CHAPITRE 1 : INTRODUCTION

Le terme «recherche opérationnelle», d'origine militaire, n'est pas très suggestif, c'est le moins qu'on puisse dire. La recherche opérationnelle constitue autant une façon d'aborder un problème à l'aide d'outils mathématiques et informatiques qu'une discipline bien définie telle que l'anthropologie ou la linguistique. Ainsi on peut traiter par une approche de RO des problèmes de gestion, de politique électorale, de biologie moléculaire, d'ingénierie, de logistique, d'informatique (vision par ordinateur, infographie, télématique), etc. La RO est une discipline polyvalente, qui emprunte beaucoup aux mathématiques, évidemment, mais également à l'informatique et à l'économie. Elle le leur rend bien.

Dans le cadre de ce cours, nous allons étudier un certains nombre de modèles mathématiques pouvant s'appliquer à des situations concrètes. L'emphase sera mise sur la modélisation et les algorithmes de résolution, parfois au détriment, bien malheureusement, de la rigueur mathématique.

Considérons l'exemple d'un réseau routier urbain fort congestionné. Un candidat aux élections municipales propose, s'il est élu, d'améliorer l'écoulement du trafic par la construction d'une voie rapide. Au cours de l'élection, le débat porte sur la pertinence du projet. Il est admis par tous qu'une voie rapide ne peut que réduire le temps de transport des 6000 automobilistes qui se rendent de leur origine A à leur destination D (voir Figure 1). Par contre, le prix à payer pour la construction de la voie rapide est-il trop élevé pour la diminution de temps de transport appréhendée ?

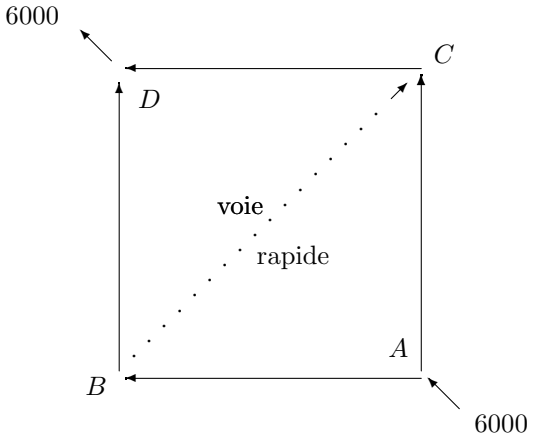


FIGURE 1 – Le réseau du paradoxe de Braess

Afin de répondre de façon éclairée à cette question, il faut pouvoir évaluer l'impact du projet sur les chemins utilisés par les utilisateurs. Et pour connaître ces chemins, il faut d'abord savoir quels sont les critères de choix de ceux-ci. On suppose que les usagers ne tiennent compte que du temps de parcours et choisirons des chemins de temps de parcours minimum. Si t_p représente le temps de parcours du chemin p, $t_{\min}$ le temps de parcours minimum et x_p le nombre d'utilisateurs du chemin p, ce **principe de comportement** peut se traduire mathématiquement de la façon suivante :

$$t_p > t_{\min} \quad \Longrightarrow \quad x_p = 0$$

ou encore

$$
\begin{aligned}
& t_p \geq t_{\min} \qquad p = 1, 2, 3 \\
& x_p(t_p - t_{\min}) = 0 \qquad p = 1, 2, 3 \\
& \sum_{p=1}^{3} x_p = 6000 \\
& x_p \geq 0.
\end{aligned}
$$

La première équation stipule qu'on ne peut avoir un chemin dont le délai serait inférieur au délai à l'équilibre. La seconde nous assure qu'il ne peut y avoir de flot sur des chemins dont le délai est supérieur au délai à l'équilibre. Les deux suivantes, satisfaction de la demande et non négativité des flots, se justifient d'elles-mêmes.
Cette formulation mathématique est loin d'être triviale. Elle implique le bon choix de variables de décision (les flots de véhicules sur les chemins menant de A à D) ainsi que la traduction en équations d'une notion qui semble relever plus de la sociologie que de la mathématique. Ceci constitue la partie la plus ardue de l'exercice de modélisation. Une fois cette étape franchie, il ne reste plus qu'à raffiner un peu notre **modèle d'affectation du trafic**.

Spécifions d'abord la forme du temps de transport t_p. Supposons que ce temps soit égal à la somme des temps de transport sur les arcs formant le chemin p, ce qui paraît assez naturel. Ainsi :

$$t_{ACD} = t_{AC} + t_{CD}.$$

Puisque le réseau est congestionné, il est également naturel de supposer que les délais sur les arcs augmentent avec les flots sur ces arcs. On introduit, pour chaque arc du réseau, une **fonction de congestion** qui spécifie, pour chaque usager de l'arc considéré, le temps de parcours :

$$
\begin{aligned}
t_{AB}(x_{AB}) &= \frac{1}{100}\, x_{AB} \\
t_{AC}(x_{AC}) &= 50 + \frac{1}{1000} x_{AC} \\
t_{BC}(x_{BC}) &= 10 + \frac{1}{1000} x_{BC} \\
t_{BD}(x_{BD}) &= 50 + \frac{1}{1000} x_{BD} \\
t_{CD}(x_{CD}) &= \frac{1}{100}\, x_{CD}.
\end{aligned}
$$

Ces temps de parcours peuvent également s'exprimer en fonction des flots sur les chemins. Par exemple, puisque l'arc AB appartient à la fois aux chemins ABD et ABC, on obtient $x_{AB} = x_{ABD} + x_{ABCD}$ et $x_{BD} = x_{ABD}$, d'où l'on dérive le délai du chemin ABD :

$$t_{ABD}(x_{ABD}, x_{ACD}, x_{ABCD}) = \frac{1}{100}\,(x_{ABD} + x_{ABCD}) + (50 + \frac{1}{1000} x_{ABD}).$$

On dérive de la même façon les délais des chemins ACD et $ABCD$.

On peut maintenant décrire formellement notre modèle d'**affectation du trafic**

$$\begin{aligned}
t_p &\geq t_{\min} \quad \forall p \in \{ABD, ACD, ABCD\} \\
x_{ABD}\Big(t_{ABD}(x_{ABD}, x_{ACD}, x_{ABCD}) - t_{\min}\Big) &= 0 \\
x_{ACD}\Big(t_{ACD}(x_{ABD}, x_{ACD}, x_{ABCD}) - t_{\min}\Big) &= 0 \\
x_{ABCD}\Big(t_{ABCD}(x_{ABD}, x_{ACD}, x_{ABCD}) - t_{\min}\Big) &= 0 \\
x_{ABD} + x_{ACD} + x_{ABCD} &= 6000 \\
x_{ABD}, x_{ACD}, x_{ABCD} &\geq 0.
\end{aligned}$$

Si les trois chemins sont utilisés, il suffit de résoudre le système linéaire

$$\begin{aligned}
t_{ABD} &= t_{ACD} \\
t_{ABD} &= t_{ABCD} \\
x_{ABD} + x_{ACD} + x_{ABCD} &= 6000,
\end{aligned}$$

dont l'unique solution est

$$x_{ABD} = x_{ACD} = x_{ABCD} = 2000.$$

On vérifie aisément que le délai commun est alors de 92 (minutes). Bien. Mais qu'en était-il du délai *avant* la construction de la voie rapide BC ? Par symétrie, il est facile de voir qu'un **équilibre** sera atteint lorsque $x_{ABD} = x_{ACD} = 3000$ et que le délai correspondant est égal à 83 (minutes). La construction de la voie rapide aura donc eu comme conséquence, non seulement de faire élire notre conseiller municipal,[1] mais également d'augmenter le temps de trajet de tous les automobilistes ! De tels paradoxes ont effectivement été observés en pratique.

Cet exemple simpliste donne une petit idée de ce que la recherche opérationnelle peut accomplir. Le modèle précédent peut être raffiné pour tenir compte des caractéristiques suivantes d'un réseau urbain :

- la présence de plusieurs modes de transport (automobile, camion, autobus, bicyclette) ;
- les aspects aléatoires du trafic (incidents, accidents, conditions météorologiques) ;
- les variations horaires, journalières et saisonnières de la demande ;
- les modes de contrôle du trafic (feux de circulation, panneaux à messages variables, systèmes d'information) ;
- le caractère dynamique du trafic ;
- la présence de différentes classes d'usagers, chacune avec ses propres critères et comportements.

Plus un modèle est raffiné, plus il risque de se rapprocher d'une situation réelle. Par contre, plus il risque d'être difficile à résoudre. De plus, un problème commun aux modèles détaillés est celui de la collecte de données fiables. Un modèle trop complexe risque d'être lourd et inutilisable.

1. Dans une élection subséquente, celui-ci aura bien entendu rejeté le tort sur les ingénieurs responsables du projet.

Les modèles de RO s'expriment souvent (pas toujours, loin de là) comme des problèmes d'optimisation. Dans l'espace euclidien à n dimensions, on obtient typiquement la formulation [2]

$$\begin{array}{lll} \max\limits_{x} & f(x_1, \ldots, x_n) & \\ & g_1(x_1, \ldots, x_n) & \leq \quad 0 \\ & \vdots & \vdots \quad \vdots \\ & g_m(x_1, \ldots, x_n) & \leq \quad 0 \end{array}$$

où $x = (x_1, \ldots, x_n)$ est le vecteur des **variables**, f la **fonction objectif** et $g_1, \ldots, g_m$ les contraintes du problème. Puisqu'il est inutile de produire une solution qui ne respecte pas les contraintes et que les contraintes s'expriment en fonction des variables de décision, les étapes du processus de construction d'un modèle doivent être, *dans l'ordre* :

- le choix des variables de décision ;
- la représentation du domaine réalisable (faisabilité d'un vecteur x) à l'aide des fonctions de contrainte g_i ;
- le choix d'un objectif.

Une fois le modèle posé, eh bien, il faut le résoudre. Et ici, on quitte presque le domaine de la RO pour entrer dans celui des mathématiques (programmation mathématique, théorie des graphes, analyse numérique, processus stochastiques) et de l'informatique (structures de données, interfaces usager-machine, programmation). Il n'y a pas de limite aux techniques que peut piller la RO. Le but est de résoudre un problème. La technique de résolution est *secondaire*.

Dans les chapitres qui suivent, nous étudierons un certain nombre de modèles ainsi que de techniques de résolution de ces modèles. Ces modèles et techniques doivent être perçus comme une boîte à outils. Il ne faut pas hésiter à enrichir celle-ci au besoin.

Maintenant, place aux modèles !

2. Noter qu'une contrainte d'égalité $g(x) = 0$ peut être représentée par les deux contraintes d'inégalité $g(x) \leq 0$ et $-g(x) \leq 0$.

CHAPITRE 2 : PROGRAMMATION LINÉAIRE

À tout seigneur tout honneur ! La programmation linéaire constitue la pierre angulaire de toute la recherche opérationnelle. Il faut bien sûr éviter de forcer tout modèle à être linéaire. Par contre, un très grand nombre de modèles constituent des extensions de programmes linéaires. Sa compréhension est essentielle à la compréhension de modèles plus sophistiqués.

Un **programme linéaire** s'exprime de façon générique sous la forme

$$\begin{array}{lrcccl} \max\limits_x & c_1x_1 & + \cdots + & c_nx_n & & \\ & a_{11}x_1 & + \cdots + & a_{1n}x_n & \leq & b_1 \\ & \vdots & & \vdots & & \\ & a_{m1}x_1 & + \cdots + & a_{mn}x_n & \leq & b_m \end{array}$$

objectif

contraintes

ou, sous une forme plus compacte :

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, m \end{aligned}$$

et, pour adultes vaccinés avec droit de vote, la forme matricielle :

$$\begin{array}{lll} \max\limits_x & cx & \\ & Ax & \leq b \end{array}$$

c : vecteur ligne $1 \times n$
x : vecteur colonne $n \times 1$
A : matrice $m \times n$
b : vecteur colonne $m \times 1$

Exemple (voir Figure 1) **:**

$$\begin{array}{lrcccll} \max & 3x_1 & + & 5x_2 & & & \\ & x_1 & & & \leq & 4 & \\ & & & 2x_2 & \leq & 12 & \\ & 3x_1 & + & 2x_2 & \leq & 18 & \\ & x_1 & & & \geq & 0 & (-x_1 \leq -0) \\ & & & x_2 & \geq & 0 & (-x_2 \leq -0) \end{array}$$

Un point **réalisable** (ou **admissible**) est **extrémal** (on parle parfois d'un **sommet**) s'il ne peut se trouver à l'intérieur d'un segment (non nul) entièrement contenu dans la région admissible (**polyèdre**). Dans l'exemple de la figure 1 il y a cinq points extrémaux : $x^1 = (0, 0)$, $x^2 = (0, 6)$, $x^3 = (2, 6)$, $x^4 = (4, 3)$ et $x^5 = (4, 0)$.

Le but du jeu est d'augmenter la valeur de l'objectif. L'observation fondamentale est que l'objectif augmente tout en satisfaisant aux contraintes si et seulement si l'on se dirige dans une direction admissible [1] d qui fait un angle aigu avec le vecteur c, ce qui advient si et seulement si le produit scalaire des vecteurs c et d est positif. Dans la figure 2, la direction d^1 est une direction d'amélioration, d^2 une direction de détérioration et d^3 une direction de constance. Il est important de noter que

– la droite D est le lieu géométrique des points pour lesquels l'objectif vaut $c_1x_1 + c_2x_2$. Ceci correspond à une **courbe de niveau** (ou **droite de constance**) sur une carte topographique ;
– les courbes de niveau sont parallèles ;

1. Une direction est admissible s'il est possible de s'y engager sans quitter le domaine réalisable. Il se peut que l'extrémité du vecteur définissant cette direction se trouve à l'extérieur du domaine réalisable.

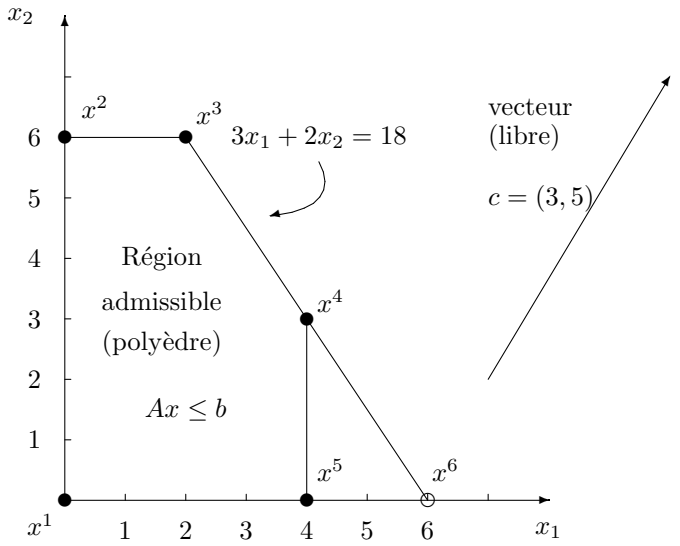


FIGURE 1 – La géométrie d'un programme linéaire

- le vecteur c est le **vecteur normal** (terme synonyme de **perpendiculaire** ou **orthogonal**) associé à la droite D. À chaque point de l'espace correspond une courbe de niveau, c'est-à-dire une droite de constance dans le cas qui nous intéresse. Ainsi la droite D_1 est une courbe de niveau correspondant à une valeur supérieure de l'objectif ;
- pour un déplacement de longueur unitaire, la plus grande amélioration est obtenue en se dirigeant dans la direction du vecteur c. Cette direction est appelée **direction de la plus grande pente** et est orthogonale aux droites de constance.

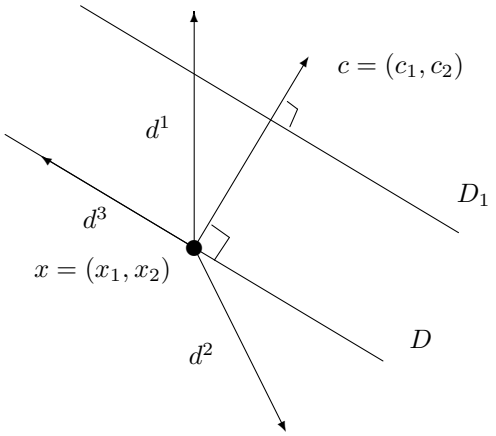


FIGURE 2 – Directions

En suivant la direction *admissible* de plus grande amélioration (cette direction est unique) et en s'interdisant de quitter la région admissible, on atteint une solution optimale en un nombre fini d'itérations (voir la trajectoire ABCD illustrée à la figure 3). Une solution est optimale si et seulement s'il n'y a pas de direction d'amélioration admissible en ce point.

Pour tout programme linéaire, l'une ou l'autre des quatre situations suivantes doit se présenter :

- il existe une solution optimale et celle-ci est unique ;
- l'objectif est borné et il existe une infinité de solutions optimales qui sont situées sur une **face** non triviale du polyèdre des contraintes. Dans l'exemple de la figure 1 il existe cinq faces triviales de dimension nulle (points extrémaux), cinq faces de dimension un (côtés du pentagone) et une face de dimension deux (le pentagone lui-même). Dans l'espace, on peut rencontrer des faces de dimensions zéro (points extrémaux), un (arêtes), deux (faces au sens géométrique traditionnel) et trois (le polyèdre lui-même) ;
- il n'existe aucune solution réalisable ;
- le problème est **non borné**, c'est-à-dire que l'objectif peut augmenter indéfiniment en suivant une demi-droite faisant un angle aigu avec le vecteur objectif et entièrement située dans le domaine admissible. Il est clair que cette situation ne peut se produire que si le domaine lui-même n'est pas borné.

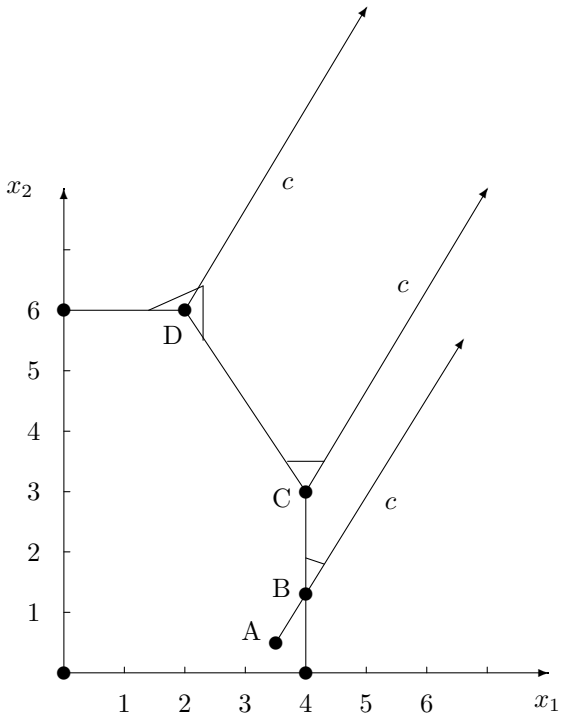


FIGURE 3 – Trajectoire menant à une solution optimale

Il n'est pas difficile de démontrer que s'il existe une solution optimale bornée, il existe alors une solution optimale extrémale. Pour résoudre le problème, il suffirait donc d'énumérer les points extrémaux (sommets) du domaine, qui sont en nombre fini. Las ! Ils sont en nombre exponentiel en fonction de la taille (nombre de contraintes) du problème.

Le premier algorithme proposé pour la programmation linéaire visite de façon «efficace» des points extrémaux adjacents, tout en améliorant l'objectif. Dans l'exemple précédent, deux suites partant du point $(0, 0)$ et satisfaisant ces deux critères sont possibles : $\{x^1, x^2, x^3\}$ et $\{x^1, x^5, x^4, x^3\}$.

Cet algorithme ressemble singulièrement à l'algorithme d'élimination de Gauss-Jordan pour la résolution de systèmes d'équations linéaires sous-déterminés, c'est-à-dire contenant plus de variables que de contraintes et admettant, en général, une infinité de solutions réalisables.

Appliquons l'algorithme du simplexe à notre exemple. L'algorithme utilise une **forme canonique** («standard») où les variables sont toutes positives ou nulles et les autres contraintes sont toutes des contraintes

d'égalité. En ajoutant des **variables d'écart** (x_3, x_4 et x_5 dans notre exemple) aux contraintes d'inégalité "$\leq$" et en remplaçant toute variable libre x_i par la différence $x_i^+ - x_i^-$ de deux variables non négatives, il est facile de se ramener à cette forme. [2]

$$
\begin{array}{lrcrcrcrcr}
\max & 3x_1 & + & 5x_2 & & & & & & \\
 & & & & & & & & & \\
 & x_1 & & & + & x_3 & & & = & 4 \\
 & & & 2x_2 & & & + & x_4 & = & 12 \\
 & 3x_1 & + & 2x_2 & & & & & = & 18 \\
 & & & & & & & & & \\
 & x_1, & & x_2, & & x_3, & & x_4, & \geq & 0
\end{array}
$$

Comme dans la phase finale de la méthode de Gauss-Jordan, on exprime des variables liées (**variables de base**) en fonction de variables libres (**variables hors-base**), ce qui nous donne le **dictionnaire** suivant :

$$
\begin{array}{lrcrcrcr}
\max & 3x_1 & + & 5x_2 & & & & \\
 & x_3 & = & 4 & - & x_1 & & \\
 & x_4 & = & 12 & & & - & 2x_2 \\
 & x_5 & = & 18 & - & 3x_1 & - & 2x_2 \\
 & x_1, & x_2, & x_3, & x_4, & x_5 & \geq & 0
\end{array}
$$

Une **solution de base** est obtenue en posant $x_j = 0$ pour toutes les variables hors-base, c'est-à-dire les variables qui se trouvent à droite des signes d'égalité. Tout dictionnaire correspond à un point extrémal. [3] Le dictionnaire précédent correspond au point extrémal $(x_1, x_2, x_3, x_4, x_5) = (0, 0, 4, 12, 18)$. En deux dimensions, ceci correspond au point $x^1 = (0, 0)$.

L'algorithme du simplexe substitue à une variable de base (à gauche du dictionnaire) une variable hors-base (à droite du dictionnaire). On choisit évidemment une variable qui fasse augmenter l'objectif. On a le choix entre les variables x_1 et x_2. Choisissons la variable x_1. Afin de maintenir la non négativité des variables de base, x_1 ne peut dépasser la valeur $x_1 = 4$. Une fois cette valeur atteinte, la variable de base x_3 devient nulle et doit quitter la base. En posant $x_1 = 4 - x_3$ (première équation du dictionnaire), on obtient le nouveau dictionnaire :

$$
\begin{array}{lrcrcrcr}
 & x_1 \uparrow 4 & & x_3 \downarrow 0 & & & & \\
 & & & & & & & \\
 & x_1 & = & 4 - x_3 & & & & \\
 & & & & & & & \\
\max & 3(4 - x_3) + 5x_2 & = & 12 + 5x_2 - 3x_3 & & & & \\
 & & & & & & & \\
 & x_1 & = & 4 - x_3 & & = & 4 - x_3 \\
 & x_4 & = & 12 - 2x_2 & & = & 12 - 2x_2 \\
 & x_5 & = & 18 - 3(4 - x_3) - 2x_2 & & = & 6 - 2x_2 + 3x_3
\end{array}
$$

qui correspond au point extrémal $x^5 = (4, 0, 0, 12, 6)$. À la deuxième itération, x_2 entre dans la **base** et x_5 la quitte.

2. N'ayons pas peur d'ajouter des variables. Je rappelle que nous traitons de *gros* problèmes. Enfin, il faut s'entendre : le terme ≪nous≫ désigne un ordinateur qui a été efficacement programmé.

3. Cependant, un point extrémal peut avoir plusieurs représentations comme dictionnaire. Noter qu'un dictionnaire comportant des variables de base *négatives* ne correspond pas à un point extrémal ; voir par exemple le point x^6 de la figure 1.

$$x_2 \uparrow 3 \quad x_5 \downarrow 0$$

$$\begin{array}{lcl} x_2 & = & 3 + \frac{3}{2}x_3 - \frac{1}{2}x_5 \\ \max & & 27 + \frac{9}{2}x_3 - \frac{5}{2}x_5 \end{array}$$

$$\begin{array}{lcl} x_2 & = & 3 + \frac{3}{2}x_3 - \frac{1}{2}x_5 \\ x_1 & = & 4 - x_3 \\ x_4 & = & 6 - 3x_3 + x_5 \end{array}$$

Ce dictionnaire correspond au point extrémal $x^4 = (4, 3, 0, 6, 0)$.

On peut également adopter un format de **tableau** qui évite d'écrire les noms des variables. Plusieurs formats de tableau sont possibles. Vous trouverez ci-dessous le dictionnaire précédent sous forme de tableau ainsi que le tableau optimal obtenu en effectuant une troisième itération où la variable x_3 réintègre la base. L'élément encerclé du tableau de gauche, situé à l'intersection de la quatrième ligne (variable quittant la base) et de la deuxième colonne (variable entrant dans la base) est appelé **pivot** et joue un rôle important dans la dérivation de règles arithmétiques permettant de passer d'un tableau à un autre.

		variables hors-base	
		x_3	x_5
objectif	27	$\frac{9}{2}$	$-\frac{5}{2}$
variables de base x_2	3	$\frac{3}{2}$	$-\frac{1}{2}$
variables de base x_1	4	-1	0
variables de base x_4	6	$\textcircled{-3}$	1

		x_4	x_5
objectif	36	$-\frac{3}{2}$	-1
x_2	6	$-\frac{1}{2}$	0
x_1	2	$\frac{1}{3}$	$-\frac{1}{3}$
x_3	2	$-\frac{1}{3}$	$\frac{1}{3}$

optimum : x^3

Les coefficients de la première ligne se trouvant sous les variables hors-base sont appelés **coûts réduits**. Lorsque ceux-ci sont tous négatifs ou nuls, la solution correspondante est bien sûr optimale. Attention : La contraposée de cette affirmation n'est pas nécessairement vraie ! En effet, une solution dont certains coûts réduits sont positifs peut être optimale.

DIVERSES CONSIDÉRATIONS

Comme vous avez pu le constater (enfin je l'espère) à la lecture de ces quelques pages, la programmation linéaire, ce n'est pas vraiment très compliqué. Il y a cependant un certain nombre de points à éclaircir.

1. **Initialisation**

   (a) Une contrainte d'inégalité $ax \geq b$ peut être mise sous la forme $-ax \leq -b$;

   (b) Une contrainte d'égalité $ax = b$ peut être remplacée par deux contraintes d'inégalité $ax \leq b$ et $ax \geq b$;

   (c) Une variable x_j qui n'est pas contrainte en signe peut être remplacée par la différence $x_j = x'_j - x''_j$ de deux variables non négatives.

Maintenant, que faire si la base correspondant aux variables d'écart n'est pas admissible ? Par exemple soit les contraintes

$$\begin{array}{rcrcl} x_1 & + & x_2 & \leq & 2 \\ x_1 & - & x_2 & \geq & 1 \end{array}$$

dont la forme canonique est, après ajout de deux variables d'écart :

$$\begin{array}{rcrcrcrcl} x_1 & + & x_2 & + & x_3 & & & = & 2 \\ -x_1 & + & x_2 & & & + & x_4 & = & -1. \end{array}$$

Malheureusement, la base constituée des variables d'écart n'est pas réalisable. En changeant le signe de la deuxième contrainte et en ajoutant une variable **artificielle** (fictive) x_5 on obtient le système linéaire

$$\begin{array}{rcrcrcrcrcl} x_1 & + & x_2 & + & x_3 & & & & & = & 2 \\ x_1 & - & x_2 & & & - & x_4 & + & x_5 & = & 1 \end{array}$$

qui possède la base admissible $x = (0, 0, 2, 0, 1)$. Une solution x pour le problème modifié sera admissible pour le problème originel si et seulement si la variable artificielle x_5 est nulle. Il est naturel d'obtenir une telle solution par la résolution du programme linéaire

$$\begin{array}{lr} \min & x_5 \\ & \\ \begin{array}{rcrcrcrcrcl} x_1 & + & x_2 & + & x_3 & & & & & = & 2 \\ x_1 & - & x_2 & & & - & x_4 & + & x_5 & = & 1 \\ x_1, & & x_2, & & x_3, & & x_4, & & x_5 & \geq & 0. \end{array} & \end{array}$$

Si la valeur optimale de ce programme est positive, alors le polyèdre originel est vide. Sinon, on élimine les colonnes correspondant aux variables artificielles [4], on exprime l'objectif originel en fonction des variables hors-base et l'on entame la ≪phase II≫ de la méthode.

Cette technique se généralise à des situations impliquant plusieurs variables artificielles. Il suffit alors de minimiser la somme des variables artificielles.

2. **Convergence**

Toute solution de base réalisable correspond à un point extrémal. Si toutes les solutions de base ont des valeurs (objectif) distinctes, alors l'algorithme du simplexe ne peut visiter une base plus d'une fois et doit donc converger en améliorant l'objectif à chaque itération. Notons que le nombre de solutions de base (réalisables ou non) est énorme : $\binom{m}{n} = \frac{m!}{n!(m-n)!}$. En pratique cependant le nombre de bases visitées semble linéaire par rapport au nombre de contraintes.

Un point extrémal peut admettre plusieurs représentation de base. Il se pourrait alors que que l'on passe d'une base à l'autre sans améliorer l'objectif. Ceci ne peut se produire que si au moins une des variables de base est nulle (une telle solution de base est dite **dégénérée**). Le danger est alors de visiter de façon cyclique un ensemble de bases dégénérées, ce qui est possible, mais peu probable en pratique. Noter qu'il existe des règles permettant théoriquement d'éviter le phénomène de cyclage.

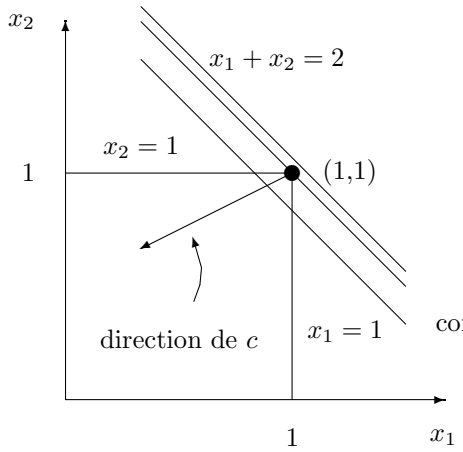
On trouve à la figure 4 un exemple de point extrémal dégénéré (le point $x = (1, 1)$) et le programme linéaire correspondant. Soient x_3, x_4 et x_5 les variables d'écart du problème. Le dictionnaire associé au point $(1, 1)$ est

$$\begin{array}{lrcrcrcrcl} \max & & & -3 & + & x_3 & + & x_5 & \\ & & & & & & & & \\ & x_1 & = & 1 & - & x_3 & & & \\ & x_2 & = & 1 & + & x_3 & - & x_5 & \\ & x_4 & = & 0 & - & x_3 & + & x_5 & . \end{array}$$

4. Des ajustements techniques sont nécessaires si une variable artificielle de valeur nulle se trouve dans la base.

Si on introduit la variable x_3 dans la base, celle-ci prend la valeur zéro. Il y a changement de base (mais pas de point extrémal). À l'itération suivante, il y aura (vérifiez-le) changement de point extrémal et amélioration de l'objectif.

Si on perturbe l'une des contraintes satisfaites à égalité par le point dégénéré, la dégénérescence disparaît. Par exemple, si la contrainte $x_1 + x_2 \leq 1$ est remplacée par la contrainte $x_1 + x_2 \leq 1 - \epsilon$, soit le point dégénéré $(1, 1)$ est remplacé par les deux points extrémaux $(1, 1 - \epsilon)$ et $(1 - \epsilon, 1)$ si ϵ est positif (voir figure 4), soit la contrainte ne joue plus aucun rôle dans le problème ainsi que dans l'algorithme. Cette technique de perturbation du vecteur b peut être utilisée en théorie ou en pratique pour éliminer la dégénérescence.



$$\begin{aligned} \max \quad & -2x_1 - x_2 \\ & x_1 \leq 1 \\ & x_2 \leq 1 \\ & x_1 + x_2 \leq 2 \\ & x_1,\, x_2 \geq 0 \end{aligned}$$

contrainte perturbée

FIGURE 4 – Un exemple de dégénérescence

3. **Efficacité**

   – L'algorithme du simplexe, bien que de complexité théorique exponentielle, est très efficace en pratique ;
   – Il existe des algorithmes polynomiaux pour la programmation linéaire, basés sur des principes entièrement différents de ceux de la méthode du simplexe. Entre autres, ces algorithmes «efficaces» ne produisent une solution extrémale qu'à la dernière itération.

4. **Implantation numérique**

   Les problèmes pratiques impliquent fréquemment des milliers, voire des millions de variables et de contraintes. Une implantation naïve peut se révéler désastreuse, suite à des problèmes numériques. Il est en effet important de savoir si une variable, un coût réduit, est positif, nul ou négatif. Les logiciels de programmation linéaire utilisent des techniques sophistiquées pour choisir les variables d'entrée et de sortie ainsi que pour effectuer les changements de base, une opération qui semble pourtant triviale sur de petits exemples.

5. **Solution non bornée**

   Si une variable hors-base de coût réduit positif n'est pas limitée dans son augmentation (ce qui arrive si tous les coefficients de la colonne du dictionnaire correspondant à cette variable sont positifs ou nuls) alors l'objectif peut augmenter indéfiniment le long d'une demi-droite dont l'extrémité est le point extrémal courant. L'algorithme du simplexe reconnaît cette situation et envoie un message approprié à l'utilisateur.

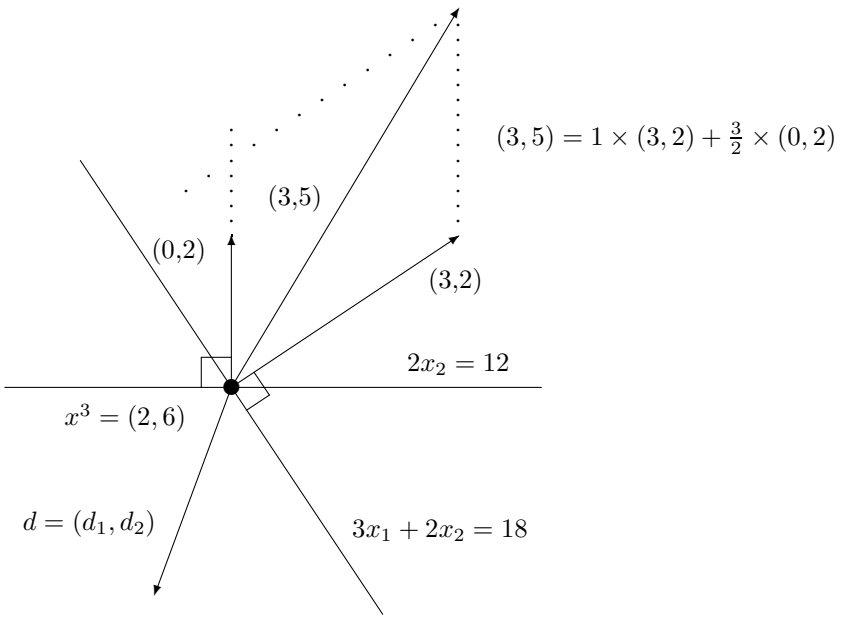


FIGURE 5 – La dualité

6. **Dualité**

À l'optimum, il n'existe pas de direction d'amélioration admissible, c'est-à-dire :

$$(3,5)(d_1,d_2) \leq 0 \qquad \forall \quad d = (d_1,d_2) \text{ admissible.}$$

Un théorème stipule (voir l'illustration en Figure 5) que cette condition est satisfaite si et seulement si le vecteur objectif $c = (3,5)$ peut s'exprimer comme une combinaison linéaire *non-négative* de vecteurs normaux (orthogonaux) aux contraintes **actives**. Dans notre cas, on a :

$$(3,5) = y_2(0,2) + y_3(3,2),$$

d'où l'on déduit : $y_2 = 3/2$ et $y_3 = 1$. En posant $y_1 = y_4 = y_5 = 0$, on obtient

$$(3,5) = y_1(1,0) + y_2(0,2) + y_3(3,2) + y_4(-1,0) + y_5(0,-1),$$

avec y_2, $y_3 \geq 0$ et $y_1 = y_4 = y_5 = 0$ ou, matriciellement :

$$(3,5) = \begin{pmatrix} y_1 & y_2 & y_3 & y_4 & y_5 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 2 \\ 3 & 2 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Dans le cas général, on obtient (A^i désigne la ligne i de la matrice A) :

$$c = yA \qquad y \geq 0 \qquad \text{et} \quad y_i = 0 \quad \text{si} \quad A^i x < b_i,$$

ce qu'on peut exprimer synthétiquement comme :

$$\begin{aligned} yA &= c \\ y &\geq 0 \\ y(b - Ax) &= 0. \end{aligned}$$

Le vecteur de **multiplicateurs** y est aussi appelé **vecteur dual**. Un vecteur dual est admissible s'il satisfait aux contraintes

$$yA = c \qquad y \geq 0.$$

Un **vecteur primal** x est alors optimal si et seulement s'il existe un vecteur dual y tel que

$$\boxed{\begin{array}{rcll} Ax & \leq & b & \text{primal réalisabilité} \\ yA & = & c & \text{dual} \\ y & \geq & 0 & \quad \text{réalisabilité} \\ y(b - Ax) & = & 0 & \text{complémentarité.} \end{array}}$$

Puisque $yA = c$, la condition de complémentarité peut également s'exprimer comme $yb - cx = 0$ ou $yb = cx$. Noter que $yb \geq cx$ et que l'égalité ne peut être réalisée qu'à l'optimum. Pour tout vecteur dual-réalisable y, la quantité yb représente donc une bonne supérieure sur la valeur optimale de l'objectif (primal) cx. On obtient la meilleure borne en résolvant le **programme linéaire dual** :

$$\boxed{\begin{array}{ll} \min\limits_{y} & yb \\ & yA = c \\ & y \geq 0. \end{array}}$$

Puisque le problème dual est un programme linéaire, on peut le mettre sous forme canonique :

$$\boxed{\begin{array}{ll} \max\limits_{y} & -yb \\ & \\ & yA \leq c \\ & -yA \leq -c \\ & -y \leq 0 \end{array}} \qquad \begin{array}{l} \text{variables duales du dual} \\ \\ u' \\ u'' \\ v \end{array}$$

et calculer son programme dual

$$\begin{array}{ll} \min\limits_{u',u'',v} & cu' - cu'' \\ & Au' - Au'' - v = -b \\ & u', u'', v \geq 0 \end{array} \qquad \text{ou} \qquad \begin{array}{ll} \max\limits_{u',u'',v} & c(u'' - u') \\ & A(u'' - u') + v = b \\ & u', u'', v \geq 0. \end{array}$$

En effectuant la substitution $x = u'' - u'$ on élimine les contraintes de signe sur les variables u' et u'', ce qui donne :

$$\begin{array}{ll} \max\limits_{x,v} & cx \\ & Ax + v = b \\ & v \geq 0, \end{array}$$

qui est équivalent (remarquer que v est une variable d'écart) au programme primal

$$\begin{array}{ll} \max\limits_{x} & cx \\ & Ax \leq b. \end{array}$$

À partir d'un dictionnaire optimal, il est possible de recouvrer la solution optimale du problème dual. Cette opération est effectuée par tous les logiciels de programmation linéaire.

Puisque le programme dual est un programme linéaire, on peut le résoudre comme le problème primal. Il est parfois avantageux de résoudre le dual plutôt que le primal, par exemple si le dual comporte moins de variables.

7. Le diagramme ci-dessous représente les diverses possibilités pour un couple primal-dual de programmes linéaires.

		dual			
		fini	$-\infty$	∞	
primal	fini	×			∞ : primal non borné ou dual non réalisable
	∞			×	$-\infty$: primal non réalisable ou dual non borné
	$-\infty$		×	×	

8. **Formes non canoniques**

   Si le programme linéaire n'est pas sous forme canonique, les règles suivantes s'appliquent :

   - à une contrainte d'inégalité dans le primal correspond une variable duale non négative ;
   - à une contrainte d'égalité dans le primal correspond une variable duale libre ;
   - à une variable libre dans le primal correspond une contrainte d'égalité dans le dual ;
   - à une variable non négative dans le primal correspond une contrainte d'inégalité dans le dual ;
   - les signes des contraintes principales (pas les contraintes $x \geq 0$) sont inversés ;
   - il y a complémentarité entre les contraintes d'inégalité du primal et les variables duales correspondantes, et vice versa. Ainsi, pour la forme utilisée dans l'algorithme du simplexe, on obtient

primal	**dual**	**complémentarité**
$\max_x \; cx$	$\min_y \; yb$	$(c - yA)x = 0$
$Ax = b$	$yA \geq c$	
$x \geq 0$		

   L'exemple suivant illustre tous les cas de figure.

primal	**dual**	**complémentarité**
min $2x_1 - 3x_2 + x_3 - 6x_4$	max $y_1 + 5y_2$	
$x_1 + x_2 + x_3 + x_4 = 1$	$y_1 + y_2 = 2$	$y_2(x_1 - 3x_2 + 2x_4 - 5) = 0$
$x_1 - 3x_2 + 2x_4 \geq 5$	$y_1 - 3y_2 \leq -3$	$x_2(-3 - y_1 + 3y_2) = 0$
	$y_1 \leq 1$	$x_3(1 - y_1) = 0$
$x_2, x_3 \geq 0$	$y_1 + 2y_2 = -6$	
x_1, x_4 libres		
	$y_2 \geq 0$	
	y_1 libre	

9. **Analyse paramétrique**

   Soit x une solution optimale d'un programme linéaire. Si l'on perturbe légèrement le vecteur c (c devient c') et que la base optimale n'en est pas modifiée, la valeur optimale du programme perturbé est $c'x$. Même si la solution x cesse d'être optimale, le nombre $c'x$ nous donne une borne *inférieure* sur la valeur optimale du problème perturbé.

   Symétriquement, si le vecteur de ressources b est perturbé (b devient b') et que la base demeure *réalisable*, l'objectif $cx = yb$ devient yb', où y est une solution optimale du problème dual. Si la base cesse d'être réalisable, alors yb' fournit une borne *supérieure* sur la valeur du nouvel objectif. On

réalise ici l'importance du vecteur dual-optimal y. La composante y_i représente la **valeur marginale** d'une unité supplémentaire de la $i$ième ressource.

Pour toutes ces raisons, il est important de connaître l'information duale et de l'intégrer au dictionnaire final, pour former un **dictionnaire final étendu** (voir figure 6). Noter que la numérotation des variables duales doit correspondre à l'ordre des contraintes. Dans notre exemple, l'ordre des contraintes avait été modifié à la deuxième itération, ce qui explique la numérotation des variables duales. Noter que ce n'est pas le fruit du hasard si les composantes positives du vecteur dual optimal sont égales aux coûts réduits des variables hors-base, au signe près, que l'on trouve dans la première ligne du tableau final.

		x_4	x_5		
max	36	$-\frac{3}{2}$	-1		
x_2	6	$-\frac{1}{2}$	0	1	y_3
x_1	2	$\frac{1}{3}$	$-\frac{1}{3}$	0	y_1
x_3	2	$-\frac{1}{3}$	$\frac{1}{3}$	$\frac{3}{2}$	y_2

optimum : x^3

FIGURE 6 – Un dictionnaire étendu

10. **Logiciels**

Il existe de nombreux logiciels pour résoudre des problèmes linéaires. Pour vous amuser, vous pouvez tester l'une des nombreuses applets Java, disponibles sur Internet et permettant de résoudre de petits programmes linéaires. Plus sérieusement, vous pouvez utiliser des logiciels spécialisés tels que LINDO, CPLEX ou GAMS. Des versions étudiantes gratuites de GAMS sont disponibles sur Internet.

Lorsque les problèmes deviennent grands, il est nécessaire d'utiliser une syntaxe évoluée. Lorsqu'ils deviennent très grands, il est indispensable d'utiliser un langage spécialisé qui construit, pour une application spécifique, la matrice des contraintes selon la syntaxe du logiciel et les spécifications de l'usager. Ces langages sont appelés «générateurs de matrices». Enfin, je mentionne que les tableurs les plus populaires, tels que Excel, permettent de résoudre des problèmes linéaires. Avis aux amateurs des produits Microsoft.

11. **Forme matricielle**

Nous terminons ce chapitre par un exposé de certains résultats basés sur une notation matricielle-vectorielle. Pour changer le mal de place, nous considérons un objectif à minimiser plutôt qu'à maximiser.

$$\begin{array}{lll} \min\limits_{x} & cx & \\ & Ax = b & c: \quad 1 \times n \\ & x \geq 0 & x: \quad n \times 1 \\ & & A: \quad m \times n \\ & & b: \quad m \times 1 \end{array} \qquad \text{(P)}$$

On suppose que la matrice A (dont la $j$ième colonne sera notée A_j) est de plein rang. Soit B une sous-matrice carrée (**base**) formée de m colonnes indépendantes de A. On associe à B les décompositions $A = [B|N]$ et $x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}$. Si l'on fixe le vecteur des **variables hors base** x_N à zéro, on obtient la **solution de base** $x = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$. Une solution de base est **admissible** si $x_B \geq 0$. Réécrivons le programme linéaire (P) en mettant en évidence la décomposition de A et de x :

$$\begin{aligned} \min_{x_B, x_N} \quad & c_B x_B + c_N x_N \\ & B x_B + N x_N = b \\ & x_B, x_N \geq 0 \end{aligned}$$

ou, après élimination de x_B à l'aide des contraintes d'égalité :

$$\begin{aligned} \min_{x_N \geq 0} \quad & c_B B^{-1} b + (c_N - c_B B^{-1} N) x_N \\ & B^{-1} b - B^{-1} N x_N \geq 0. \end{aligned}$$

Puisque $x_B = B^{-1}b \geq 0$, la solution $x_N = 0$ du programme ci-dessus est réalisable.
L'**algorithme du simplexe** tente d'améliorer la solution en intégrant dans la base une **variable hors-base** x_{j^*} ($j^* \in N$) dont le **coût réduit** $c_{j^*} - c_B B^{-1} A^{j^*}$ est négatif. La valeur maximale que peut prendre x_{j^*} est donnée par

$$x_{j^*} = \min_{i \in B} \left\{ \frac{(B^{-1}b)_i}{(B^{-1}A^{j^*})_i} : (B^{-1}A^{j^*})_i > 0 \right\}.$$

Si $t = +\infty$, le programme linéaire (P) est non borné inférieurement. Sinon, on fait sortir de la base une variable x_{i^*} ($i^* \in B$) telle que

$$(B^{-1}b)_{i^*} - x_{j^*}(B^{-1}A^{j^*})_{i^*} = 0.$$

La nouvelle base est : $B \leftarrow B \cup A^{j^*} - A^{i^*}$.
On continue le processus jusqu'à ce que le vecteur des coûts réduits $c - c_B B^{-1} A$ soit non négatif. [**remarque :** la partie de ce vecteur correspondant aux variables de base est nulle puisque $c_B - c_B B^{-1} B = 0$.]
Dualité : Soit y un vecteur $1 \times m$. Une borne *inférieure* sur la valeur du programme (P) s'obtient en substituant aux contraintes $Ax = b$ l'unique contrainte $yAx = yb$. Si de plus on a que $yA \leq c$, on obtient trivialement la borne inférieure $cx \geq yAx = yb$. Si l'on veut maintenant obtenir la *meilleure* borne inférieure à partir de cette technique, on est amené à étudier le **programme linéaire dual**

$$\boxed{\begin{aligned} \max_{y} \quad & yb \\ & yA \leq c \end{aligned}} \qquad \text{(D)}$$

Remarquons que le dual du programme linéaire (D) est (P). De plus, si le problème primal (P) possède un minimum, on a le résultat :

$$\max_y yb = \min_x cx.$$

Les conditions **primales-duales** suivantes caractérisent l'optimalité d'une solution :

$$\left.\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned}\right\} \quad \text{primal réalisable}$$

$$\left.\begin{aligned} yA &\leq c \end{aligned}\right\} \quad \text{dual réalisable}$$

$$\left.\begin{aligned} yb &= cx \\ \text{ou } (c - yA)x &= 0 \end{aligned}\right\} \quad \begin{array}{l} \text{écarts complémentaires} \\ \text{(orthogonalité)} \end{array}$$

Les variables y sont appelées **variables duales** ou parfois **multiplicateurs du simplexe**. Si B est une base optimale, c'est-à-dire que les coûts réduits sont tous positifs ou nuls, alors $y = c_B B^{-1}$ est une solution optimale du problème dual. Finalement, on dit qu'une base (ou la solution de base correpondante) est **dégénérée** si au moins une des variables de base est nulle.

CHAPITRE 3 : PROGRAMMATION NON LINÉAIRE

Un programme non linéaire prend la forme générique

$$\begin{array}{lllll} \max\limits_{x} & f(x_1,\ldots,x_n) & & \\ & g_1(x_1,\ldots,x_n) & \leq & b_1 \\ & \vdots & \vdots & \vdots \\ & g_m(x_1,\ldots,x_n) & \leq & b_m \end{array}$$

où au moins une des fonctions f et $g_1,\ldots,g_m$ est non linéaire.

L'équivalent de la notion de droite de constance en programmation linéaire est la notion de **courbe de niveau** qui constitue le lieu géométrique des points admettant une valeur commune de l'objectif. Un résultat *fondamental* est que le **gradient** de la fonction f au point $x = (x_1,\ldots,x_n)$, c'est-à-dire le vecteur ligne de ses dérivées partielles

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1},\ldots,\frac{\partial f}{\partial x_n}\right)$$

est orthogonal à la courbe de niveau α passant par le point x (voir Figure 1), d'équation

$$\{y : f(y) = f(x) = \alpha\}.$$

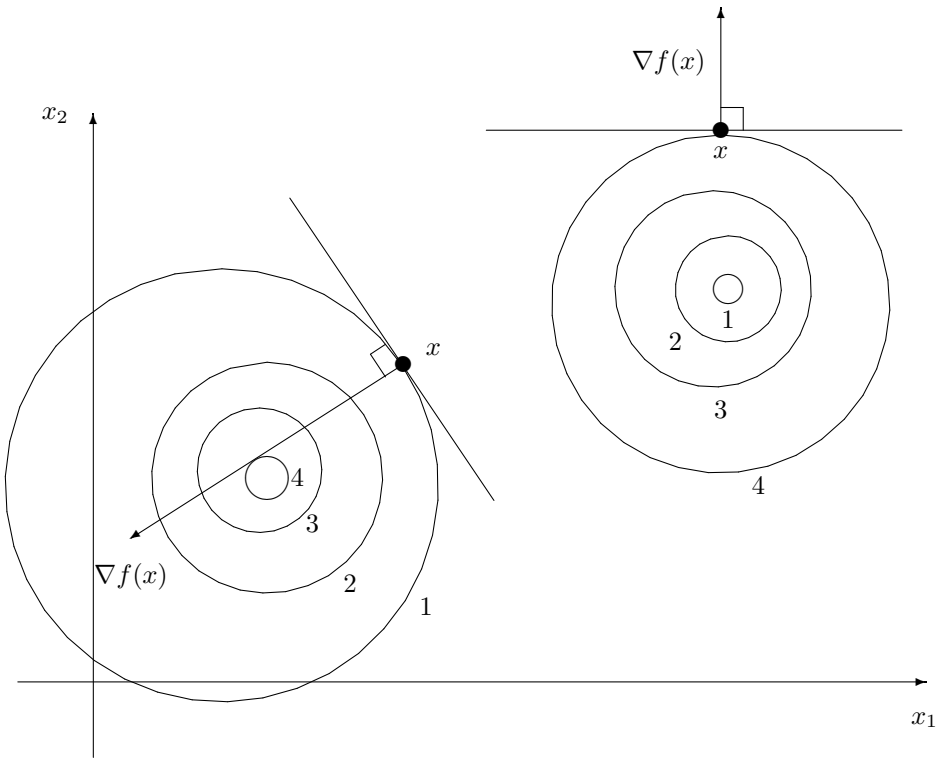


FIGURE 1 – Exemples de courbes de niveau

La direction du gradient est la direction de plus grande pente (ascendante). Une direction d est une direction de montée au point x si le produit scalaire $\nabla f(x)d$ est positif, et une direction de descente si $\nabla f(x)d$ est

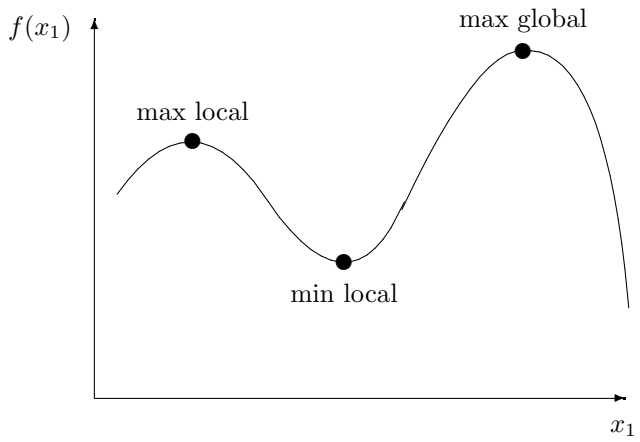


FIGURE 2 – Optima locaux et globaux

négatif. L'objet mathématique $\nabla f(x)d$ est la **dérivée directionnelle** de la fonction f dans la direction d au point x.

La géométrie d'un programme non linéaire est beaucoup plus complexe que celle d'un programme linéaire. En particulier :
– la solution n'est plus forcément extrémale ; celle-ci peut se situer à l'intérieur du domaine réalisable, là où aucune contrainte n'est satisfaite à égalité ;
– il peut exister des optima locaux (voir Figure 2).
Une condition *nécessaire* (mais pas nécessairement suffisante) pour qu'un point x^* soit optimal est qu'il n'existe pas de direction d'amélioration admissible en x^* (voir Figure 3) :

$$\nabla f(x^*)d \leq 0 \qquad \forall d \text{ admissible.}$$

Sous certaines hypothèses techniques, une condition nécessaire pour que cette condition soit satisfaite, et donc que x^* soit un candidat à l'optimalité, est que x^* soit une solution optimale du programme linéaire obtenu en linéarisant l'objectif et les contraintes du programme non linéaire originel :

$$\begin{array}{lrcl} \max\limits_x & f(x^*) + \nabla f(x^*)(x - x^*) & & \\ & & & \\ & g_1(x^*) + \nabla g_1(x^*)(x - x^*) & \leq & b_1 \\ & \vdots \qquad\qquad \vdots & & \vdots \\ & g_m(x^*) + \nabla g_m(x^*)(x - x^*) & \leq & b_m \end{array}$$

où, sous forme compacte et canonique :

$$\boxed{\begin{array}{lrcl} \max\limits_x & \nabla f(x^*)x & & \\ & \nabla g_i(x^*)x & \leq & b_i - g_i(x^*) + \nabla g_i(x^*)x^* \quad i = 1, \ldots, m \end{array}}$$

Les conditions d'optimalité de ce programme linéaire sont

$$\begin{array}{rcll} \nabla g_i(x^*)x & \leq & b_i - g_i(x^*) + \nabla g_i(x^*)x^* \quad i = 1, \ldots, m & \bullet \text{ primal réalisable} \\ \sum\limits_{i=1}^{m} y_i \nabla g_i(x^*) & = & \nabla f(x^*) & \bullet \text{ dual} \\ y_i & \geq & 0 \qquad\qquad i = 1, \ldots, m & \text{réalisable} \\ y_i(b_i - g_i(x^*)) & = & 0 & \bullet \text{ complémentarité} \end{array}$$

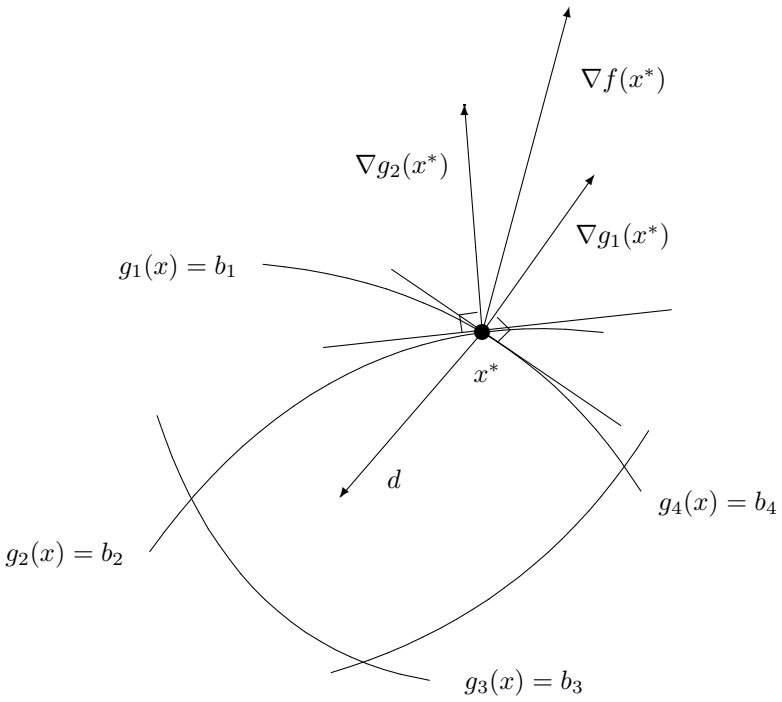


FIGURE 3 – Il n'existe pas de direction de montée admissible

En posant $x = x^*$ dans le système ci-dessus on obtient les **conditions nécessaires d'optimalité de Kuhn et Tucker**, qui constituent le résultat fondamental de la programmation non linéaire :

$$\begin{array}{rcll} g_i(x^*) & \leq & b_i & i = 1, \ldots, m \\ \sum_{i=1}^{m} y_i \nabla g_i(x^*) & = & \nabla f(x^*) & \\ y_i & \geq & 0 & i = 1, \ldots, m \\ y_i(b_i - g_i(x^*)) & = & 0 & i = 1, \ldots, m \end{array}$$

Géométriquement, le gradient $\nabla f(x^*)$ de l'objectif s'exprime comme une combinaison linéaire non négative des gradients $\nabla g_i(x^*)$ des contraintes actives (saturées), comme en programmation linéaire.

Dans le cas où l'objectif est **concave** et les contraintes sont **convexes** [1] , un point satisfaisant aux conditions de Kuhn et Tucker constitue un maximum *global* du programme non linéaire.

Une fonction d'une seule variable est convexe (voir la figure 4) si sa dérivée est croissante ou si sa dérivée seconde est toujours positive ou nulle. Une fonction de plusieurs variables est convexe si elle est convexe dans toutes les directions. Un critère utile de convexité d'une fonction f est que la matrice hessienne $H(x)$(matrice des dérivées secondes) soit semidéfinie positive pour toute valeur de x, c'est-à-dire que toutes les valeurs propres de $H(x)$ soient non négatives. Un cas particulier important est celui des formes quadratiques $x^t Ax$ où la matrice A est semidéfinie positive, c'est-à-dire que toutes les valeurs propres de $A + A^t$ sont non négatives.

Une fonction f est concave si et seulement si son opposée $-f$ est convexe. Noter qu'une fonction linéaire est à la fois convexe et concave.

1. On parle alors d'un problème **convexe**.

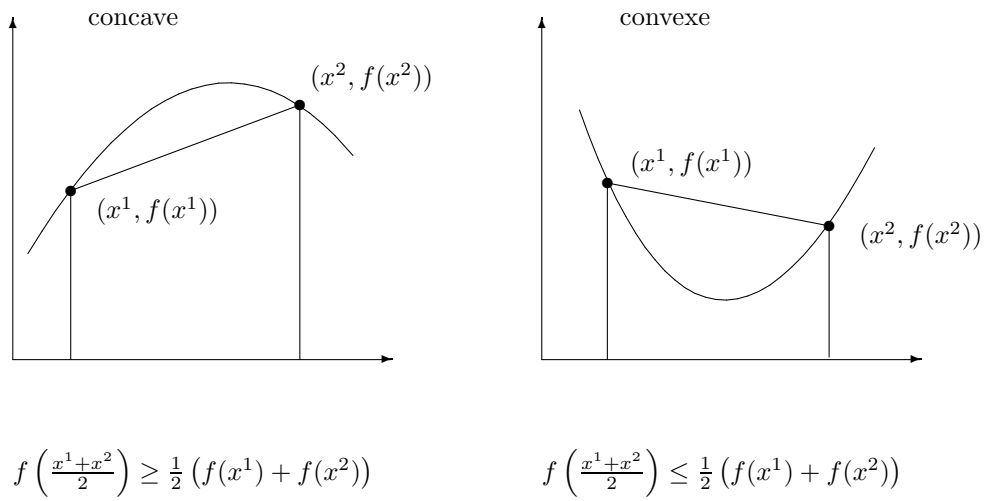


FIGURE 4 – Fonction concave, fonction convexe

Appliquons les conditions de Kuhn et Tucker à l'exemple suivant :

$$\begin{array}{lrcl} \max & x_1^2 + x_2^2 & & \\ & & & \\ & x_1 + 2x_2 & \leq & 5 \\ & -x_1 & \leq & 0 \\ & -x_2 & \leq & -1 \end{array}$$

On obtient

$$\begin{array}{rrr} x_1 + 2x_2 \leq 5 & y_1 - y_2 = 2x_1 & y_1(5 - x_1 - 2x_2) = 0 \\ -x_1 \leq 0 & 2y_1 - y_3 = 2x_2 & y_2 x_1 = 0 \\ -x_2 \leq -1 & & y_3(-1 + x_2) = 0 \\ & y_1, y_2, y_3 \geq 0. & \end{array}$$

On vérifie facilement que le point $x = (3, 1)$ et le vecteur de multiplicateurs $y = (6, 0, 10)$ satisfont aux conditions de KKT. Géométriquement (figure 5), on a que le gradient (6,0,10) est bien coincé entre les vecteurs orthogonaux aux deux contraintes satisfaites à égalité :

$$\sum_{i=1}^{3} y_i \nabla g_i(x^*) = \nabla f(x^*).$$

Cette solution n'est pas unique. En effet, le point $(0, 5/2)$ en est un optimum local. Il existe même une troisième solution : la projection de l'origine sur la première contrainte. Noter que ce point n'est ni un minimum (même local) ni un maximum (local). Il constitue un minimum global sur le segment joignant les points (0,5/2) et (3,1). Noter que les deux directions vers les extrémités du segment sont des directions de montée, mais qu'il est impossible de le savoir à partir de la seule connaissance du gradient au point courant. Une telle situation ne peut se produire si la fonction objectif est concave et les contraintes convexes.

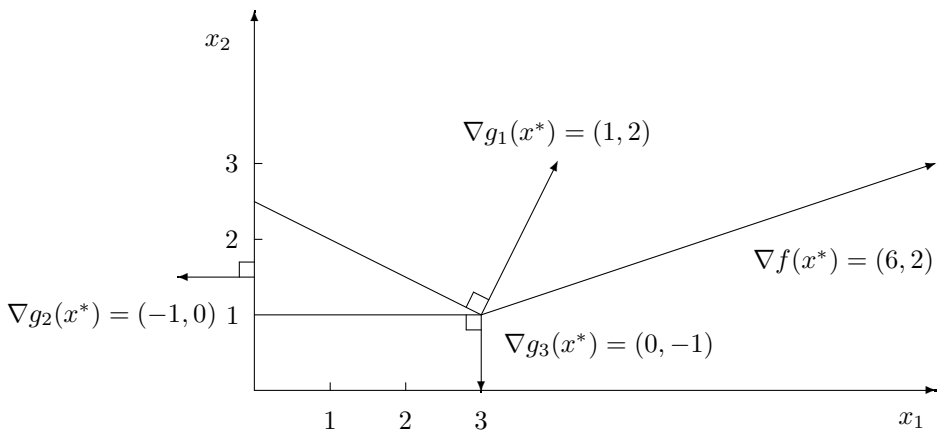


FIGURE 5 – Les conditions de Kuhn et Tucker

ALGORITHMES

Ce n'est certainement pas le but de ce cours de traiter en profondeur des algorithmes de résolution pour les programmes non linéaires, ce qui occuperait le cours dans son entier. Cependant, je vais décrire très brièvement trois stratégies de résolution utilisées en pratique.

1. UNE STRATÉGIE DE PÉNALISATION DES CONTRAINTES

   Soit le programme mathématique

$$\begin{aligned} \max_x \quad & f(x) \\ & g_i(x) \leq b_i \qquad i = 1, \ldots m. \end{aligned}$$

   Une façon simple de se débarasser des contraintes est d'intégrer dans l'objectif une fonction qui pénalise les vecteurs qui violent les contraintes :

$$\max_x \quad f(x) - M \sum_{i=1}^{m} P_i(x),$$

   où $P_i(x)$ prend une valeur positive lorsque $g_i(x)$ est strictement positif, ce qui décourage l'adoption de ces vecteurs x. On peut par exemple poser

$$P_i(x) = \max\{0, g_i(x) - b_i\}.$$

   Lorsque M est très grand, il y a intérêt à satisfaire les contraintes, quitte à détériorer l'objectif Si l'on souhaite que l'objectif pénalisé soit différentiable, on peut élever l'expression précédente au carré. Pour résoudre le problème pénalisé, il suffit alors de trouver les points où le gradient de son objectif s'annule. Mais ceci est une longue histoire qui est du ressort du cours IFT 3515 et non celui d'un premier cours de recherche opérationnelle.

2. UNE PREMIÈRE STRATÉGIE DE LINÉARISATION (FRANK-WOLFE)

   Soit le problème

$$\max_{x \in X} \quad f(x).$$

   Si l'ensemble X est un polyèdre, on peut se ramener à un problème de programmation linéaire en faisant une approximation linéaire de l'objectif. Ceci correspond à un développement de Taylor du premier ordre et conduit au sous-problème

$$\max_{y \in X} \quad f(x) + \nabla f(x)(y - x).$$

Soit $y(x)$ une solution de ce programme linéaire. On démontre aisément que la direction $d = y(x) - x$ est une direction d'amélioration. On maximise alors la fonction dans la direction d :

$$\max_{\alpha \in [0,1]} f(x + \alpha d),$$

ce qui n'est pas difficile car la fonction $f(x + \alpha d)$ ne comporte que la variable α. La solution de cette **recherche linéaire** conduit au nouveau point x^+ d'où l'on redémarre le processus (voir figure 6). Noter que si la valeur optimale de α est inférieure à 1, alors le gradient $\nabla f(x^+)$ doit forcément être orthogonal à la direction d (pourquoi ?).

Cette méthode est très appréciée dans le domaine du transport où le sous-problème linéaire est fréquemment un problème à la structure très simple qui peut être résolu par des algorithmes spécialisés.

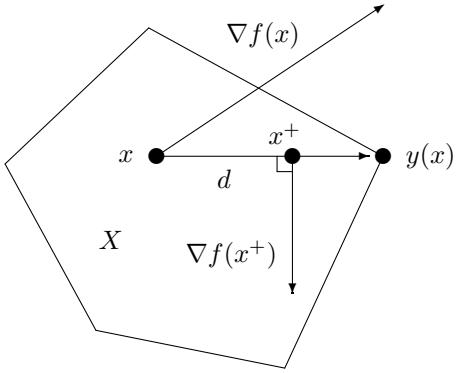


FIGURE 6 – Un premier algorithme de linéarisation

3. UNE SECONDE STRATÉGIE DE LINÉARISATION

Une fonction f est **additive** si elle peut se mettre sous la forme

$$f(x_1, \ldots, x_n) = f_1(x_1) + \ldots + f(x_n).$$

Considérons le programme mathématique

$$\begin{aligned} \max_{x} \quad & f(x) \\ & Ax \leq b, \end{aligned}$$

où toutes les fonctions impliquées sont additives et **concaves**, et où les contraintes sont linéaires. Il est alors possible d'approximer chaque fonction, sur un intervalle $[a, b]$, par une fonction linéaire par morceaux et de substituer au problème de départ un programme linéaire (voir figure 7).

Les nouvelles variables sont maintenant les λ_k. On démontre, si l'objectif est convexe, qu'au plus deux variables λ_k peuvent prendre des valeurs non nulles, et que ces variables doivent avoir des indices consécutifs. Si ce n'était pas le cas, l'approximation linéaire pourrait être invalide. Noter enfin que, plus le nombre de points d'évaluation x^k est élevé, plus l'approximation sera fine et plus le programme linéaire à résoudre sera grand, et que la solution obtenue peut être sous-optimale.

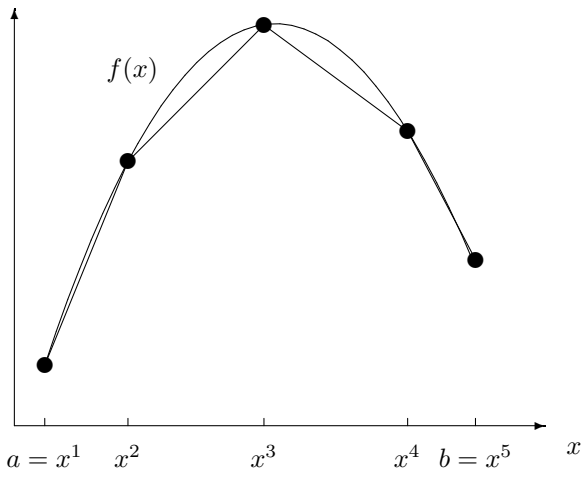


FIGURE 7 – Approximation linéaire par morceaux d'une fonction concave

Cette technique se généralise à des problèmes impliquant des programmes non linéaires plus généraux, où l'objectif est concave et les fonctions définissant les contraintes sont convexes :

$$\begin{aligned} \max_x \quad & f(x) \\ & g_i(x) \leq b_i \qquad i = 1, \ldots, m. \end{aligned}$$

On exprime alors x comme la **combinaison convexe** de K points réalisables x^k :

$$x = \sum_{k=1}^{K} \lambda_k x^k \qquad \sum_{k=1}^{K} \lambda_k = 1 \qquad \lambda_k \geq 0, \quad k = 1, \ldots, K,$$

et on approxime le problème originel par

$$\begin{aligned} \max_\lambda \quad & \sum_{k=1}^{K} \lambda_k f(x^k) \\ & \sum_{k=1}^{K} \lambda_k g_i(x^k) \leq b_i \qquad i = 1, \ldots, m \\ & \sum_{k=1}^{K} \lambda_k = 1 \\ & \lambda_k \geq 0 \qquad k = 1, \ldots, K. \end{aligned}$$

4. AUTRES CONSIDÉRATIONS

Il n'y a pas de stratégie universelle qui permette de résoudre efficacement tous les problèmes non linéaires. Il est important de tenir compte de la structure particulière que l'on traite. Par exemple, il existe des algorithmes spécialisés pour résoudre des programmes quadratiques, c'est-à-dire des programmes où l'objectif est quadratique et où les contraintes sont linéaires. Les conditions de Kuhn et Tucker de ces programmes sont très semblables à celles de programmes linéaires et il est possible de les résoudre par des variantes de la méthode du simplexe. Pour en savoir plus, vous pouvez consulter l'excellent livre "Nonlinear Programming" de Bazaraa, Sherali et Shetti.

CHAPITRE 4 : THÉORIE DES JEUX

7 octobre 2003

La théorie des jeux constitue un domaine fascinant où l'on cherche à décrire par des équations le comportement stratégique d'êtres humains. On distingue les jeux **non coopératifs** où les coalitions ne sont pas admises et les **jeux coopératifs** où les joueurs peuvent former des coalitions afin de maximiser un revenu commun à la coalition et qu'ils se partagent par la suite, à l'intérieur de chaque coalition. Dans cette section, nous ferons l'hypothèse que les joueurs annoncent *simultanément* leur stratégie.

Jeux non coopératifs

Dans cette section, on ne donne pas de description détaillée du déroulement du jeu, ce qui rendrait impossible son analyse mathématique. On se contente de donner le résultat du jeu (gains) correspondant aux choix stratégiques des joueurs. Ainsi, dans le jeu d'échecs, une stratégie correspond à une règle qui détermine, pour chaque position, le mouvement à effectuer. D'un point de vue mathématique, ce jeu n'a que peu d'intérêt et n'a que 3 issues possibles : soit il existe une stratégie gagnante pour les blancs, soit il existe une stratégie gagnante pour les noirs, soit toutes les parties entre deux joueurs «parfaits» se terminent par une nulle.

Considérons un jeu impliquant n joueurs cherchant chacun à maximiser un objectif qui lui est propre. L'objectif $f_i(x_1, \ldots, x_n)$ du joueur i dépend à la fois de sa **stratégie** x_i ainsi que des stratégies de ses adversaires. Si l'on suppose que le domaine X_i des stratégies admissibles du joueur i ne dépend pas des stratégies de ses adversaires, celui-ci est confronté au programme mathématique

$$\max_{x_i \in X_i} f_i(x_1, \ldots, x_i, \ldots, x_n).$$

On fera l'hypothèse que ce programme admet une solution unique. Un ensemble de stratégies $(x_1^*, \ldots, x_n^*)$ est un *équilibre de Nash* si aucun joueur ne peut améliorer son sort en modifiant *unilatéralement* sa stratégie, c'est-à-dire, pour tout $i \in \{1, \ldots, n\}$, on a $x_i^* \in X_i$ et

$$f_i(x_1^*, \ldots, x_{i-1}^*, x_i^*, x_{i+1}^*, \ldots, x_n^*) \geq f_i(x_1^*, \ldots, x_{i-1}^*, x_i, x_{i+1}^*, \ldots, x_n^*) \qquad \forall x_i \in X_i.$$

Considérons par exemple un *duopole* où deux firmes tentent d'écouler un produit sur un marché. Le prix de vente du produit sur le marché est spécifié par une fonction décroissante de la demande totale. Cette fonction est appelée **loi de demande inverse**. Si $c_1(x_1)$ et $c_2(x_2)$ représentent, respectivement, les coûts de productions des deux firmes, les objectifs des firmes s'expriment comme

$$f_i(x_1, x_2) = x_i p(x_1 + x_2) - c_i(x_i).$$

Ainsi, si $p(x) = 200 - x$, $c_1(x_1) = x_1^2 + 20$ et $c_2(x_2) = 2x_2^2 + 10$, on obtient les deux problèmes d'optimisation :

$$\begin{aligned} \max_{x_1 \geq 0} \quad & x_1(200 - (x_1 + x_2)) - (x_1^2 + 20) \\ \max_{x_2 \geq 0} \quad & x_2(200 - (x_1 + x_2)) - (2x_2^2 + 10). \end{aligned}$$

En cherchant à annuler les dérivées de chacun des objectifs, on obtient le système linéaire

$$\begin{array}{rcrcrcrcl} 200 & - & 2x_1 & - & x_2 & - & 2x_1 & = & 0 \\ 200 & - & x_1 & - & 2x_2 & - & 4x_2 & = & 0 \end{array}$$

dont la solution est

$$(x_1, x_2) = (1000/23, 600/23).$$

Les profits respectifs des deux firmes sont approximativement 3760 et 2031. On vérifie que les profits des deux joueurs sont non négatifs. Si ce n'était pas le cas, il faudrait reprendre l'analyse en éliminant l'une des deux firmes car, si un profit est négatif, il est préférable de ne rien produire [1]. Il se peut même qu'à l'équilibre, la production des deux firmes soit nulle.

1. Paradoxe : ce n'est pas nécessairement la firme de profit négatif qui sera éventuellement sortie du marché !

Il est à noter qu'en fusionnant, les deux firmes augmenteraient leur profit global en résolvant le programme

$$\max_{x_1 \geq 0, x_2 \geq 0} x_1(200 - (x_1 + x_2)) - (x_1^2 + 20) + x_2(200 - (x_1 + x_2)) - (2x_2^2 + 10)$$

dont la solution $(x_1, x_2) = (40, 20)$ fournit un profit total de 5970 (3980 pour la première firme et de 1990 pour la seconde). Il existe des situations où le profit de chacune des firmes augmente après fusion. Ceci nous rappelle le paradoxe de Braess en équilibre du trafic.

Il n'est pas dans notre propos de traiter des algorithmes permettant de trouver un équilibre de Nash, sinon pour indiquer qu'il est possible de procéder par tâtonnement, en résolvant successivement les problèmes des deux joueurs, en espérant que le processus converge !

Jeux matriciels

[Dans cette section on adopte la convention que tous les vecteurs sont des vecteurs colonne et l'on utilisera l'opérateur de transposition]

Peut-être la forme la plus simple d'un jeu est celle où chaque joueur possède un nombre fini de stratégies et où les gains associés à chaque combinaison de stratégies est connue des deux joueurs. Si le gain du premier joueur correspond au déboursé du second, on dit que le jeu est à **somme nulle**. (Noter que le duopole de la section précédente n'est pas à somme nulle.) Un tel jeu peu se représenter sous forme matricielle, le premier joueur étant associé aux lignes de la matrice et le second à ses colonnes. Les coefficients de la matrice représentent les gains du premier joueur, qui cherche à maximiser ses gains, alors que le second joueur tente de minimiser ses pertes, c'est-à-dire les gains du joueur ligne.

Considérons le jeu matriciel associé à la matrice

$$\begin{bmatrix} 4 & 3 & 10 \\ 6 & \boxed{5} & 7 \end{bmatrix}.$$

Il est facile de vérifier que le couple stratégique $(2, 2)$ constitue un équilibre de Nash, et que le profit du joueur ligne, qu'on appelle la **valeur du jeu**, est égal à 5. Le coefficient $a_{22} = 5$ de la matrice est un **point de selle**, c'est-à-dire qu'il constitue le minimum de sa ligne et le maximum de sa colonne.

Considérons maintenant la matrice

$$\begin{bmatrix} -1 & 1 \\ 3 & -2 \end{bmatrix},$$

qui ne possède pas de point de selle. Le jeu correspondant ne possède donc pas d'équilibre de Nash. Une façon élégante de corriger la situation est d'élargir l'ensemble des stratégies en introduisant un élément aléatoire dans le jeu. Une **stratégie mixte** est un vecteur de probabilités dont la dimension est égale au nombre de stratégies. L'ensemble des stratégies mixtes du premier joueur est

$$X = \{x = (x_1, \ldots, x_m) \, : \, \sum_{i=1}^{m} x_i = 1, x \geq 0\}.$$

Ainsi, la probabilité que le joueur ligne adopte la stratégie i est égale à x_i. Symétriquement, l'ensemble des stratégies mixtes du joueur colonne est

$$Y = \{y = (y_1, \ldots, y_m) \, : \, \sum_{j=1}^{n} y_j = 1, y \geq 0\}.$$

La probabilité que le couple stratégique (i, j) soit observé est égale à $x_i y_j$, et le gain moyen (espérance de gain) du joueur ligne est donnée par l'expression

$$\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_i y_j = x^t A y.$$

Un équilibre de Nash (x, y) devra satisfaire aux conditions d'optimalité des programmes

$$\begin{array}{ll} \textbf{joueur 1} & \textbf{joueur 2} \\ \max_x \quad x^t Ay & \min_y \quad x^t Ay \\ \sum_{i=1}^{m} x_i = 1 & \sum_{j=1}^{n} y_j = 1. \\ x \geq 0 & y \geq 0 \end{array}$$

Les conditions d'optimalité de ces programmes linéaires sont

$$\begin{array}{ll} \textbf{joueur 1} & \textbf{joueur 2} \\ \sum_{i=1}^{m} x_i = 1 & \sum_{j=1}^{n} y_j = 1 \\ x \geq 0 & y \geq 0 \\ w \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \geq Ay & v \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \leq A^t x \\ x^t Ay = w & x^t Ay = v. \end{array}$$

On en déduit que la valeur du jeu est égale à $v = w = x^t Ay$. Par une coïncidence extraordinaire, en interchangeant le deuxième groupe de contraintes de chacun de ces systèmes et en oubliant la dernière ligne, on obtient les programmes linéaires primal et dual suivants :

$$\begin{array}{lll} \max & v & \min \quad w \\ \sum_{i=1}^{m} x_i & = & 1 \qquad \sum_{j=1}^{n} y_j = 1 \\ x & \geq & 0 \qquad y \geq 0 \\ v \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} & \leq & A^t x \qquad w \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \geq Ay. \end{array}$$

Ces deux programmes ont une interprétation intuitive. En effet, chaque joueur y détermine une stratégie visant à maximiser son gain minimal ou à minimiser sa perte maximale. De telles stratégies sont appelées, respectivement, **maximin** et **minimax**. Chaque joueur se prémunit contre la pire situation possible. Ces programmes linéaires sont équivalents aux programmes

$$\max_{x \in X} \min \{A^t x\}_i \qquad \min_{y \in Y} \max \{Ay\}_i.$$

Dans le petit exemple précédent, on obtient

$$\max_{x \in X} \min \{-x_1 + 3x_2, x_1 - 2x_2\} \qquad \min_{y \in Y} \max \{-y_1 + y_2, 3y_1 - 2y_2\}.$$

En posant $x_2 = 1 - x_1$ on obtient le problème unidimensionnel

$$\max_{x \in [0,1]} \min \{3 - 4x_1, 3x_1 - 2\},$$

dont la solution est $x_1 = 5/7$. Le vecteur stratégique optimal est donc $x = (5/7, 2/7)$ et la valeur du jeu est égale à $1/7$ (voir figure 1). De la même façon, on obtient le vecteur stratégique $y = (3/7, 4/7)$ pour le second joueur.

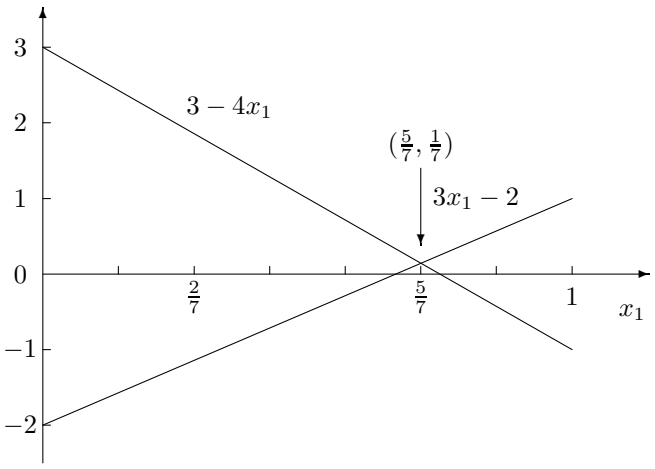


FIGURE 1 – Calcul graphique de l'équilibre

Jeux coopératifs

Le concept d'équilibre de Nash ne permet pas de modéliser les situations où des coalitions de joueur peuvent se former. La forme caractéristique d'un jeu a été introduite pour traiter correctement de telles situations. Un tel jeu est spécifié entièrement par sa **fonction caractéristique** v qui associe à chaque coalition de joueurs $S \subseteq N = \{1, \ldots, n\}$ la valeur minimale qu'elle peut obtenir indépendamment du comportement des joueurs qui ne font pas partie de la coalition.

On fait l'hypothèse raisonnable que la fonction caractéristique est **superadditive**, c'est-à-dire que, pour tout couple de coalitions A et B ne partageant aucun membre commun, on a

$$v(A \cup B) \geq v(A) + v(B).$$

En d'autres mots, l'union fait la force ! Une **imputation** $x = (x_1, \ldots, x_n)$ est une répartition du gain maximal $v(N)$ parmi les joueurs, qui satisfait la condition supplémentaire

$$x_i \geq v(\{i\}).$$

Cette dernière condition est tout à fait raisonnable. En effet, aucun joueur rationnel n'accepterait de recevoir moins que ce qu'il peut obtenir en ne coopérant pas. On peut généraliser cette condition à toutes les coalitions, c'est-à-dire

$$\sum_{i \in S} x_i \geq v(S)$$

pour toutes les coalitions S (on a l'égalité si $S = N$). L'ensemble des imputations satisfaisant cette dernière condition est appelé le **cœur** du jeu.

Exemple

Un inventeur (joueur 1) cherche une compagnie (joueur 2 ou joueur 3) pour commercialiser son produit. Si le produit est commercialisé, il rapportera 10^6 euros. La fonction caractéristique du jeu est, naturellement :

$$v(\emptyset) = v(\{1\}) = v(\{2\}) = v(\{3\}) = v(\{2,3\}) = 0$$
$$v(\{1,2\}) = v(\{1,3\}) = v(\{1,2,3\}) = 10^6.$$

Les équations définissant le cœur du jeu sont

$$\begin{aligned} x_1 + x_2 &\geq 10^6 \\ x_1 + x_3 &\geq 10^6 \\ x_2 + x_3 &\geq 0 \\ x_1 + x_2 + x_3 &= 10^6 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

dont l'unique solution est $x = (10^6, 0, 0)$.

Dans ce cas, le cœur contient un seul élément, ce qui est rassurant. Malheureusement, il est facile de construire des exemples où le cœur du jeu (un polyèdre convexe) contient une nombre infini d'éléments ou, pire, est vide. Plusieur propositions ont été faites afin de remédier à cette situation. Elles ont toutes leurs qualités et leurs défauts.

Extensions

Ce qui précède n'est évidemment qu'un bref survol d'un domaine très riche. En vrac, voici quelques notions qui ont fait l'objet d'études sérieuses :

jeux répétés Jeux où les joueurs peuvent adapter leurs stratégies en observant le comportement des adversaires.

jeux dynamiques Jeux qui se déroulent dans le temps.

jeux différentiels Jeux qui se déroulent en temps réels et sont décrits par des systèmes d'équations différentiels.

menaces Modélisation d'un élément important !

négociation Important dans les jeux coopératifs.

indices de pouvoir Concepts alternatifs à celui de cœur.

propension au risque Certains joueurs ont une aversion au risque et préfère ≪jouer sûr≫, c'est-à-dire accepter un gain moins élevé associé à un risque (variance du gain) plus faible.

CHAPITRE 5 : PROGRAMMATION LINÉAIRE AVEC VARIABLES ENTIÈRES

dernières modifications : 30 mars 2016

Lorsque les variables d'un programme mathématique représentent des décisions stratégiques, ou simplement lorsque certaines variables ne peuvent prendre des valeurs fractionnaires, il faut intégrer dans le modèle des contraintes d'intégralité. Ceci modifie profondément la nature des programmes linéaires.

Prenons l'exemple, peu édifiant je l'avoue, du cambrioleur muni d'un sac qui lui sert à transporter le butin tiré d'un coffre-fort [1] : bijoux, argent, caviar, portos anciens, objets divers. Son problème consiste à maximiser la valeur totale des objets qu'il emporte, sans toutefois dépasser une limite de poids b correspondant à ses capacités physiques. Si l'on associe à l'objet j une valeur c_j et un poids w_j, la combinaison optimale d'objets à emporter sera obtenue en résolvant le programme mathématique

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} w_j x_j \leq b \\ & x_j \text{ entier non négatif.} \end{aligned}$$

La solution de ce problème semble triviale : pourquoi ne pas choisir de façon prioritaire les objets dont le rapport qualité-poids est le plus avantageux, quitte à tronquer pour obtenir une solution entière (il n'y a pas de demi-bouteilles de porto dans le coffre-fort) ? Et bien, tout simplement parce que cette solution peut se révéler sous-optimale, voire mauvaise, comme on le constate sur l'exemple suivant.

$$\begin{array}{lrcrcrl} \max & 2x_1 & + & 3x_2 & + & 7x_3 & \\ & 3x_1 & + & 4x_2 & + & 8x_3 & \leq 14 \\ & x_1, & & x_2, & & x_3 & \text{entiers non négatifs.} \end{array}$$

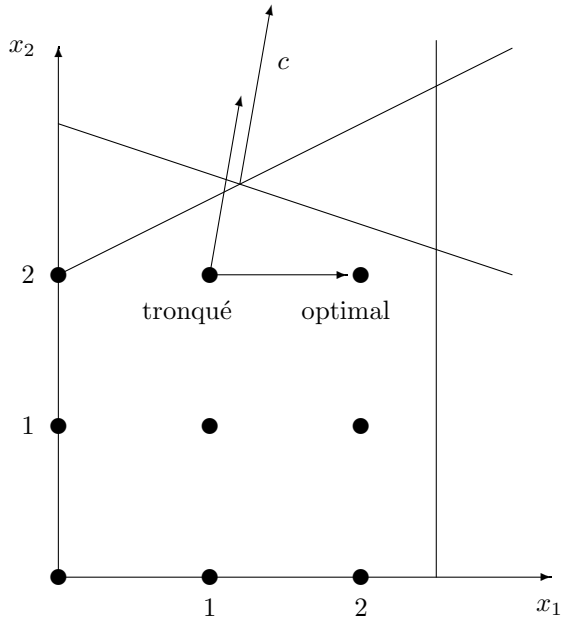


FIGURE 1 – Un programme en variables entières

1. Le coffre-fort, immense, contient un nombre incalculable d'objets de chaque type.

Si l'on oublie la contrainte d'intégralité, la solution, de valeur $12\frac{1}{4}$, est $x = (0, 0, 14/8)$. En tronquant, on obtient la solution entière $x = (0, 0, 1)$ de valeur 7. Il est facile de trouver de meilleures solutions. Une représentation géométrique d'un autre exemple, impliquant 2 variables, est donné à la figure 1.

La solution optimale du problème du cambrioleur peut s'obtenir en énumérant toutes les solutions admissibles et en conservant la meilleure (voir tableau ci-dessous, où les solutions inefficaces laissant la possibilité d'ajouter un objet n'ont pas été considérées).

x_1	x_2	x_3	objectif
0	1	1	10
2	0	1	11
0	3	0	9
2	2	0	10
3	1	0	9
4	0	0	8

La solution optimale entière, $x = (2, 0, 1)$, diffère passablement de la solution optimale linéaire. Cependant, il est clair que cette technique d'énumération ne peut s'appliquer à des problèmes de grande taille. J'y reviendrai après avoir formulé d'autres problèmes où certaines variables ne peuvent prendre des valeurs fractionnaires.

Affectation des équipages

Un problème important des compagnies aériennes consiste à constituer de façon efficace des équipages pour ses vols. Pour un équipage donné, une **rotation** consiste en une succession de **services de vol** débutant et se terminant en une même ville. Chaque service de vol est constitué d'une séquence de **segments de vol** (voir figure 2).

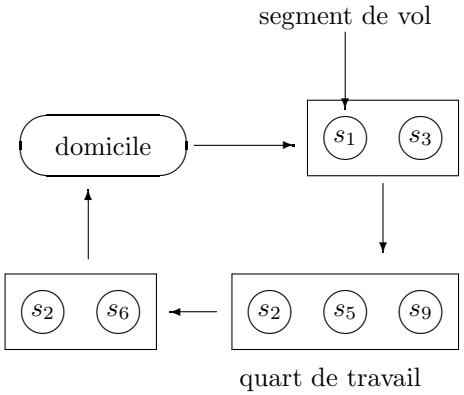


FIGURE 2 – Un exemple de rotation

Soit T l'ensemble des rotations, $T(i)$ l'ensemble des rotations contenant le segment de vol i et c_j le coût associé à la rotation j. Soit x_j une variable binaire prenant la valeur 1 si la rotation j est utilisée. Le problème d'affectation des équipages aux rotations peut s'exprimer comme le problème de **recouvrement**

$$\begin{aligned} \min \quad & \sum_{j \in T} c_j x_j \\ & \sum_{j \in T(i)} x_j \geq 1 \\ & x_j \in \{0, 1\}. \end{aligned}$$

En pratique, il faut tenir compte des contraintes imposées par les conventions collectives, ce qui complique singulièrement le problème.

Localisation

Une entreprise envisage plusieurs sites de construction pour des usines qui serviront à approvisionner ses clients. À chaque site potentiel i correspond un coût de construction a_i, une capacité de production u_i, un coût de production unitaire b_i et des coûts de transport c_{ij} des usines vers les clients.

Soit y_i une variable binaire prenant la valeur 1 si un entrepôt est construit sur le site i, d_j la demande de l'usine j et x_{ij} la quantité produite à l'usine i et destinée au marché j (flot de i à j). Un plan de construction optimal est obtenu en résolvant le programme

$$\begin{aligned}
\min \quad & \sum_i a_i y_i + \sum_i b_i \sum_j x_{ij} + \sum_i \sum_j c_{ij} x_{ij} \\
& \sum_i x_{ij} = d_j \\
& \sum_j x_{ij} \leq u_i y_i \\
& x_{ij} \geq 0 \qquad y_i \in \{0,1\}.
\end{aligned}$$

Cette formulation contient deux éléments intéressants : un coût fixe (construction) modélisé par une variable binaire y_i ainsi qu'une contrainte logique forçant les flots provenant d'un site à être nuls si aucune usine n'est construite en ce site. Notons aussi que certaines variables sont entières alors que d'autres (flots) sont réelles. Un tel programme linéaire est appelé **mixte**.

Contraintes logiques

Des variables binaires peuvent servir à représenter des contraintes logiques. En voici quelques exemples, où p_i représente une proposition logique et x_i la variable logique (binaire) correspondante. Vous pouvez agrandir cette liste par vous-même.

contrainte logique	forme algébrique
$p_1 \oplus p_2 =$ vrai	$x_1 + x_2 = 1$
$p_1 \vee p_2 \vee \ldots \vee p_n =$ vrai	$x_1 + x_2 + \ldots + x_n \geq 1$
$p_1 \wedge p_2 \wedge \ldots \wedge p_n =$ vrai	$x_1 + x_2 + \ldots + x_n \geq n$ (ou $= n$)
$p_1 \Rightarrow p_2$	$x_2 \geq x_1$
$p_1 \Leftrightarrow p_2$	$x_2 = x_1$

Fonctions linéaires par morceaux

Considérons une fonction objectif à *maximiser* de la forme illustrée à la figure ci-contre. Dans chaque intervalle $[a_{i-1}, a_i]$ la fonction est linéaire, ce qu'on peut exprimer par :

$$\begin{aligned}
& x = \lambda_{i-1} a_{i-1} + \lambda_i a_i \\
& \lambda_{i-1} + \lambda_i = 1 \\
& \lambda_{i-1}, \lambda_i \geq 0 \\
& f(x) = \lambda_{i-1} f(a_{i-1}) + \lambda_i f(a_i).
\end{aligned}$$

On peut généraliser cette formule sur tout l'intervalle de définition de la fonction f en contraignant les variables λ_i à ne prendre que deux valeurs non nulles, et ce pour deux indices consécutifs. Ceci se fait en introduisant des variables binaires y_i associées aux intervalles de linéarité $[a_{i-1}, a_i]$ (voir figure 3) :

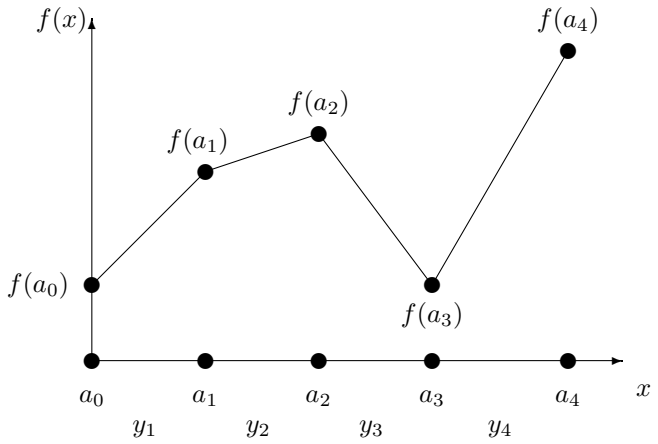


FIGURE 3 – Fonction linéaire par morceaux

$$
\begin{aligned}
x &= \sum_{i=0}^{n} \lambda_i a_i \\
f(x) &= \sum_{i=0}^{n} \lambda_i f(a_i) \\
\sum_{i=0}^{n} \lambda_i &= 1 \\
\lambda_i &\geq 0 \qquad i = 0, \ldots, n \\
\lambda_0 &\leq y_1 \\
\lambda_1 &\leq y_1 + y_2 \\
&\vdots \quad \vdots \quad \vdots \\
\lambda_{n-1} &\leq y_{n-1} + y_n \\
\lambda_n &\leq y_n \\
\sum_{i=1}^{n} y_i &= 1 \qquad \text{(un seul intervalle « actif »)} \\
y_i &\in \{0,1\} \qquad i = 1, \ldots, n.
\end{aligned}
$$

Si la fonction f est concave, les variables binaires y_i et les contraintes associées peuvent être éliminées de la formulation. Cette approche est particulièrement intéressante lorsqu'on optimise des fonctions de plusieurs variables de la forme $\sum_i f_i(x_i)$.

Une première stratégie de résolution : l'énumération implicite

Considérons un programme mathématique mixte. Par paresse, il est naturel de laisser de côté, dans un premier temps, les contraintes d'intégralité, et de se contenter de résoudre un programme linéaire (**relaxation linéaire** notée RL) du programme en variables entières). Si la solution optimale de ce programme satisfait aux contraintes d'intégralité, alors cette solution est aussi solution optimale du programme avec variables entières. Sinon, il doit exister au moins une variable x_j dont la valeur α est fractionnaire. On sépare alors le problème en deux : un sous-problème contiendra la contrainte $x_j \leq \lfloor \alpha \rfloor$ et le second la contrainte $x_j \geq \lceil \alpha \rceil = \lfloor \alpha \rfloor + 1$. Il est clair que ceci crée une partition du problème relaxé.[2] On répète

2. Bon, je sais que cette description n'est pas très rigoureuse, mais je pense qu'elle est compréhensible.

le processus pour chacun des sous-problèmes. Cette procédure est habituellement représentée sous forme d'un arbre binaire où, à chaque niveau, une partition du sommet père s'effectue suivant la règle décrite précédemment. Il s'agit alors de parcourir cet arbre d'énumération afin d'y trouver la solution optimale.

L'exploration d'un chemin de l'arbre peut prendre fin pour trois raisons :
- la solution devient entière ;
- le domaine admissible d'un sous-problème devient vide ;
- la valeur de l'objectif correspondant à la solution optimale du problème relaxé est inférieure (moins bonne) à celle d'une solution admissible connue, possiblement obtenue à un autre sommet de l'arbre.

Dans chacun de ces trois cas on dit que le sommet est **sondé**, et il est inutile de pousser plus loin dans cette direction. L'algorithme s'arrête lorsque tous les sommets sont sondés. La meilleure solution obtenue au cours du déroulement de l'algorithme est alors l'optimum *global* de notre problème.

Cette technique d'énumération partielle est connue sous le vocable de **Branch-and-Bound**. Appliquons-la au problème du cambrioleur (voir figure 4). Dans la figure, la solution optimale de chaque relaxation linéaire est indiquée sous le nœud correspondant. Cette solution est obtenue en favorisant les variables de rapport qualité/poids élevé tout en respectant les contraintes rajoutées sur les branches de l'arbre d'énumération.

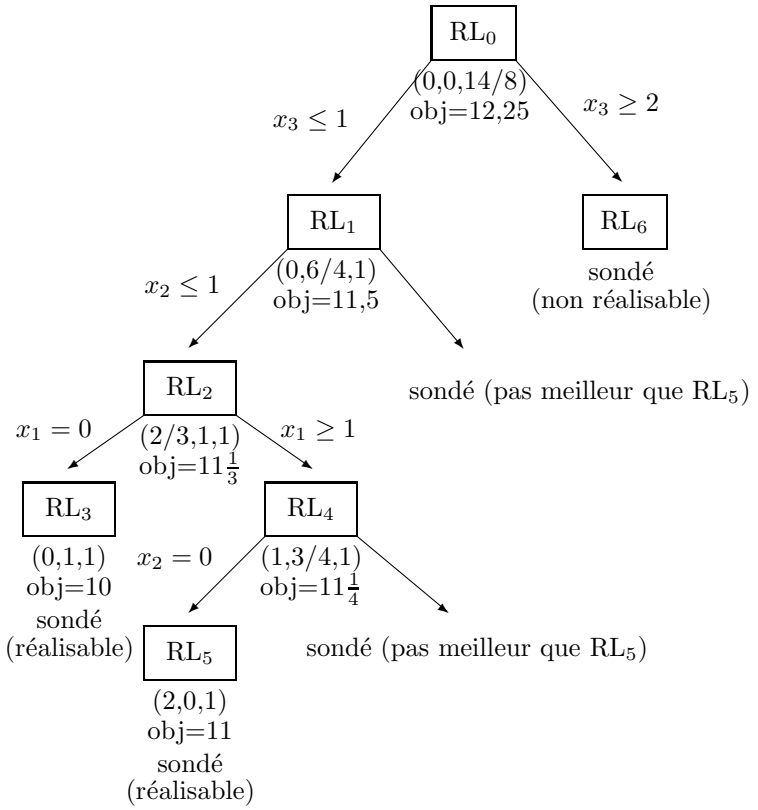


FIGURE 4 – La procédure de Branch-and-Bound

Plusieurs remarques sont de rigueur :

1. Dans l'exemple, l'algorithme ne semble pas plus efficace que l'énumération complète. En effet il y a 17 solutions admissibles (vérifiez-le !), dont seulement 6 sont efficaces, alors que l'arbre de Branch-and-Bound possède 7 sommets et exige une certaine quantité de travail pour traiter chaque sommet. Sur des problèmes de grande taille, par contre, le nombre de sommets visités est en général beaucoup plus faible que le nombre de solutions réalisables.

2. On peut tirer profit de la structure du problème pour obtenir des informations additionnelles sur la valeur des variables. Ainsi, il est clair que x_3 ne peut excéder 1. On peut alors remplacer une contrainte de la forme $x_3 \geq 1$ par $x_3 = 1$, ce qui fixe la valeur de x_3 et l'élimine du sous-problème.
3. Le programme linéaire associé à un sommet ne diffère du problème père que par une contrainte. Il est alors judicieux de résoudre ce problème par l'algorithme dual du simplexe, c'est-à-dire l'algorithme du simplexe appliqué au problème dual.
4. L'ordre dans lequel l'arbre est exploré est important. Si RL_5 n'avait pas été visité avant le descendant droit de RL_2, celui-ci n'aurait pu être sondé aussi tôt.
5. La solution optimale est obtenue au nœud RL_5. Il faut cependant pousser l'exploration un plus loin pour démontrer que cette solution est bien optimale. [3] Dans plusieurs applications, on perd beaucoup plus de temps à démontrer qu'une solution est optimale qu'à la trouver !
6. Si l'objectif du problème linéaire relaxé vaut z, la solution entière ne peut valoir plus que $\lfloor z \rfloor$ puisque les coefficients de l'objectif sont entiers. Ceci ne serait pas le cas si les coefficients étaient fractionnaires.
7. Deux stratégies d'exploration sont utilisées en pratique : fouille en profondeur (c'est celle utilisée dans l'exemple) ou exploration prioritaire des sommets correspondant aux meilleures valeurs de l'objectif. Cette dernière stratégie a l'avantage de limiter le nombre de sommets visités. Par contre, elle ne permet pas de tirer profit de la structure du problème père, à moins de garder en mémoire les dictionnaires optimaux de tous les sommets, ce qui est coûteux.
8. Au sommet initial, aucune solution admissible n'est encore disponible. On pourrait en obtenir à l'aide d'algorithmes heuristiques. Dans le problème du cambrioleur, on pourrait par exemple choisir les variables dans l'ordre décroissant de leur rapport qualité-prix, en exigeant que les variables soient entières. Ceci donnerait la solution $x = (0, 1, 1)$ de valeur 10. Cette valeur pourrait servir à sonder de futurs sommets. Puisque les sous-problèmes de chaque sommet possèdent la structure du problème de départ, cette technique pourrait être appliquée à chaque sommet de l'arbre.
9. De façon surprenante, la méthode de Branch-and-Bound fonctionne souvent bien. Malgré la taille théoriquement énorme de l'arbre d'énumération, on parvient fréquemment à une solution optimale ou quasi-optimale après avoir exploré quelques dizaines de sommets.
10. La méthode peut facilement être adaptée pour trouver *toutes* les solutions optimales du problème.

Une seconde stratégie : les hyperplans coupants

Considérons le programme mathématique

$$\begin{array}{ll} \max & 4x_1 + \frac{5}{2}x_2 \\ & x_1 + x_2 \leq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1, x_2 \text{ entiers non négatifs.} \end{array}$$

Le dictionnaire optimal correspondant à la relaxation linéaire de ce programme contient les deux contraintes

$$\begin{aligned} x_1 &= \frac{15}{4} + \frac{5}{4}x_3 - \frac{1}{4}x_4 \\ x_2 &= \frac{9}{4} - \frac{9}{4}x_3 + \frac{1}{4}x_4, \end{aligned}$$

où x_3 et x_4 sont des variables d'écart. Puisque la variable de base x_1 n'est pas entière, cette solution de base n'est pas admissible. On réécrit la première contrainte sous la forme

$$x_1 - \frac{5}{4}x_3 + \frac{1}{4}x_4 = \frac{15}{4}.$$

3. Encore que, dans le problème du cambrioleur, il est clair que la partie de l'arbre située à droite du sommet initial RL_0 ne sera jamais admissible.

En utilisant l'identité

$$a = \lfloor a \rfloor + (a - \lfloor a \rfloor),$$

où $a - \lfloor a \rfloor$ représente la partie fractionnaire de a ($0 \leq a - \lfloor a \rfloor < 1$), on obtient

$$x_1 + \left(\left\lfloor -\frac{5}{4} \right\rfloor + \frac{3}{4} \right) x_3 + \left(\left\lfloor \frac{1}{4} \right\rfloor + \frac{1}{4} \right) x_4 = \left(\left\lfloor \frac{15}{4} \right\rfloor + \frac{3}{4} \right),$$

c'est-à-dire, en mettant tous les coefficients entiers à gauche et les coefficients fractionnaires à droite :

$$x_1 - 2x_3 - 3 = \frac{3}{4} - \frac{3}{4}x_3 - \frac{1}{4}x_4.$$

Puisque les variables x_3 et x_4 sont non négatives, la partie fractionnaire (constante du membre de droite) est inférieure à 1, le membre de droite est strictement inférieur à 1. Puisque le membre de gauche est entier, le membre de droite doit aussi être entier. Or un entier inférieur à 1 doit être inférieur *ou* égal à zéro. On en déduit une contrainte additionnelle qui doit être satisfaite par toute solution admissible du problème originel, et que ne satisfait pas la solution de base courante :

$$\frac{3}{4} - \frac{3}{4}x_3 - \frac{1}{4}x_4 \leq 0.$$

En utilisant les identités $x_3 = 6 - x_1 - x_2$ et $x_4 = 45 - 9x_1 - 5x_2$ on obtient la coupe sous sa forme géométrique :

$$3x_1 + 2x_2 \leq 15.$$

Cette contrainte linéaire rend inadmissible la solution courante inadmissible [4] , sans éliminer aucune autre solution entière (voir figure 5). Si la solution du nouveau problème est entière, c'est la solution optimale de notre problème. Sinon, on construit une nouvelle coupe et on recommence.

Noter qu'il est possible d'obtenir une seconde coupe en utilisant le plafond plutôt que le plancher. Les détails sont laissés au lecteur.

La résolution du nouveau programme linéaire s'effectue, comme dans l'algorithme de Branch-and-Bound, à l'aide de l'algorithme du simplexe appliqué au problème dual, afin de tirer profit de la solution duale optimale obtenue précédemment.

Cette technique seule ne permet pas de résoudre des problèmes de grande taille. Cependant elle peut être combinée de façon efficace avec la technique de Branch-and-Bound.

4. Dans la solution courante, les variables hors-base x_3 et x_4 sont nulles, ce qui donne $3/4 \leq 0$!

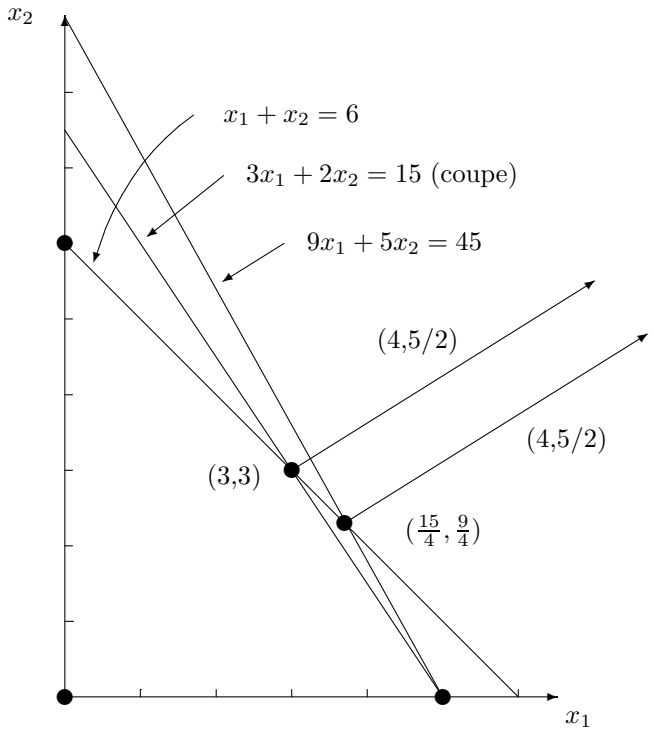


FIGURE 5 – Une droite coupante

CHAPITRE 6 : RÉSEAUX

19 novembre 2003

Les graphes sont un outil puissant de modélisation. Ils interviennent naturellement dans la représentation de réseaux de transport ou de télécommunication, par exemple, mais également dans la représentation de structures relationnelles abstraites. Mathématiquement, un graphe G prend la forme $G = (N, A)$ où N est un ensemble de **sommets** et $A \subset N \times N$ un ensemble d'**arcs**. Le terme **réseau** est un terme générique désignant un graphe dont les sommets ou arcs possèdent des attributs : coûts, capacités, longueurs, etc. Ce qui est amusant avec les réseaux, c'est qu'ils permettent de visualiser des situations abstraites. [1]

Dans ce chapitre nous allons étudier quatre problèmes classiques de la théorie des réseaux. Dans le texte, les termes «distance», «coût» et «longueur» sont synonymes.

1. Le problème de flot à coût minimum

Un très grand nombre de programmes linéaires possèdent une composante «réseau» très importante. Nous allons étudier dans cette section le problème consistant à approvisionner, au moindre coût, des entrepôts à partir des unités de production (usines) d'une compagnie. Les données du problème sont :
– le réseau de transport ;
– la localisation et la capacité de production des usines (sources) ;
– la localisation et la demande des entrepôts (destinations) ;
– les coûts de transport c_{ij} sur les arcs du réseau.
Il reste à déterminer les productions des usines ainsi que les chemins utilisés pour approvisionner les entrepôts. Désignons par b_i la **demande** associée au sommet i. Par convention, un sommet d'**offre** (usine) possède une demande négative et un sommet de **passage** (ni usine ni entrepôt) possède une demande nulle. Pour simplifier, nous supposerons que l'offre est égale à la demande, c'est-à-dire : $\sum_i b_i = 0$.

Il semble naturel de formuler ce problème en terme de flots sur les chemins reliant les sources et les destinations du résau. Or, le nombre de chemins étant astronomique, il est plus simple d'utiliser les flots sur les arcs comme variables, quitte à récupérer les flots de chemins par la suite.

Soit x_{ij} le flot sur l'arc (i, j). De façon naturelle, les flots doivent être non négatifs. Ils doivent également satisfaire aux **conditions de conservation** qui stipulent que le flot ne peut être créé qu'aux sources et absorbé aux destinations. En tout autre sommet du réseau, la flot entrant doit correspondre au flot sortant. (voir figure 1). Le PFCM prend ainsi la forme du programme linéaire

$$\min_{x \in X} cx$$

où

$$X = \{x \geq 0 \,:\, \sum_j x_{ji} - \sum_j x_{ij} = b_i,\ \forall i \in N\} = \{x \geq 0 \,:\, Ax = b\}.$$

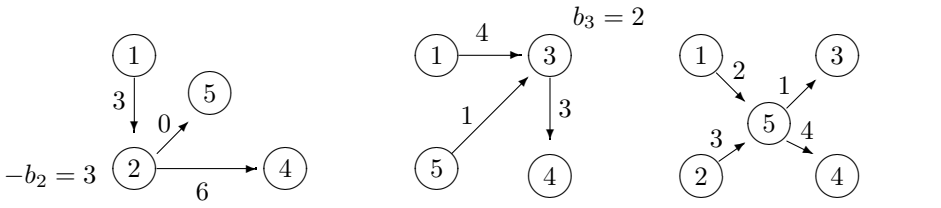


FIGURE 1 – Conservation du flot (offre, demande, passage)

1. Pour obtenir une liste des termes les plus fréquents de la théorie des graphes, vous pouvez consulter mes notes du cours IFT 1063.

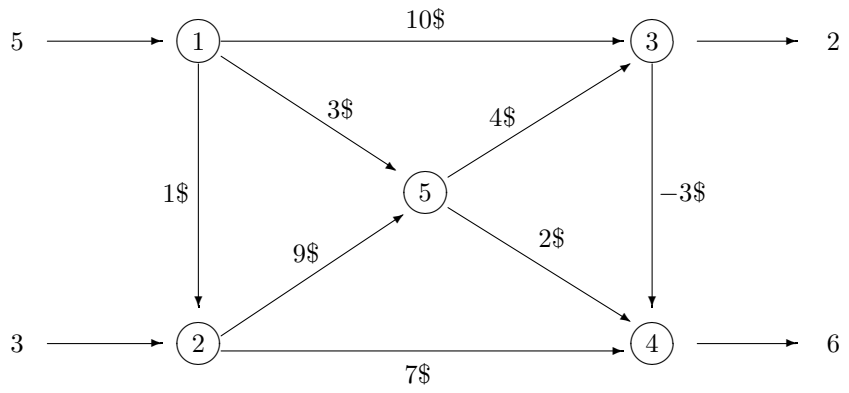


FIGURE 2 – Exemple de PFCM

Ci-dessous vous trouverez toutes les données de l'exemple que nous allons tenter de résoudre (voir figure 2). La matrice A représente la **matrice d'incidence sommets-arcs** du graphe. Chaque colonne décrit un arc alors que la ligne i contient l'information pertinente au sommet i.

$$A = \begin{bmatrix} -1 & -1 & -1 & & & & & \\ +1 & & & -1 & -1 & & & \\ & +1 & & & & -1 & +1 & \\ & & & +1 & & +1 & & +1 \\ & & +1 & & +1 & & -1 & -1 \end{bmatrix}$$

$$b = \begin{bmatrix} -5 \\ -3 \\ +2 \\ +6 \\ 0 \end{bmatrix} \qquad c = [1, 10, 3, 7, 9, -3, 4, 2].$$

Ce programme étant linéaire, il est naturel de lui appliquer l'algorithme du simplexe. Celui-ci prend une forme très particulière sur les réseaux. Sans entrer dans les détails techniques, mentionnons qu'un point extrémal (solution de base) est obtenu en affectant le flot uniquement sur un arbre du réseau touchant à tous les sommets. À chaque arbre correspond une et une seule affectation du flot qui satisfait aux offres et aux demandes du problème. Par exemple, sur l'arbre de la figure 3, le flot sur les arcs est obtenu en parcourant les arcs dans l'ordre inverse de leur niveau dans l'arbre (ordre topologique inverse, c'est-à-dire en allant des feuilles vers la racine). [2] Le coût de cette solution initiale est de 65$.

On introduit, à chaque sommet i, un nombre y_i représentant la longueur de l'unique chemin allant de la racine au sommet i dans l'arbre. Si un arc est utilisé à rebours, son coût change de signe. La distance entre deux sommets i et j du réseau, en passant par l'arbre, est alors $y_j - y_i$. Par exemple, la distance de 5 à 3 peut être obtenue de 2 façons (voir figure 4) :

$$\begin{array}{rcll} \text{distance} & = & y_3 - y_5 & \text{calcul direct} \\ & = & -c_{51} + c_{13} & \text{en passant par le chemin } 5 - 1 - 3. \end{array}$$

Les variables associées aux arcs de l'arbre sont les variables de base. Suivant la philosophie de l'algorithme du simplexe, l'on se pose la question suivante : que se passe-t-il si l'on augmente la valeur d'une variable hors-base, par exemple la variable x_{53} ? L'objectif diminuera si le coût de l'arc $(5, 3)$ (coût direct) est inférieur au coût obtenu en passant par l'arbre pour se rendre de 5 à 3. Le coût «dans l'arbre» étant égal à $y_3 - y_5 = 10 - 3 = 7$, il est avantageux d'utiliser l'arc (5,3) dont le coût est $4 < 7$. De façon générale, l'augmentation d'une variable hors-base fera diminuer l'objectif si et seulement si

$$c_{ij} < y_j - y_i,$$

2. La racine de l'arbre est choisie de façon arbitraire (sommet 1 dans l'exemple).

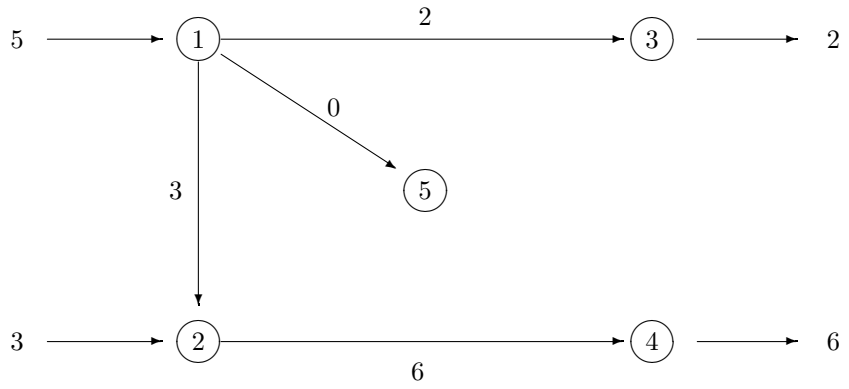


FIGURE 3 – Solution de base (point extrémal)

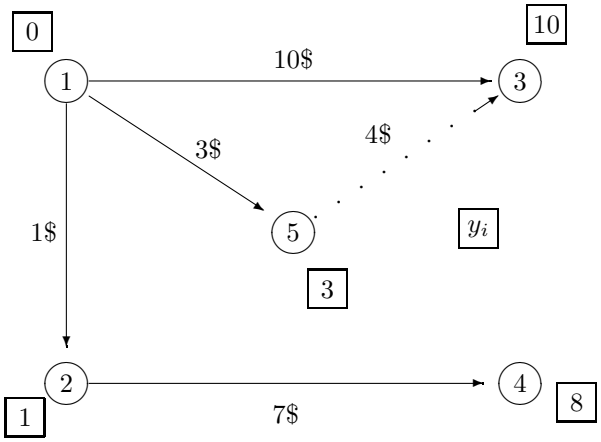


FIGURE 4 – Distance entre deux sommets dans l'arbre

c'est-à-dire

$$\bar{c}_{ij} = c_{ij} + y_i - y_j < 0.$$

Le nombre $\bar{c}_{ij}$ est le **coût réduit** associé à l'arc (variable) hors-base (i, j). Dans notre cas :

$$\bar{c}_{53} = 4\$ + 3\$ - 10\$ = -3\$.$$

En augmentant le flot d'une quantité ϵ sur l'arc (5,3), il est nécessaire d'effectuer certains ajustements pour maintenir la conservation du flot. Ceci ne peut se faire que d'une seule façon : en diminuant le flot d'une quantité ϵ sur le chemin allant de 5 à 3 *dans l'arbre*, tout en tenant compte du sens des arcs (une diminution du flot d'un arc parcouru dans le mauvais sens correspond à une augmentation du flot).

Mais voyons les choses de façon légèrement différentes. En introduisant l'arc (5,3), on crée un **cycle** unique $(5 - 3 - 1 - 5)$ dans le réseau. En modifiant le flot par une même quantité (positive pour les arcs dans le même sens que (5,3) et négative pour les autres) sur tous les arcs du cycle, on obtient un flot admissible, à la condition que tous les flots **demeurent non négatifs**. On augmente donc le flot x_{53} de la plus grande quantité ϵ qui satisfasse à cette dernière contrainte. Lorsque x_{53} prend la valeur ϵ, les flots du circuit prennent les valeurs respectives $x_{13} = 2 - \epsilon$ (sens contraire) et $x_{15} = 0 + \epsilon$ (bon sens). Le nombre ϵ ne peut donc dépasser la valeur 2. Pour $\epsilon = 2$, la variable x_{53} intègre la base où elle prend la valeur 2, la variable x_{13} quitte la base, et la variable de base x_{15} voit sa valeur passer de 0 à 2 (voir figure 5). Le résultat, comme vous pouvez le constater, est une nouvelle solution de base (arbre).

La règle générale déterminant simultanément la valeur maximale de ϵ et la variable quittant la base est :

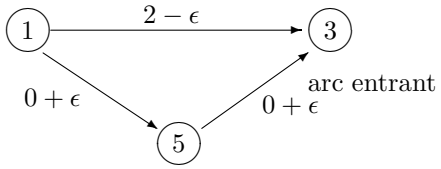


FIGURE 5 – Modification du flot dans le cycle

$$\epsilon = \min_{(k,l)\in \text{cycle}} \{x_{kl} \,:\, (k,l) \text{ est dans le sens inverse de l'arc entrant}\}.$$

L'opération décrite ci-dessus correspond à un changement de base dans l'algorithme du simplexe. Le nouvel objectif est obtenu en additionnant à l'ancien objectif le produit du coût réduit par la modification du flot sur l'arc entrant (ϵ) :

$$\text{nouvel objectif} = 65\$ + (-3\$) \times 2 = 59\$.$$

Une fois le pivot effectué, on met à jour les nombres y_i [3] et on se remet à la recherche d'une variable de coût réduit $\bar{c}_{ij} < 0$. On découvre (voir figure 6) :

$$\bar{c}_{34} = c_{34} + y_3 - y_4 = -3\$ + 7\$ - 8\$ = -4\$ < 0.$$

En parcourant le cycle dont le premier arc est (3,4), on réalise (voir figure 6) que la valeur maximale de ϵ est

$$\min\{6 - \epsilon, 3 - \epsilon\} = 3.$$

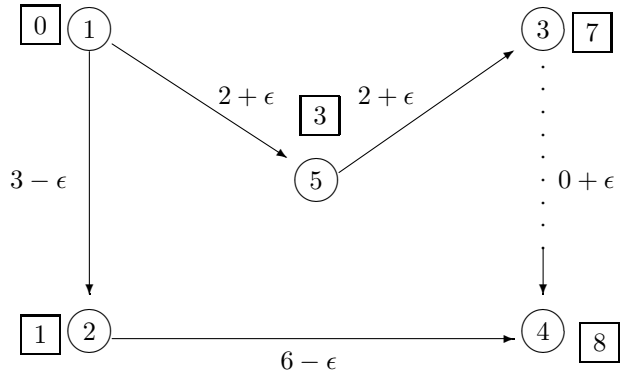


FIGURE 6 – Deuxième itération

Le nouvel objectif est égal à

$$59\$ + \bar{c}_{34}\epsilon = 59\$ - 4\$ \times 3 = 47\$.$$

On met à jour l'arbre, on réévalue les nombres y_i et on obtient l'arbre de la figure 7. Les coûts réduits des variables hors base sont maintenant :

$$\begin{array}{lcccccl} \bar{c}_{12} & = & 1\$ + 0\$ - (-3\$) & = & 4\$ & \geq & 0 \\ \bar{c}_{13} & = & 10\$ + 0\$ - 7\$ & = & 3\$ & \geq & 0 \\ \bar{c}_{25} & = & 9\$ + (-3\$) - 3\$ & = & 3\$ & \geq & 0. \end{array}$$

Puisque ceux-ci sont tous positifs ou nuls, la solution est optimale.

3. Si on utilise le sommet 5 plutôt que le sommet 1 comme racine on obtient : $y_5 = 0$, $y_1 = y_5 - c_{15}$ (l'arc (1,5) est emprunté à contre-sens), $y_2 = y_1 + c_{12} = -2$, $y_3 = 4$ et $y_4 = 5$. On vérifie que les coûts réduits des variables hors-base ne sont pas modifiés car tous les y_i ont diminué d'une *même* quantité 3.

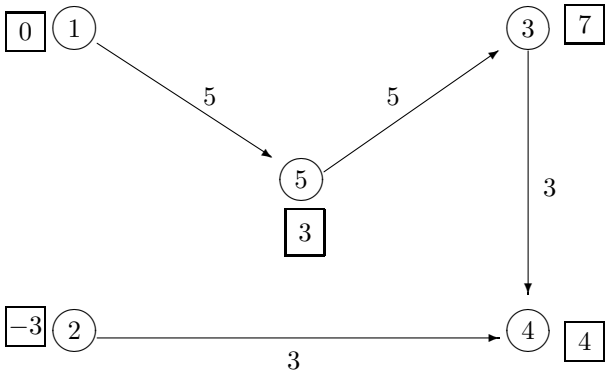


FIGURE 7 – Solution optimale

Remarques

1. Uniquement la différence $y_i - y_j$ importe. On pourrait ajouter une constante à tous les nombres y_i sans changer quoi que ce soit à l'algorithme. C'est ce qui se passe lorsqu'on change de racine.
2. Le choix de la racine d'un arbre est arbitraire. Les seules considérations sont d'ordre informatique. En effet, il est parfois utile, pour minimiser les modifications lors du pivot, de changer de racine. Ceci implique des manipulations non triviales des structures de données associées au graphe
3. En cas d'égalité, le choix de la variable sortante est arbitraire. La solution de base subséquente est alors dégénérée.
4. Si, à une itération donnée, les variables du cycle augmentent toutes, alors le problème est non borné.
5. Pour trouver une solution de base initiale, on peut utiliser la méthode dite de la phase I, qui consiste à mettre tout le flot sur des arcs artificiels liant les sources aux destinations. Les arcs artificiels ont un coût de 1 alors que les autres arcs ont un coût nul. La technique dite du «grand M» associerait aux arcs artificiels un coût de M alors que les autres arcs conserveraient leur coût originel.
6. En pratique, les arcs possèdent fréquemment des bornes supérieures sur le flot qu'ils peuvent supporter. Dans ce cas, on utilise une variante de la méthode du simplexe, dite «simplexe avec bornes sur les variables» où une variable est hors-base lorsque sa valeur est égale soit à zéro, soit à sa borne supérieure. Le choix des variables d'entrée est légèrement modifié pour tenir compte de cette nouvelle définition.
7. Il est facile de trouver des itinéraires des flots (chemins) qui soient compatibles avec les flots sur les arcs obtenus par l'algorithme (exercice).

2. Flot maximum dans un réseau

Ce problème ultra classique consiste à maximiser le flot d'une source (origine) s à une destination t dans un réseau dont les arcs sont munis de capacités u_{ij} (voir figure 8). Il s'exprime comme un programme linéaire similaire à celui du PFCM :

$$\begin{aligned} \max_{v,x} \quad & v \\ & \sum_j x_{ji} - \sum_j x_{ij} = \begin{cases} -v & \text{si } i = s \\ v & \text{si } i = t \\ 0 & \text{sinon} \end{cases} \\ & 0 \le x_{ij} \le u_{ij}. \end{aligned}$$

En fait, il est possible de formuler ce problème comme un PFCM. Cependant, nous allons présenter un algorithme spécialement adapté au problème du flot maximum.

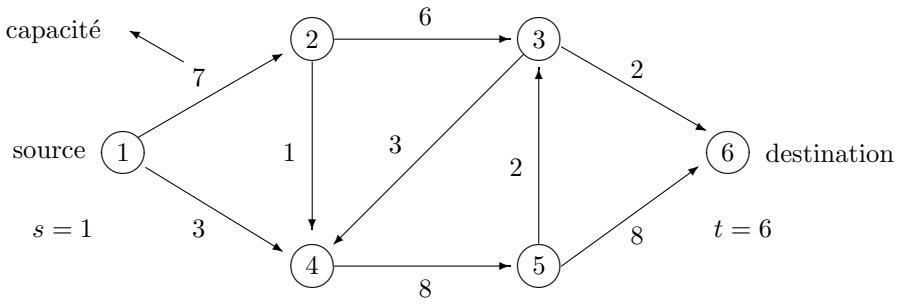


FIGURE 8 – Un problème de flot maximal dans un réseau

Pour résoudre le problème, il est naturel de trouver un chemin non saturé et d'y pousser le plus grand flot possible. Ceci s'effectue par une procédure d'**étiquetage**. Un sommet i est étiqueté s'il existe un chemin non saturé menant de la source s au sommet i. Soit S l'ensemble des sommets étiquetés ($s \in S$ par définition) et T son complément. Si $(i, j) \in S \times T$ et que $x_{ij} < u_{ij}$, on retire le sommet j de T pour l'intégrer à S, en conservant l'information sur le prédécesseur (sommet i) afin de pouvoir retracer le chemin. La procédure d'étiquetage se termine si soit
– t est étiqueté ;
– il n'y a plus aucun sommet à étiqueter et $t \in T$.
Dans le deuxième cas, l'algorithme se termine. Dans le premier cas, on a découvert un chemin

$$p = (i_0 = s, i_1, \ldots, i_{l-1}, i_l = t)$$

de capacité résiduelle positive. L'augmentation maximale de flot sur ce chemin est égale à

$$\epsilon = \min_{1 \leq k \leq l} \{u_{i_{k-1} i_k} - x_{i_{k-1} i_k}\}.$$

La figure 9 illustre l'algorithme. Au départ, le flot est nul. Puis, par la procédure d'étiquetage, on trouve le chemin d'augmentation $1-2-5-6$ dont la capacité est 4. On augmente le flot de 4 unités le long de ce chemin et on recommence. À la deuxième itération, on trouve le chemin $1-4-5-6$ de capacité résiduelle 1. On augmente le flot de une unité et on s'arrête car plus aucun ne chemin ne posséde de capacité résiduelle positive.

Malheureusement, la solution obtenue par cette procédure n'est pas optimale. Il est facile de vérifier qu'en affectant un flot de 2 sur chacun des chemins $1-2-3-6$, $1-4-5-6$ et $1-2-5-6$ on obtient un flot de 6 unités. Une modification de l'algorithme s'impose donc. Nous allons permettre d'étiqueter un sommet en utilisant un arc dans le *sens inverse*, sous la condition que le flot sur cet arc est strictement positif. On peut en effet, comme dans la méthode du simplexe de la section précédente, maintenir la conservation de flot en *diminuant* le flot des arcs utilisés à rebours. Grâce à cette astuce, on identifie le chemin d'augmentation $1-4-5-2-3-6$ dans la figure 9. Le sommet 2 est étiqueté à partir du sommet 5 et de l'arc (2,5). La règle permettant de déterminer l'augmentation maximale ϵ doit être modifiée pour interdire à un flot qui diminue de devenir négatif. Soit p^+ l'ensemble des arcs du chemin parcourus dans le sens naturel et p^- l'ensemble des arcs du chemin parcourus dans le sens inverse. On a

$$\begin{aligned} \epsilon_1 &= \min_{(i,j) \in p^+} \{u_{ij} - x_{ij}\} \\ \epsilon_2 &= \min_{(i,j) \in p^-} \{x_{ij}\} \\ \epsilon &= \min\{\epsilon_1, \epsilon_2\}. \end{aligned}$$

Le flot optimal est maintenant tel qu'illustré à la figure 10.

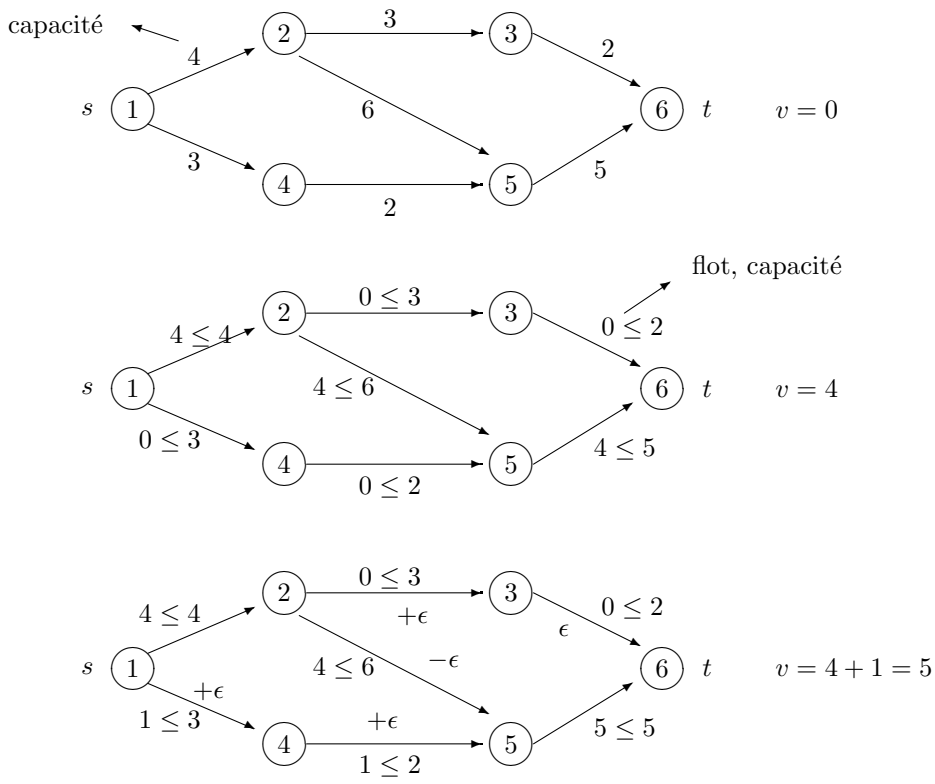


FIGURE 9 – Un problème de flot maximal dans un réseau

Remarques

1. On peut associer à chaque sommet j une étiquette f_j représentant une augmentation possible du flot jusqu'à j. On initialise y_s à $= +\infty$ et y_j à $-\infty$ $j \neq s$. Lors du balayage du sommet i, on révise les étiquettes des voisins de i :
   – $y_j = \max\{y_j, \min\{y_i, u_{ij} - x_{ij}\}\}$ si l'arc (i, j) existe ;
   – $y_j = \max\{y_j, \min\{y_i, x_{ji}\}\}$ si l'arc (j, i) existe.
   Si l'étiquette est modifiée, on note le sommet prédécesseur i afin de pouvoir retracer le chemin.

   Ce procédé permet d'identifier le chemin d'augmentation ayant la plus grande capacité résiduelle. Cette stratégie gloutonne n'est pas toujours la meilleure et peut même ne pas converger si les capacités ne sont pas entières ! Une stratégie qui converge dans tous les cas consiste à utiliser un chemin d'augmentation possédant le plus petit nombre d'arcs.
2. Soit S l'ensemble des sommets étiquetés lorsque l'algorithme se termine. Puisque la solution est optimale, $t \notin S$, c'est-à-dire : $t \in T$. Définissons :

$$C = A \cap (S \times T).$$

   Puisque $s \in S$ et $t \in T$, l'ensemble C n'est pas vide. Cet ensemble, que l'on appelle une **coupe**, déconnecte la source s de la destination t. Il s'ensuit que la somme des capacités des arcs de C est une borne supérieure sur le flot maximal. À l'optimum, cette borne est égale à la valeur maximale du flot, et on nomme la coupe correspondante **coupe minimale** (voir figure 11). Un théorème célèbre, dû à Ford et Fulkerson, affirme que le valeur maximale du flot est égale à la capacité minimale d'une coupe (MAX FLOT = MIN COUPE). Ce résultat établit l'égalité des objectifs primal et dual lorsqu'il existe une solution optimale.

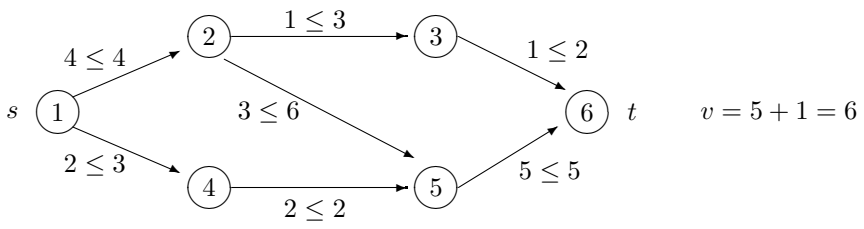


FIGURE 10 – Le flot optimal

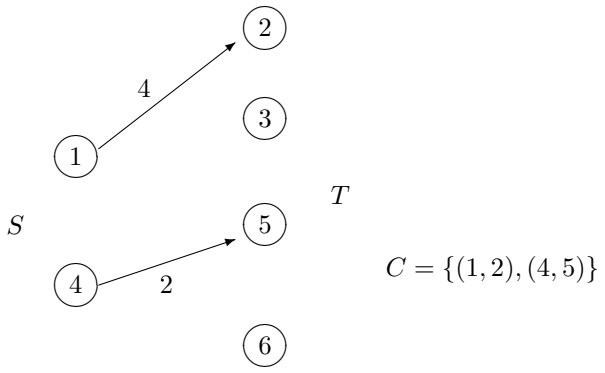


FIGURE 11 – Une coupe minimale

3. Les plus courts chemins

Le problème de la recherche d'un plus court chemin entre deux sommets s et t d'un réseau est l'un des plus étudiés en recherche opérationnelle. Il peut s'exprimer comme un programme linéaire en variables entières :

$$\min_{x} \quad \sum_{(i,j)\in A} c_{ij} x_{ij}$$

$$\sum_{j} x_{ji} - \sum_{j} x_{ij} = \begin{cases} -1 & \text{si } i = s \\ +1 & \text{si } i = t \\ 0 & \text{sinon} \end{cases}$$

$$\sum_{(i,j)\in A} x_{ij} = 1$$

+ contraintes assurant que le graphe est connexe (pas facile)

Dans le cas où il n'existe pas de circuit dont la somme des longueurs d'arc est négative, on peut ignorer la contrainte $\sum_{(i,j)\in A} x_{ij} = 1$ et remplacer les contraintes $x_{ij} \in \{0, 1\}$ par $x_{ij} \geq 0$, tout simplement.

En pratique, il est (presque) aussi simple de calculer les plus courts chemins de s vers tous les autres sommets de N que de s vers t. De plus, comme la solution de ce problème plus général est un arbre de plus courts chemins et donc une solution de base, les contraintes d'intégralité sur les variables ne sont pas requises. On peut donc appliquer l'algorithme du simplexe à ce cas particulier de PFCM. Nous allons plutôt présenter un schéma algorithmique combinatoire très général qui partage certains points communs avec la méthode du simplexe. Ce schéma attribue à chaque sommet i une étiquette y_i qui, à la fin de l'algorithme, correspond à la longueur d'un plus court chemin de s à i. Afin de retracer les plus courts chemins, on met à jour le vecteur des prédécesseurs *pred*.

ALGORITHME GÉNÉRIQUE

1. $B \leftarrow \{s\}$ $y_s \leftarrow 0$ $y_i \leftarrow +\infty \quad \forall i \neq s$
2. **si** $B = \emptyset$ STOP **sinon** choisir $i \in B$
3. BALAYER LE SOMMET i.
   $\forall (i,j) \in A$ faire :
   si $y_i + c_{ij} < y_j$ **alors**
   - $y_j \leftarrow y_i + c_{ij}$
   - $B \leftarrow B \cup \{j\}$
   - $pred(j) \leftarrow i$
4. Retirer i de B et retourner à 2.

Appliquons l'algorithme à l'exemple de la figure 12.

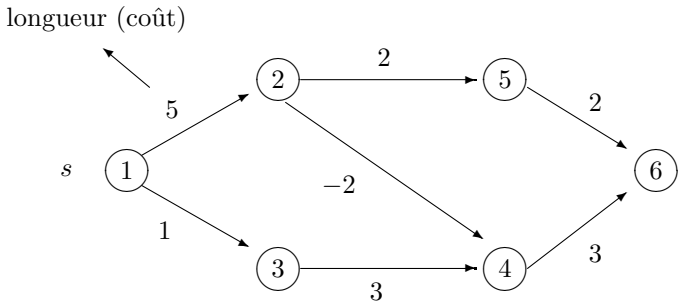


FIGURE 12 – Réseau (plus court chemins)

- choisir le sommet 1 et le balayer
  $B \leftarrow \{1\}$ $\quad y \leftarrow \{0, \infty, \infty, \infty, \infty, \infty\}$
- balayer 1
  $y_1 + c_{12} = 0 + 5 < y_2 = \infty \Rightarrow B \leftarrow \{1,2\}$, $y_2 \leftarrow 5$, $pred(2) \leftarrow 1$
  $y_1 + c_{13} = 0 + 1 < y_3 = \infty \Rightarrow B \leftarrow \{1,2,3\}$, $y_3 \leftarrow 1$, $pred(3) \leftarrow 1$
  $B \leftarrow B - \{1\} = \{2,3\}$
- choisir le sommet 3 et le balayer
  $y \leftarrow (0,5,1,4,\infty,\infty)$, $pred \leftarrow (-,1,1,3,-,-)$, $B \leftarrow \{2,4\}$
- choisir le sommet 4 et le balayer
  $y \leftarrow (0,5,1,4,\infty,7)$, $pred \leftarrow (-,1,1,3,-,4)$, $B \leftarrow \{2,6\}$
- choisir le sommet 2 et le balayer
  $y \leftarrow (0,5,1,3,7,7)$, $pred \leftarrow (-,1,1,2,2,4)$, $B \leftarrow \{4,5,6\}$
  (réétiquetage du sommet 3)
- choisir le sommet 4 et le balayer
  $y \leftarrow (0,5,1,3,7,6)$, $pred \leftarrow (-,1,1,2,2,4)$, $B \leftarrow \{5,6\}$
  (réétiquetage du sommet 6)
- choisir le sommet 5 et le balayer
  $y \leftarrow (0,5,1,3,7,6)$, $pred \leftarrow (-,1,1,2,2,4)$, $B \leftarrow \{6\}$
  (aucune modification d'étiquettes ou de prédécesseurs)
- choisir le sommet 6 et le balayer
  $y \leftarrow (0,5,1,3,7,6)$, $pred \leftarrow (-,1,1,2,2,4)$, $B \leftarrow \emptyset$ (fin de l'algorithme)

Remarques

1. Le choix du sommet à balayer influence l'efficacité de l'algorithme. Le choix du sommet d'étiquette minimale est celui adopté dans l'algorithme de **Dijkstra**. Si toutes les longueurs d'arc sont non

négatives, alors l'étiquette d'un sommet balayé devient permanente. Ce résultat permet de démontrer facilement que la complexité de la méthode est $O(n^2)$, où n est le nombre de sommets du graphe.

Si les sommets sont choisis suivant l'ordre de leur apparition dans l'ensemble B, alors on obtient l'algorithme de **Ford-Bellman-Moore** dont la complexité est $O(n^3)$ mais qui est souvent plus efficace que la méthode de Dijkstra en pratique.

2. Soit $pred^k(v)$ le $k$ième prédécesseur du sommet v. S'il existe un indice k tel que $pred^k(v) = 0$, on a alors détecté un circuit de longueur négative passant par v. Si les sommets sont balayés dans leur ordre numérique (algorithme de Ford-Bellman-Moore) et qu'une étiquette est améliorée à l'itération $n+1$, il doit forcément exister un circuit négatif.
3. Un algorithme permet de résoudre élégamment le problème qui consiste à trouver les plus courts chemins entre *tous* les couples de sommets d'un réseau. Cet algorithme est inspiré de l'algorithme de Warshall permettant de déterminer la fermeture transitive d'un graphe (revoir les notes du cours IFT 1065). L'algorithme de **Floyd-Warshall** construit les longueurs $w_{ij}^{[k]}$ des plus courts chemins du sommet i au sommet j n'utilisant comme sommets intermédiaires que des sommets de l'ensemble $\{1, 2, \ldots, k\}$. On pose : $w_{ij}^{[0]} = c_{ij}$ et l'on détermine itérativement

$$w_{ij}^{[k]} = \min\{w_{ij}^{[k-1]}, w_{ik}^{[k-1]} + w_{kj}^{[k-1]}\}.$$

Par construction, l'algorithme se termine à l'itération n avec la matrice W des plus courtes distances entre tous les couples de sommets. Si l'on souhaite retracer les chemins, il suffit de mettre à jour simultanément une matrice de prédécesseurs. La complexité de cet algorithme est $O(n^3)$.

4. Tous les livres de recherche opérationnelle traitent de gestion de projet par la technique du «chemin critique». Ce triste chapitre de la recherche opérationnelle m'ennuie profondément. C'est pourquoi je n'en parlerai plus...

4. Arbre sous-tendant de poids minimal

Soit le problème consistant à relier au moindre coût un certain nombre de villes par un réseau de fibre optique, sachant que l'installation des câbles se fera le long des routes du réseau routier. Soit c_{ij} le coût de construction unitaire sur l'arc (i, j). La solution du problème sera certainement un arbre, c'est-à-dire un sous-graphe acyclique comportant $n-1$ arcs. Le «meilleur» arbre sera solution du programme mathématique

$$\begin{aligned} \min_x \quad & \sum_{i,j} c_{ij} x_{ij} \\ & \sum_i \sum_j x_{ij} = n - 1 \\ & \sum_{(i,j) \in S \times S} x_{ij} \leq |S| - 1, \quad \forall S \subseteq N \\ & x_{ij} \in \{0, 1\}, \end{aligned}$$

où $|S|$ représente la cardinalité (nombre d'éléments) de l'ensemble S.

Supposons que nous soyions en possession d'une solution partielle du problème (voir figure 13) qui forme une **forêt** [4] .

On appelle **admissible** un arc qui ne fait pas partie de la forêt et dont l'ajout ne crée pas de cycle. Soit T un arbre de la forêt. Puisque T doit être relié au reste du graphe, il est naturel de considérer l'arc admissible adjacent à T qui soit de coût minimum. À chaque arbre T est ainsi associé un **arc intéressant** indiqué par un arc pointillé dans la figure 13.

4. Une forêt est un graphe acyclique. Une forêt est l'union d'arbres dont certains peuvent ne contenir qu'un seul point et aucun arc.

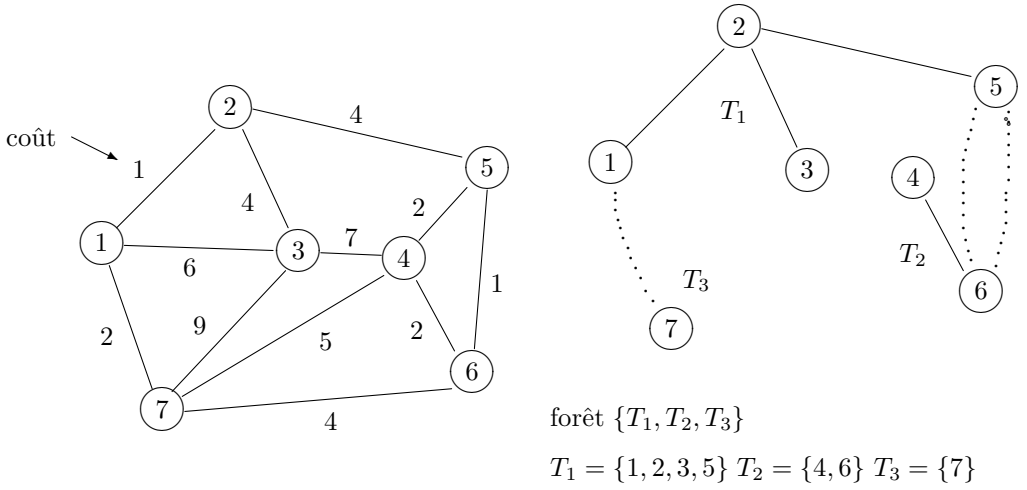


FIGURE 13 – Graphe pour le problème de l'arbre sous-tendant minimal

Et maintenant, pourquoi ne pas insérer les arcs intéressants dans la solution ? De façon surprenante, on peut démontrer que cette stratégie ≪gloutonne≫ conduit à la solution optimale. Nous allons en considérer trois variantes.

1. À la première itération, insérer dans la solution l'arc de coût minimum. Aux itérations subséquentes, insérer l'arc intéressant adjacent au seul arbre non trivial. Cette stratégie, attribuée à **Prim**, fait croître un seul arbre dans la forêt.
2. À chaque itération, insérer l'arc admissible de coût minimum. Cette stratégie est attribuée à **Kruskal**.
3. Insérer simultanément tous les arcs intéressants.[5] Cet algorithme, proposé en 1926 par **Borůvka** est en quelque sorte le plus moderne : son auteur aurait-il eu la prémonition du parallélisme ?

Appliquons ces trois algorithmes au problème de la figure 13. Les itérés successifs sont présentés dans les tableaux ci-dessous. On note que les solutions, bien que de même coût total (14), sont différentes. Ceci peut se produire même si les coûts des arcs sont tous distincts.

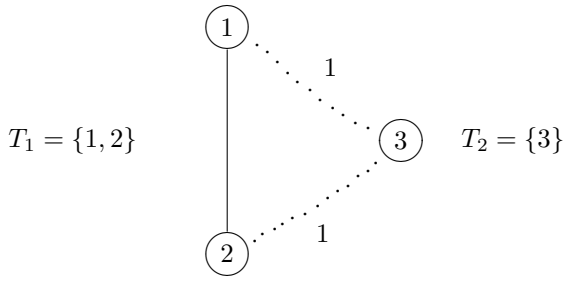


FIGURE 14 – Mauvais choix dans l'algorithme de Borůvka

5. Attention à ne pas créer de cycle en cas d'égalité de coût (voir figure 14 où l'insertion simultanée des arcs (1,3) et (2,3) crée un cycle.)

itération	1	2	3	4	5	6
arc inséré	(1,2)	(1,7)	(2,3)	(2,5)	(5,6)	(5,4)
alternative	(5,6)		(2,5)	(7,6)		(4,6)
			(7,6)			
coût	1	2	4	4	1	2

Algorithme de Prim

itération	1	2	3	4	5	6
arc inséré	(1,2)	(5,6)	(4,6)	(1,7)	(6,7)	(2,3)
alternative	(5,6)		(5,4)		(2,3)	
			(1,7)		(2,5)	
coût	1	1	2	2	4	4

Algorithme de Kruskal

itération	1							2
arcs insérés	(1,2)	(2,1)	(3,2)	(4,5)	(5,6)	(6,5)	(7,1)	(2,5)
alternative								
alternative			(1,7)		(2,5)			
coût	1	–	4	2	1	–	2	4

Algorithme de Borůvka

CHAPITRE 7 : PROGRAMMATION DYNAMIQUE

26 novembre 2003

La programmation dynamique est une technique d'exploration des solutions d'un programme mathématique qui, comme un chat, est plus facile à reconnaître qu'à décrire. Je vais en donner quatre exemples classiques.

1. Le problème du sac-à-dos

Dans cette section, le problème du sac-à-dos

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} w_j x_j \leq b \\ & x_j \text{ entier non négatif} \end{aligned}$$

est résolu par une technique différente de celle (Branch-and-Bound) étudiée précédemment. À l'itération k du processus, on détermine une solution optimale correspondant à un sac-à-dos de capacité k. Si $z(k)$ est la valeur de la solution optimale, on obtient trivialement l'équation de récurrence

$$z(k) = \max_{j:w_j \leq k} \{c_j + z(k - w_j)\}$$

avec, comme conditions initiales :

$$z(k) = 0 \quad \text{si } k < \min_j \{w_j\}.$$

À chaque itération, on note $j(k)$ un objet pour lequel le maximum a été atteint. Appliquons la technique au problème

$$\begin{array}{lrcrcrl} \max & 2x_1 & + & 3x_2 & + & 7x_3 & \\ & 3x_1 & + & 4x_2 & + & 8x_3 & \leq 14 \\ & x_1, & & x_2, & & x_3 & \text{entiers non négatifs.} \end{array}$$

k	$z(k)$	$j(k)$
0	0	–
1	0	–
2	0	–
3	2	1
4	$\max\{2 + z(4-3), 3 + z(4-4)\} = 3$	2
5	$\max\{2 + z(5-3), 3 + z(5-4)\} = 3$	2
6	$\max\{2 + z(6-3), 3 + z(6-4)\} = 4$	1
7	$\max\{2 + z(4), 3 + z(3)\} = 5$	1,2
8	$\max\{2 + z(5), 3 + z(4), 7 + z(0)\} = 7$	3
9	$\max\{2 + z(6), 3 + z(5), 7 + z(1)\} = 7$	3
10	$\max\{2 + z(7), 3 + z(6), 7 + z(2)\} = 7$	1,2,3
11	$\max\{2 + z(8), 3 + z(7), 7 + z(3)\} = 9$	1,3
12	$\max\{2 + z(9), 3 + z(8), 7 + z(4)\} = 10$	2,3
13	$\max\{2 + z(10), 3 + z(9), 7 + z(5)\} = 10$	2,3
14	$\max\{2 + z(11), 3 + z(10), 7 + z(6)\} = 11$	1,3

On retrace la solution optimale en procédant à rebours : $j(14) = 1$, $j(14 - w_1) = j(11) = 1$, $j(11 - w_1) = j(8) = 3$. La solution optimale est donc $x^* = (2, 0, 1)$.

Cet algorithme permet non seulement d'obtenir la solution optimale du problème de départ, mais également les solutions optimales correspondant à tous les sacs-à-dos de capacité inférieure ou égale à 14. Notons qu'il est possible d'améliorer l'efficacité de l'algorithme.

2. Les plus courts chemins

Soit t un sommet destination et $y_i^{[k]}$ la distance d'un plus court chemin du sommet i au sommet t qui utilise *au plus* k arcs. La formule récursive suivante permet de calculer $y_i^{[k]}$ [1]

$$y_i^{[k]} = \min_j \{c_{ij} + y_j^{[k-1]}\}$$

et on note $j_i^{[k]}$ un sommet pour lequel le minimum est atteint. Les condition initiales sont

$$y_i^{[0]} = \begin{cases} 0 & \text{si } i = t \\ \infty & \text{sinon.} \end{cases}$$

À moins qu'il n'existe des **circuits négatifs**, l'algorithme basé sur cette équation récursive trouvera après au plus $n-1$ itérations (n désigne le nombre de sommets dans le réseau) un arbre de plus courts chemins de tout sommet i vers la destination t. En effet, s'il existe un plus court chemin, il doit exister un plus court chemin **simple** (sans circuit) et un tel chemin ne peut comporter plus de $n-1$ arcs.

Appliquons cette technique au réseau ci-dessous, avec $t = 6$.

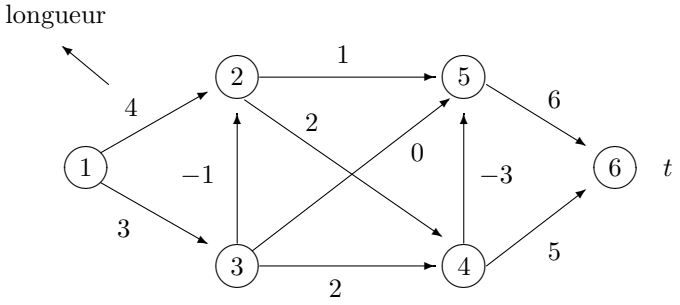


FIGURE 1 – Le réseau de l'exemple de plus courts chemins

Les résultats sont indiqués au tableau 1. Le premier nombre indique la longueur d'un chemin partiel. Le nombre entre parenthèses correspond au successeur $succ^{[k]}(i)$ du sommet i dans le chemin obtenu à l'itération k, et permet de retracer le chemin optimal (1,3,2,4,5,6). Je donne au-dessous du tableau deux exemples de calcul de $y_i^{[k]}$ et $succ^{[k]}(i)$:

		5	4	3	2	1	0	← itérations
	1	7(3)	8(3)	9(3)	∞	∞	∞	
	2	5(4)	5(4)	5(4)	7(4)	∞	∞	
sommets →	3	4(2)	4(2)	5(4)	6(5)	∞	∞	
	4	3(5)	3(5)	3(5)	3(5)	5(6)	∞	
	5	6(6)	6(6)	6(6)	6(6)	6(6)	∞	
	6	0(6)	0(6)	0(6)	0(6)	0(6)	0(6)	

TABLE 1 – Programmation dynamique pour les plus courts chemins : $y_i^{[k]}$ $(succ^{[k]}(i))$

1. Le minimum est pris sur tous les indices de sommets adjacents à i, c'est-à-dire pour lesquels il existe un arc de i à j. Par convention, on suppose que i est relié à lui-même par un arc de longueur nulle.

$$
\begin{aligned}
y_2^{[2]} &= \min\{c_{22} + y_2^{[1]}, c_{24} + y_4^{[1]}, c_{25} + y_5^{[1]}\} \\
&= \min\{0 + \infty, 2 + 5, 1 + 6\} = 7 \\
&\quad succ^{[2]}(2) = 4 \text{ ou } 5
\end{aligned}
$$

$$
\begin{aligned}
y_1^{[4]} &= \min\{c_{11} + y_1^{[3]}, c_{12} + y_2^{[3]}, c_{13} + y_3^{[3]}\} \\
&= \min\{0 + 9, 4 + 5, 3 + 5\} = 8 \\
&\quad succ^{[4]}(1) = 3
\end{aligned}
$$

Remarques

- Si, lors d'une itération complète, il n'y a aucune amélioration, alors on peut arrêter l'algorithme.
- Si on poursuit l'algorithme après la $(n-1)$ième itération et qu'il y a amélioration, alors le réseau doit forcément comporter un circuit négatif, c'est-à-dire un circuit dont la somme des coûts des arcs est négative. L'algorithme détectera ce circuit.
- La complexité de l'algorithme est $O(n^3)$.
- Noter la similarité de cet algorithme avec l'algorithme générique du chapitre 6 où l'on choisit les sommets à balayer dans l'ordre de leur insertion dans l'ensemble B. En fait, si on autorise l'ensemble B à contenir plusieurs copies d'un même sommet [2] alors les deux algorithmes sont identiques.
- Le problème du sac-à-dos peut se formuler sous forme d'un problème de plus court chemin en introduisant $b+1$ sommets indexés par les entiers de 0 à b et des arcs $(k, k+w_j)$ de coût c_j. L'algorithme de programmation dynamique tient compte de la structure particulière de ce graphe (**graphe topologique**, ça devrait vous rappeler des bons souvenirs du cours IFT 1063 : ordres partiels, diagrammes de Hasse, etc.). Ainsi, à l'itération k, la quantité $z(k)$ (qui joue le rôle de $y_i^{[k]}$) est-elle fixée de façon définitive.

3. Gestion de stocks

Considérons un problème de gestion de stocks sur horizon infini où tout (demande par unité de temps d, coût unitaire d'inventaire h, coût fixe de commande c) est constant. On souhaite déterminer le niveau et l'intervalle optimal des commandes. Il est facile de démontrer qu'une stratégie optimale consiste à commander, à intervalle fixe de durée I, une quantité x du produit. Le niveau de l'inventaire est alors donné par la fonction linéaire par morceaux illustrée à la figure 2.

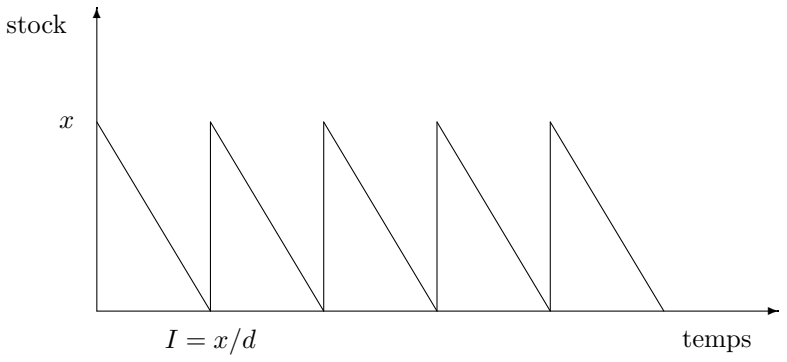


FIGURE 2 – Solution d'un problème simple de gestion de stocks

La demande étant uniforme, le niveau de stock s'annule x/d unités de temps après la commande. Puisque le niveau moyen du stock est égal à $x/2$, le coût moyen unitaire d'inventaire est égal à $hx/2$. Le coût unitaire moyen du système prend la forme

2. ... ce qui est inefficace, car un sommet i pourrait être balayé avec une valeur y_i qui est périmée.

$$f(x) = \frac{c}{x/d} + h\frac{x}{2} = \frac{cd}{x} + \frac{h}{2}x.$$

Le minimum de cette fonction est atteint lorsque la dérivée $f'(x) = -\frac{cd}{x^2} - \frac{h}{2}$ s'annule, c'est-à-dire

$$x = \sqrt{\frac{2cd}{h}}.$$

Pas grand-chose à voir avec la programmation dynamique. Cependant, les choses deviennent plus intéressantes et plus réalistes lorsque les paramètres varient, par exemple s'il y a des économies d'échelle ou des variations saisonnières de la demande. Pour simplifier un peu, on suppose que les commandes sont passées à intervalles réguliers et que l'horizon de planification est fini. On utilise les notations suivantes :

$[k-1, k]$	**période** $k \quad (k = 1, \ldots, n)$
s_k	niveau de l'inventaire à la fin de la période k (**état** du système)
$c_k(x)$	coût de commande pour une quantité x
$h_k(s)$	coût d'inventaire pour un niveau moyen s
d_k	demande à la période k
$f^{[k]}(s)$	coût d'une politique optimale de commande débutant en *début* de période k (instant $k-1$) à l'état s
$x^{[k]}(s)$	commande optimale au début de la période k

L'évolution de l'inventaire est illustrée à la figure 3.

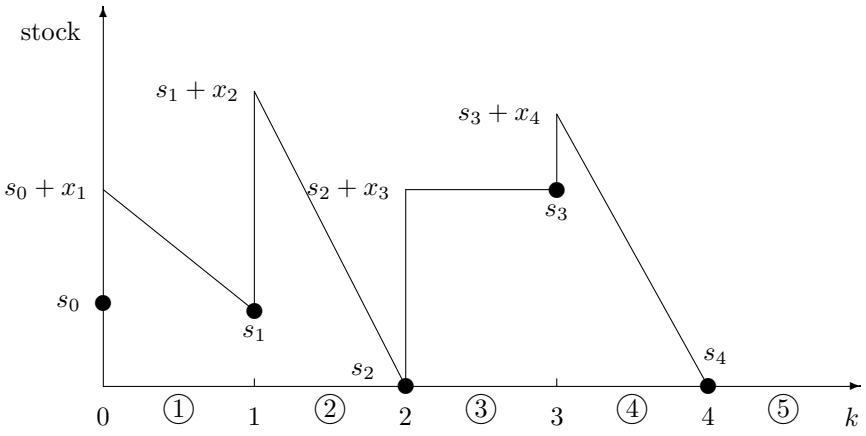


FIGURE 3 – Évolution typique du niveau d'inventaire

L'équation de la programmation dynamique pour ce problème prend la forme [3], pour toute valeur possible du niveau de stock s :

$$f^{[n+1]}(s) = 0 \quad \forall s$$
$$f^{[k]}(s) = \min_{x \text{ admissible}} \left\{ c_k(x) + h_k \times \frac{s + x + s_k}{2} + f^{[k+1]}(s_k) \right\} \qquad k = n, n-1, \ldots, 2, 1$$

avec l'équation d'évolution $s_k = s + x - d_k$. Dans l'équation ci-dessus, $(s + x + s_k)/2$ représente le niveau de stock moyen en période k.

3. Une commande x est admissible si elle respecte la contrainte de capacité et si elle permet de satisfaire la demande pour la période k.

Ces équations de récurrence donnent, pour chaque période k et chaque niveau d'inventaire s, le coût d'une politique de commande optimale du début de la période k jusqu'à la fin de l'horizon de planification. Dans le cas où tous les paramètres et variables sont entiers, ces équations sont faciles à résoudre.

Considérons l'exemple sur 3 périodes correspondant aux données suivantes : $h_k(s) = 2s$, vecteur de demande= (2,1,4), capacité de stockage = 5 et $c_k(x) = c(x)$ avec comme fonction de coût de commande

x	0	1	2	3	4	5
$c(x)$	0	10	16	20	22	23

.

On suppose que le stock initial est nul. Une solution optimale est donnée au tableau 2. Le premier chiffre indique le coût de la solution partielle. Il est suivi, entre parenthèses, du niveau de commande optimal $x^{[k]}(s)$. Il est facile de démontrer que le niveau de stock terminal est nul. Le niveau de commande de la dernière période (période 3) s'obtient donc trivialement. Aux autres périodes (2 et 1) on obtient successivement :

s	$f^{[1]}(s)$	$f^{[2]}(s)$	$f^{[3]}(s)$	$f^{[4]}(s)$
5		13(0)	6(0)	–
4		21(0)	4(0)	0
3		25(0)	14(1)	0
2		27(0)	20(2)	0
1		27(0)	24(3)	0
0	51(3)	36(5)	26(4)	0

TABLE 2 – Solution optimale du problème de gestion de stock : $f^{[k]}(s_k)$ $(x^{[k]}(s))$

Voici quelques exemples de calcul des valeurs optimales :

$$\begin{aligned} f^{[2]}(0) = \min\{&c(1) + h\left(\tfrac{1+0}{2}\right) + f^{[3]}(0+1-1), \\ &c(2) + h\left(\tfrac{2+1}{2}\right) + f^{[3]}(0+2-1), \\ &c(3) + h\left(\tfrac{3+2}{2}\right) + f^{[3]}(0+3-1), \\ &c(4) + h\left(\tfrac{4+3}{2}\right) + f^{[3]}(0+4-1), \\ &c(5) + h\left(\tfrac{5+4}{2}\right) + f^{[3]}(0+5-1)\} \end{aligned} \qquad x \text{ optimal}$$

$$= \min\{10+1+26, 16+3+24, 20+5+20, 22+7+14, 23+9+4\} = 36 \qquad 5$$

$$f^{[2]}(1) = \min\{0+1+26, 10+3+24, 16+5+20, 20+7+14, 22+9+4\} = 27 \qquad 0$$

$$f^{[2]}(2) = \min\{0+3+24, 10+5+20, 16+7+14, 20+9+4\} = 27 \qquad 0$$

$$f^{[2]}(3) = \min\{0+5+20, 10+7+14, 16+9+4\} = 25 \qquad 0$$

$$f^{[2]}(4) = \min\{0+7+14, 10+9+4\} = 21 \qquad 0$$

$$f^{[2]}(5) = \min\{0+9+4\} = 13 \qquad 0$$

$$f^{[1]}(0) = \min\{16+2+36, 20+4+27, 22+6+27, 23+8+25\} = 51 \qquad 3$$

Dans ces équations, le premier nombre représente le coût de commande, le second le coût d'inventaire et le troisième le **coût futur**. La somme des deux premiers coûts représente le **coût immédiat**. Noter que le minimum n'est pris que sur les valeurs admissibles du niveau de commande : il faut satisfaire la demande à venir sans dépasser la capacité de l'entrepôt (5). Ainsi, si le niveau d'inventaire est nul au début de la deuxième période, il faut commander au moins une unité pour satisfaire à la demande de cette période.

On trouve la solution optimale en partant de la cellule de gauche, en se dirigeant vers la droite et en respectant l'équation d'évolution de l'inventaire. Ainsi, $x_1 = 3$, $x_2 = 0$ et $x_3 = 4$. La mise en œuvre informatique de cet algorithme est d'une simplicité déconcertante.

On remarque une similitude entre cet algorithme et l'algorithme de programmation dynamique présenté à la section précédente (plus courts chemins). En fait, ces algorithmes sont identiques. On peut en effet formuler le problème d'inventaire comme un problème de plus court chemin en en créant, à chaque début de période, des nœuds correspondant aux niveaux d'inventaire. À chaque niveau de commande x correspond un arc liant un sommet (k, s) au sommet $(k+1, s_k = s + x - d_k)$ de coût $c_k(x_k) + h_k((s + x + s_k)/2)$.

4. Un modèle d'achats stochastique

Les situations où tous les paramètres d'un problème sont connus avec certitude sont peu fréquentes. Par exemple, dans le modèle précédent, il est sans doute irréaliste de faire l'hypothèse que la demande est une constante immuable. Dans cette section, nous considérons la situation où un détaillant achète un produit d'un grossiste pour le revendre. Malheureusement, le commerçant doit passer ses commandes au début du mois, *avant* de connaître le niveau de la demande mensuelle. Par expérience, il en connaît cependant la distribution de probabilité. S'il achète une trop grande quantité du produit et que la demande est faible, les coûts d'inventaire seront élevés et il risque de se retrouver avec des invendus. S'il achète peu du produit, il risque de perdre des ventes. Notre algorithme de programmation dynamique devra donc lui suggérer une politique d'achat qui maximise son *gain moyen*, étant donné un niveau de stock au début de chaque mois. On introduit les notations suivantes :

$r_k(v)$	revenu au mois k pour un niveau de vente v
$\text{Prob}\,(d_k = d)$	probabilité que le niveau de demande soit égal à d
v_k	niveau de vente au mois k

On suppose que toutes les quantités sont entières et non négatives. On a

$$\begin{aligned} &\text{Prob}\,(d_k = d) \geq 0 \\ &\sum_d \text{Prob}\,(d_k = d) = 1. \end{aligned}$$

Il faut modifier l'équation d'évolution pour interdire au niveau de stock de devenir négatif :

$$s_k = \max\{s_{k-1} + x_k - d_k, 0\}$$

et introduire une équation spécifiant le niveau de vente v_k au mois k correspondant à une demande d :

$$v_k = \min\{d_k, s_{k-1} + x_k\}.$$

Au début du mois k, le gain maximal s'obtient en résolvant l'équation de récurrence probabiliste

$$f^{[k]}(s) = \max_{x \text{ admissible}} \sum_d \text{Prob}\,(d_k = d) \left[r_k(v_k) - c_k(x) - h_k \times \left(\frac{s + x + s_k}{2} \right) + f^{[k+1]}(s_k) \right].$$

Une commande x est admissible si elle respecte la capacité de l'entrepôt ($s_{k-1} + x \leq$ capacité).

À chaque itération, on maximise le gain moyen jusqu'à la fin de la période de planification. Il est important de noter que la politique optimale n'est pas fixée au début de la période de planification mais s'ajuste, mois après mois, aux aléas du marché de la demande. Noter que v_k et s_k sont des fonctions de d.

Il est maintenant plus que temps de considérer un exemple concret dont les paramètres sont
- Stock initial nul.
- Prix de vente unitaire : 4.
- Horizon de planification : 3 mois.
- Capacité de l'entrepôt : 3.
- Coût unitaire d'inventaire : 2.

– Répartition de la demande et coûts de commande ($c_k(x) = c(x)$ pour tout k)

d	0	1	2	3
$\text{Prob}\,(d_k = d)$	$\frac{1}{8}$	$\frac{2}{8}$	$\frac{3}{8}$	$\frac{2}{8}$

x	0	1	2	3
$c(x)$	0	2	3	3

Comme les calculs sont fastidieux, je me contenterai de donner quelques éléments du tableau contenant les $f^{[k-1]}(s_{k-1})$ (voir tableau 3) et d'illustrer le calcul de certains de ses éléments. Je vous encourage fortement à écrire un petit programme qui vous permette de compléter ce tableau.

s	$f^{[1]}(s)$	$f^{[2]}(s)$	$f^{[3]}(s)$	$f^{[4]}(s)$
3		?	$\frac{22}{8}(0)$	0
2		?	$\frac{28}{8}(0)$	0
1		$\frac{199}{64}(1)$	$\frac{19}{8}(0)$	0
0	?	?	$\frac{4}{8}(2)$	0

TABLE 3 – Solution optimale partielle du problème de gestion des achats stochastique

Le calcul de l'élément $f^{[3]}(1)$ s'effectue de la façon suivante, en notant qu'on ne peut satisfaire une demande supérieure à 1 si le niveau de stock est nul et que l'on ne commande rien. Il suffit donc de ne considérer, dans le calcul du premier élément du gain moyen, que deux cas : soit la demande est nulle (avec probabilité 1/8) soit la demande est supérieure ou égale à 1 (avec probabilité 7/8). Si l'on commande une unité, il y a trois possibilités de vente (0, 1 ou 2). Si l'on commande 2 unités, on pourra répondre à toute demande pendant le troisième mois. Comme dans le cas déterministe, le gain comporte deux composantes : un gain immédiat (revenu de la vente − coût de commande − coût d'inventaire) et un gain à venir. Ceci nous donne :

$$\begin{aligned} f^{[3]}(1) &= \max\{\tfrac{1}{8}(0-0-2)+\tfrac{7}{8}(4-0-1), && x=0 \\ &\qquad \tfrac{1}{8}(0-2-4)+\tfrac{2}{8}(4-2-3)+\tfrac{5}{8}(8-2-2), && x=1 \\ &\qquad \tfrac{1}{8}(0-3-6)+\tfrac{2}{8}(4-3-5)+\tfrac{3}{8}(8-3-4),+\tfrac{2}{8}(12-3-3)\} && x=2 \\ &= \max\{\tfrac{19}{8},\tfrac{12}{8},\tfrac{4}{8}\}=\tfrac{19}{8} \qquad x^{[3]}(1)=0. \end{aligned}$$

Si le niveau de stock au début du troisième mois est égal à 2, il est inutile de considérer l'achat de plus d'une unité du produit :

$$\begin{aligned} f^{[3]}(2) &= \max\{\tfrac{1}{8}(0-0-4)+\tfrac{2}{8}(4-0-3)+\tfrac{5}{8}(8-0-2), && x=0 \\ &\qquad \tfrac{1}{8}(0-2-6)+\tfrac{2}{8}(4-2-5)+\tfrac{3}{8}(8-2-4),+\tfrac{2}{8}(12-2-3)\} && x=1 \\ &= \max\{\tfrac{28}{8},\tfrac{6}{8}\}=\tfrac{28}{8} \qquad x^{[3]}(2)=0. \end{aligned}$$

Et ainsi de suite :

$$f^{[3]}(3)=\tfrac{1}{8}(0-0-6)+\tfrac{2}{8}(4-0-5)+\tfrac{3}{8}(8-0-4)+\tfrac{2}{8}(12-0-3)\}=\tfrac{22}{8} \quad x=0 \qquad x^{[3]}(3)=0.$$

$$\begin{aligned} f^{[2]}(1) &= \max\{\tfrac{1}{8}(0-0-2+\tfrac{19}{8})+\tfrac{7}{8}(4-0-1+\tfrac{4}{8}), && x=0 \\ &\qquad \tfrac{1}{8}(0-2-4+\tfrac{28}{8})+\tfrac{2}{8}(4-2-3+\tfrac{19}{8})+\tfrac{5}{8}(8-2-2+\tfrac{4}{8}), && x=1 \\ &\qquad \tfrac{1}{8}(0-3-6+\tfrac{22}{8})+\tfrac{2}{8}(4-3-5+\tfrac{28}{8})+\tfrac{3}{8}(8-3-4+\tfrac{19}{8})+\tfrac{2}{8}(12-3-3+\tfrac{4}{8})\} && x=2 \\ &= \max\{\tfrac{199}{64},\tfrac{182}{64},\tfrac{127}{64}\}=\tfrac{199}{64} \qquad x^{[2]}(1)=0. \end{aligned}$$

Il existe plusieurs variantes et extensions de ce modèle stochastique. Par exemple, on pourrait attribuer une valeur aux invendus à la fin de l'horizon de planification. On pourrait également pénaliser l'entreprise en cas de rupture de stock. Ou considérer une demande qui varie dans le temps (demande saisonnière).

La programmation dynamique

Tel que promis en début de section, je n'ai pas donné de définition formelle de la programmation dynamique, comme je l'avais fait, par exemple, pour la programmation linéaire. On peut cependant identifier deux points fondamentaux communs aux situations considérées, qui permettent d'utiliser un schéma algorithmique relativement général :

- une division de l'horizon de planification en périodes ;
- une fonction objectif additive qui, au début d'une période donnée, est la somme de deux termes : un gain passé et un gain à venir. Cette forme fonctionnelle permet d'optimiser en ne tenant compte que de l'état présent, indépendamment des décisions qui ont mené à cet état.

La définition des périodes et états est souvent naturelle. Parfois, cependant, un effort d'imagination est nécessaire pour mettre le problème sous une forme qui se prête à la programmation dynamique. C'est le cas du problème de sac-à-dos où il n'y a pas d'aspect dynamique « naturel ». Dans cet exemple « dégénéré », les périodes correspondent à la capacité résiduelle du sac et il n'y a pas vraiment d'états. On peut en fait considérer qu'au début de la période, il n'existe qu'un seul état k correspondant à un sac-à-dos vide, et qui ne joue donc aucun rôle.

En conclusion, je vais élaborer brièvement sur les avantages et inconvénients de la programmation dynamique.

Avantages

- Très flexible. Il est facile d'ajouter des contraintes, d'incorporer des fonctions quelconques.
- Fournit une solution globalement optimale.
- Fournit non seulement une solution optimale, mais un ensemble de solutions optimales à chaque étape intermédiaire et pour tous les états possibles. Ainsi, il est possible d'utiliser les résultats dans les situations où des modifications du système mènent à des changements d'état impromptus.
- Permet de traiter des problèmes stochastiques.
- S'adapte à des situations où le temps, les états et les décisions varient continûment. Mais là, c'est pas mal plus complexe.

Inconvénients

- Ne s'applique que sous les hypothèses d'additivité et d'amnésie (le passé n'influence pas le futur).
- Parfois lent et donc inefficace.
- Ne se prête pas bien à la réoptimisation, contrairement à la programmation linéaire.