnées de k points $x^1, ..., x^k \in I\!R^n$, on appelle *rang de* M, noté rang(M), le nombre maximum de lignes (ou de colonnes) de M linéairement indépendantes. Conséquemment, si rang(M)=k, les points sont linéairement indépendants. Dans le cas où $k = n$, les points sont linéairement indépendants si le déterminant de la matrice M est non-nul.

Soit $x \in I\!R^n$. On dit que x est une *combinaison linéaire* des points $x^1, ..., x^k \in I\!R^n$ s'il existe k scalaires $\lambda_1, \lambda_2, \ldots, \lambda_k \in I\!R$ tels que $x = \sum_{i=1}^{k} \lambda_i x^i$. Si, de plus, $\sum_{i=1}^{k} \lambda_i = 1$, alors on dit que x est une *combinaison affine* de ces points.

On peut noter que, dans la représentation euclidienne classique, la combinaison linéaire concerne des vecteurs et la combinaison affine des points. Néanmoins, un point x peut-être vu comme un vecteur en considèrant le vecteur $x - 0$ où 0 est l'origine du repère.

Tout d'abord rappelons-nous qu'un ensemble de points linéairement indépendants est affinement indépendant, par contre la réciproque n'est pas vraie. En fait, on a la proposition suivante.

Proposition 13.4.1 Les deux assertions suivantes sont équivalentes :
i) $(x^1, ..., x^n)$ n points affinement indépendants.
ii) $(x^2 - x^1, ..., x^n - x^1)$ $n - 1$ points (vecteurs) linéairement indépendants.

En fait, si 0 n'appartient pas à l'enveloppe affine des points $x^1, ..., x^n$, alors les points $x^1, ..., x^n$ sont affinement indépendants si et seulement s'ils sont linéairement indépendants. (En effet dans ce cas, si $x^1, ..., x^n$ affinement indépendants, $0, x^1, ..., x^n$ affinement indépendants et donc $x^1, ..., x^n$ linéairement indépendants).

• Dimension d'un polyèdre

Définition 13.4.2 Un polyèdre P de $I\!R^n$ est de *dimension* d s'il y a au maximum $d+1$ points affinement indépendants dans P. On écrit alors $dim(P) = d$. Un polyèdre P est dit de *pleine dimension* si $dim(P) = n$.

Considérons maintenant un polyèdre $P \subseteq I\!R^n$ et considérons une description de P par un système de m_1 inéquations et m_2 équations, ainsi :

$$P = \left\{ x \in I\!R^n \mid \begin{array}{ll} A_i x \leq b_i, & i = 1, ..., m_1 \\ B_j x = d_j, & j = 1, ..., m_2 \end{array} \right\}.$$

Cette écriture implique que, pour toute inéquation $A_i x \leq b_i$, $i \in \{1, ..., m_1\}$, il existe une solution $\tilde{x}$ de P telle que $A_i \tilde{x} < b_i$ (sinon, elle serait rangée parmi les équations).

On note dans cette section A, B, b et d les matrices et vecteurs correspondant à cette écriture de P.

• Points intérieurs

Définition 13.4.3 Un point $x^* \in P$ est appelé *point intérieur* de P si $A_i x^* < b_i$ pour $i = 1, ..., m_1$.

Proposition 13.4.4 Si P est non vide, alors P contient un point intérieur.

Preuve. Pour $i = 1, ..., m_1$, soit $x^i \in P$ tel que $A_i x^i < b_i$. Posons $\bar{x} = \frac{1}{m_1} \sum_{i=1}^{m_1} x^i$. Pour $i = 1, ..., m_1, A_i \bar{x} = \frac{1}{m_1} \sum_{i=1}^{m_1} A_i x^i < \frac{1}{m_1} m_1 b_i < b_i$. De plus, pour $j = 1, ..., m_2$, $B_j \bar{x} = d_j$. Donc $\bar{x}$ est un point intérieur. □

• Rang et Dimension

Pour une matrice donnée M, on appelle *rang de* M, noté rang(M), le nombre maximum de lignes (ou de colonnes) de M linéairement indépendantes.

On peut remarquer que si M est de rang r et s'il existe un vecteur l tel que $\{x \in I\!R^n \mid Mx = l\} \neq \emptyset$, alors le nombre maximum de points affinement indépendants dans l'ensemble $\{x \in I\!R^n \mid Mx = l\}$ est $n - r + 1$.

Proposition 13.4.5 Si $P \neq \emptyset$, alors $dim(P) = n - rang(B)$.

Preuve. Posons $k = \text{rang}(B)$. Par la remarque précédente, le système $Bx = 0$ admet $n - k + 1$ solutions $x^1, ..., x^{n-k+1}$ affinement indépendantes. Comme $P \neq \emptyset$, par la proposition 13.4.3, P contient un point intérieur $\bar{x}$, donc $A_i \bar{x} < b_i$ pour $i = 1, ..., m_1$. Alors il existe $\epsilon > 0$ tel que $A_i \bar{x} + \epsilon A_i x^j \leq b_i$, pour tout $i = 1, ..., m_1$ et $j = 1, ..., n - k + 1$. On considère alors les points $y^j = \bar{x} + \epsilon x^j$ pour $j = 1, ..., n - r + 1$. Nous avons $y^j \in P$ pour $j = 1, ..., n - r + 1$. De plus, comme les points $x^1, ..., x^{n-k+1}$ sont affinement indépendants, les points $y^1, ..., y^{n-k+1}$ le sont aussi. Ce qui implique que $\dim(P) \leq n - k$. Par conséquent, $\dim(P) = n - k$. □

En conséquence de cette proposition, un polyèdre est de pleine dimension si et seulement s'il contient un point vérifiant toutes les contraintes du polyèdre avec inégalité stricte.

Si on arrive à donner la dimension d'un polyèdre, on ne connaît pas forcément les inégalités qui le décrivent. Etudions les propriétés de ces inégalités.

13.5 Face d'un polyèdre

On considère dans cette section un polyèdre $P \subseteq \mathbb{R}^n$, et supposons que nous connaissons un système qui décrit P et qui est écrit comme dans la section précédente. Nous allons étudier quelles sont les inégalités de ce système essentielles à la description de (P).

• Face

Définition 13.5.1 Soit $ax \leq \alpha$ une inégalité valide pour P. Le sous-ensemble $F = \{x \in P \mid ax = \alpha\}$ est appelé *face* de P. On dit aussi que F est la face définie par $ax \leq \alpha$. Si $F \neq \emptyset$ et $F \neq P$, on dit que la face F est *propre*.

Par convention, l'ensemble vide et le polytope P lui-même sont considérés comme des faces de P.
Le résultat suivant montre que, dans toute description d'une face, il existe un sous-sytème d'inégalités correspondant à cette face.

Proposition 13.5.2 Un sous-ensemble F non vide de P est une face de P si et seulement s'il existe un sous-système $A'x \leq b$ de $Ax \leq b$ tel que $F = \{x \in P \mid A'x = b'\}$.

Preuve. ($\Leftarrow$) Supposons qu'il existe un sous-système $A'x \leq b$ tel que $F = \{x \in P \mid A'x = b'\}$. Alors, par la convention d'écriture du système $Ax \leq b$, pour tout point $x \in P \setminus F$, il existe au moins une contrainte parmi les inégalités de $A'x \leq b'$ qui ne soit pas vérifiée à l'égalité par x. Considérons l'inégalité $ax \leq \alpha$ obtenue en sommant les inégalités de $A'x \leq b'$. Il est clair que $ax = \alpha$ pour tout $x \in P \setminus F$.
($\Rightarrow$) Soit F une face non vide de P. On veut produire le sous-système composé des inégalités de $Ax \leq b$ telles que tout point de F vérifie ces inégalités à l'égalité. Pour cela, considèrons $ax \leq \alpha$ une inégalité valide pour P telle que $F = \{x \in P \mid ax = \alpha\}$. On considère alors le programme linéaire $max\{ax \mid x \in P\}$. Les solutions optimales de ce programme linéaire sont précisément les éléments de F. Soit (y^1, y^2) une solution optimale duale de ce programme linéaire où y_1 et y_2 sont les vecteurs duaux correspondant respectivement aux systèmes $Ax \leq b$ et $Bx = d$. Par les conditions des écarts complémentaires en programmation linéaire, le sous-système $A'x \leq b'$ de $Ax \leq b$ dont les variables duales ont une valeur positive est précisément le sous-système recherché, i.e. tel que $F = \{x \in P \mid A'x = b'\}$. □

• Face de dimension 0

En fait un point extrême d'un polyèdre peut être aussi défini comme étant une face.

Proposition 13.5.3 Un point $\bar{x} \in P$ est un point extrême de P si et seulement si $\bar{x}$ est une face de dimension 0.

Preuve. Soit $\bar{x}$ un point quelconque de P. On associe à $\bar{x}$ le polyèdre $\bar{P}$ obtenu à partir de P en transformant toute inéquation vérifiée à l'égalité par $\bar{x}$ en une égalité. Notons alors $\bar{A}x \leq \bar{b}, \bar{B}x = \bar{d}$ le système des inéquations et des équations décrivant $\bar{P}$. Remarquons qu'alors $\bar{A}\bar{x} < \bar{b}$.
($\Rightarrow$) Supposons par contraposée que $\bar{x}$ n'est pas une face de dimension 0, alors $dim(\bar{P}) > 0$ et,ainsi, par la proposition 13.4.5, $rang(\bar{B}) < n$. Par conséquent, il existe un point $y \neq 0$ tel que $\bar{B}y = 0$. Comme $\bar{A}x < \bar{b}$, il existe un scalaire $\epsilon > 0$ tel que $\bar{A}y + \epsilon\bar{A}y \leq \bar{b}$, et ainsi $\bar{x}^1 = \bar{x} + \epsilon y \in \bar{P} \subset P$. Aussi, ϵ peut être choisi suffisamment petit pour que $\bar{x}^2 = \bar{x} - \epsilon y \in P$. Comme ainsi $\bar{x} = \frac{1}{2}(\bar{x}^1 + \bar{x}^2)$, $\bar{x}$ n'est pas un point extrême de P.
($\Leftarrow$) Si $\bar{x}$ est une face de dimension 0, alors $\bar{P}$ est un polyèdre de dimension 0 et ainsi $rang(\bar{B}) = n$. Alors $\bar{x}$ est la solution unique du système $\bar{B}x = d$. Supposons qu'il existe $\bar{x}^1$ et $\bar{x}^2$ dans P tels que $\bar{x} = \frac{1}{2}(\bar{x}^1 + \bar{x}^2)$. Or $\bar{x}^1$ et $\bar{x}^2$ sont des solutions de $\bar{B} = \bar{d}$. (En effet, on peut le déduire de $\bar{B}\bar{x}^1 \leq \bar{d}$, $\bar{B}\bar{x}^2 \leq \bar{d}$ et $\bar{B}\bar{x}^1 + \bar{B}\bar{x}^2 = 2\bar{d}$.) Par conséquent, $\bar{x} = \bar{x}^1 = \bar{x}^2$, et donc $\bar{x}$ est un point extrême de $\bar{P}$. □

• **Face "minimale"**

En fait, les faces peuvent être contenues les unes dans les autres. On peut tout d'abord se poser la question des faces *minimales*, c'est-à-dire celles qui ne contiennent strictement pas d'autre face.

Pour régler cette question, on a les résultats suivants.

Proposition 13.5.4 Un sous-ensemble F non vide de P est une face minimale de P si et seulement s'il existe un sous-système $A'x \leq b'$ de $Ax \leq b$ tel que $F = \{x \in \mathbb{R}^n \mid A'x = b', Bx = d\}$.

Corollaire 13.5.5 *Toute face minimale non vide de P est de dimension $n - rang \begin{pmatrix} A \\ B \end{pmatrix}$.*
Les faces minimales non vides d'un polytope sont des points extrêmes.

La notion de face minimale se ramène donc en fait à la recherche des points extrêmes. Ce n'est pas étonnant, ce sont les faces qui sont nécessaires à une formulation "minimale" du problème, cela revient en fait à énumérer tous les points extrêmes, or cela est très coûteux et cela abandonne l'outil de la Programmation Linéaire. Cherchons plutôt quelles sont les inégalités essentielles à la formulation.

13.6 Facettes et description minimale

Dans la suite, nous allons voir que les inégalités qui sont nécessaires à la description de P sont celles qui définissent des faces maximales.
Remarquons que si F est une face propre de P, alors dim(F)≤dim(P)-1.

• Redondance dans une formulation linéaire

Une inégalité est dite *redondante* dans un système $Ax \leq b$ définissant un polyèdre P, si le sous-système, obtenu à partir de $Ax \leq b$ en supprimant cette inégalité, définit le même polyèdre P. Soit $x^* \in \mathbb{R}^n$. Inversement, une inégalité non-redondante est dite *essentielle.*

• Facette et redondance

Définition 13.6.1 Si F est propre et $dim(F) = dim(P) - 1$, alors F est appelée *facette* de P.

Proposition 13.6.2 Supposons que $P \neq \emptyset$. Alors toute contrainte valide pour P qui ne définit pas de facette de P est redondante.

Preuve : Soit $A_i x \leq b_i$ une contrainte valide essentielle à la description de P. On veut montrer que $A_i x \leq b_i$ définit une facette de P. Pour cela, remarquons que, comme cette inégalité est nécessaire dans P, alors il doit exister un point $x^* \in \mathbb{R}^n \setminus P$ tel que

$$\begin{aligned} &A_j x^* \leq b_j, \forall j \in \{1, ..., m_1\} \setminus \{i\}, \\ &A_i x^* > b_i, \\ &Bx^* = d. \end{aligned}$$

Puisque P est non vide, il contient un point intérieur, disons $\bar{x}$, et donc $A_i \bar{x} < b_i$. Soit z un point sur le segment entre x^* et $\bar{x}$ tel que $A_i z = b_i$, i.e., il existe $0 < \lambda < 1$ avec $z = \lambda \bar{x} + (1 - \lambda) x^*$. Ainsi

$$\begin{aligned} &A_j z < b_j, \forall j \in \{1, ..., m_1\} \setminus \{i\}, \\ &A_i z = b_i, \\ &Bz = d. \end{aligned}$$

Ceci implique que z appartient à la face de P définie par $A_i x \leq b_i$. Aussi notons que le système donné par les équations de F est $A_i x = b, Bx = d$ avec $A_i x = b$ linéairement indépendante des lignes de B (sinon, l'inégalité $A_i x \leq b_i$ ne serait pas essentielle). Par conséquent le rang de ce système est $rang(B) + 1$. Donc $dim(F) = n - (rang(B) + 1)$

et comme $dim(P) = n - rang(B)$, $dim(F) = dim(P) - 1$. $A_i x \leq b_i$ est bien une facette de P. □

• Description minimale du polyèdre par des inégalités

Pour pouvoir prouver les résultats suivants, nous avons besoin des 2 lemmes de Farkas.

Lemme 13.6.3 (Lemme de Farkas pour les inéquations). *Etant donnés une matrice $m \times n$ A et un vecteur $b \in \mathbb{R}^n$, le système $Ax \leq b$ admet une solution si et seulement s'il n'existe pas un vecteur $y \geq 0$ de $\mathbb{R}^m$ tel que $yA = 0$ et $yb < 0$.*

Lemme 13.6.4 (Lemme de Farkas) *Le système $Ax = b$ admet une solution (resp. solution positive) si et seulement s'il n'existe pas un vecteur y tel que $yA = 0$ et $yb < 0$ (resp. $yA \geq 0$ et $yb < 0$).*

Le résultat suivant montre qu'une facette d'un polyèdre P est au moins définie par une contrainte valide pour P.

Proposition 13.6.5 Pour toute facette F de P, une des inéquations définissant F est nécessaire dans la description de P.

Preuve : Soit $I \subset \{1, ..., m_1\}$ l'ensemble des indices des inégalités $A_i x \leq b_i$ de $Ax \leq b$ qui définissent F. Soit $\bar{P}$ le polyèdre définie par ces inégalités. Nous allons montrer que $P \setminus \bar{P}$ n'est pas vide, ce qui montre qu'au moins une des inégalités parmi $A_i x \leq b_i$, $i \in I$, doit apparaître dans la description de P. Par la suite, nous considérons une contrainte $A_i x \leq b_i$ pour un $i \in I$.
Comme $F \neq \emptyset$, prenons x^0 un point intérieur de F. De plus, comme $F \neq P$, A_i est indépendant des lignes de B, c'est-à-dire que le système $yB = A_i$ n'a pas de solution. Par le lemme de Farkas, le système $\{Bx = 0, A_i x > 0\}$ admet une solution, disons x^1. Comme $A_k x^0 < b_k$ pour tout $k \in \{1, ..., m_1\} \setminus I$, il existe $\epsilon > 0$ tel que $A_k x^0 + \epsilon A_k x^1 \leq b_k$ pour $k \in \{1, ..., m_1\} \setminus I$.
Notons $x^2 x^0 + \epsilon x^1$. D'abord, remarquons que $Bx^2 = 0$. Alors x^2 est un point du segment reliant x^0 à x^1 qui est dans $\bar{P}$. En effet, $A_k x^2 \leq b_k$, pour $k \in \{1, ..., m_1\}$. De plus, comme $A_i x^1 > 0$ et $A_i x^0 = b_i$, on a $A_i x^2 = A_i x^0 + \epsilon A_i x^1 > b_i$. Par conséquent, $x^2 \notin P$ donc $x^2 \in P \setminus \bar{P}$. □

• Inégalités équivalentes

En fait une seule inégalité est nécessaire par facette dans une description de P. En effet, on peut définir que deux inégalités valide $a_1 x \leq \alpha_1$ et $a_2 x \leq \alpha_2$ sont *équivalentes*

s'il existe $\lambda > 0$ et un vecteur μ de $\mathbb{R}_2^m$ tel que $a_2 = \lambda a_1 + \mu B$ et $\alpha_2 = \lambda \alpha_1 + \mu d$. (Dans les notations de l'écriture de P). On a alors le résultat suivant.

Proposition 13.6.6 Supposons que $P \neq \emptyset$. Si deux inégalités valides de P définissent la même facette, alors elles sont équivalentes.

D'après cette proposition, pour toute facette F de P, une et une seule inégalité définissant F est nécessaire dans un système décrivant P. Comme conséquence des propositions précédentes, on a le théorème suivant.

Théorème 13.6.7 *Le système définissant P est minimal (en nombre d'inéquations et d'équations) si et seulement si les lignes de B sont linéairement indépendantes et toute inégalité $A_i x \leq b_i$, $i = 1, ..., m_1$ définit une facette distincte de P.*

La description d'un polyèdre de pleine dimension est même unique à une multiplication d'un scalaire près.

Corollaire 13.6.8 *Si P est un polyèdre de pleine dimension, alors il existe un système linéaire minimal unique (à des multiplications par des scalaires près) qui décrit P. De plus, toute contrainte de ce système définit une facette distincte de P.*

13.7 Etude faciale sur $conv(S)$

Comme on vient de le voir, les résulats concernant les faces et facettes d'un polytope permet de bien déterminer le système des inégalités linéaires permettant de décrire le polytope des solutions. On peut même limiter l'étude de la validité et des facettes aux seuls points de S. On a en effet les deux lemmes suivants.

Lemme 13.7.1 *Une inégalité $ax \leq \alpha$ est valide pour S si et seulement si elle est valide pour $conv(S)$.*

Lemme 13.7.2 *Si F est une face non vide de $conv(S)$ de dimension $p - 1$, alors il existe p points affinement indépendants dans $S \cap F$.*

Ces deux lemmes nous indiquent que l'on peut démontrer si une inégalité est valide et si elle définit une facette d'un polyèdre des solutions en s'intéressant uniquement aux solutions du problème.

13.8 Approches polyèdrales

Voici une rapide description de ce que l'on nomme une approche polyédrale d'un problème d'optimisation combinatoire.

• Formulation entière

En pratique, nous avons vu dans la première section qu'un problème d'optimisation combinatoire $\mathcal{P}$ peut souvent s'écrire comme un programme linéaire en nombres entiers de la forme

$$(P_1)\ \max\{cx \mid Ax \leq b, 0 \leq x \leq 1, x \text{ entier}\}$$

où $Ax \leq b$ est un ensemble d'inégalités linéaires qui peut être de taille exponentielle par rapport au nombre n de variables.

Cette formulation PLNE (P_1) définit donc un ensemble de solutions entières $\mathcal{S}$ qui sont très exactement les solutions du problème $\mathcal{P}$. La contrainte "x entier" est dite *contrainte d'intégrité* (ou intégralité ou entièreté).

• Relaxation linéaire et points extrêmes fractionnaires

Si on relaxe la contrainte d'intégrité dans (P_1), on obtient la *relaxation linéaire* (P_1^*) de la formulation entière.
Elle peut parfois fournir une solution entière. C'est le cas par exemple des matrices de flot si les capacités sont entières ou des matrices correspondants au problème d'affectation. Plus généralement, c'est le dans certains types de matrices : matrices TDI (Totally Dual Integral), matrices TU (Totally Unimodular),...
Mais ce n'est pas le cas en général. Cela signifie que le polyèdre définit par (P_1^*) possède des *points extrêmes fractionnaires*. Bien entendu, il peut exister des instances dont la solution optimale correspond par chance à un point entier parmi un ensemble de points qui seraient majoritairement fractionnaires, mais ce n'est pas le cas général. On peut donc dire que cette relaxation linéaire n'est pas une description suffisante. L'approche polyèdrale consiste à "renforcer" cette formulation pour que la relaxation linéaire propose un point extrême entier. Plus exactement, on cherche à ajouter suffisamment de contraintes de l'enveloppe convexe dans le but d'avoir produit celles dont l'intersection des faces va donner un point extrême entier optimal, c'est-à-dire une solution optimale du problème.

• Définition théorique du polytope associé

Le problème $\mathcal{P}$ peut alors s'écrire également

$$\max \{cx \mid x \in \mathcal{S}\},$$

où c est une fonction coût associée aux variables du problème. Considérons l'enveloppe convexe $conv(\mathcal{S})$ des solutions de $\mathcal{P}$. Le problème $\mathcal{P}$ est alors équivalent au programme linéaire

$$\max \{cx \mid x \in conv(\mathcal{S})\}.$$

Par conséquent, si nous caractérisons le polyèdre $conv(\mathcal{S})$ par un système d'inégalités linéaires, alors nous ramenons le problème $\mathcal{P}$ à la résolution d'un programme linéaire.

De plus les inégalités de $Ax \leq B$ sont toutes valides par rapport au polyèdre $conv(\mathcal{S})$, certaines peuvent même être des inégalités définissant des facettes de ce polyèdre des solutions.

L'*approche polyédrale*, introduite par Edmonds en 1965 [3], dans le cadre du problème du couplage, consiste à étudier le polytope $conv(\mathcal{S})$ afin de pouvoir résoudre $\mathcal{P}$ comme un programme linéaire. La caractérisation complète du polytope $conv(\mathcal{S})$ est généralement difficile à obtenir. Par ailleurs, une description complète du polyèdre peut comporter un nombre exponentiel d'inégalités. Cependant, un nombre réduit de ces inégalités peut être suffisant pour résoudre le problème à l'aide d'une *méthode de coupes* ou *de coupes et branchement* (Branch&Cut method) (voir section correspondante).

13.9 Exercices

13.9.1 Points extrêmes et facettes du polytope du stable

Soit $G = (V, E)$ un graphe simple non orienté sans boucle.

a). Soit $P(G)$ le polytope des stables de G, i.e.,

$$P(G) = conv\{\chi^s \mid S \text{ est un stable de } G\}.$$

Montrer que $P(G)$ est de pleine dimension.

Réponse : Il suffit de produire $n + 1$ vecteurs de $\mathbb{R}^n$ affinement indépedants. Par exemple, les vecteurs d'incidence de l'ensemble vide $\emptyset$ et des ensembles $E_u = \{u\}$, $u \in V$. En effet, ces ensembles sont bien tous des stables. De plus, les vecteurs $\chi^{E_u} - \chi^{\emptyset}$ sont bien linéairement indépendants car la matrice formée par ces vecteurs-colonne est la matrice identité qui est inversible.

b). Montrez que les contraintes

$$x(v) \geq 0 \quad \text{pour tout sommet } v \text{ de } V, \tag{13.3}$$

$$x(u) + x(v) \leq 1 \quad \text{pour toute arête } uv \text{ de } E, \tag{13.4}$$

sont valides pour $P(G)$.

Réponse : Les contraintes triviales sont clairement valides. Les contraintes de type (13.4) sont valides pour $P(G)$ car tout vecteur d'incidence d'un stable les vérifie. En effet, les deux extrémités d'un arête ne peuvent toutes deux appartenir à un même stable.

c). Montrer que les contraintes (13.3) définissent des facettes de $P(G)$.

Réponse : Soit $v \in V$. Posons $F = \{\chi^S \in I\!R^n \mid S \text{ stable et } \chi^S(v) = 0\}$ la face associée à la contrainte $x(v) \geq 0$. Pour prouver que cette face est une facette, il faut montrer qu'elle est propre et qu'elle contient n vecteurs affinement indépendants.
F est bien propre car elle n'est pas vide (par exemple l'ensemble vide est un stable et son vecteur (nul) d'incidence appartient à F) et $F \neq P(G)$ (en effet, le vecteur d'incidence χ^{E_v} du stable E_v appartient à $P(G)$ mais pas à F).
De plus, les vecteurs $\chi^{\emptyset}$ et χ^{E_u}, $u \in V \setminus \{v\}$, sont bien n vecteurs d'incidence de stables affinement indépendants.

d). Soit K une clique de G (sous-graphe complet de G). Montrer que la contrainte

$$\sum_{v \in K} x(v) \leq 1, \tag{13.5}$$

est valide. Montrer de plus qu'elle définit une facette de $P(G)$ lorsque K est une clique maximale au sens de l'inclusion (c'est-à-dire que K n'est pas contenue dans une clique plus grande). En déduire que les contraintes (13.4) peuvent ne pas définir des facettes.

Réponse : La contrainte est bien valide car un stable ne peut pas contenir plus d'un sommet d'un graphe complet. Sa face F associée est propre car elle n'est pas vide et est différente de $P(G)$ (ne contient pas $\chi^{\emptyset}$). De plus, considérons les vecteurs E_u, $u \in K$ qui sont $|K|$ vecteurs contenus dans F. Il manque donc $n - |K|$ vecteurs à exhiber. Or comme K est un sous-graphe complet maximal dans G, cela signifie que pour tout sommet $w \in V \setminus K$, il existe un sommet u_w dans K tel que w et u_w ne sont pas adjacents (en effet, dans le cas contraire, K ne serait pas maximale). Considérons alors les n stables E_u, $u \in K$ et les $\{w, u_w\}$, $w \in V \setminus K$ qui sont n vecteurs affinement indépendants dans F. La contrainte définit bien une facette de $P(G)$.
En fait, les contraintes d'arêtes (13.4) sont des sous-graphes complets à 2 éléments.

Par conséquent, ces contraintes ne vont définir des facettes que si elles sont maximales dans G. Et inversement, si une arête est contenue dans un sous-graphe complet plus grand, alors sa face associée n'est pas une facette.

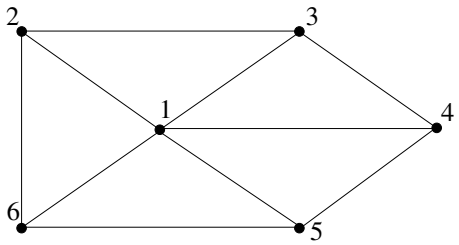
e). Soit C un cycle impair de G. Montrer que la contrainte :

$$\sum_{v \in V(C)} x(v) \leq \frac{|C|-1}{2} \tag{13.6}$$

est valide pour $P(G)$.
Proposer un algorithme de séparation pour utiliser une telle contrainte.

Réponse : Cette contrainte est équivalente au fait qu'un stable sur un cycle impair de taille k ne peut contenir qu'au plus $\frac{k-1}{2}$ sommets du cycle. En effet, s'il en contenait d'avantage, il y aurait nécessairement deux sommets adjacents dedans, une contradiction. On peut également prouver ce résultat par la somme de Chvàtal-Gomory.
On peut montrer que l'ensemble des contraintes de cycles impairs peut être séparé en temps polynomial. Cela revient à rechercher des cycles impairs dans un graphe particulier.

f). On considère le problème du stable de poids maximum sur le graphe $H_1 = (V, E, c)$ (où le c est un poids 1 sur chaque sommet) suivant :



Montrer que le polyèdre défini par les contraintes (**??**), (13.4) et (13.6) associé à H_1 n'a pas tous ses sommets à coordonnées entières. Comment peut-on couper ce point fractionnaire ?

Réponse : Le point $(\frac{1}{5}, \frac{2}{5}, \frac{2}{5}, \frac{2}{5}, \frac{2}{5}, \frac{2}{5})$ est bien fractionnaire. Chacune des contraintes triviales et d'arêtes est vérifiée. De plus les contraintes associées aux 6 cycles impairs élémentaires de H_1 sont vérifiées à l'égalité. Il s'agit des cycles reliant les sommets $(1,2,3), (1,3,4), (1,4,5), (1,5,6), (1,6,2)$ et $(2,3,4,5,6)$. De plus la matrice composée des vecteurs de ces 6 contraintes est inversible (calcul du déterminant). Donc ce point est bien un point extrême fractionnaire de H_1.
On peut constater que le sommet 1 joue le rôle du centre relié universellement aux 5

autres sommets (on appelle ce graphe une roue). En fait, un stable sur H_1 contient soit le sommet 1 seul, soit 2 sommets parmi les sommets du cycle extérieur. Par conséquent, la contrainte $2x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 2$ est valide pour $P(G)$. En effet, la contrainte est bien vérifiée par tous les stables de H_1. De plus, cette contrainte coupe bien le point extrême fractionnaire voulu.

13.9.2 Facettes du polytope du sac-à-dos en 0-1

Considérons un problème de sac à dos P en 0-1 (0-1 knapsack polytope).

$$\begin{array}{c} Max \quad c_1x_1 + c_2x_2 + ... + c_nx_n \\ a_1x_1 + a_2x_2 + ... + a_nx_x \leq b \\ x_i \in \{0,1\} \text{ pour } i = 1, ..., n \end{array}$$

On suppose que $a_i > 0$ pour $i = 1, ..., n$ et $b > 0$. On note le polytope du knapsack, comme étant l'enveloppe convexe des vecteurs entiers $\chi \in \{0,1\}^n$ qui satisfont la contrainte de sac-à-dos, i.e.

$$P_K(a,b) = conv\{\chi \in \{0,1\}^n \mid a\chi \leq b\}.$$

Un sous-ensemble R de $\{1, ..., n\}$ est dit *recouvrement* (cover) de P si $\sum_{i \in R} a_i > b$.

1). Montrer que si R est un recouvrement de P alors la contrainte suivante est valide pour $P_K(a,b)$, i.e

$$\sum_{i \in R} x_i \leq |R| - 1.$$

Réponse : Supposons le contraire, c'est-à-dire, qu'il existe une solution du sac-à-dos χ^* telle que $\sum_{i \in R} \chi_i^* > |R| - 1$. Ainsi $\sum_{i \in R} \chi_i^* = |R|$, c'est-à-dire ici $\chi_i^* = 1$ pour tout $i \in R$. Donc, par définition, $\sum_{i \mid \chi_i^* = 1} a_i > b$, une contradiction.

2). Soit le problème de sac-à-dos Q suivant

$$\begin{array}{c} Max \quad z = 10x_1 + 19x_2 + 12x_3 + 12x_4 + x_5 + 3x_6 + x_7 \\ 11x_1 + 6x_2 + 6x_3 + 5x_4 + 5x_5 + 4x_6 + x_7 \leq 19 \\ x_i \in \{0,1\} \text{ pour } i = 1, ..., 7. \end{array}$$

Les ensembles suivants sont-ils des recouvrements pour le problème de sac-à-dos Q : $R_1 = \{1,2,3\}$, $R_2 = \{2,3,4\}$ et $R_3 = \{3,4,5,6\}$? Donner les contraintes de recouvrement correspondantes pour ceux qui le sont.

Réponse : $R_1 = \{1,2,3\}$ et $R_3 = \{3,4,5,6\}$ sont bien des recouvrements car $11+6+6 > 19$ et $6+5+5+4 > 19$. En revanche, $R_2 = \{2,3,4\}$ n'est pas un recouvrement car $6+6+5 \leq 19$.
$c_1 : x_1 + x_2 + x_3 \leq 2$
$c_3 : x_3 + x_4 + x_5 + x_6 \leq 3$

3). Donner pour chacune des contraintes données dans 2) un point extrême du domaine de Q' (la relaxation linéaire de Q) qui peut être coupé par cette contrainte. Justifier votre réponse.

Réponse : Pour $c_1 : x_1 + x_2 + x_3 \leq 2$: le point fractionnaire $(1, 1, \frac{1}{3}, 0, 0, 0, 0)$ vérifie à l'égalité les contraintes triviales associées aux indices $1, 2, 4, 5, 6, 7$. De plus, il vérifie à l'égalité la contrainte de sac-à-dos $(11 + 6 + 2 = 19)$. Donc ce point vérifie à l'égalité 7 contraintes du système. De plus, ces 7 contraintes sont clairement linéairement indépendantes. Par conséquent, c'est bien un point fractionnaire. En revanche, comme $1 + 1 + \frac{1}{3} > 2$, ce point est coupé par la contrainte c_1, c'est-à-dire que si l'on ajoute la contrainte c_1 au système, ce point fractionnaire ne sera plus un point extrême du domaine de définition du système.

4). Soit R un recouvrement. Donnez la dimension du polyèdre

$$P_K^R(a,b) = P_K(a,b) \cap \{x \in I\!R^n \mid x(i) = 0, \forall i \in \{1,...,n\} \setminus R\}.$$

Réponse : Ce polyèdre possède seulement $|R|$ composantes. Il est de pleine dimension si on peut donner $|R| + 1$ points solutions affinement indépendants dans $P_K^R(a,b)$. Considérons les vecteurs canoniques χ^j, $j \in R$ tel que $\chi^j(j) = 1$ et $\chi^j(i) = 0$, $i \in \{1,...,n\} \setminus \{j\}$. On remarque qu'ils appartiennent tous à $P_K^R(a,b)$. De plus, le vecteur nulle est aussi solution et appartient à $P_K^R(a,b)$. Comme ces $|R|+1$ vecteurs sont clairement linéairement indépendants, ils sont aussi affinement indépendants et donc $P_K^R(a,b)$ est de dimension R, c'est-à-dire de pleine dimension.

5). Donnez une condition suffisante pour qu'une contrainte de recouvrement définisse une facette de $P_K^R(a,b)$.

Réponse : Soit R un recouvrement. Posons la face associée à la contrainte de recouvrement basée sur R :

$$F_R = \{\chi \in \{0,1\}^n \mid a\chi \leq b \text{ et } \sum_{i \in R} \chi(i) = |R| - 1\}.$$

Cette face est un sous-ensemble du polyèdre $P_K^R(a,b)$ qui est de dimension $|R| + 1$. Pour prouver que cette face est une facette, il faut tout d'abord montrer qu'elle soit

propre, c'est-à-dire prouver qu'elle contienne au moins un élément. Pour cela, il faut qu'il existe un vecteur $\chi \in \{0,1\}^n$ tel que $a\chi \leq b$ et qui soit tel que $\chi(i) = 1$ pour exactement $|R| - 1$ élements de R. Il faut donc ajouter des conditions sur R de façon à ce qu'il existe un tel R. En fait, il faut qu'il existe $i \in R$ tel que $R \setminus \{i\}$ soit une solution du problème. Pour que la face soit propre, il faut aussi que la face ne soit pas tout l'espace, mais comme le vecteur nul n'est pas dans F_R, c'est bien le cas.
De plus, pour que la face soit une facette, elle doit contenir $|R|$ vecteurs affinement indépendants. Une façon suffisante est de répéter n fois l'idée du paragraphe précédent, donc R doit être tel que pour tout élément i de R, $R \setminus \{i\}$ soit une solution (i.e. n'est plus un recouvrement). On dit alors que R est un *recouvrement minimal*.
Supposons donc que R est un recouvrement minimal. Avec une telle hypothèse R est propre et, d'autre part, les vecteurs d'incidence $R \setminus \{i\}$, $i \in R$, appartiennent tous à F_R et sont linéairement indépendants. On a donc prouvé que si R est minimal alors la contrainte de recouvrement définit une facette de $P_K^R(a,b)$.

6). Est-ce que cette condition est suffisante ?

Réponse : Supposons que R n'est pas minimal. C'est-à-dire qu'il existe dans R un élément i_0 tel que $R \setminus \{i_0\}$ soit encore un recouvrement. Alors posons R' ce recouvrement. On remarque alors que si contrainte de recouvrement $\sum_{i \in R'} x_i \leq |R'| - 1$ est vérifié alors celle associée à R l'est aussi. Comme la contrainte $\sum_{i \in R} x_i \leq |R| - 1$ est dominée par une autre contrainte, alors elle ne peut pas définir une facette. La condition sur la minimalité de R est donc nécessaire et suffisante pour que la contrainte de recouvrement associée définisse une facette.

Chapitre 14

Caractérisation de polyèdre

Dans ce chapitre, nous nous intéressons à la caractérisation d'un polyèdre au travers de l'exemple historique du polytope des solutions du problème de couplage maximal.

14.1 Définitions et outils de preuve

14.1.1 Définitions

On appelle *caractérisation* (ou caractérisation complète) d'un polyèdre le fait de décrire l'ensemble des inégalités qui composent ce polyèdre. Ainsi le PL composé par ces inégalités est entier.

Dans le cas où l'on connaît la caractérisation d'un polyèdre et que l'on sait énumérer ou séparer polynomialement toutes les inégalités de cette caractérisation, on a alors déterminé que le problème est polynomial.

Comme cité dans la section 8, il existe plusieurs techniques pour caractériser un polyèdre :
- montrer que les solutions optimales d'un PL et les points extrêmes du polyèdre sont en bijection ; puis montrer que ce PL est TU ou TDI.
- énumérer toutes les familles de contraintes définissant des facettes du polyèdre.

Nous présentons ici cette dernière technique au travers de l'exemple historique du polytope du couplage.

14.1.2 Outils de preuve

Cette section contient quelques résultats et notations qui seront utilisés dans les preuves des sections de ce chapitre. Néanmoins, toutes les notations seront réexpliquées dans les autres sections, il s'agit ici plus de les expliciter que de les définir.

Soit $\mathcal{P}$ un problème combinatoire consistant à recherche une solution optimale dans un ensemble $\mathcal{S} \subset 2^E$ de solutions où E est un ensemble fini de taille n et valué par c, i.e. $\mathcal{P} : max\{c(S) \mid S \in \mathcal{S}\}$.
On pose alors χ^S le vecteur d'incidence de S tel que $\chi^S(e) = 1$ si $e \in S$ and $\chi^S(e) = 0$ si $e \in E \setminus S$. Le polytope $P(\mathcal{S})$ associé à ce problème se définit comme l'enveloppe convexe des vecteurs d'incidence des élements de $\mathcal{S}$, i.e.

$$P(\mathcal{S}) = conv\{\chi^S \in \{0,1\}^n \mid S \in \mathcal{S}\}.$$

Pour prouver qu'une contrainte définit une facette de $P(\mathcal{S})$, on peut, par exemple, utiliser le résultat suivant :

Lemme 14.1.1 *Soit $ax \leq \alpha$ une contrainte définissant une facette F de $P(\mathcal{S})$.*
Soit $a'x \leq \alpha'$ une contrainte valide de $P(\mathcal{S})$ et soit F' la face associée à $a'x \leq \alpha'$.
Alors si $F \subset F'$ alors $F = F'$.

Le lemme précédent est simplement dû aux propriétés de maximalité (au sens de l'inclusion) des facettes d'un polyèdre. Ce lemme est une des clés de certaines preuves du fait qu'une face d'une contrainte est une facette.

Le lemme suivant, en revanche, est spécifique aux polytopes de pleine dimension. On utilise ce lemme sous les deux formes proposées i) et ii) qui ne sont en fait que la contraposée l'une de l'autre.

Lemme 14.1.2 *Supposons que $P(\mathcal{S})$ soit de pleine dimension.*
Soit $ax \leq \alpha$ une contrainte définissant une facette de $P(\mathcal{S})$.
Posons $\mathcal{C}_a$ l'ensemble des solutions de $\mathcal{S}$ dont le vecteur d'incidence associé vérifie l'inégalité $ax \leq \alpha$ à l'égalité, i.e.

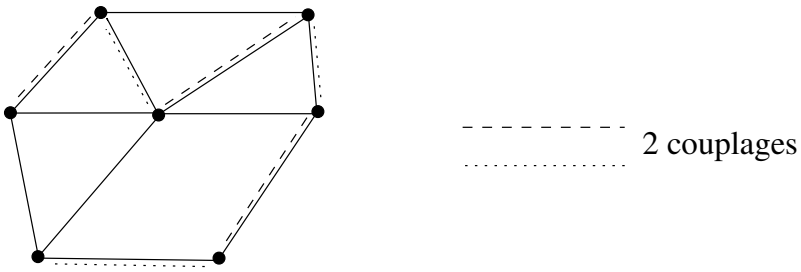
$$\mathcal{C}_a = \{S \subset E \mid C \text{ solution de } \mathcal{S} \text{ et } ax^C = \alpha\}.$$

Soit $a'x \leq \alpha'$ une contrainte valide de $P(\mathcal{S})$.
i). Si pour toute solution de $C \in \mathcal{C}_a$, $a'x^C = \alpha'$, alors $ax \leq \alpha$ et $a'x \leq \alpha'$ sont identiques à un coefficient multiplicateur positif prêt.
ii). Si $ax \leq \alpha$ et $a'x \leq \alpha'$ ne sont pas identiques à un coefficient multiplicateur positif prêt, alors il existe$C \in \mathcal{C}_a$ tel que $a'x^C < \alpha'$.

Le petit ii) de ce théorème est fréquemment la clef des preuves de caractérisation.

14.2 Le polytope du couplage

Soit $G = (V, E)$ un graphe. Un *couplage* (*matching*, en anglais) est un sous-ensemble d'arêtes deux à deux non-adjacentes. Si chaque arête e de G est munie d'un certain poids $c(e)$, *le problème du couplage maximum* dans G consiste à déterminer un couplage dont le poids total des arêtes est maximum, i.e. $\sum_{e \in E} c(e)$ maximum.



14.2.1 Formulation et points extrêmes

Le problème du couplage dans un graphe est historiquement le premier problème d'optimisation combinatoire qui a été étudié par une approche polyédrale par Jack Edmonds en 1965. Il propose une caractérisation complète du polyèdre.

Donnons tout d'abord une formulation du problème comme un programme linéaire en variables binaires.

Soit $\chi(e)$, $e \in E$, des variables binaires associées aux arêtes de G telle que

$$\chi(e) = \begin{cases} 1 & \text{si } e \text{ prise dans le couplage} \\ 0 & \textit{sinon} \end{cases}$$

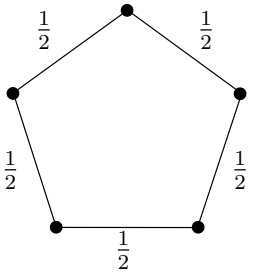
Comme dans un couplage, il y a au plus une arête incidente à chaque sommet, le problème du couplage maximum est équivalent au PLNE $\mathcal{P}$ suivant.

$$\begin{aligned} \text{Max} \sum_{e \in E} c(e)x(e) & \\ \sum_{e \in \delta(u)} x(e) \leq 1, & \quad \forall v \in V, \qquad (14.1) \\ 0 \leq x(e) \leq 1, & \quad \forall e \in E, \qquad (14.2) \\ x(e) \text{ entier}, & \quad \forall e \in E. \qquad (14.3) \end{aligned}$$

Considérons le polyèdre associé au problème du couplage maximum. On note $P_M(G)$ l'enveloppe convexe des vecteurs d'incidence des couplages de G, i.e.

$$P_M(G) = con\{\chi^M \in I\!R^n \mid M \text{ couplage de } G\}.$$

Malheureusement, la relaxation linéaire $\mathcal{P}^*$ de $\mathcal{P}$ n'est pas une description du polyèdre $P_M(G)$. En effet, $\mathcal{P}^*$ contient des points extrêmes fractionnaires dans le cas général. Par exemple, consédirons l'exemple suivant.



Soient $e_1, ..., e_5$ les 5 arêtes de ce graphe. Les valeurs fractionnaires représentent le vecteur $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ où toutes les composantes correspondent à une arête. Il est facile de voir que ce vecteur vérifie toutes les contraintes de $\mathcal{P}^*$. De plus, il vérifie à l'égalité les 5 contraintes de type (14.1) associée à chacun des 5 sommets. Comme de plus, ces contraintes sont linéairement indépendantes, ce point est bien un point extrême du domaine de définition de $\mathcal{P}^*$. Comme ce point est fractionnaire, il ne peut appartenir à $P_M(G)$. (Remarquons que si le poids était $c(e_i) = 1$, $i = 1, ..., 5$, ce point serait même la solution maximale de $\mathcal{P}^*$.).
On veut donc "couper" ce point extrême fractionnaire dans une description de $P_M(G)$. Pour cela, on peut remarquer qu'un couplage dans ce cycle contient au plus deux arêtes. Par conséquent, l'inégalité $\sum_{i=1}^{5} \leq 2$ est valide pour le polyèdre $P_M(G)$. De plus, son ajout dans la description coupe ce point fractionnaire indésirable.

Tentons de généraliser cette contrainte (appelée dans ce cas une coupe). En fait si on considère un sous-ensemble de sommets $S \subseteq V$ avec $|S|$ impair. Alors il est claire qu'un couplage de G ne peut pas contenir plus de $\frac{S-1}{2}$ arêtes de $E(S)$. Par conséquent, considérons l'inégalité

$$\sum_{e \in E(S)} x(e) \leq \frac{|S| - 1}{2}, \text{pour tout } S \subseteq V \text{ avec } |S| \text{ impair} \tag{14.4}$$

qui est valide pour $P_M(G)$. On peut remarquer que cette contrainte généralise bien la contrainte basée sur le cycle du petit exemple.

14.2.2 Etude faciale de $P_M(G)$

Mais comment savoir si ces contraintes citées sont suffisantes pour décrire tout $P_M(G)$. Et comment savoir si ces contraintes sont elles-mêmes nécessaires. Pour cela,

nous allons approfondir l'étude faciale de $P_M(G)$.

Tout d'abord, étudions la dimension du polyèdre $P_M(G)$.

Proposition 14.2.1 $P_M(G)$ est de pleine dimension, i.e. $\dim(P_M(G)) = m$.

Preuve : On veut déterminer $m+1$ vecteurs affinement indépendants dans $P_M(G)$. Constatons tout d'abord que l'ensemble vide $\emptyset$ est un couplage de G. Donc le vecteur d'incidence de $\emptyset$ $\chi^{\emptyset} = (0, ..., 0) \in P_M(G)$. Il suffit alors de trouver m vecteurs de $P_M(G)$ linéairement indépendants.
Si l'on note $M_e = \{e\}$, $e \in E$, les singletons arêtes de G. Alors M_e, $e \in E$, est un couplage de G. De plus, la matrice formé par les vecteurs-lignes M_e, $e \in E$, est la matrice identité qui est bien inversible. Donc les vecteurs M_e, $e \in E$, sont bien linéairement indépendants. Par conséquent, $P_m(G)$ est de pleine dimension. □

Nous connaissons trois familles, (14.2), (14.1) et (14.4), de contraintes valides pour $P_M(G)$, mais définissent-elles des facettes de $P_M(G)$.

Proposition 14.2.2 Les contraintes triviales $x(e) \geq 0$, $e \in E$, définissent des facettes de $P_M(G)$.

Preuve : Tout d'abord, il faut prouver que ces contraintes sont bien valides. Ce qui est évident dans ce cas présent.
Soit $e \in E$. Il s'agit de montrer que la face $F = \{x \in P_M(G) \mid x(e) = 0\}$ est bien une facette.
Nous montrons en premier que F est propre. En effet, $F \neq \emptyset$ car $\chi^{\emptyset} \in P_M(G)$ par exemple. De plus, $F \neq P_M(G)$ car $\chi^{M_e}(e) = 1$ et donc $\chi^{M_e} \in P_M(G) \setminus F$.
Enfin, nous exhibons m vecteurs affinement indépendants dans F. Ce sont par exemple, les vecteurs $\chi^{\emptyset}$ et $\chi^{M'_e}$, $e \in E \setminus \{e\}$. Ainsi, la dimension de F est $dim(F) = m - 1$ et F est bien une facette. □

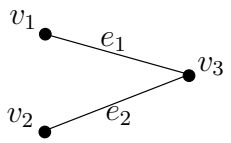
Proposition 14.2.3 Les contraintes triviales $x(e) \leq 1$, $e \in E$ ne définissent pas des facettes.

Preuve : En effet, les contraintes $x(e) \leq 1$, $e \in E$ sont redondantes par rapport aux contraintes de type (14.1). Inversement, les contraintes de type (14.1) ne peuvent pas être toutes obtenues comme somme des contraintes triviales. On dit que les contraintes de type (14.1) dominent les contraintes triviales. Par conséquent, les contraintes triviales $x(e) \leq 1$, $e \in E$ ne définissent pas de facettes. □

Certaines études faciales de contraintes sont par contre moins catégoriques. Par exemple, pour les contraintes de type (14.1) qui semblent nécessaires à la formulation, on peut se rendre compte qu'il existe des cas où ces contraintes sont redondantes et ne définissent donc pas de facettes.

Proposition 14.2.4 Les contraintes de type (14.1) ne définissent pas en général de facettes.

Preuve : Il suffit de produire un contre-exemple. Considérons donc un graphe limité à deux arêtes adjacentes par une extrémité comme indiqué sur la figure suivante : Si



l'on prend les contraintes de type (14.1) associée aux sommets v_1, v_2 et v_3. On obtient le système

$$\begin{array}{rcrl} x(e_1) & + & x(e_2) & \leq 1 \\ x(e_1) & & & \leq 1 \\ & & x(e_2) & \leq 1 \end{array}$$

On voit donc que la première de ces trois contraintes dominent les deux autres. Ces deux dernières ne définissent donc pas des facettes. □

14.2.3 Caractérisation complète de $P_M(G)$

En fait, ces trois types de contraintes sont suffisantes. Ce résultat donné par Edmonds en 1965 et redémontré de la façon suivante par Lovász en 1979.

Théorème 14.2.5 *Pour tout graphe $G = (V, E)$, le polyèdre des couplages $P_M(G)$ est donné par les inégalités (14.2), (14.1) et(14.4).*

En d'autres termes, le problème du couplage maximum dans un graphe G est équivalent à un programme linéaire !

Pour prouver ce résultat, nous utilisons le lemme suivant, qui est l'adaptation au cas du couplage du lemme 14.1.2. Pour $ax \leq \alpha$ une contrainte définissant une facette de $P_M(G)$, on pose C_a l'ensemble des couplages dont le vecteur d'incidence associé vérifie l'inégalité $ax \leq \alpha$ à l'égalité, i.e.

$$C_a = \{C \in E \mid C \text{ couplage de } G \text{ et } ax^C = \alpha\}.$$

Lemme 14.2.6 *Soit $ax \leq \alpha$ une contrainte définissant une facette de $P_M(G)$. Soit $a'x \leq \alpha'$ une contrainte valide de $P_M(G)$.*
i) Si pour tout couplage $C \in C_a$, $a'x^C = \alpha'$, alors $ax \leq \alpha$ et $a'x \leq \alpha'$ sont identiques à un coefficient multiplicateur positif prêt.
ii) Si $ax \leq \alpha$ et $a'x \leq \alpha'$ ne sont pas identiques à un coefficient multiplicateur positif prêt, alors il existe un couplage $C \in C_a$ tel que $a'x^C = \alpha'$.

Avant de nous lancer dans la preuve du théorème, donnons quatre lemmes techniques, spécifiques au problème du couplage qui nous seront nécessaires pour la preuve.

Lemme 14.2.7 *Supposons $ax \leq \alpha$ une contrainte définissant une facette de $P_M(G)$ qui soit différente des contraintes triviales (14.2).*
Alors $a(e) \geq 0$ pour tout $e \in E$.

Preuve : Pour cela, nous allons montrer que s'il existe une arête $e \in E$ telle que $a(e) < 0$, alors $ax \leq \alpha$ est de type (14.2), ce qui constituera évidemment une contradiction.
Supposons donc qu'il existe $e_0 \in E$ tel que $a(e_0) < 0$. Soit C un couplage de C_a. Supposons que $e_0 \in C$ et posons alors $C' = C \setminus \{e_0\}$ qui est encore un couplage. Comme $ax^C = \alpha$, on a $ax^{C'} = ax^C - a(e_0) = \alpha$ et donc comme $a(e_0) < 0$, on a $ax^{C'} > \alpha$ ce qui contredit le fait que C' soit encore un couplage. Donc $e_0 \notin C$.
On vient donc de prouver que tout couplage de C_a ne contient pas e_0, c'est-dire que $x(e_0) \geq 0$ est vérifiée à l'égalité par tous les couplages de C_a, c'est-dire que la face définie par $ax \leq \alpha$ est contenue dans la facette définie par $x(e_0) \geq 0$. Par le lemme 14.1.1, $ax \leq \alpha$ est donc de type (14.2), une contradiction. □

Lemme 14.2.8 *Supposons $ax \leq \alpha$ une contrainte définissant une facette de $P_M(G)$ qui soit différente des contraintes de couplage (14.1).*
Alors, pour tout sommet v, il existe un couplage C de C_a tel que C ne contienne pas d'arête incidente à v.

Preuve : Pour cela, nous allons montrer que, dans ce cas, $ax \leq \alpha$ est alors de type (14.1).
Supposons qu'il existe un sommet $v_0 \in V$ tel que tout couplage de $c \in C_a$, C contient une arête incidente à v_0. C'est-à-dire, $x^C(\delta(v)) = 1$. Ce qui revient à dire par le lemme 14.1.1 que l'inégalité $ax \leq \alpha$ est de type (14.1). □

On a besoin d'étudier plus précisemment le *graphe support* G_a associé à l'inégalité $ax \leq \alpha$. On le définit comme le graphe composé des arêtes $e \in E$ telles que $a(0) > 0$.

Lemme 14.2.9 *G_a est connexe.*

Preuve : Supposons que G_a soit l'union de deux graphes G_1 et G_2 disjoints. On pose alors a_1 (respectivement a_2) le vecteur obtenu depuis a en mettant à zéro les coefficients associés aux arêtes de G_1 (respectivement G_2). On note alors $\alpha_i = max\{a_i\chi^C | C$ couplage de $G\}$, pour $i = 1, 2$.
Alors $\alpha = \alpha_1 + \alpha_2$ et $a = a_1 + a_2$ donc $ax \leq \alpha$ s'obtient en sommant deux inégalités valides pour $P(G)$. Ce qui est impossible car elle définit une facette de $P(G)$. □

Lemme 14.2.10 *Soit $e = uv$ une arête de G telle que u et v soit dans G_a et telle qu'il existe un couplage $C \in C_a$ non incident à u et v. Alors l'arête e n'est pas dans G_a.*

Preuve : Comme C n'est ni incident à u, ni à v, alors $C' = C \cup \{e\}$ est encore un couplage de G, donc, par validité, $a\chi^{C'} \leq \alpha$. Or, comme $C \in C_a$, $a\chi^C = \alpha$ et $a\chi^{C'} = a\chi^C - a(e)$, on a $a(e) \leq 0$. Or, par définition du graphe support, $a(e) > 0$, une contradiction. □

Et, enfin, la preuve du théorème qui est assez technique.

Preuve du théorème 14.2.5 :
Tout d'abord, il faut prouver que toutes les contraintes sont bien valides, ce qui est bien le cas.
Nous allons en fait montrer que toute facette de P est définie par une de ces contraintes. Soit $ax \leq \alpha$ une contrainte définissant une facette de $P_M(G)$. On suppose que $ax \leq \alpha$ définit une facette n'est pas de type (14.2) et (14.1), on va alors prouver que dans ce cas $ax \leq \alpha$ est de type (14.4). Pour prouver cela, il suffit d'exhiber un sous-ensemble $S \subset V$, on a $x^C(\delta(S)) = \frac{|S|-1}{2}$ pour tout couplage de $C \in C_a$, ce qui prouvera alors que $ax \leq \alpha$ est de type (14.4).
En fait cet ensemble est S_0, l'ensemble des sommets de V incidents à une arête e de G_a (i.e. telle que $a(e) > 0$). Considérons alors la contrainte (14.4) associée à S_0, c'est-à-dire

$$x(E(S_0)) \leq \frac{|S_0| - 1}{2} \tag{14.5}$$

Supposons donc que notre contrainte $ax \leq \alpha$ soit aussi différente de (14.5), donc il existe un couplage $C_1 \in C_a$ qui ne vérifie pas à l'égalité (14.5), i.e. tel que $\chi^{C_1}x(E(S_0)) < \frac{|S_0|-1}{2}$. Par conséquent, comme $\frac{|S_0|-1}{2}$ désigne le nombre de paire disctincte dans S_0, S_0 contient au moins une paire u et v de sommets qui ne sont incidentes à aucune arête de C_1. De plus, par le lemme 14.2.9, G_a est connexe. On va supposer alors qu'on a choisi C_1 de manière à ce que la distance entre u et v dans G_a selon le poids a est la plus

courte possible.
Par le lemme 14.2.10, on sait donc que u et v ne sont pas adjacents dans G_a. Alors il existe un sommet z différent de u et v dans G_a sur la plus courte chaîne reliant u et v. Comme $ax \leq \alpha$ est différente des contraintes (14.1), par le lemme 14.2.8, il existe un couplage C_2 de C_a tel que C_2 ne contienne pas d'arête incidente à z.
On peut alors montrer que C_2 est incident u et v. En effet, supposons que, par exemple, C_2 ne couvre pas u, dans ce cas, u et z sont deux sommets de G_a non couvert par C_2. De plus, la chaîne qui les relie dans G_a est de taille strictement inférieure à celle reliant u et v (car $a(e) > 0$ pour tout e dans G_a), d'où contradiction. On obtient la même contradiction pour v.
Considérons alors l'ensemble d'arête $M = C_1 \cup C_2$. On peut déduire des deux paragraphes précédents que les sommets u, v et z sont de degré 1 dans le graphe G^* formé par les arêtes de M.
Sans perte de généralité, on peut supposer que la composante connexe de G^* contenant u consiste en une chaîne Q ne passant pas par z. Considérons alors les deux couplages

$$\bar{C_1} = (C_1 \setminus Q) \cup (C_2 \cap Q)$$

et

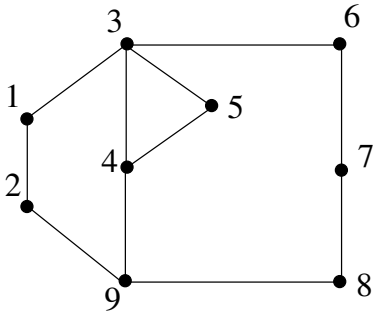
$$\bar{C_2} = (C_2 \setminus Q) \cup (C_1 \cap Q),$$

obtenus à partir de C_1 et de C_2 en permutant le statut "dedans" ou "dehors" des arêtes de Q pour chacun des couplages. Puisque $\bar{C_1} \cup \bar{C_2} = C_1 \cup C_2$, alors $\bar{C_1}$ et $\bar{C_2}$ sont dans C_a. Comme, $\bar{C_2}$ ne couvre ni u, ni z, cela contredit le choix de M_1 selon une chaîne minimale. □

14.3 Exercices

14.3.1 Caractérisation partielle du polytope du stable

Considérons le graphe $H_2 = (V, E)$ suivant :



Le but de cet exercice est de montrer que le polytope du stable pour ce graphe est donné par les contraintes (**??**), (13.4) et (13.6).
Soit $ax \leq \alpha$ une contrainte, différente des contraintes (**??**) et(13.4), qui définit une facette de $P(H_2)$.
1) Montrer que $a_v \geq 0$ pour tout v de V.
2) Montrer que $a_1 = a_2$ et $a_6 = a_7 = a_8$.
3) Montrer que si les coefficients du cycle (3,4,5) (resp. (1,2,9,8,7,6,3)) sont tous positifs alors la contrainte $ax \leq \alpha$ correspond à la contrainte (13.6) associée à ce cycle.
4) Soit H_a le sous-graphe induit par les sommets dont les coefficients sont positifs. Montrer que si H_a possède une arête telle qu'un de ses sommets soit de degré 1, alors $ax \leq \alpha$ correspond à la contrainte (13.4) associée à cette arête.
5) En déduire que le polytope du stable pour H_2 est donné par les contraintes (**??**), (13.4) et (13.6). Donner ce polytope.

Réponse :
1) Soit $v \in V$. Comme que $ax \leq \alpha$ est différente de la contrainte triviale associée à v, alors il existe un stable S tel que $a\chi^S = \alpha$ et $\chi^S(v) \neq 0$, c'est-à-dire S contient v. Posons alors $S' = S \setminus \{v\}$ qui est un stable et donc $a\chi^{S'} \leq \alpha$. Comme $a\chi^{S'} = a\chi^S - a_v \leq \alpha$ et $a\chi^S = \alpha$, alors $a_v \geq 0$.

2) Montrons que $a_1 = a_2$ ($a_6 = a_7$ et $a_7 = a_8$ peut être montré de manière identique). On peut supposer qu'un parmi a_1 et a_2 est strictement positif, par exemple $a_1 > 0$ (sinon $a_1 = a_2 = 0$). Comme $ax \leq \alpha$ est différente de la contrainte d'arête associée à l'arête $(1, 3)$, alors il existe un stanle S tel que $ax^S = \alpha$ et ne vérifiant pas la contrainte à l'égalité, c'est-à-dire que S ne contient pas 1 sommet parmi 1 et 3, donc S ne contient ni 1, ni 3.
On peut alors prouver que $2 \in S$. En effet, dans le cas contraire, $S \cup \{1\}$ est un stable et donc $ax^{S\cup\{1\}} = ax^S + a_1 \leq \alpha$ et, comme $ax^S = \alpha$, $a_1 = 0$, une contradiction.
On peut alors poser $S'' = (S \setminus \{2\}) \cup \{1\}$ et alors S'' est un stable. Comme $ax^{S''} = ax^S - a_2 + a_1 \leq \alpha$, alors $a_1 \leq a_2$.
Comme $0 < a_1 \leq a_2$, on peut reproduire symétriquement ce raisonnement pour prouver que $a_2 \leq a_1$. Et par conséquent, $a_1 = a_2$.

3) Supposons en premier que $a_3 > 0, a_4 > 0$ et $a_5 > 0$. Comme $ax \leq \alpha$ est différente de la contrainte de cycle associée au cycle passant par 3,4 et 5, alors il existe un stable S ne contenant ni 3, ni 4, ni 5. Mais dans ce cas, $S \cup \{5\}$ est encore un stable et $ax^S + a_5 \leq \alpha$, d'où $a_5 = 0$, une contradiction.
Supposons à présent que les coefficients dans a des sommets $W = \{1, 2, 3, 6, 7, 8, 9\}$ sont tous strictement positifs. Comme $ax \leq \alpha$ est différente de la contrainte associée

au cycle correspondant, alors il existe un stable S tel que $ax^S = \alpha$ et tel que $x^S(W) = |S \cap W| < 3$. Il y a donc deux cas :
- Si $|S \cap W| = 0$ ou $|S \cap W| = 1$. Dans ce cas, il existe toujours un sommet u de W en dehors de S qui n'est adjacent à aucun sommet de S. Dans ce cas, $S \cap \{u\}$ est encore un stable et on obtient comme dans le cas précédent $a_u = 0$, ce qui est une contradiction.
- Si $|S \cap W| = 2$, alors, d'après la configuration de H_2, il y a deux cas
  - Soit $S \cap W \neq \{1, 7\}$ et $S \cap W \neq \{2, 7\}$. Dans ce cas, il existe toujours un troisième sommet de W hors de S et non adjacent à aucun sommet de S, on obtient la même contradiction qu'auparavant.
  - Soit $S \cap W = \{1, 7\}$ (ou $S \cap W = \{2, 7\}$, cas similaire). Si $4 \notin S$, alors $S \cup \{9\}$ est un stable et on arrive à la contradiction habituelle. Mais si $4 \in S$, on ne peut pas ajouter le sommet 9 pour obtenir la contradiction. En revanche, on voit que S est précisemment le stable $S = \{1, 7, 4\}$, donc $a_1 + a_7 + a_4 = \alpha$. Or, on a prouvé que $a_7 = a_8$, donc $a_1 + a_8 + a_4 = \alpha$. Or $\{1, 4, 8\}$ est un stable et $S' = \{1, 4, 8, 6\}$ est aussi un stable. Ainsi $ax^{S'} = a_1 + a_8 + a_4 + a_6 \leq \alpha$, et donc on obtient $a_6 = 0$, une contradiction.

    Dans tous les cas, nous sommes arrivés à une contradiction, si ces coefficients sont tous positifs, alors $ax \leq \alpha$ est de type (13.6).

4) Supposons que H_a possède une arête pendante uv, c'est-à-dire telle qu'un de ses sommets, par exemple u est de degré 1. Soit S un stable de H_2 tel que $ax^S = \alpha$ et tel que S ne contienne ni u, ni v. Alors $S' = S \cup \{u\}$ est encore un stable et alors $ax^{S'} = ax^S + a_1 \leq \alpha$, donc $a_u = 0$, ce qui contredit l'hypothèse. Donc tous les stables S tels que $ax^S = \alpha$ intersectent l'arête uv et donc vérifient la contrainte d'arête (13.4) associée à cette arête.

5) On suppose que $ax \leq \alpha$ définit une contrainte d'un type différent de (**??**), (13.4) et (13.6). On va prouver qu'à chaque cas, on arrive à une contradiction.
- Si $a_3 = 0$, alors le graphe H_a est un sous-graphe d'un graphe composé de trois chemins issus de 9. Donc si l'arête $(2, 1)$ est pendante dans H_a, $ax \leq \alpha$ est de type (13.4), donc $a_1 = 0$ et conséquemment par 2), $a_2 = 0$. De même, on prouve que $a_6 = a_7 = a_8 = 0$ et $a_5 = a_4 = 0$. Si $a_9 > 0$, $ax^S \leq \alpha$ serait soit équivalente à $x(9) \leq 1$ et si $a_9 = 0$, la face associée à $ax \leq \alpha$ serait $P(H_2)$, un contradiction avec le fait qu'elle soit propre.
- Si $a_9 = 0$, on est dans un cas similaire, donc impossible.
- Considérons donc que $a_3 > 0$ et $a_0 > 0$.

  - Si $a_4 = 0$, alors $a_5 = 0$ (sinon $(3, 5)$ arête pendante). Dans ce cas, H_a est le cycle impair passant par $1, 2, 3, 6, 7, 8, 9$. Si un des coefficients des sommets du cycle est nul, alors on aurait une arête pendante dans H_a, une contradiction. Donc

tous les coefficients du cycle sont strictement positifs. Mais, par 3), $ax \leq \alpha$ serait alors de type (13.6).

- Si $a_4 > 0$, alors $a_5 = 0$, sinon on aurait $a_3, a_4, a_5 > 0$ et $ax \leq \alpha$ serait de type (13.6).
  - Si $a_1 = a_2 > 0$, alors on aurait $a_3, a_4, a_9, a_1, a_2 > 0$ et on peut prouver comme au 3) que $ax \leq \alpha$ est du type (13.6) associé au cycle impair passant par $3, 4, 9, 1$ et 2. Donc $a_1 = a_2 = 0$.
  - Si $a_6 = a_7 = a_8 = 0$, alors $(3, 9)$ est pendante, contradiction. Donc $a_6 = a_7 = a_8 > 0$. Donc H_a est le cycle pair passant par les sommets 3,4,9,8,7 et 6. Or tout stable S vérifiant $ax^S = \alpha$ contient nécessairement 3 sommets parmi ces sommets (sinon on obtient une contradiction avec la positivité des coefficients). Donc un tel stable va vérifier par exemple l'inégalité $x_6 + x_7 \leq 1$ à l'égalité, ce qui est la dernière contradiction.

Annexe A

Annexe : Méthodes heuristiques pour l'optimisation combinatoire

A.1 Introduction

A.1.1 Motivation et cadre d'utilisation

Un problème d'Optimisation Combinatoire peut être issu de l'industrie, de la gestion de la cîté, de la biologie, des nouvelles technologies en général ou être purement théorique. Suivant les cas, celui qui étudie ce problème doit se demander de quel temps d'étude il dispose, de l'intérêt induit par cette étude mais aussi du cadre (machine, temps de calcul,...) dont il dipose pour résoudre des instances de ce problème.

En effet, dans de nombreux cas, la détermination d'une solution exacte prend trop de temps de calcul pour qu'il soit raisonnable de chercher une telle solution. De même, la difficulté ou l'intérêt scientifique réduit du problème (de part sa complexité pratique, sa structure difficile,...) ou bien sa réalité pratique (trop de données, taille énorme des instances,...) peut demander de trouver rapidement un algorihme efficace pour déterminer simplement une "bonne" solution et non la meilleure. La notion de "bonne" solution n'est pas théorique et repose plus sur le fait qu'on sait produire une solution non triviale et de valeur supérieure à celle qu'un cerveau humain pourrait construire.

On se contente alors souvent d'une solution approchée, en essayant de faire en sorte qu'elle soit la meilleure possible dans un temps acceptable. Malheureusement ce compromis est souvent impossible à évaluer numériquement a priori. Si on sait évaluer la qualité de la solution déterminée de manière théorique, on parle *d'algorithme d'approximation*. Si cette évaluation ne peut se faire qu'expérimentalement ou alors par un évaluateur humain, on parle alors *d'heuristiques combinatoires*.

Ces algorithmes heuristiques fournissent donc rapidement des solutions réalisables pour un problème pratique. On calibre alors l'heuristique en fonction du temps de

calcul possible, de la puissance des machines etc. Par exemple, si la solution doit être fournie à une machine dite "en temps réel", il n'y a que quelques centième de secondes possibles. L'industrie ou la gestion de la cîté (circulation urbaine,...) offre plutôt des horizons de quelques minutes, voir de quelques heures. D'autre part, les heuristiques sont souvent les seules façons de traiter de très grandes instances venant de l'industrie (circuits électroniques, gestion de productions,...) ou des nouvelles technologies (génomiques,...).

A.1.2 Méthodologie

L'étude commence bien évidemment par étudier si le problème en question n'a pas une réponse polynomiale simple, efficace et exacte ! ! On découvre parfois des logiciels commerciaux donnant des solutions heuristiques à des problèmes de plus courts chemins ou de flots maximum ! !

Une bonne heuristique est bien souvent spécifique à un problème : elle est par exemple la traduction informatique du savoir d'un spécialiste du domaine (par exemple des spécialistes du planning d'un hôpital,...). Apprendre à construire de telles heuristiques se fait par l'expérience de nombreux cas concrets.

En comparant ces heuristiques particulières entre elles, les spécialistes ont néanmoins déduits de grands principes que l'on appelle *méthodologie* de la construction heuristique. Il s'agit principalement de concevoir des heuristiques en validant étape par étape leur proximité avec la réalité du problème qu'elles résolvent. Principalement, il est intéressant de valider la justesse de son heuristique avec des solutions exactes (quand on les connaît) ou avec des solutions fournies par l'expérience. Néanmoins, il est théoriquement impossible d'évaluer réellement une solution heuristique.

A.1.3 Méta-heuristiques

Si certaines heuristiques sont spécifiques à un problème, d'autres ont pour vocation de pouvoir être adaptées à divers problèmes. On appelle parfois ces dernières des *méta-heuristiques* ou encore heuristiques générales. En fait, il s'agit de principes algorithmiques permettant d'obtenir une solution en respectant certains principes de construction.

Les principales sont les méhodes gloutonnes ; les méthodes de recherche locale (ou d'améliorations itératives) telle que la méthode tabou (ou ses améliorations comme Grasp) ou les méthodes de descentes (recuit simulé) ; les méthodes évolutives (algorithmes génétiques) ; et la simulation (objet ou continue).

A.2 Algorithmes gloutons

Les algorithmes gloutons (greedy algorilhms en anglais) sont des algorithmes pour lesquels, à chaque itération, on fixe la valeur d'une (ou plusieurs) des variables décrivant le problème sans remettre en cause les choix antérieurs.
Le principe est donc de partir d'une solution incomplète (éventuellement totalement indéterminée) que l'on complète de proche en proche en effectuant des choix définitifs : à chaque étape, on traite (on « mange ») une partie des variables sur lesquelles on ne revient plus.

Par exemple, l'algorithme de Kruskal pour la recherche d'un arbre couvrant de poids minimum est glouton : on traite successivement les arêtes sans revenir sur les décisions prises à propos des arêtes précédentes. Cet algorithme conduit néanmoins à une solution exacte ! Mais ce n'est que rarement le cas. Pour le problème du voyageur de commerce par exemple, on peut proposer l'algorithme glouton suivant (dite heuristique de Christofides) : on part d'un sommet u pris au hasard et on le relie à son plus proche voisin qu'on appelle u_2 ; à une itération courante, si on appelle u_i le sommet atteint à l'itération précédente, on repart de u_i et on le relie à son plus proche voisin parmi les sommets non encore rencontrés ; lorsqu'on atteint tous les sommets, on ferme le cycle hamiltonien à l'aide de l'arête $\{u_n, u_1\}$ si on appelle u_n le dernier sommet rencontré. On peut remarquer que la dernière arête utilisée peut être la plus longue du graphe... rendant ainsi la solution très mauvaise.

Si les algorithmes de ce type sont souvent rapides, en revanche la solution qu'ils déterminent peut être arbitrairement loin de la solution. On les utilise néanmoins fréquemment pour obtenir rapidement une solution réalisable. Par exemple, elle servent à initialiser une méthode itérative. Mais dans certains types d'instances réelles, la solution gloutonne est parfois très bonne.

A.3 Méthodes de recherche locale

Le principe des *méthodes de recherche locale* (ou méthodes d'amélioration itérative) est inspirée des méthodes d'optimisation continue. Ces méthodes consistent à déterminer itérativement la solution d'une fonction continue en utilisant des outils comme les dérivées partielles ou les gradients, suivant que la fonction soit ou non dérivable.

La description de ces méthodes à partir d'une solution de départ X_0, à engendrer une suite (finie) de solutions (X_n) déterminées de proche en proche, c'est-à-dire itérativement (X_{i+1} étant déterminée à partir de X_i). Le choix de la solution X_{i+1} se fait dans un ensemble "localement proche" de la solution X_i et de manière à avoir une solution X_{i+1} "plus intéressante" au sens de la recherche locale. Les deux expressions

mises entre guillemets dans la phrase précédente se formalisent à partir de la notion de *voisinage* et de *recherche dans un voisinage*.

La construction essentielle de ces méthodes reposent en effet sur la détermination d'un bon voisinage. Ensuite, suivant la façon de choisir une solution dan le voisinage, on obtient différentes méthodes de recherche locale : méthode tabou, descente "pure", descente stochastique, recuit simulé,...

A.3.1 Voisinage d'une solution

On définit généralement le voisinage d'une solution à l'aide d'une transformation élémentaire (ou locale). On appelle *transformation* toute opération permettant de changer une solution X de S en une autre solution X' de S. Une transformation sera considérée comme élémentaire (ou locale) si elle ne modifie que "faiblement" la structure de la solution à laquelle on l'applique.

Autrement dit, les transformations locales constituent un sous-ensemble de l'ensemble des transformations. Elles sont locales en ce sens qu'elles ne perturbent pas globalement la structure de la solution qu'elles changent, mais ne la modifient que localement. Par exemple, si X est un entier codé sous la forme d'une chaîne de 0-1, une transformation locale peut consister à changer un élement en 0-1 de la chaîne en son complémentaire.

Le choix de la transformation élémentaire dépend a priori du problème à traiter. On peut soumettre le choix de la transformation élémentaire à deux critères qu'elle devra respecter. Dans la mesure où cette transformation est appliquée de nombreuses fois, on doit d'abord pouvoir en évaluer rapidement les conséquences, ce qui entraîne qu'elle doit être relativement simple (c'est pour cette raison qu'on considère plus volontiers des transformations locales que globales).

D'autre part, elle doit permettre d'engendrer tout l'ensemble S, ou du moins tout l'ensemble dans lequel on cherche la solution optimale. C'est la propriété *d'accessibilité*. On veut aussi pouvoir revenir sur ses pas à la solution d'origine : propriété de *réversibilité*. Par exemple, la transformation consistant à changer, dans une chaîne de n 0-1, un 0-1 en son complémentaire permet d'engendrer n'importe quelle configuration en au plus n changements bien choisis à partir de n'importe quelle solution initiale.

La transformation élémentaire étant choisie, on peut définir le voisinage d'une solution. Etant donnée une transformation locale, le *voisinage* $V(X)$ d'une solution X est l'ensemble des solutions que l'on peut obtenir en appliquant à X cette transformation locale.

La notion de voisinage dépend donc de la transformation locale considérée. Ainsi,

pour la transformation élémentaire évoquée plus haut dans l'exemple, le voisinage d'une chaîne X de n 0-1 est l'ensemble des n chaînes de n 0-1 possédant exactement $n-1$ 0-1 en commun avec X. Une autre transformation pourrait induire un voisinage de cardinal différent. Par exemple, celle définie par le changement simultané de deux 0-1 conduirait à un voisinage possédant $\frac{n(n-1)}{2}$ éléments...

En pratique, on recherche en général des voisinages de tailles réduites dans laquelle l'exploration peut être réalisée en temps polynomial ($O(n), O(n^2)$ au pire $O(n^3)$). Mais le tout est d'avoir un voisinage bien pensée et efficace.

Un autre problème sous-jacent au voisinage est la façon de coder une solution. En effet, il peut y avoir différentes façon de coder la valeur d'une solution ce qui induit plusieurs techniques de recherche. Par exemple, un tour du TSP peut être codé soit par les arêtes utilisées, soit par la liste des villes dans l'ordre de la visite...

Quelques idées de voisinages possibles

Pour définir le voisinage, on trouve fréquemment une ou plusieurs des opérations suivantes mises en oeuvre, selon la nature du problème et du codage des solutions :
- **complémentation** (remplacement) : pour les solutions codées sous forme d'une chaîne en 0-1. On a évoqué plus haut cette transformation : on remplace un 0-1 quelconque de la chaîne par son complémentaire. Par exemple,

1 0 0 1 1 1 0 0 1 devient 1 0 0 1 1 0 0 0 1

par complémentation du sixième élément. On peut généraliser cette transformation, lorsque la solution est codée sous la forme d'une chaîne de caractères : en remplaçant un (éventuellement plusieurs) caractère(s) quelconque(s) de la chaîne par un autre caractère (par autant d'autres caractères).
- **échange** : lorsque la solution est codée sous la forme d'une chaîne de caractères, l'échange consiste à intervertir les caractères situés en deux positions données de la chaîne. Ainsi

A B C D E F G devient A E C D B F G

par échange des positions 2 et 5.
- **insertion-décalage** : supposons de nouveau que la solution est codée sous la forme d'une chaîne de caractères. L'insertion-décalage consiste alors à choisir deux positions i et j, à insérer en position i le caractère situé en position j puis à décaler tous les caractères anciennement situés entre i (inclus) et j (exclus) d'un cran à droite si $i < j$, d'un cran à gauche sinon. Par exemple

A B C D E F G devient A B F C D E G

par insertion-décalage avec $i = 3$ et $j = 6$.
- **inversion** : supposons encore que la solution est codée sous la forme d'une chaîne de caractères. L'inversion consiste à choisir deux positions i et j avec $i < j$, puis à inverser l'ordre d'écriture des caractères situés aux positions i, $i + 1$,...,$j - 1$, j. Par exemple

A B C D E F G devient A E D C B F G

par inversion avec $i = 2$ et $j = 5$.

Ces transformations locales dépendent d'un (pour la première) ou de deux (pour les autres) paramètres. On peut concevoir des transformations locales dépendant d'un plus grand nombre de paramètres. Pour de nombreux problèmes, les conséquences de ces changements sont faciles à évaluer et la répétition de ces transformations (en les choisissant convenablement) un nombre de fois suffisant permet bien d'engendrer n'importe quelle solution. Ces transformations (ou les codages adoptés pour représenter les solutions) ont parfois un inconvénient, provenant généralement du fait que les configurations ainsi engendrées ne sont pas toujours toutes pertinentes (un exemple étant celui du remplacement d'un caractère par un autre, dans le cas où la chaîne représentant une solution réalisable ne peut admettre plusieurs fois un même caractère).

En pratique, un voisinage doit s'inspirer de la structure même du problème. Il est par exemple important de choisir le bon codage de la solution pour X. Par exemple, pour le problème du voyageur de commerce, il est possible de coder la solution par l'ordre des villes visité ou par un vecteur en 0-1 sur les arêtes du graphe...

Exemple de voisinage : le problème du voyageur de commerce

Pour le problème du voyageur de commerce, il est habituel de considérer comme transformation élémentaire celle qui consiste à choisir deux arêtes non adjacentes dans le cycle hamiltonien (de la solution courante) et de les remplacer par les deux arêtes qui permettent de reconstituer un cycle hamiltonien.

Cette transformation est appelée 2-opt. Elle définit, pour chaque solution, un voisinage de $\frac{n(n-3)}{2}$ éléments. Elle peut en fait être considérée comme un cas particulier de l'inversion évoquée plus haut. On peut la généraliser en envisageant une transformation qu'on pourrait appeler k-opt et qui consisterait à remplacer simultanément k arêtes par k autres convenablement choisies.

A.3.2 Initialisation et réitération des méthodes de recherche locale

Pour appliquer une méthode de recherche locale, il faut une solution de départ. Celle-ci peut être calculée tout à fait aléatoirement, ou bien provenir d'une autre méthode approchée, par exemple d'un algorithme glouton.

Lorsque la solution initiale est (au moins partiellement) aléatoire, on peut alors appliquer plusieurs fois la descente en changeant à chaque fois la configuration de départ, et ne conserver finalement que la meilleure solution rencontrée depuis la première descente. Cette répétition permet alors d'atténuer l'inconvénient majeur de la recherche locale, c'est-à-dire d'être locale.

Il est également possible de concevoir une méthode gloutonne qui compléterait une un morceau réalisable d'une solution. Par exemple, dans le cas d'un vecteur à n 0-1, on ne conserverait que quelques valeurs des 0-1 et on s'autoriserait à remplacer les autres de manière gloutonnes. Ainsi, on peut utiliser le schéma suivant appelé GRASP (Greedy Random Adaptative Procedure), initié par Feo et Resende en 1989.

GRASP est composée de 2 étapes : une étape de construction gloutonne, suivie par une étape de recherche locale. On réitère ensuite ces 2 étapes en utilisant à pour le départ de l'une la solution courante de la fin de l'autre. Cette idée permet de cumuler les avantages de plusieures méthodes.

A.3.3 La méthode Tabou

Principe de la méthode Tabou

Même si les premières idées concernant la méthode Tabou datent peut-être de 1977, on peut plus sûrement la faire remonter aux alentours de 1986. Elle a été proposée indépendamment par F. Glover d'une part, par P. Hansen puis P. Hansen et B. Jaumard d'autre part (sous un nom différent).

L'idée de départ est simple. Elle consiste à se déplacer de solution en solution en s'interdisant de revenir en une configuration déjà rencontrée. Plus précisément, supposons qu'on a défini un voisinage $V(X)$ pour chaque solution X. Supposons en outre qu'on dispose à toute itération de la liste T de toutes les configurations rencontrées depuis le début de l'exécution de la méthode. Alors, à partir de la configuration courante X, on choisit dans $V(X) \setminus T$ la solution X' qui minimise la fonction H, puis on ajoutant X' à la liste T. Autrement dit, on choisit parmi les configurations voisines de X mais non encore rencontrées celle qui "descend" le plus fortement si X n'est pas un minimum local (par rapport à la transformation élémentaire qui définit le voisinage), ou celle qui "remonte" le moins sinon, et on ajoute X' à T pour s'interdire d'y revenir.

La liste Tabou

En fait, il est rarement possible de pouvoir mettre en oeuvre ce principe : conserver toutes les configurations rencontrées consomme en général trop de place mémoire et trop de temps de calcul pour savoir quelle configuration choisir dans le voisinage de la solution courante, puisqu'il faut comparer chaque voisin à chaque élément de la liste T.

D'un autre côté, supposons qu'on passe de X à Y avec $Y \in V(X)$ et $H(Y) > H(X)$, il alors existe en Y au moins une direction de descente dans son voisinage, celle qui redescend en X ! Et on risque ainsi de boucler si on ne s'interdit pas, au moins provisoirement, un retour en X.

Pour éviter ces inconvénients sans être obligée de conserver en mémoire toutes les configurations rencontrées depuis le début, la méthode Tabou préconise la tactique suivante. Au lieu d'ajouter la solution courante X à la liste T des configurations interdites (taboues), on se contente, quand on remonte, de conserver en mémoire la transformation élémentaire qui a permis de passer de la configuration courante à la suivante et on s'interdit d'appliquer son inverse : ce sont donc désormais des mouvements qui sont tabous et non plus des configurations.

Cette modification présente à son tour certains inconvénients. En interdisant des mouvements, on s'interdit aussi d'aller vers certaines solutions qui pourraient être intéressante. Pour ne pas trop appauvrir le voisinge de la onfiguration courante, on limite la taille de la liste T que l'on gère comme une file (premier entré, premier sorti). En pratique, on choisit souvent une taille assez petite, qu'il faut décider en la paramétrant. Suivant les cas, on préconise autour de la 10aine, de la centaine, rarement plus. Une taille plus petite risque de ne pas pouvoir empêcher le bouclage, et une taille plus grande semble en général trop appauvrir les voisinages.

Il est nécessaire de prévoir un critère d'arrêt : par exemple un nombre d'itérations que l'on s'autorise à effectuer, où la stagnation de la meilleure valeur trouvée depuis un certain nombre d'itérations...

Améliorations

Il existe peu de paramètres à fixer sur une méthode tabou (à part la taille de la liste). Mais comme l'on explore systématiquement tout un voisinage, ce voisinage joue un rôle majeur.

De nombreuses variantes peuvent être ajoutées à cette méthode : manipuler plusieurs listes Tabou, interdire toute transformation inverse, associer à chaque transformation élémentaire un compteur indiquant pendant combien d'itérations elle est taboue,...

On peut également s'autoriser à passer outre le caractère tabou d'un mouvement dans certains cas. On appelle ce procédé *l'aspiration* : on lui donne un paramètre m qui indique au bout de combien d'itération on s'autorise à passer à une solution taboue mais qui est intéressante par rapport à la valeur courante. Il existe d'autres techniques intéressantes pour améliorer la puissance de la méthode tabou, en particulier, l'intensification et la diversification. Toutes les deux se basent sur l'utilisation d'une mémoire à long terme et se différencient selon la façon d'exploiter les informations de cette mémoire. *L'intensification* se fonde sur l'idée d'apprentissage de propriétés favorables : les propriétés communes souvent rencontrées dans les meilleurs configurations visitées sont mémorisées au cours de la recherche, puis favorisées pendant la période d'intensification. Une autre manière d'appliquer l'intensification consiste à mémoriser une liste de solutions de bonne qualité et à retourner vers une des ces solutions. La *diversification* a un objectif inverse de l'intensification : elle cherche à diriger la recherche vers des zones inexplorées. Sa mise en oeuvre consiste souvent à modifier temporairement la fonction de coût pour favoriser des mouvements n'ayant pas été effectués ou à pénaliser les mouvements ayant été souvent répétés. L'intensification et la diversification jouent donc un rôle complémentaire.

A.3.4 Les méthodes de descente

L'idée générale d'une méthode de descente est de toujours prendre dans un voisinage une solution meilleure que la solution courante. En ce qui concerne l'exploration du voisinage de la solution courante, plusieurs, attitudes peuvent être adoptées : exploration aléatoire, exploration systématique pour déterminer un voisin meilleur, ou exploration exhaustive pour déterminer le meilleur voisin.

Descente "simple", descentes itérées et descente stochastique

• Descente simple :

La méthode de descente simple est inspiré de la minimisation de fonctions continues, la descente simple consiste tout simplement à choisir systématiquement un sommet du voisinage qui améliore le plus la solution courante. L'algorithme s'arrête donc quand il n'est plus possible d'améliorer la solution.

Cette descente simple n'est intéressante que dans le cas où le voisinage est suffisemment petit, car il faut systématiquement explorer le voisinage. En pratique, elle amène à un optimum local souvent intéressant.

• Descentes itérées :

Une méthode dite "des descentes itérées" consistent à relancer plusieurs fois une descente simple à partir de solutions générées aléatoirement. Cette méthode facile à mettre

en œuvre est souvent très efficace.

• Descente stochastique :
L'exploration stochastique (aléatoire) consiste à choisir aléatoirement une solution voisine, puis de tester si elle améliore la solution courante :

Initialiser une valeur réalisable pour X
Tant que nécessaire faire
    Choisir aléatoirement et uniformément Y dans $V(X)$
    si $H(Y) < H(X)$ alors $X \leftarrow Y$

Cette descente aléatoire évite de visiter systématiquement un voisinage qui serait trop grand.

Le critère d'arrêt peut être un nombre d'itérations sans amélioration jugé suffisant, ou l'obtention d'une solution acceptable. Le défaut de ces méthodes de descentes pures est de rester dans un minimum local sans trouver le minimum global. Elles sont néanmoins très utilisées pour être une brique de GRASP (voir A.3.2).

Algoritme du recuit simulé

Le recuit simulé (ou méthode de Monte-Carlo) provient de la physique moléculaire où des molécules se positionnent de façon à minimiser leur énergie quand la température baisse (c'est le principe de frappe des métaux en ferronerie : chauffer, puis refroidir avant de modeler le métal). L'algorithme de principe est le suivant où RND est une fonction qui donne un échantillon uniforme et indépendant dans $[0, 1]$:

T est une "température"
X est l'état initial,
$RX \leftarrow X$ (état record)
Tant que nécessaire faire
    Choisir aléatoirement et uniformément $Y \in V(X)$
    si $H(Y) < H(RX)$ alors $RX \leftarrow Y$
    si $H(Y) < H(X)$ alors $X \leftarrow Y$
    sinon
        si $RND \leq exp(-\frac{H(Y)-H(X)}{T})$ alors $X \leftarrow Y$
    Générer une nouvelle température T

A chaque itération, on prend un voisin au hasard. Si ce voisin est meilleur, il devient le nouvel état courant. Sinon, il devient le nouvel état courant avec une certaine pro-

babilité qui dépend à la fois de la différence des coûts et de la température courante. Pour cette loi de probabilité, on utilise souvent cette fonctoon appelée “dynamique de Metropolis”, mais d'autres fonctions sont possibles.

La température de départ doit être suffisamment élevée pour permettre d'accepter régulièrement des mauvaises transitions au début de l'algorithme. Cette température décroît progressivement vers 0 durant le déroulement de l'algorithme, diminuant ainsi la probabilité d'accepter des transitions défavorables.

Généralement, l'algorithme doit s'arrêter quand l'état est "gelé". C'est-à-dire que l'état courant n'est plus modifié (la température est trop basse pour accepter de mauvaises transitions).
En pratique, on peut décider d'une baisse de la température par pallier réguliers où la température s'abaisse dès qu'un objectif est atteint (nombre d'itérations, valeurs, stagnation,...).

Si la température baisse suffisamment lentement, et si le système de voisinage vérifie certaines propriétés, le recuit converge en probabilité vers l'optimum. En effet, Hajek (1988) a énoncé le théorème suivant :
Théorème de convergence (Hajek 1988) : Si le système de voisinage vérifie les propriétés d'accessibilité et de réversibilité, et si à partir d'un certain rang n, $T \geq \frac{D}{ln(n+1)}$ où D représente la profondeur maximale d'un minimum local, alors le recuit simulé converge en probabilité vers l'ensemble des solutions optimales.
Malheureusement, il est difficile (voire impossible) en pratique de vérifier ces hypothèses.

A.4 Les méthodes évolutives

Le terme "algorithmes évolutifs" englobe une autre classe assez large de métaheuristiques. Ces algorithmes sont basés sur le principe du processus d'évolution naturelle. Les algorithmes évolutifs doivent leur nom à l'analogie entre leur déroulement et le mécanisme de sélection naturelle et de croisement des individus d'une population vivante sexuée.

Un algorithme évolutif typique est composé de trois éléments essentiels :
1) une population constituée de plusieurs individus représentant des solutions potentielles (configurations) du problème donné.
2) un mécanisme d'évaluation de l'adaptation de chaque individu de la population à l'égard de son environnement extérieur
3) un mécanisme d'évolution composé d'opérateurs permettant d'éliminer certains individus et de produire de nouveaux individus à partir des individus sélectionnés.

Du point de vue opérationnel, un algorithme évolutif débute avec une population initiale souvent générée aléatoirement et répète ensuite un cycle d'évolution suivant les principes en 3 étapes séquentielles :
- Evaluation : mesurer l'adaptation (la qualité) de chaque individu de la population
- Sélection : sélectionner une partie des individus
- Reproduction : produire de nouveaux individus par des recombinaisons d'individus sélectionnés

Ce processus se termine quand la condition d'arrêt est vérifiée, par exemple, quand un nombre de cycles (générations) ou quand un nombre d'évaluations est atteint ou quand des solutions suffisamment bonnes sont trouvées. Si l'on s'imagine que le processus suit le principe d'évolution naturelle, la qualité des individus de la population doit s'améliorer au fur et à mesure du processus.

Parmi les composantes d'un algorithme évolutif, la population et la fonction d'adaptation correspondent respectivement à la notion de configuration et à la fonction d'évaluation dans la recherche locale. La notion de mécanisme d'évolution est proche de celle du mécanisme de parcours du voisinage de la recherche locale mais les opérateurs sont sensiblement différents. En effet, un algorithme évolutif comporte un ensemble d'opérateurs tels que la sélection, la mutation et éventuellement le croisement.
La *sélection* a pour objectif de choisir les individus qui vont pouvoir survivre ou/et se reproduire pour transmettre leurs caractéristiques à la génération suivante. La sélection se base généralement sur le principe de conservation des individus les mieux adaptés et d'élimination des moins adaptés.
Le *croisement* ou recombinaison cherche à combiner les caractéristiques des individus parents pour créer des individus enfants avec de nouvelles potentialités dans la génération future.
La *mutation* effectue de légères modifications de certains individus.

Suivant l'imagination des ses utilisateurs, on appelle ces méthodes “algorithmes génétiques”, “programmation évolutive”, “stratégies d'évolution”, “essaim de particules”, “colonie de fourmis”,... Nous prendrons ici comme exemple, largement suffisant, des algorithmes dits “génétiques”. Notons qu'il s'agit ici d'une présentation combinatoire de ces algorithmes qui ont aussi une version en optimisation continue.

A.4.1 Cadre des algorithmes génétiques

Les algorithmes génétiques peuvent se définir à partir d'un codage sous forme de chaînes 0/1 de longueur fixe et un ensemble d'opérateurs "génétiques" : la mutation,

l'inversion et le croisement Un individu sous ce codage est un chromosome, un gène la composante de base du chromosome et un allèle la valeur effective d'un gène (0 ou 1 ici). En d'autres termes, un chromosome, un gène et un allèle représentent respectivement une solution (configuration), un attribut de la solution et la valeur de l'attribut.

Les opérateurs génétiques sont définis de manière à opérer stochastiquement sur le codage sans aucune connaissance sur le problème. Par exemple, le croisement "bi-points" consiste à choisir aléatoirement deux points de croisement et à échanger les segments des deux parents déterminés par ces deux points. La mutation consiste simplement à changer aléatoirement la valeur de certains gènes. Le rôle de la mutation dans les algorithmes génétiques est essentiellement de réintroduire de nouvelles valeurs pour des gènes alors que le croisement réalise uniquement des recombinaisons de valeurs existantes.

Un cycle d'évolution complet d'un algorithme génétique est formé par l'application des opérateurs de sélection, croisement et mutation sur une population de chromosomes.

L'universalité d'un tel algorithme pose des problèmes d'efficacité en pratique. En effet, en tant que méthode d'optimisation, un algorithme génétique basé sur des opérateurs génétiques "aveugles" est rarement en mesure de produire des résultats comparables à ceux de la recherche locale. Une technique pour remédier à ce problème consiste à spécialiser l'algorithme génétique au problème donnée. Plus précisément, à la place des opérateurs aléatoires, la mutation et le croisement sont adaptés en se basant sur des connaissances spécifiques du problème. De cette manière, la recherche est mieux guidée et donc plus efficace.

A.4.2 Mise en oeuvre

Le choix du codage est important et souvent délicat. L'objectif est bien sûr d'abord de pouvoir coder n'importe quelle solution. Mais il est souhaitable, au-delà de cette exigence, d'imaginer soit un codage tel que toute chaîne de caractères représente bien une solution réalisable du problème, soit un codage qui facilite ensuite la conception du croisement de telle sorte que les « enfants » obtenus à partir de la recombinaison de leurs « parents » puissent être associés à des solutions réalisables, au moins pour un grand nombre d'entre eux.

L'idée majeure à conserver dans un algorithme génétique, en opposition aux méthodes de descentes, est de travailler plus sur la structure des solutions que sur leurs valeurs (on parle parfois de “schéma”). C'est-à-dire qu'il faudrait pouvoir conserver une bonne structure au travers des individus : typiquement un morceau de solutions donnant une bonne valeur. D'autre part, le codage devrait être tel qu'une petite variation dans le chromosome n'entraîne pas une trop grande variation dans la configuration

associée à ce chromosome. Ainsi, une représentation en binaire des entiers ne s'accorde pas bien à ce principe.

En général, il n'est pas facile de construire un codage essayant de tenir compte de tous ces critères, si bien qu'on est parfois amené à ne pas prendre en. considération certaines de ces indications.

La sélection

La sélection des individus sur lesquels on va appliquer le croisement fait intervenir la fonction à minimiser : la probabilité de choisir l'individu X sera d'autant plus grande que $H(X)$ sera faible. Une façon habituelle de faire intervenir sont de répartir les individus/solutions $X_1, X_2, ..., X_p$ selon des catégories et des tirages au sort.

La probabilité de sélectionner X, est proportionnelle à $1-\frac{H(X_i)}{F}$, avec $F = \sum_{i=1}^{p} H(X_i)$ et vaut donc $\frac{F-H(X_i)}{F(p-1)}$. Ce qui donne à chaque individu sa chance d'être sélectionné. Attention il faut des valeurs de H positives. Il faut souvent adapter cette idée en fonction des fonctions H.

Le but de la sélection est de savoir quels individus sont conservés et lesquels serviront à la reproduction.

Le croisement

L'objectif du croisement est de recombiner d'une certaine façon les chromosomes de deux (rarement plus) parents procréateurs afin de former les chromosomes d'un ou de deux (rarement plus) enfants. Le croisement s'inspire du mécanisme observé dans le crossing-over de la génétique (et est d'ailleurs aussi appelé ainsi) : on extrait une partie du code associé à chacun des parents, et on réorganise ces parties entre elles de façon à former de nouveaux individus qui jouent le rôle des enfants.

Un croisement que l'on rencontre souvent est le « croisement à un point ». Afin de décrire celui-ci, imaginons que l'on décide de croiser deux individus A et B (les parents) représentés chacun par un chromosome, respectivement C_A et C_B. Le croisement à un point consiste à déterminer aléatoirement une position (un gène) après laquelle on coupe C_A et C_B : on obtient donc quatre morceaux de chromosomes C_A^1 et C_A^2 issus de C_A, et C_B^1 et C_B^2 issus de C_B. On peut alors facilement former deux nouveaux individus D et E (les enfants), obtenus par un échange des morceaux de chromosomes : D aura pour chromosome la concaténation de C_A^1 et de C_B^2, et E aura pour chromosome celle de C_B^1 et de C_A^2.

Ce type de croisement fait que les couples dé gènes n'ont pas tous la même probabilité de rester ensemble (ainsi les extrémités d'un chromosome seront-elles systématiquement séparées), ce qui n'est pas toujours souhaitable. On peut alors, modifier ce croisement en un « croisement à deux points » (et plus généralement à k points)...

La mutation

Le dernier opérateur, la mutation, est conçu pour apporter une certaine diversité dans la population et empêcher que celle-ci ne converge trop vite vers un même individu, ou vers un petit groupe d'individus, ce qui rendrait inopérant le croisement (si du moins celui-ci conserve les caractéristiques des procréateurs). Il agit aléatoirement sur le codage d'un individu, en remplaçant un ou plusieurs des symboles du codage par autant d'autres symboles de l'alphabet. Par exemple dans le cas d'un entier codé en binaire, la mutation peut être de changer un 1 en 0 ou un 0 en 1. Pour des problèmes de position, la mutation consistera à modifier localement la solution associée à l'individu à muter par une opération appropriée.

Par exemple, pour le voyageur de commerce, on pourra considérer comme mutation une transformation du type 2-opt (c'est-à-dire un échange en croix). Plus généralement, on pourra s'inspirer des transformations élémentaires que sont les voisinages d'une solution réalisable. Attention, pour ne pas trop perturber la composition de la population et ne pas transformer les algorithmes génétiques en une errance aléatoire, la mutation doit avoir une faible probabilité p,d'être appliquée. On pourra, selon les cas, envisager au plus une mutation par chromosome ou au contraire envisager éventuellement une mutation par gène. La valeur de probabilité devra être choisie en conséquence (plus faible dans le second cas que dans le premier).

Autres opérateurs

D'autres opérateurs sont possibles. Par exemple, celui qui consiste à améliorer séparément les individus de la population, souvent à l'aide d'une méthode d'amélioration itérative (mais on pourrait évidemment envisager de lui substituer une méthode plus sophistiquée, comme le recuit simulé ou la méthode Tabou). Dans leur conception originelle, les algorithmes génétiques n'intègrent pas cette idée qui modifie sensiblement la philosophie de la méthode : le croisement peut alors paraître réduit à un moyen d'échapper à un minimum local que risquerait d'atteindre la méthode d'amélioration itérative sans pouvoir en sortir. On peut éventuellement interpréter cette phase d'amélioration individuelle comme une sorte d'apprentissage permettant à l'individu de mieux s'adapter à l'environnement (que traduit la fonction à optimiser).

A.5 La simulation

On appelle *moteur de simulation* un algorithme reproduisant un système complexe. Par exemple, les simulateurs permettent de représenter un réseau de files d'attente entre machines d'usine afin d'optimiser l'ordre dans lequel les pièces circulent entre les machines. Un autre exemple est de représenter le plus fidèlement possible la circulation

des voitures (par l'utilisation de la programmation object) afin d'étudier virtuellement les embouteillages.

Un système sur lequel on essaye de résoudre un problème d'optimisation combinatoire demande alors de faire fonctionner le moteur de simulation afin d'observer pour quelle valeur (ordonnancement, solution,...) on obtient les meilleures évaluation.

Cette méthode est en quelque sorte celle qu'on utilise quand on ne sait rien faire face à la difficulté d'une structure combinatoire.

Bibliographie

[1] Alain Billionnet, "Optimisation Discrète", *Dunod Applications & Métiers* (2007).

[2] W. Cook, W. Cunningham, W. Pulleyblank et A. Schrijver ; *Combinatorial Optimization*, Wiley-Interscience (1997).

[3] J. Edmonds, "Maximum matching and a polyhedron with 0-1 vertices", *Journal of Research of the National Bureau of Standards 69 B* (1965) 125-130.

[4] M. Grötschel, L. Lovász et A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization", *Combinatorica 1*, (1981) 70-89.

[5] A.R. Mahjoub, "Approches Polyédrales en Optimisation Combinatoire", *Optimisation combinatoire . 1 , concepts fondamentaux* (2005) Hermes science publ. Lavoisier.

[6] H.Marchand, A.Martin, R. Weismantel et L. Wolsey "Cutting Planes in Integer and Mixed Integer Programming", *Cours post-doctoral donné au laboratoire CORE pour le réseau Adonet* (2003).

[7] M. Minoux, "Programmation mathématique : théorie et algorithmes", Ed. Lavoisier, 2008 (1ère édition 1983).

[8] M. Sakarovitch, "Optimisation combinatoire", Ed. Hermann, 1997 (non réédité).

[9] L. Wolsey, *Integer Programming*, Wiley-Interscience, (1998).