Enseignement de l'informatique

POLYCOPIE

Méthodes d'optimisation

LEBBAH Yahia
Professeur, Université d'Oran 1

Département Informatique, Faculté des Sciences Exactes et Appliquées, Université Oran1
B.P. 1524, El-M'Naouar Oran, Algérie

Version du 5 novembre 2025

Table des matières

Chapitre 1

Préliminaires sur l'optimisation : problèmes faciles et problèmes difficiles[Prins, 1994]

1.1 Problèmes faciles

Exploration d'un graphe Donnée : Un graphe orienté $G = (X, U)$, deux sommets s et t de X. Question : Existe-t-il un chemin de s à t ? Algorithme en $O(M)$.

Chemin de coût minimal Données : $G = (X, U, C)$ un graphe orienté valué, et deux sommets s et t de X. Question : Trouver un chemin de coût minimal de s à t. Algorithme de Bellman en $O(NM)$. Algorithme de Dijkstra en $O(N^2)$.

Flot maximal Données : Un réseau de transport $G = (X, U, C, s, t)$. Question : Maximiser le débit du flot qui peut s'écouler dans le réseau entre s et t. Algorithme de Ford-Fulkerson en $O(NM^2)$.

Arbre recouvrant de poids minimal Données : $G = (X, E, W)$ un graphe simple valué. Question : Trouver un arbre recouvrant de poids minimal. Algorithme de Prim en $O(N^2)$. Algorithme de Kruskal en $O(M \log N)$. Si G est orienté, on pourrait calculer une arborescence recouvrante en $O(MN)$ avec l'algorithme de Edmonds.

Couplage Données : Soit $G = (X, E)$ un graphe simple. Un couplage de G (matching) est un sous-ensemble d'arêtes tel que deux quelconques d'entre elles n'aient aucun sommet commun. Question : Trouver un couplage de cardinalité maximale.

Parcours eulériens et chinois Données : Un parcours eulérien passe une fois par chaque arc ou arête. Le problème d'existence est solvable en $O(M)$. Un parcours chinois visite au moins une fois chaque arête.

Test de planarité Données : Un graphe simple $G = (X, E)$. Question : G est-il planaire, c'est-à-dire dessinable dans le plan sans croisement d'arêtes ? Un algorithme en $O(M)$ dû à Hopcroft et Tarjan.

Test de bipartisme On peut démontrer qu'un graphe est biparti ssi ne contient pas de cycle impair. Algorithme en $O(M)$.

Recherche d'une information parmi N Il s'agit de la recherche d'un élément dans un tableau de N éléments.

Tri de N **nombre** Solvable avec l'algorithme du tri par tas en $O(n \log n)$.

Programmation linéaire Il s'agit de résoudre le problème d'optimisation :

$$\begin{cases} min\ c.x \\ A.x \leq b \\ x \in I\!R^n, x \geq 0 \end{cases}$$

L'algorithme du simplexe est performant en moyenne, mais exponentiel dans le pire des cas. Karmarkar a proposé en 1984 un algorithme polynomial en $O(n^{3.5}L)$ où L est le nombre de bits pour coder A, b et c.

1.2 Problèmes difficiles

Stable maximal Données : Un graphe simple $G = (X, E)$. Un sous-ensemble de sommets S est un ensemble stable s'il n'y a pas d'arête entre deux sommets quelconques de S.

Transversal minimal Données : Un graphe simple $G = (X, E)$. Un sous-ensemble de sommets T est un Transversal si toute arête de G a au moins une extrémité dans T.

Clique maximale Données : Un graphe simple $G = (X, E)$. Un sous-ensemble de sommets Q est une clique si toute paire de sommets de Q est reliée par une arête. Q engendre donc un graphe complet.

Coloration minimale Données : Un graphe simple $G = (X, E)$. G est k-colorable si on peut colorer ses sommets avec k couleurs distinctes, sans que deux sommets voisins aient la même couleur. Le plus petit k pour lequel G est k-colorable est le nombre chromatique de G.

Problèmes hamiltoniens Données : $G = (X, U, C)$ un graphe orienté valué. Le problème d'existence d'un parcours hamiltonien dans un graphe G est difficile. Le problème du voyageur de commerce consiste à trouver un circuit ou un cycle hamiltonien, de coût minimal, dans un graphe valué complet.

Problème SAT Données : une formule clausale. Question : Peut-on affecter à chaque variable propositionnelle de façon à rendre toutes les clauses vraies.

Sac à dos en variables entières Il s'agit de résoudre le problème :

$$\begin{cases} min\ c.x \\ a.x \leq b \\ x \text{ entier} \end{cases}$$

Bin packing On donne n objets de poids a_i et un nombre non limité de boites de capacité b. Le but est de répartir les objets en un nombre minimal de boites.

1.3 NP-complétude et les classes P et NP [Prins, 1994]

Certains problèmes d'optimisation combinatoire disposent d'algorithmes polynomiaux, tandis que d'autres n'en ont toujours pas. Existe-t-il réellement une classe de problèmes combinatoires pour lesquels on ne trouvera jamais d'algorithmes polynomiaux, ou est ce que les problèmes difficiles ont en fait de tels algorithmes, mais non encore découverts ? On conjecture depuis longtemps l'existence d'une classe de problèmes intrinsèquement difficiles, car plusieurs problèmes difficiles (comme le problème du voyageur de commerce) résistent depuis plus de 50 ans à l'assaut des travaux de recherche : malgré ces efforts, aucun algorithme polynomial n'a été trouvé.

La théorie de la complexité a été développée dans les années 1970 pour répondre à cette question. Le principal résultat est que tous les problèmes difficiles sont liées : la découverte d'un algorithme polynomial pour un seul problème difficile permettrait de déduire des algorithmes polynomiaux pour tous les autres.

La théorie de la complexité ne traite que des problèmes d'existence, à réponse oui/non. Ceci n'est pas gênant pour les problèmes d'optimisation. Un algorithme efficace pour le problème d'existence peut être utilisé pour résoudre efficacement son problème d'optimisation associé, par une simple dichotomie sur les valeurs de la fonction objectif.

1.4 La classe P et NP [Prins, 1994, Cori et al., 2001]

Etant donné une fonction $f : \mathbb{N} \rightarrow \mathbb{N}$, on dira qu'un problème appartient à la classe $DTIME(f)$ s'il existe une machine déterministe qui sur tout entrée de longueur n, résout le problème en $O(f(n))$ pas de calcul.

L'ensemble des problèmes d'existence qui admettent des algorithmes polynomiaux forment la classe P. $On peut poser P = \cup_{k\geq 0} DTIME(n \longmapsto n^k)$. Il faut pouvoir vérifier en temps polynomial une proposition de réponse Oui.

La classe NP est celles des problèmes d'existence dont une proposition de solution Oui est vérifiable polynomialement. On définit également la classe $NP = \cup_{k\geq 0} NTIME(n \longmapsto n^k)$, où N signifie non-déterministe. Le modèle de calcul non-déterministe est enrichi par une instruction de "choix" où il existe une façon immédiate pour conduire à la solution.

Les problèmes qui ne sont pas dans NP existent, mais ne présentent qu'un intérêt théorique pour la plupart. NP inclut P.

Pour un problème sans algorithme efficace, il faut procéder comme suit pour prouver l'appartenance à NP :

1. proposer un codage de la solution (appelé certificat) ;
2. proposer un algorithme qui va vérifier la solution au vu des données du certificat ;
3. montrer que cet algorithme a une complexité polynomiale.

Considérons le problème suivant : **étant donné un ensemble S de n nombres entiers et un entier b, existe-t-il un sous-ensemble T de S dont la somme des éléments est égale à b ?** On ne connait pas d'algorithme polynomial pour résoudre ce problème. Il n'empêche qu'il est dans NP, car **vérifier qu'une somme d'un ensemble d'entiers T, sous-ensemble d'un ensemble S de cardinalité n, est égale à b, est en $O(n^2)$.** Dans cette vérification, il faut s'assurer que tous les éléments de T sont dans S, qui nécessitera au plus n^2 tests.

1.5 Les problèmes NP-complets [Prins, 1994, Cori et al., 200

Il s'agit des problèmes les plus difficiles de NP, le "noyau dur". La notion de problème NP-complet est basée sur celle de transformation polynomiale d'un problème. Un problème d'existence P_1 se transforme polynomialement en un autre P_2 s'il existe un algorithme polynomial A transformant toute donnée pour P_1 en une pour P_2, en conservant la réponse Oui et Non. Par exemple, un stable d'un graphe simple G est une clique dans le graphe complémentaire G_c.

Un problème NP-complet est un problème de NP en lequel se transforme polynomialement tout autre problème de NP. La classe des problèmes NP-complets est notée NPC. On rencontre dans la littérature le terme NP-difficile pour les problèmes d'optimisation : un problème d'optimisation combinatoire est NP-difficile si le problème d'existence associé est NP-complet.

La conséquence pratique forte de la classe NPC : si on trouverait un algorithme polynomial A pour un seul problème NP-complet X, on pourrait en déduire un autre pour tout autre problème difficile Y de P. Il suffit de transformer polynomialement les données de Y en données pour X, puis exécuter l'algorithme pour X.

Une question immédiate est de savoir si de tels problèmes existent réellement dans NP. Le logicien Cook (et Levin) a montré en 1970 que **le problème SAT est NP-complet**. Depuis cette date, les travaux de recherche ont montré que la plupart des problèmes d'existence associés aux problèmes difficiles sans algorithmes polynomiaux connus sont NP-complets.

Problème SAT

Données Soit un ensemble de de variables $\{x_1, ..., x_n\}$. Soit une formule logique F sous forme normale conjonctive $F = C_1 \wedge C_2 \wedge ... \wedge C_l$, avec chaque clause $C_i = (y_{i,1} \vee y_{i,2} \vee ... \vee y_{i,k_i})$, où chaque $y_{i,j}$ est un littéral, c'est-à-dire $y_{i,j} = x_i$ ou $y_{i,j} = \neg x_i$.

Résultat "oui" ssi F est satisfaisable, qu'il existe une affectation de valeurs de vérités aux variables qui rende F vraie.

Proposition 1. *Le problème SAT est NP-complet.*

Preuve : Premier problème démontré NP-complet. Preuve réalisée par Cook et Levin [Cook, 1971, Levin, 1973]. □

Chapitre 2

Optimisation linéaire et résolution exacte par séparation/évaluation

Les sections 2.1, 2.3, 2.2 sont extraits intégralement du document [Bastin, 2010] téléchargeable ici : `https://www.iro.umontreal.ca/~bastin/Cours/IFT1575/IFT1575.pdf`

2.1 Présentation générale [Bastin, 2010]

Certaines quantités ne peuvent s'écrire sous forme de nombres réels, issus d'un domaine continu. Au contraire, certaines décisions sont par nature discrètes, et doivent se représenter à l'aide de nombres entiers. Considérons par exemple une **entreprise de transport**, qui décide de renouvellement sa flotte de camions. Le **nombre de camions à acheter est un nombre naturel**.

La présence de telles variables entières modifie profondément la nature des programmes sous-jacents. Lorsqu'un problème est linéaire avec des variables entières, nous parlerons de *programmation mixte entière*. Si toutes les variables sont entières, nous utiliserons la terminologie de de *programmation (pure) en nombres entiers*. Si les variables entières sont à valeurs 0 ou 1 (binaires), nous parlerons de **programmation 0–1 (binaire)**.

Nous pourrions bien entendu considérer le cas **non-linéaire**, mais les complications sont telles qu'à ce jour, **aucune méthode pleinement satisfaisante n'existe**, même si d'intenses recherches sont actuellement conduites à ce niveau, notamment au sein de l'entreprise IBM.

Exemple 1 (Problème du sac à dos)**.** *Considérons un transporteur muni d'un sac (unique) pour transporter son butin. 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. Supposons qu'il y a* n **type d'objets** *que le transporteurpourrait emporter, et que ceux-ci sont en nombre tel que quelle que soit la nature de l'objet considéré, la seule limite au nombre d'unités que le transporteur peut prendre est que le* **poids total reste inférieur à** b*. 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*

$$\begin{aligned} \max_x \; & \sum_{j=1}^{n} c_j x_j \\ s.c. \; & \sum_{j=1}^{n} w_j x_j \leq b \\ & x_j \in \mathcal{N}, \; j = 1, \ldots, n. \end{aligned}$$

Ici, $x_j \in \mathcal{N}$ **signifie que** x_j **est un naturel**, *autrement dit un entier non négatif.* **Intuitivement, nous pourrions penser que la solution consiste à choisir en premier lieu les objets dont le rapport qualité-poids est le plus avantageux**, *quitte à* **tronquer** *pour obtenir une solution entière (nous ne pouvons pas diviser un objet). Cette solution peut malheureusement se révéler* **sous-optimale, voire mauvaise**, *comme on le constate sur l'exemple suivant.*

$$\begin{aligned} \max_x \; & 2x_1 + 3x_2 + 7x_3 \\ s.c. \; & 3x_1 + 4x_2 + 8x_3 \leq 14, \\ & x_1, \; x_2, \; x_3 \in \mathcal{N}. \end{aligned}$$

Si nous oublions la contrainte d'intégralité, la solution, de valeur 12.25, *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, par exemple* $x = (1, 0, 1)$.

La solution optimale du problème du transporteur peut s'obtenir en **énumérant toutes les solutions admissibles** *et en conservant la meilleure (voir Table 2.1, , 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

Table 2.1 – Problème du sac à dos : énumération des solutions

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**.

Exemple 2. *Une entreprise doit* **choisir de nouveaux emplacements pour construire des usines et des entrepôts**. *Elle a le choix entre deux emplacements : Oran (LA) et Constantine (SF). Nous ne pouvons construire un entrepôt que dans une* **ville où nous disposons d'une usine**, *et nous ne pouvons* **pas**

	Valeur estimée (millions \$)	Coût de construction (millions \$)
Usine à LA	9	6
Usine à SF	5	3
Entrepôt à LA	6	5
Entrepôt à SF	4	2
Limite maximum	-	10

Table 2.2 – Données du problème

construire plus d'un entrepôt. *Nous associons à chaque construction (d'une usine ou d'un entrepôt dans chacun des lieux envisagés)* **sa valeur estimée et son coût** *de construction. L'objectif est de* **maximiser la valeur totale estimée**, *en ne dépassant pas une limite maximum sur les coûts. Les données du problème sont résumées dans la Table 2.2. Les* **variables** *sont*

$$x_j = \begin{cases} 1 & \textit{si la décision } j \textit{ est approuvée ;} \\ 0 & \textit{si la décision } j \textit{ n'est pas approuvée.} \end{cases}$$

L'objectif est de **maximiser la valeur estimée totale** *:*

$$\max z = 9x_1 + 5x_2 + 6x_3 + 4x_4.$$

Les contraintes fonctionnelles sont

1. *la* **limite maximum sur les coûts** *de construction :*

$$6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10;$$

2. *nous ne pouvons construire* **plus d'un entrepôt** *:*

$$x_3 + x_4 \leq 1;$$

3. *l'entrepôt ne sera* **à LA que si l'usine est à LA** *:*

$$x_3 \leq x_1;$$

4. *l'entrepôt* **ne sera à SF que si l'usine est à SF** *:*

$$x_4 \leq x_2 :$$

5. *contraintes 0–1 (intégralité) :*

$$x_j \in \{0, 1\}, \;\; j = 1, 2, 3, 4;$$

ou encore

$$0 \leq x_j \leq 1 \textit{ et } x_j \textit{ entier}, \;\; j = 1, 2, 3, 4.$$

Par conséquent, nous avons le **programme mathématique**

$$\begin{aligned} max\ z &= 9x_1 + 5x_2 + 6x_3 + 4x_4 \\ s.c.\ & 6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10; \\ & x_3 + x_4 \leq 1; \\ & -x_1 + x_3 \leq 0; \\ & -x_2 + x_4 \leq 0; \\ & x_1, x_2, x_3, x_4 \leq 1; \\ & x_1, x_2, x_3, x_4 \geq 0; \\ & x_1, x_2, x_3, x_4 \ \textit{entiers}. \end{aligned}$$

Le modèle illustre deux cas classiques d'illustrations de **variables binaires** *:*

- *alternatives* **mutuellement exclusives** *: nous ne pouvons construire plus d'un entrepôt, i.e.*

$$x_3 + x_4 \leq 1;$$

- **décisions contingentes** *: nous ne pouvons construire un entrepôt que là où nous avons construit une usine :*

$$x_3 \leq x_1; x_4 \leq x_2 :$$

Exemple 3 (Localisation). *Une entreprise envisage plusieurs sites de construction pour des usines qui serviront à approvisionner ses clients. A 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*

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

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**.

Exemple 4 (**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**.

contrainte logique	**forme algébrique**
$p_1 \oplus p_2 = vrai$	$x1 + x2 = 1$
$p_1 \vee p_2 \vee \cdots \vee p_n = vrai$	$x_1 + x_2 + \ldots + x_n \geq 1$
$p_1 \wedge p_2 \wedge \ldots \wedge p_n = vrai$	$x_1 + x_2 + \ldots + x_n \geq n$ *(ou* $= n$*)*
$p_1 \Rightarrow p_2$	$x_2 \geq x_1$
$p_1 \Leftrightarrow p_2$	$x_2 = x_1$

Exemple 5 (**Fonctions linéaires par morceaux**). *Considérons une fonction objectif à maximiser, pour laquelle dans chaque* **intervalle** $[a_{i-1}, a_i]$ **la fonction est linéaire**, *ce que nous pouvons exprimer par :*

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

Car, il est bien établi que les points d'un segment d'extrémités A et B :

$$[A, B] = \{(1-t)A + tB | t \in [0, 1]\}.$$

Nous pouvons 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** $[a_{i-1}, a_i]$ *:*

$$\begin{aligned}
x &= \sum_{i=0}^{n} \lambda_i a_i, \\
f(x) &= \sum_{i=0}^{n} \lambda_i f(a_i), \\
\sum_{i=0}^{n} \lambda_i &= 1, \\
\lambda_i &\geq 0,\ i = 0, \ldots, n \\
\lambda_0 &\leq y_1, \\
\lambda_1 &\leq y1 + y2, \\
\vdots\ \vdots &\quad\ \vdots \\
\lambda_{n-1} &\leq y_{n-1} + y_n, \\
\lambda_n &\leq y_n, \\
\sum_{i=1}^{n} y_i &= 1 \textit{(un seul intervalle ``actif'')} \\
y_i &\in 0, 1,\ i = 1, \ldots, n.
\end{aligned}$$

2.2 Contraintes mutuellement exclusives [Bastin, 2010]

2.2.1 Deux contraintes

Prenons l'exemple de deux contraintes. **L'une ou l'autre des deux contraintes doit être satisfaite, mais pas les deux simultanément.** Par exemple,

- soit $3x_1 + 2x_2 \leq 18$;
- soit $x_1 + 4x_2 \leq 16$.

Soit M **un très grand nombre** ; le système précédent **est équivalent à**

- soit

$$3x_1 + 2x_2 \leq 18,$$
$$x_1 + 4x_2 \leq 16 + M;$$

- soit

$$3x_1 + 2x_2 \leq 18 + M,$$
$$x_1 + 4x_2 \leq 16.$$

En introduisant une variable binaire y, nous obtenons le **système équivalent**

$$3x_1 + 2x_2 \leq 18 + M(1 - y),$$
$$x_1 + 4x_2 \leq 16 + My.$$

La signification de cette variable est

- $y = 1$, si la première contrainte est satisfaite ;
- $y = 0$, si la seconde contrainte est satisfaite.

Nous avons de la sorte construit deux alternatives mutuellement exclusives.

Nous aurions pu aussi introduire deux variables binaires :

- $y_1 = 1$, si la première contrainte est satisfaite ;
- $y_2 = 1$, si la seconde contrainte est satisfaite.

Nous devons avoir

$$y_1 + y_2 = 1.$$

Afin de se ramener au modèle précédent, il suffit de poser

$$y_1 = y;$$
$$y_2 = 1 - y.$$

Il s'agit d'un cas particulier de la situation suivante : K parmi N contraintes doivent être satisfaites. Dans ce cas plus général, nous introduisons N variables binaires.

2.2.2 K contraintes parmi N

Soit les N **contraintes**

$$f_j(x_1, x_2, \ldots, x_n) \leq d_j, \;\; j = 1, 2, \ldots, N.$$

Nous introduisons N variables binaires, avec $y_j = 1$ **si la j^e contrainte est satisfaite** :

$$f_j(x_1, x_2, \dots, x_n) \leq d_j + M(1 - y_j),\ j = 1, 2, \dots, N.$$

Il reste à spécifier que seulement K **de ces contraintes peuvent êtres satisfaites** :

$$\sum_{j=1}^{N} y_j = K.$$

2.2.3 Fonction ayant N valeurs possibles

Soit la **contrainte**

$$f(x_1, x_2, \dots, x_n) = d_1, \text{ ou } d_2 \text{ ou } \dots \text{ ou } d_N.$$

Nous introduisons N variables binaires, avec $y_j = 1$ si la fonction vaut d_j. La contrainte s'écrit alors

$$f(x_1, x_2, \dots, x_n) = \sum_{j=1}^{N} d_j y_j,$$

avec

$$\sum_{j=1}^{N} y_j = 1.$$

Exemple 6 (Wyndor Glass)**.** *Supposons que le temps de production maximum à l'usine 3 n'est pas toujours 18h, mais pourrait également être 6h ou 12h. Cette contrainte s'écrit alors*

$$3x_1 + 2x_2 = 6 \textit{ ou } 12 \textit{ ou } 18.$$

Nous introduisons alors trois variables binaires

$$3x_1 + 2x_2 = 6y_1 + 12y_2 + 18y_3,$$
$$y_1 + y_2 + y_3 = 1.$$

2.2.4 Objectif avec coûts fixes

Supposons que le coût associé à un produit j est composé de deux parties :

1. un coût fixe initial k_j encouru dès qu'une unité de j est produite ;
2. un coût c_j proportionnel au nombre d'unités de j produites.

Le coût total associé à la production de x_j unités est

$$f(x_j) = \begin{cases} k_j + c_j x_j & \text{si } x_j > 0, \\ 0 & \text{si } x_j = 0. \end{cases}$$

Supposons de plus que l'objectif consiste à minimiser la **somme de n fonctions avec coûts fixes**

$$\min z = \sum_{j=1}^{n} f_j(x_j).$$

Nous introduisons alors n variables binaires :

$$y_j = \begin{cases} 1 \text{ si } x_j > 0, \\ 0 \text{ si } x_j = 0. \end{cases}$$

L'objectif s'écrit alors

$$\min z = \sum_{j=1}^{n} c_j x_j + k_j y_j.$$

Les valeurs de x_j et de y_j dépendent l'une de l'autre : il s'agit d'un exemple de *décisions contingentes*. Nous devons en particulier avoir une contrainte qui précise que $x_j = 0$ **si** $y_j = 0$. Toutefois, les deux variables ne sont plus binaires, vu que x_j peut être quelconque. Soit M_j **une borne supérieure sur la valeur de** x_j. Nous pouvons écrire la **relation entre les deux variables** de cette manière :

$$x_j \leq M_j y_j.$$

Ainsi,
- si $y_j = 0$, alors $x_j = 0$;
- si $y_j = 1$, alors $x_j \leq M_j$;
- si $x_j > 0$, alors $y_j = 1$;
- si $x_j = 0$, alors toute soluton optimale satisfait $y_j = 0$ lorsque $k_j > 0$ (si $k_j = 0$, la variable y_j est inutile).

Nous obtenons par conséquent le programme

$$\begin{aligned} \min\ z = & \sum_{j=1}^{n} c_j x_j + k_j y_j \\ \text{s.c. } & x_j \leq M_j y_j, \\ & y_j \in \{0,1\},\ j = 1, 2, \ldots, n. \end{aligned}$$

2.2.5 Variables entières en variables 0–1

Soit x une **variable entière générale bornée** :

$$0 \leq x \leq u,$$

et soit N l'entier **tel que** $2^N \leq u \leq 2^{N+1}$. La **représentation binaire de** x est

$$x = \sum_{j=0}^{N} 2^j y_j.$$

L'intérêt de cette transformation est que les méthodes de programmation 0–1 sont souvent plus efficaces que les méthodes de programmation en nombres entiers. Elle engendre néanmoins une **multiplication du nombre de variables**.

Exemple 7. *Nous considérons* **trois types de produits, et deux usines** *; nous exprimons le* **profit par unité de produit** *en milliers de dollars. Nous connaissons les* **ventes potentielles par produit** *(unités/semaine), et la* **capacité de**

	Produit 1 temps de production (h/unité)	Produit 2 temps de production (h/unité)	Produit 3 temps de production (h/unité)	Capacité de production production (h/semaine)
Usine 1	3	4	2	30
Usine 2	4	6	2	40
Profit/unité (1000$)	5	7	3	–
Ventes potentielles (par semaine)	7	5	9	–

TABLE 2.3 – Exemple de production avec variables entières.

production par usine *(h/semaine). Nous avons toutefois comme contrainte que* **pas plus de deux produits ne peuvent être fabriqués**, *et* **une seule des deux usines doit être exploitée**. *Les données du problème sont résumées dans la Table 2.3. Les variables sont* x_j, **le nombre d'unités fabriquées du produit** j. *Pour représenter la* **contrainte "pas plus de deux produits"**, *nous devons introduire des variables 0–1 :*

$$y_j = \begin{cases} 1 & si\ x_j > 0; \\ 0 & si\ x_j = 0. \end{cases}$$

Afin de représenter la contrainte **"une seule des deux usines"**, *nous devons ajouter une variables 0–1 :*

$$y_4 = \begin{cases} 1 & si\ l'usine\ 1\ est\ choisie; \\ 0 & si\ sinon. \end{cases}$$

*L'***objectif** *est*

$$\max z = 5x_1 + 7x_2 + 3x_3.$$

Les **ventes potentielles** *sont*

$$x_1 \leq 7,\ \ x_2 \leq 5,\ \ x_3 \leq 9.$$

L'exigence **interdisant d'avoir plus de deux produits** *se traduit mathématiquement par*

$$y_1 + y_2 + y_3 \leq 2.$$

La **relation entre les variables continues et les variables 0–1** *s'exprime par*

$$x_1 \leq 7y_1,\ \ x_2 \leq 5y_2,\ \ x_3 \leq 9y_3.$$

La contrainte portant sur l'utilisation d'une seule usine est

- *soit* $3x_1 + 4x_2 + 2x_3 \leq 30$;
- *soit* $4x_1 + 6x_2 + 2x_3 \leq 40$.

En utilisant la variable 0–1 (et M très grand), elle se traduit par

$$3x_1 + 4x_2 + 2x_3 \leq 30 + M(1 - y_4),$$
$$4x_1 + 6x_2 + 2x_3 \leq 40 + My_4.$$

En résumé, nous avons le modèle

$$\begin{aligned}
\max z &= 5x_1 + 7x_2 + 3x_3 \\
s.c.\ & x_1 \leq 7y_1,\ x_2 \leq 5y_2,\ x_3 \leq 9y_3 \\
& y_1 + y_2 + y_3 \leq 2, \\
& 3x_1 + 4x_2 + 2x_3 \leq 30 + M(1 - y_4), \\
& 4x_1 + 6x_2 + 2x_3 \leq 40 + My_4, \\
& x_1, x_2, x_3 \geq 0, \\
& y_j \in \{0, 1\}, j = 1, 2, 3, 4.
\end{aligned}$$

Exemple 8. **Nous considérons à nouveau trois types de produits, pour lesquels nous pouvons placer cinq annonces publicitaires, avec un maximum de trois annonces par produit.** *L'estimation des revenus publicitaires est donnée dans la Table 2.4, où les profits sont exprimés en millions de dollars. Les*

Nombre d'annonces	Produit 1	Produit 2	Produit 3
0	0	0	0
1	1	0	-1
2	3	2	2
3	3	3	4

Table 2.4 – Revenus publicitaires.

variables du problèmes sont le nombre d'annonces pour le produit i, dénoté par x_i, mais l'hypothèse de proportionnalité est alors violée : nous ne pouvons représenter l'objectif sous forme linéaire uniquement avec ces variables.

Prenons tout d'abord comme variables

$$y_{ij} = \begin{cases} 1 & si\ x_i = j; \\ 0 & sinon. \end{cases}$$

L'objectif est

$$\max z = y_{11} + 3y_{12} + 3y_{13} + 2y_{22} + 3y_{23} - y_{31} + 2y_{32} + 4y_{33}.$$

Nous utiliserons les **5 annonces disponibles** *:*

$$\sum_{i=1}^{3} \sum_{j=1}^{3} j y_{ij} = 5.$$

Enfin, on a droit à **un seul type d'annonce par produit, la définition des variables 0–1** *donne*

$$\sum_{j=1}^{3} y_{ij} \leq 1, \ \ i = 1, 2, 3.$$

Soit une autre modélisation *en prenant comme variables*

$$y_{ij} = \begin{cases} 1 & si\ x_i \geq j; \\ 0 & sinon. \end{cases}$$

Autrement dit, nous avons remplacé l'égalité dans la première condition par une inégalité. Cette définition implique

$$\begin{aligned} x_i = 0 &\Rightarrow y_{i1} = 0, y_{i2} = 0, y_{i3} = 0; \\ x_i = 1 &\Rightarrow y_{i1} = 1, y_{i2} = 0, y_{i3} = 0; \\ x_i = 2 &\Rightarrow y_{i1} = 1, y_{i2} = 1, y_{i3} = 0; \\ x_i = 3 &\Rightarrow y_{i1} = 1, y_{i2} = 1, y_{i3} = 1. \end{aligned}$$

Ce qui peut encore s'énoncer somme

$$y_{i(j+1)} \leq y_{ij}, \ \ i = 1, 2, 3, \ \ j = 1, 2.$$

Supposons que $x_1 = 3$ (il y a trois annonces pour le produit 1). *Le profit associé à cette valeur doit être 3. Mais $x_1 = 3$* **veut aussi dire que chaque variable binaire associée au produit 1 vaut 1** *; comment dès lors comptabiliser correctement la contribution de ces trois variables au profit ?* **La solution consiste à prendre comme profit associé à la variable y_{ij} la différence $c_{ij+1} - c_{ij}$**, *où c_{ij} est le revenu net si nous plaçons j annonce pour le produit i. Dans notre exemple, le profit associé à*

— y_{11} *est 1-0 = 1 ;*
— y_{12} *est 3-1 = 2 ;*
— y_{13} *est 3-3 = 0 ;*

Nous obtenons ainsi le programme mathématique suivant :

$$\begin{aligned} \max\ z = y_{11} + 2y_{12} + 2y_{22} + y_{23} - y_{31} + 3y_{32} + 2y_{33} \\ s.c.\ y_{i(j+1)} \leq y_{ij}, i = 1, 2, 3, \ \ j = 1, 2, \\ \sum_{i=1}^{3}\sum_{j=1}^{3} y_{ij} = 5, \\ y_{ij} \in \{0, 1\}, \ \ i = 1, 2, 3, \ \ j = 1, 2. \end{aligned}$$

2.2.6 Problème de recouvrement

Exemple 9 (**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**. *Comme il y a un coût*

associé à chaque séquence de vols, *la compagnie cherche à* **minimiser les coûts d'affectation des équipages aux séquences** *tout en assurant le service sur chacun des vols.*

Considérons par exemple un problème avec 11 vols et 12 séquences de vols, dont les données sont décrites dans la Table 2.5. Les variables sont

Vol | Séquence	1	2	3	4	5	6	7	8	9	10	11	12
1	1			1			1			1		
2		1			1			1			1	
3			1			1			1			1
4				1			1		1	1		1
5	1					1				1	1	
6				1	1				1			
7							1	1		1	1	1
8		1		1	1				1			
9					1			1			1	
10			1				1	1				1
11						1			1	1	1	1
Coût	2	3	4	6	7	5	7	8	9	9	8	9

TABLE 2.5 – Affectation d'équipages.

$$x_j = \begin{cases} 1 & \textit{si la séquence de vols } j \textit{ est affectée;} \\ 0 & \textit{sinon.} \end{cases}$$

L'objectif est

$$\min z = 2x_1 + 3x_2 + 4x_3 + 6x_4 + 7x_5 + 5x_6 + 7x_7 + 8x_8 + 9x_9 + 9x_{10} + 8x_{11} + 9x_{12}.$$

Nous devons affecter **trois équipages**

$$\sum_{j=1}^{12} x_j = 3.$$

Le service doit être **assuré sur chacun des vols** *:*

$$\begin{aligned} x_1 + x_4 + x_7 + x_{10} &\geq 1 \\ x_2 + x_5 + x_8 + x_{11} &\geq 1 \\ x_3 + x_6 + x_9 + x_{12} &\geq 1 \\ x_4 + x_7 + x_9 + x_{10} + x_{12} &\geq 1 \\ &\dots \end{aligned}$$

Généralement, un **problème de recouvrement d'ensemble** met en oeuvre

— I : un ensemble d'objets (les vols dans l'exemple précédent) ;

— $\mathcal{J}$: une collection de sous-ensembles de I (e.g. les séquences de vols) ;

— J_i, $i \in I$: les sous-ensembles dans $\mathcal{J}$ qui contiennent i.

Nous avons les variables binaires x_j, **prenant la valeur 1 si le sous-ensemble j est choisi, et 0 sinon**. En considérant un objectif linéaire, avec c_j le coût associé au sous-ensemble j. Nous obtenons le programme

$$\begin{aligned}
&\min_x \sum_{j \in \mathcal{J}} c_j x_j \\
&\text{s.c.} \ \sum_{j \in J_i} x_j \geq 1, \ i \in I; \\
&\qquad x_j \in \{0, 1\}, \ j \in \mathcal{J}.
\end{aligned}$$

2.3 Stratégies de résolutions[Bastin, 2010]

2.3.1 Relaxation linéaire

Il est tentant "d'oublier" les contraintes d'intégralité, et de résoudre le problème en nombres continus ainsi produit. Nous parlerons alors de *relaxation*. Ainsi, nous pourrons construire la relaxtion en programme linéraire d'un programme mixte entier. **Une fois le programme relâché résolu, nous pourrions arrondir aux valeurs entières les plus proches.** Dans certains cas, cela peut fonctionner, mais l'exemple du sac à dos montre que **ce n'est pas toujours vérifié**. Cette méthode par arrondissement est même **parfois désastreuse**.

Exemple 10. *Considérons le programme*

$$\begin{aligned}
&\max z = x_2 \\
&s.c. \ -x_1 + x_2 \leq \frac{1}{2}, \\
&\qquad x_1 + x_2 \leq \frac{7}{2}, \\
&\qquad x_1, x_2 \geq 0 \ \text{et entiers.}
\end{aligned}$$

La relaxation en programme linéaire donne

$$\begin{aligned}
&\max z = x_2 \\
&s.c. \ -x_1 + x_2 \leq \frac{1}{2}, \\
&\qquad x_1 + x_2 \leq \frac{7}{2}, \\
&\qquad x_1, x_2 \geq 0.
\end{aligned}$$

Ce nouveau programme a pour **solution** $\left(\frac{3}{2}, 2\right)$. *Que nous arrondissions cette solution à* $(1, 2)$ *ou* $(2, 2)$, **nous n'obtenons pas de solution réalisable**, *comme illustré sur la Figure 2.1*

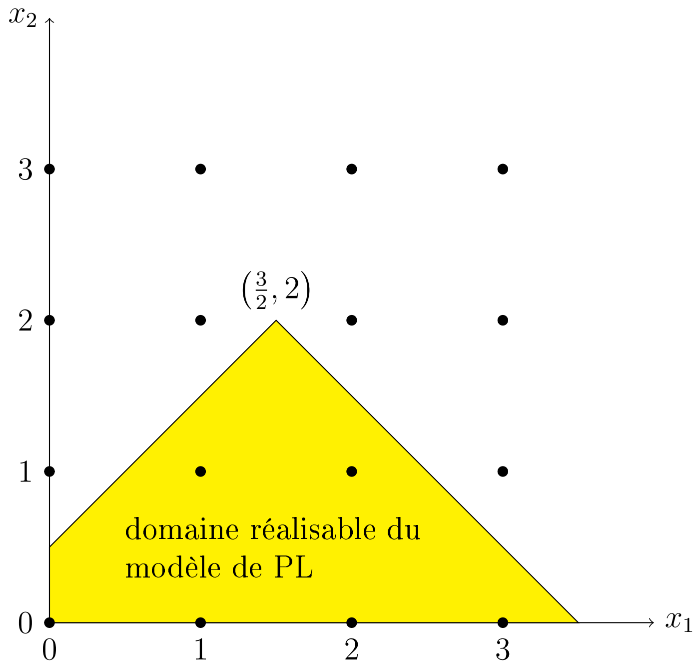


FIGURE 2.1 – Exemple de relaxation linéaire : problème d'admissibilité des solutions

Exemple 11. *Considérons le programme*

$$\begin{aligned} \max\ z &= x_1 + 5x_2 \\ s.c.\ x_1 + 10x_2 &\leq 20, \\ x_2 &\leq 2, \\ x_1, x_2 &\geq 0 \text{ et entiers.} \end{aligned}$$

La version relâchée de programme est

$$\begin{aligned} \max\ z &= x_1 + 5x_2 \\ s.c.\ x_1 + 10x_2 &\leq 20, \\ x_2 &\leq 2, \\ x_1, x_2 &\geq 0, \end{aligned}$$

qui a pour solution optimale $(2, 1.8)$. **En arrondissant à $(2, 1)$ afin de garantir l'admissibilité, nous obtenir la valeur 7 pour la fonction objectif, loin de la valeur optimale du programme mixte entier, avec pour valeur optimale 11**, *en* $(0, 2)$ *(voir Figure 2.2).*

La solution de la relaxation linéaire ne peut donc être exploité directement pour obtenir la solution exacte du problème en nombres entiers.

Cependant, la relaxation linéaire a les propriétés suivantes :

Borne supérieure La valeur de la solution optimale de la relaxation est une borne supérieure sur la valeur de la solution optimale du problème de maximisation en nombres entiers. C'est une borne inférieur sur un problème de minimisation.

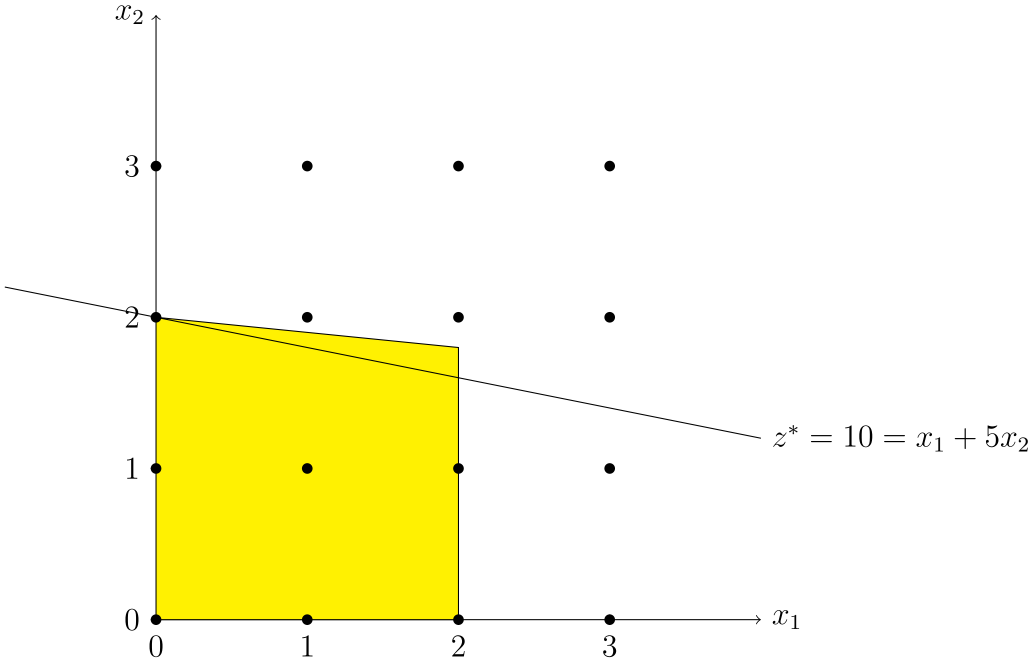


FIGURE 2.2 – Exemple de relaxation linéaire : solution médiocre

Solution entière Si la solution optimale de la relaxation est entière (donc admissible pour le problème en nombres entiers), elle est également la solution optimale du problème en nombres entiers.

Borne inférieure La valeur d'une solution admissible/faisable du problème en nombres entiers de maximisation fournit une borne inférieure sur la valeur de la solution optimale de ce problème en nombres entiers. C'est une borne supérieur sur un problème de minimisation.

D'une façon globale toute méthode de résolution qui garantit ces propriétés peut servir comme relaxation.

2.3.2 Approche par énumération

Un modèle en nombres entiers borné (par exemple, un modèle avec uniquement des variables 0–1) possède un nombre fini de solutions. Nous pourrions **envisager de toutes les énumérer**, mais le nombre de solutions explose rapidement. Pour $n = 20$ variables 0–1, il y a **plus d'un million de solutions possibles**. **Pour $n = 30$, c'est plus d'un milliard**. Comme il apparaît qu'il est vite déraisonnable de procéder à une énumération complète des solutions, nous allons essayer de mettre à profit la relaxation en programme linéaire pour éliminer certaines de ces solutions. Cette technique d'énumération partielle est connue sous le vocable de *branch-and-bound* **(B&B)**. Il s'agit d'une **approche** *diviser-pour-régner* :

- décomposition du problème en sous-problèmes plus simples ;
- combinaison de la résolution de ces sous-problèmes pour obtenir la solution du problème original.

Dans l'algorithme de branch-and-bound, chaque sous-problème correspond à un sommet dans l'arbre des solutions. Nous résolvons la relaxation linéaire de chaque sous-problème. L'information tirée de la relaxation linéaire nous permettra (peut-être) d'éliminer toutes les solutions pouvant être obtenues à partir de ce sommet.

2.3.2.1 Algorithme de branch & bound : cas 0–1

Un algorithme simple pour énumérer toutes les solutions d'un modèle 0–1 consiste à :

- choisir un sommet dans l'arbre des solutions ;
- choisir une variable x non encore fixée relativement à ce sommet ;
- générer les deux variables $x = 0$ et $x = 1$ (la variable x est dite *fixée*) : chaque alternative correspond à un sommet de l'arbre des solutions ;
- recommencer à partir d'un sommet pour lequel certaines variables ne sont pas encore fixées.

A la racine de l'arbre, aucune variable n'est encore fixée, tandis qu'aux feuilles de l'arbre, toutes les variables ont été fixées. Le nombre de feuilles est 2^n (pour n variables 0–1).

Le calcul de borne (ou évaluation) consiste à résoudre la relaxation linéaire en chaque sommet. L'élagage (ou élimination) consiste à utiliser l'information tirée de la résolution de la relaxation linéaire pour éliminer toutes les solutions émanant du sommet courant. Dés lors, le branch-and-bound est un algorithme de séparation et d'évaluation successives.

Exemple 12 (California Mfg)**.** *Rappelons le problème*

$$
\begin{aligned}
max\ z &= 9x_1 + 5x_2 + 6x_3 + 4x_4\\
s.c.\ & 6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10;\\
& x_3 + x_4 \leq 1;\\
& -x_1 + x_3 \leq 0;\\
& -x_2 + x_4 \leq 0;\\
& x_1, x_2, x_3, x_4 \leq 1;\\
& x_1, x_2, x_3, x_4 \geq 0;\\
& x_1, x_2, x_3, x_4 \ \textit{entiers.}
\end{aligned}
$$

La relaxation linéaire permet aux variables de prendre les valeurs fractionnaires entre 0 et 1, ce qui conduit à la solution

$$\left(\frac{5}{6}, 1, 0, 1\right),$$

avec comme valeur $z = 16.5$*. Branchons sur la variable* x_1*.*

Dénotons sous-problème 1 celui obtenu avec $x_1 = 0$ *:*

$$\begin{aligned} max\ z &= 5x_2 + 6x_3 + 4x_4 \\ s.c.\ & 3x_2 + 5x_3 + 2x_4 \leq 10; \\ & x_3 + x_4 \leq 1; \\ & x_3 \leq 0; \\ & -x_2 + x_4 \leq 0; \\ & x_2, x_3, x_4 \ binaires. \end{aligned}$$

La solution de la relaxation linéaire est $(x_1, x_2, x_3, x_4) = (0, 1, 0, 1)$*, avec* $z = 9$*.*

Le sous-problème 2 est celui obtenu avec $x_1 = 1$ *:*

$$\begin{aligned} max\ z &= 5x_2 + 6x_3 + 4x_4 + 9 \\ s.c.\ & 3x_2 + 5x_3 + 2x_4 \leq 4; \\ & x_3 + x_4 \leq 1; \\ & x_3 \leq 1; \\ & -x_2 + x_4 \leq 0; \\ & x_2, x_3, x_4 \ binaires. \end{aligned}$$

La solution de la relaxation linéaire est alors

$$(x_1, x_2, x_3, x_4) = \left(1, \frac{4}{5}, 0, \frac{4}{5}\right),$$

avec $z = 16 + \frac{1}{5}$*.*

Nous obtenons dès lors les bornes suivantes :
— sous-problème 1 : $Z_1 \leq 9$;
— sous-problème 2 : $Z_2 \leq 16 + \frac{1}{5}$.

Notons que toutes les variables sont binaires et tous les paramètres dans l'objectif sont des valeurs entières. Dès lors, la borne supérieure pour le sous-problème 2 est 16. Pour le sous-problème 1, la solution obtenue est entiere : c'est la meilleure solution courante. Nous savons que la valeur optimale cherchée, Z, sera au moins

$$Z^* = 9 : Z \geq Z^*.$$

Quels sous-problèmes pouvons-nous à présent considérer afin de nous approcher de la solution optimale ? Tous les sous-problèmes actuellement traités doivent-ils donner naissance à d'autres problèmes. Si un sous-problème ne donne lieu à aucun autre problème, nous parlerons d'élagage, en référence avec l'idée de couper la branche correspondante dans l'arbre d'exploration.

Considérons tout d'abord le sous-probleme 1 : la solution optimale de la relaxation PL est entière. Il ne sert donc à rien de brancher sur les autres variables, puisque toutes les autres solutions entières (avec $x_1 = 0$) sont nécessairement de valeur inférieures ou égales à 9. Nous pouvons donc élaguer ce sommet.

Pour le sous-probleme 2, la solution optimale de la relaxation PL n'est pas entière :

$$Z^* = 9 \leq Z \leq 16.$$

La branche ($x_1 = 1$) peut encore contenir une solution optimale. Mais si nous avions eu $Z_2 \leq Z*$, nous aurions pu conclure que la branche ne pouvait améliorer la meilleure solution courante.

Un sous-problème est elagué si une des trois conditions suivantes est satisfaite :
— test 1 : sa borne supérieure (valeur optimale de la relaxation PL) est inférieure ou égale à Z^* (valeur de la meilleure solution courante) ;
— test 2 : sa relaxation PL n'a pas de solution réalisable ;
— test 3 : la solution optimale de sa relaxation PL est entière.

Lorsque le test 3 est verifié, nous testons si la valeur optimale de la relaxation PL du sous-problème, Z_i, est supérieure à Z^*. Si $Z_i > Z^*$, alors $Z^* := Z_i$, et nous conservons la solution, qui devient la meilleure solution courante. En résumé, nous obtenons l'algorithme ci-dessous.

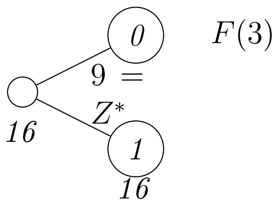
Algorithm 1. *Branch-and-Bound : cas binaire*

1. *Initialisation :*
   (a) *poser* $Z^* = -\infty$ *;*
   (b) *appliquer le calcul de borne et les critères d'élagage à la racine (aucune variable fixée).*
2. *Critere d'arrêt : s'il n'y a plus de sous-problemes non élagués, arrêter.*
3. *Branchement :*
   (a) *parmi les sous-problèmes non encore élagués, choisir celui qui a été crée le plus récemment (s'il y a égalité, choisir celui de plus grande borne supérieure) ;*
   (b) *appliquer le Test 1 : si le sous-problème est élagué, retourner en 2.*
   (c) *brancher sur la prochaine variable non fixée.*
4. *Calcul de borne :*
   (a) *résoudre la relaxation PL de chaque sous-problème ;*
   (b) *arrondir la valeur optimale si tous les paramètres de l'objectif sont entiers.*
5. *Elagage : élaguer un sous-problème si*
   (a) *la borne superieure est inférieure ou égale à* Z^* *;*
   (b) *la relaxation PL n'a pas de solution réalisable ;*
   (c) *la solution optimale de la relaxation PL est entière : si la borne supérieure est strictement supérieure à* Z^**,* Z^* *est mise à jour et la solution de la relaxation PL devient la meilleure solution courante.*
6. *Retourner en 2.*

A partir de quel noeud devrions-nous brancher ? Il y a plusieurs choix possibles ; dans cette version, on propose comme règle de sélection de choisir le sous-problème le plus récemment créé. L'avantage est que cette approche facilite la réoptimisation lors du calcul de borne, car il n'y a que peu de changements apportés par rapport au dernier sous-probleme traité. Le désavantage est que cela peut créer un grand nombre de sous-probèmes. Une autre option est la règle de la meilleure borne : choisir le sous-problème ayant la plus grande borne supérieure.

Dans cette version, la règle de branchement consiste à choisir la prochaine variable non fixée. Il est souvent plus intéressant de choisir une variable à valeur fractionnaire. En branchant sur une telle variable, il est certain que les deux sous-problèmes crées mènent à des solutions differentes de la solution courante. De nombreux critères existent pour choisir une telle variable de facon à orienter la recherche vers un élagage rapide.

Exemple 13 (suite)**.** *Jusqu'à maintenant, voici l'arbre obtenu, en branchant sur la variable* x_1*.*



$F(3)$ *indique que le sous-probleme a été élagué (*fathomed*) en raison du Test 3.*

Sélection : nous choisissons le sous-problème 2, le seul qui n'a pas encore été élagué. Nous branchons sur la prochaine variable, soit x_2*. Deux nouveaux sous-problèmes sont crées :*

— *sous-problème 3 :* $x_1 = 1,\ x_2 = 0$ *;*
— *sous-problème 4 :* $x_1 = 1,\ x_2 = 1$.

Considérons tout d'abord le sous-problème 3 ($x_1 = 1,\ x_2 = 0$*). Nous obtenons le problème*

$$\begin{aligned} \max z_3 &= 6x_3 + 4x_4 + 9 \\ s.c.\ 5x_3 + 2x_4 &\leq 4 \\ x_3 + x_4 &\leq 1 \\ x_3 &\leq 1 \\ x_4 &\leq 0 \\ x_3,\ x_4 &\ \textit{binaire.} \end{aligned}$$

La solution de la relaxation PL est

$$(x_1, x_2, x_3, x_4) = \left(1, 0, \frac{4}{5}, 0\right).$$

et

$$Z = 13 + \frac{4}{5} :\ Z_3 \leq 13.$$

Le sous-problème 4 ($x_1 = 1,\ x_2 = 1$*) devient*

$$\begin{aligned} \max z_4 &= 6x_3 + 4x_4 + 14 \\ s.c.\ 5x_3 + 2x_4 &\leq 1 \\ x_3 + x_4 &\leq 1 \\ x_3 &\leq 1 \\ x_4 &\leq 1 \\ x_3,\ x_4 &\ \textit{binaire.} \end{aligned}$$

La solution de la relaxation PL est

$$(x_1, x_2, x_3, x_4) = \left(1, 1, 0, \frac{1}{2}\right).$$

et

$$Z = 16 : \ Z_4 \leq 16.$$

Aucun des tests d'élagage ne s'applique sur ces sous-problèmes. Nous devons dès lors choisir un des deux sous-problèmes pour effectuer un branchement, puisque ce sont ceux crées le plus récemment. Nous choisissons celui de plus grande borne supérieure, soit le sous-problème 4. Nous branchons sur x_3 *et nous générons deux nouveaux sous-problèmes.*

Le sous-problème 5, défini avec $x_1 = 1$, $x_2 = 1$, $x_4 = 0$, *s'écrit La solution de la relaxation PL est*

$$(x_1, x_2, x_3, x_4) = (1, 1, 0.2, 0)$$

et

$$Z = 14.$$

Le sous-problème 6, défini avec $x_1 = 1$, $x_2 = 1$, $x_4 = 1$, *s'écrit La relaxation PL n'a pas de solution realisable : ce sous-probleme est élagué.*

Le sous-problème 5 ne peut pas être élagué. Il est crée le plus récemment parmi les sous-problèmes non élagués (3 et 5), aussi choisissons-nous le pour effectuer un branchement. Nous branchons sur x_3 *et générons les sous-problèmes suivants :*

— *sous-problème 7 :* $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$;
— *sous-problème 8 :* $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $x_4 = 0$.

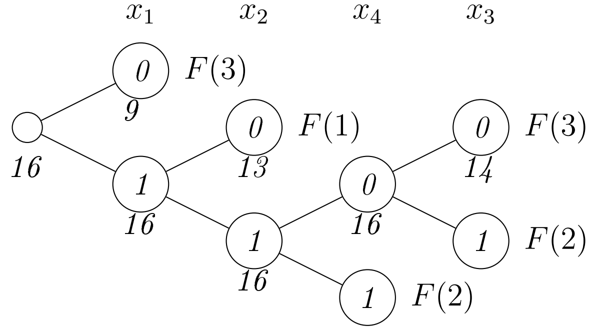
Toutes les variables sont fixées, aussi pouvons-nous directement résoudre ces sous-problèmes. Le sous-problème 7 a pour solution $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$, *pour* $Z_7 = 14$. *La solution est entière, aussi nous élaguons le sous-problème en vertu du Test 3. Puisque* $Z_7 > Z^*$, $Z^* = 14$ *et la solution du sous-problème devient la meilleure solution courante. Le sous-problème 8 a pour solution* $(x_1, x_2, x_3, x_4) = (1, 1, 1, 0)$. *Cette solution n'est pas réalisable. Le sous-problème est par conséquent élagué par le Test 2.*

Le sous-problème 3 est le seul non encore élagué. Appliquons le Test 1 : $Z_3 = 13 \leq 14 = Z*$. *Le sous-problème est donc élagué. Comme il n'y a plus de sous-problèmes non élagués, nous pouvons nous arrêter. La solution optimale est :*

$$(x_1, x_2, x_3, x_4) = (1, 1, 0, 0),$$

et la valeur optimale est $Z^* = 14$.

L'arbre obtenu suite à l'exécution de l'algorithme se présente comme suit :



$F(j)$: le sous-problème est élagué par le Test j

2.3.2.2 Algorithme de branch & bound : cas général

Considérons à présent le cas général d'un modèle de programmation (mixte) en nombres entiers : variables entières générales et variables continues. Comme précédemment, nous ignorons dans un premier temps les contraintes d'intégralité (les valeurs des variables entières sont traitées comme variables continues), et résolvons le programme linéaire résultant. 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. La procédure de branchement se généralise alors comme suit : nous séparons alors le problème relaxé en deux sous-problèmes ; un sous-problème contiendra la contrainte $x_j \leq \lfloor\alpha\rfloor$ et le second la contrainte $x_j \geq \lceil\alpha\rceil = \lfloor\alpha\rfloor + 1$. Nous répétons 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.

Algorithm 2. *Algorithme de B&B : cas général*

1. *Initialisation :*
   - (a) *Poser $Z^* = -\infty$.*
   - (b) *Appliquer le calcul de borne et les critères d'élagage à la racine (aucune variable fixée).*
   - (c) *Critere d'arrêt : s'il n'y a plus de sous-problèmes non élagués, arrêter.*
2. *Branchement :*
   - (a) *Parmi les sous-problèmes non encore élagués, choisir celui qui a été crée le plus récemment (s'il y a égalité, choisir celui de plus grande borne supérieure).*
   - (b) *Appliquer le Test 1 : si le sous-problème est élagué, retourner en 2.*
   - (c) *Brancher sur la prochaine variable entière à valeur non entière dans la relaxation PL.*
3. *Calcul de borne : résoudre la relaxation PL de chaque sous-problème.*
4. *Elagage : élaguer un sous-problème si*

(a) La borne superieure est inférieure ou égale à Z^*.

(b) La relaxation PL n'a pas de solution réalisable.

(c) Dans la solution optimale de la relaxation PL, toutes les variables entières sont à valeurs entières : si la borne supérieure est strictement supérieure à Z^*, Z^* est mise à jour et la solution de la relaxation PL devient la meilleure solution courante.

5. Retourner en 2.

Un sous-problème est elagué si une des trois conditions suivantes est satisfaite :
- test 1 : sa borne supérieure (valeur optimale de la relaxation PL) est inférieure ou égale à Z^* (valeur de la meilleure solution courante) ;
- test 2 : sa relaxation PL n'a pas de solution réalisable ;
- test 3 : la solution optimale de sa relaxation PL est entière.

2.4 Branch and bound : exemple

Un programme linéaire mixte (MIP) se présente comme suit :

$$(MIP) \equiv \begin{cases} \text{minimize} & cx \\ \text{subject to} & Ax \leq b \\ & x_i \in \mathbb{Z}, i = 1..p \\ & x_i \in \mathbb{R}, i = p+1..n \end{cases} \tag{2.1}$$

Le problème : Comment le résoudre correctement et rapidement ?

Branch and Bound (BB) Diviser pour régner. "Diviser/Branch" : subdivise le problème. "Régner/Bound" : considérer la qualité de la solution des sous-problèmes.

Branch and Cut (BC) BB en renforçant la relaxation linéaire LP d'un MIP avec des inégalités avant tout branchement. On raisonne sur les contraintes !

Branch and Price (BP) BB en se concentrant sur le choix de la variable entrante (column generation/pricing) dans la résolution de la relaxation. On raisonne sur les variables (les colonnes) !

Branch and bound
- Diviser le problème en sous-problèmes
- Calculer la relaxation LPR du sous-problème en considérant "réelles" les variables entières
  - LPR infaisable : stop.
  - LPR a une solution faisable entière : résolu, solution optimale pour le sous-problème.
  - LPR a une solution moins bonne que la meilleure solution entière : stop.
  - Sinon LPR a des composantes réelles, Diviser en sous-problèmes.

Soit le problème MIP [1] :

1. http ://www.ie.bilkent.edu.tr/ mustafap/courses/bb.pdf

$$\begin{array}{ll} \text{maximize} & z = -x_1 + 4x_2 \\ \text{subject to} & -10x_1 + 20x_2 \leq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & x_1 \leq 5 \\ & x_i \geq 0, x_1 \text{ et } x_2 \text{ sont entiers.} \end{array} \tag{2.2}$$

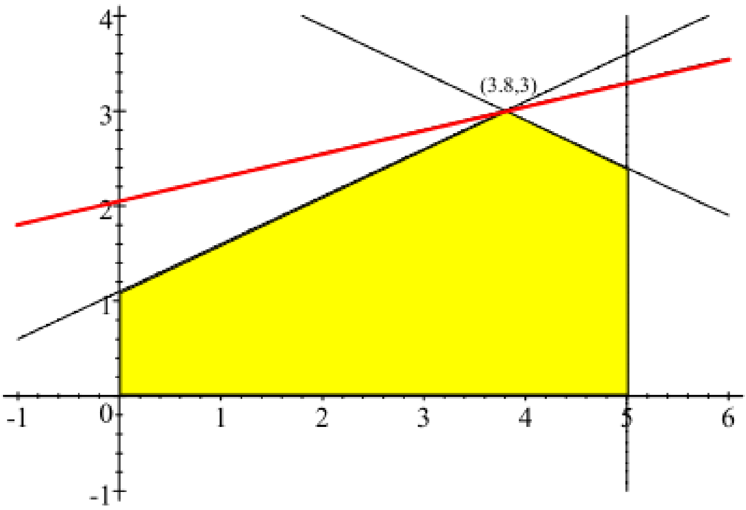
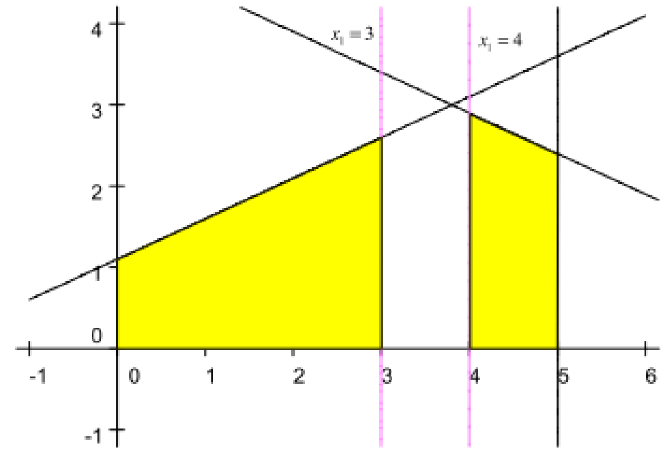


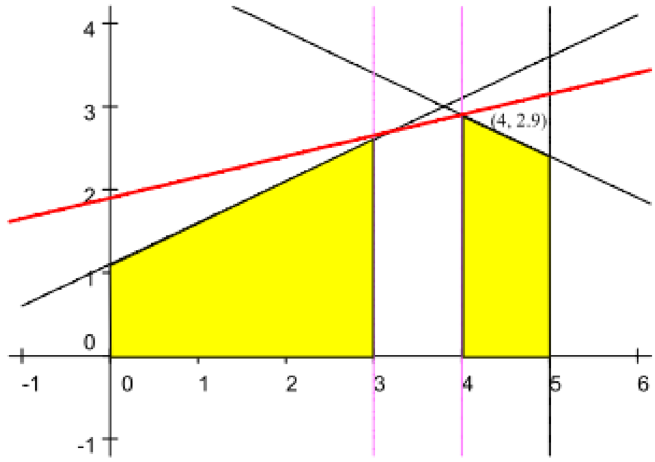
FIGURE 2.3 – Relaxation linéaire (2.3)

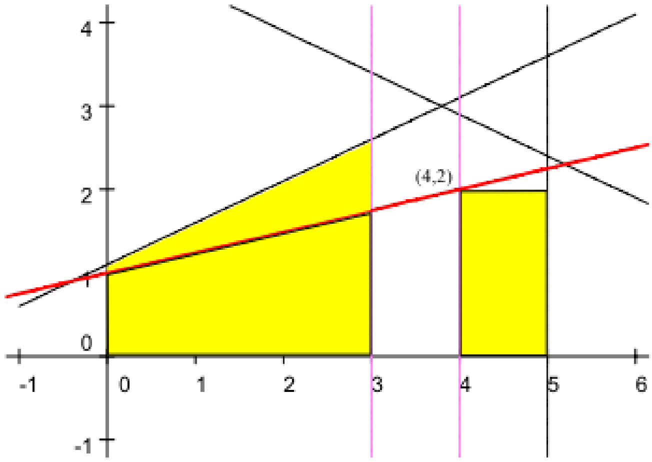
Au niveau de la relaxation

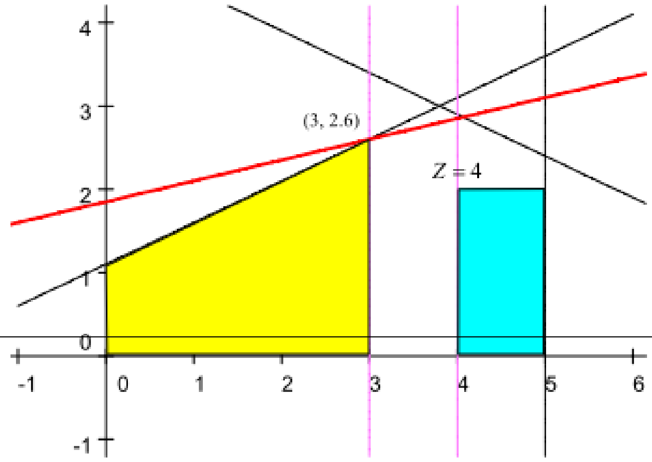
$$\begin{array}{ll} \text{maximize} & z = -x_1 + 4x_2 \\ \text{subject to} & -10x_1 + 20x_2 \leq 22 \\ & 5x_1 + 10x_2 \leq 49 \\ & x_1 \leq 5 \\ & x_i \geq 0, x_1 \text{ et } x_2 \text{ sont réelles.} \end{array} \tag{2.3}$$

La relaxation (2.3) est illustrée dans la Figure 2.3. **Sa solution est $(3.8, 3)$ avec $z = 8.2$.** Nous devons faire un "Branch" sur la variable fractionnaire x_1, en obtenant deux branches $x_1 \geq 4$ et $x_1 \leq 3$. Cette séparation (Branch) est illustrée dans la première image de la Figure 2.4.


$x_1 = 3$
$x_1 = 4$


(4, 2.9)


(4,2)


(3, 2.6)
$Z = 4$

Par la suite la procédure Branch and Bound procède comme suit :

1. Le sous-problème correspondant à $x_1 \geq 4$ est résolu sur $I\!R$. On obtient la solution $(4, 2.9)$ avec $z = 7.6$. On obtient ainsi deux sous-sous-problèmes $x_2 \geq 3$ et $x_2 \leq 2$.

2. Le sous-problème $x_2 \geq 3$ est infaisable.

3. Nous considérons donc maintenant le sous-problème $x_2 \leq 2$. Ce dernier a une solution optimale (donnée dans la deuxième image de la Figure 2.4, qui a comme solution $(4, 2)$ avec $z = 4$. Cette première solution va permettre d'avoir une meilleure borne (inférieure) $z^* = 4$ de notre problème avec une valeur $z = 4$.

4. En ce qui concerne la branche $x_1 \leq 3$, on obtient la solution $(3, 2.6)$ avec $z = 7.4$, donnant lieu à deux branches $x_2 \leq 2$ et $x_2 \geq 3$.

5. Le LPR de $x_2 \geq 3$ est infaisable.

6. En ce qui concerne $x_2 \leq 2$, on obtient la solution $(1.8, 2)$ avec $z = 6.2$, donnant lieu à deux branches $x_1 \geq 2$ et $x_1 \leq 1$.

7. Pour $x_1 \geq 2$, on obtient la solution $(2, 2)$ avec $z = 6$. La solution est entière, d'où la nouvelle borne $z^* = 6$.

8. Pour $x_1 \leq 1$, nous obtenons une solution $(1, 1.6)$ avec $z = 5.4$. 5.4 est inférieur à la meilleure borne courante z^*, d'où son élagage.

Soit un autre problème :

$$\begin{array}{ll} \text{maximize} & z = 9 + 5x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 3x_2 + 5x_3 + 2x_4 \leq 4 \\ & x_3 + x_4 \leq 1 \\ & -x_1 + x_3 \leq 0 \\ & x_2 + x_4 \leq 0 \\ & x_i \in \{0, 1\} \end{array} \tag{2.4}$$

La procédure Branch and Bound est illustré dans la Figure 2.5

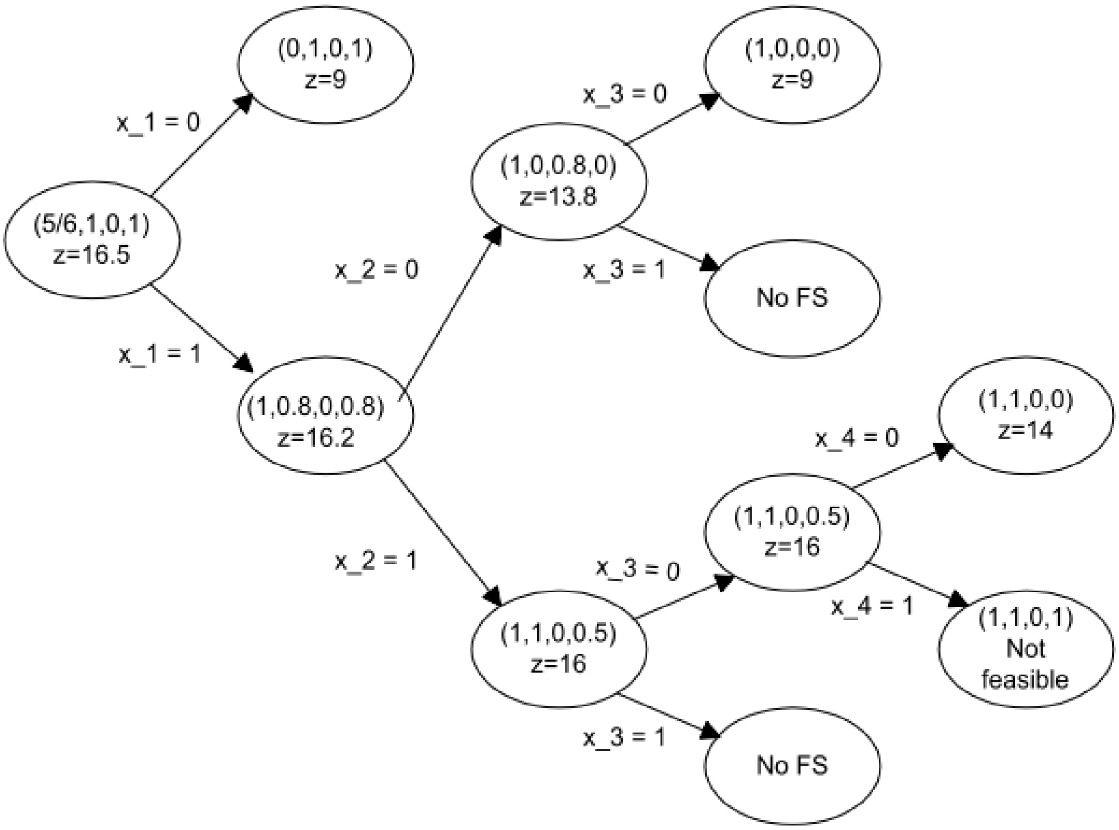


FIGURE 2.5 – Arbre de recherche de la procédure "Branche and Bound" sur le problème (2.3)

2.5 Travaux Dirigés I (Modélisation)

2.5.1 Exercice

Une entreprise de production de produits alimentaires désire orienter ses activités sur 3 lignes de produits I, II et III. Le profit moyen par produit est estimé à 300 DA par tonne pour le produit I, 200 DA pour le produit II, et 500 DA pour le produit III. Les équipements sont répartis en trois départements de production :

— Fabrication ;
— Mélange ;
— Emballage.

La durée maximale de charge pour chaque département est de 8 heures par jour. Les processus de production relatifs aux trois produits font l'objet des opérations successives suivantes :

— Produit I : Fabrication-Mélange. La production est enlevée par les utilisateurs dès qu'elle est réalisée. Chaque tonne ainsi produite exige 3 heures d'utilisation de la capacité de "fabrication" et 1 heures de capacité du département "mélange".
— Produit II : Mélange-Emballage. Le produit est réalisé à partir d'achats de composantes alimentaires non produites dans l'entreprise, fait l'objet des seules opérations de mélange et emballage. Chaque tonne produite exige 1 heure d'utilisation de capacité "mélange" et 2 heures de capacité "emballage".
— Produit III : Fabrication-Mélange-Emballage. Le produit subit les 3 séries d'opérations fabrication-mélange-emballage, chacune d'entre elles exigeant respectivement 2 heures, 1 heure et 1 heure de capacité des équipements.

1. Ecrire le programme linéaire modélisant ce problème ?

2.5.2 Exercice

Dans une entreprise de construction de matériel électrique, on dispose de 1200 heures/machine par mois et 1500 heures/ouvrier par mois. Les contacteurs nécessitent 15 heures/machine, 12 heures/ouvrier et rapportent 8 DA (par unité). Les disjoncteurs nécessitent 30 heures/machine, 30 heures/ouvrier et rapportent 20 DA (par unité). Les compteurs nécessitent 20 heures/machine, 25 heures/ouvrier et rapportent 18 DA (par unité).

1. Modéliser sous forme d'un programme linéaire PL la recherche des quantités optimales des contacteurs x_1, des disjoncteurs x_2 et des compteurs x_3 pour maximiser le gain ?
2. Formuler le dual du PL ?
3. Démontrer que la contrainte $17x_1 + 60x_2 + 45x_3 \leq 2700$ est non-redondante [2] dans le PL ?
4. Si des contraintes sont redondantes dans un PL, comment se comporterait l'algorithme du simplex ? Comment y remédier ?

2. Soit le système d'inéquations $l_i \equiv \sum_{j=1\ldots m} a_{i,j}x_j \geq c_i$, avec $i = 1\ldots n$. Une inéquation $l' \equiv \sum_{j=1\ldots m} b_j x_j \geq d$ est dite redondante si et seulement si $l' = \sum_{i=1\ldots n} \lambda_i l_i$ où $\forall i, \lambda_i$ est un réel positif.

2.5.3 Exercice

Etant donné deux variables x_1, x_2 binaires (i.e., $x_1 \in \{0,1\}$, $x_2 \in \{0,1\}$). Soit l'expression non linéaire de leur multiplication :

$$(P)\begin{cases} y = x_1 \times x_2 \\ x_1 \in \{0,1\}, x_2 \in \{0,1\}, y \in \{0,1\} \end{cases} \tag{2.5}$$

Nous proposons un système d'équations linéaires (Q)

$$(Q)\begin{cases} y \leq x_1 \\ y \leq x_2 \\ y \geq x_1 + x_2 - 1 \\ x_1 \in \{0,1\}, x_2 \in \{0,1\}, y \in \{0,1\} \end{cases} \tag{2.6}$$

Démontrer que (P) est équivalent à (Q) ?

2.5.4 Exercice

Soit le problème

$$(P)\begin{cases} \min \sum_{i=1..n} c_i x_i \\ \\ (\sum_{j=1..n} A_j x_j \leq b) \text{ OU } (\sum_{j=1..n} C_j x_j \leq d) \\ x_i \geq 0, i = 1..n \end{cases} \tag{2.7}$$

Le problème (P) n'est pas un PL car il contient la contrainte de disjonction "OU". Nous proposons un autre problème (Q)

$$(Q)\begin{cases} \min \sum_{i=1..n} c_i x_i \\ \sum_{j=1..n} A_j x_j \leq b + My \\ \sum_{j=1..n} C_j x_j \leq d + M(1-y) \\ x_i \geq 0, i = 1..n \\ y \in \{0,1\} \end{cases} \tag{2.8}$$

où M est un nombre très grand.

Démontrer que (P) est équivalent à (Q) ?

2.5.5 Exercice

Une firme *Penault* produit deux types de carrosseries C_1 et C_2. Nous nous intéressons à deux tâches indépendantes :

— *Peinture* : Cette firme ne pourra pas peindre C_2 à partir d'un maximum de 40 carrosseries C_1. Cette firme ne pourra pas aussi peindre C_1 à partir d'un maximum de 60 carrosseries C_2.

— *Fabrication* : Cette firme ne pourra pas produire C_2 à partir d'un maximum de 50 carrosseries C_1. Cette firme ne pourra pas aussi fabriquer C_1 à partir d'un maximum de 50 carrosseries C_2.

La carrosserie C_1 rapporte 300 DA et C_2 200 DA.

1. Modéliser sous forme d'un programme linéaire PL la recherche des quantités optimales x_1 de C_1 et x_2 de C_2 à produire pour maximiser le gain ? (N.B. Dans une tâche T donnée dans la firme, si la firme ne pourra pas produire C_2 à partir de k_1 C_1, et la firme ne pourra pas aussi produire C_1 à partir de k_2 C_2, la contrainte T peut être traduite en une seule contrainte linéaire $(x_1/k_1) + (x_2/k_2) \leq 1$.)
2. Tracer graphiquement le polyèdre décrivant le PL et extraire la solution optimale sans passer par l'algorithme du Simplexe ?
3. Soit la contrainte supplémentaire suivante : la firme *Penault* doit produire au moins 30 carrosseries C_1 et au moins 20 carrosseries C_2. Comment devient-il le PL ?

2.6 Travaux Dirigés II (séparation/évaluation)

2.6.1 Exercice [Problème d'affectation]

Soit un problème d'affectation ayant comme données :
— n tâches à affecter à n personnes
— Toute personne exécute 1 et 1 seule tâche.
— $c_{i,j}$ est le coût d'affectation de la tâche i à la personne j.

Le problème : trouver l'affectation des tâches aux personnes qui minimise le coût total ?

Soit l'instance du problème :

	Tâche 1	Tâche 2	Tâche 3	Tâche 4
Personne a	9	2	7	8
Personne b	6	4	3	7
Personne c	5	8	1	8
Personne d	7	6	9	4

Soit le calcul suivant :

$$LB1 = \sum_{i=1..n} \min_{j=1..m} c_{i,j}$$

Questions :

1. Démontrer que $LB1$ est une méthode de relaxation valide.
2. Dérouler l'algorithme par séparation évaluation sur l'instance donnée en exploitant $LB1$ comme relaxation.

2.6.2 Exercice [Problème de sac-à-dos]

Soit un problème de sac-à-dos ayant comme données :
— Un sac de volume V.
— Un ensemble de n objects $O = \{o_1, o_2, ..., o_n\}$.
— Tout objet o_i a un volume v_i et une utilité u_i.
— On suppose que $\sum_{i=1..n} v_i > V$.

Le problème : Trouver un sous-ensemble d'objets de O qui maximise l'utilité et qui soit borné par V.

Soit le calcul LB2 suivant :
— Soit le vecteur R des rapports entre les utilités et les volumes $[u_1/v_1, u_2/v_2, ..., u_n/v_n]$.
— Soit $x = (x_1, x_2, ..., x_n)$ les variables binaires associées aux n objets. $x_i = 0$ si l'objet i est mis dans le sac, sinon 1.
— Le calcul consiste en :
  — Trier R par ordre décroissant ;
  — $Disponible = V$;
  — Initialiser x à zéro ;
  — Pour $i = 1$ jusqu'à n faire :
    — Si ($v_i \leq Disponible$) alors $x_i = 1$; sinon $x_i = Disponible/v_i$ finsi.
    — $Disponible = Disponible - v_i * x_i$;

Questions :

1. Démontrer que $LB2$ est une méthode de relaxation valide.

2. Dérouler l'algorithme par séparation évaluation sur l'instance donnée en exploitant $LB2$ comme relaxation sur les données suivantes :

Objets	1	2	3	4
u	16	22	12	8
v	5	7	4	3
V	14			

3. Soit maintenant un autre algorithme UB2 qui reprend l'algorithme LB2 mais en trouvant une solution faisable (sans garantie d'optimalité) dans une démarche gloutonne :
   - Soit le vecteur R des rapports entre les utilités et les volumes $[u_1/v_1, u_2/v_2, ..., u_n/v_n]$.
   - Soit $x = (x_1, x_2, ..., x_n)$ les variables binaires associées aux n objets. $x_i = 0$ si l'objet i est mis dans le sac, sinon 1.
   - Le calcul consiste en :
     - Trier R par ordre décroissant ;
     - $Disponible = V$;
     - Initialiser x à zéro ;
     - Pour $i = 1$ jusqu'à n faire :
       - Si ($v_i \leq Disponible$) alors $x_i = 1$; sinon $x_i = 0$ finsi.
       - $Disponible = Disponible - v_i * x_i$;
4. Dérouler à nouveau l'algorithme par séparation évaluation sur l'instance donnée en exploitant $LB2$ et $UB2$ sur les données ci-dessus.
5. Formuler ce problème en terme d'un programme linéaire en nombres entiers ?
6. Résoudre l'instance du problème par séparation/évaluation en utilisant la relaxation linéaire. Faites vous aider avec le solveur ampl pour le calcul de borne via l'algorithme du simplexe. Comparer l'arbre de recherche utilisant la relaxation linéaire avec l'arbre via la borne $LB2$?

2.7 Travaux Pratiques (Optimisation)

2.7.1 Exercice [Un modèle simple]

Installer `AMPL` en décompressant "amplide-demo-linux32.tar.gz", la version "AMPL FOR STUDENTS" à partir du site officiel http://ampl.com, puis invoquer AMPL :

```
tar xzf amplide-demo-linux32.tar.gz
ample
```

Commençons par un exemple simple d'un problème d'optimisation.

Une compagnie de commercialisation de la peinture fournit deux types de couleurs : la couleur bleue et la couleur dorée. La couleur bleue est vendu 10 DA par gallon, et la dorée 15 DA. La compagnie a une unité de fabrication et fabrique une couleur par unité de temps. Cependant, la couleur bleue est plus facile à produire et l'unité de fabrication peut produire 40 gallons par heure, et 30 gallons pour la couleur dorée. L'unité de vente de cette compagnie informe qu'elle ne peut pas vendre plus de 860 gallons de la couleur dorée et 1000 gallons de la couleur bleue. Supposons que dans le volume horaire travaillé par semaine est de 40 heures, et le produit est toujours stocké dans la semaine qui suit. Nous voulons déterminer le nombre de gallons de couleur dorée et bleue à produire pour maximiser le gain.

Soit `PaintB` (resp. `PaintG`) le nombre de gallons à produire de la peinture de couleur bleue (resp. de couleur dorée). Le problème peut être formulé comme suit :

$$\begin{array}{ll} \max & 10PaintB + 15PaintG \\ s.t. & \frac{1}{40}PaintB + \frac{1}{30}PaintG \leq 40 \\ & 0 \leq PaintB \leq 1000 \\ & 0 \leq PaintG \leq 860 \end{array}$$

`AMPL` permet de concevoir des programmes mathématiques à saisir avec une syntaxe qui est très proche de la notation algébriques que l'on vient de voir. Pour utiliser `AMPL`, on doit créer un fichier texte qui va contenir le texte du programme mathématique (On utilisera un éditeur de texte).

Saisir le texte suivant dans l'éditeur :

```
## Example One
var PaintB; # amount of blue
var PaintG; # amount of gold

maximize profit: 10*PaintB + 15*PaintG;

subject to time: (1/40)*PaintB + (1/30)*PaintG <= 40;
subject to blue_limit: 0 <= PaintB <= 1000;
subject to gold_limit: 0 <= PaintG <= 860;
```

Sauvegarder l'exemple sous le nom `examp1.mod`.

Notez les différents opérateurs et expressions utilisés pour déclarer les variables, la fonction objectif et les contraintes.

Suivre les étapes suivantes pour manipuler notre exemple :

— `AMPL` est un environnement de programmation mathématique qui peut s'interfacer avec la plupart des solveurs connus (`Cplex`, `Minos`, `lpsolve`, ...). Précisons maintenant à `AMPL` le solveur que l'on veut utiliser :
`ample: option solver cplex;`
— Charger notre exemple
`ample: model examp1.mod;`
— Si vous avez des erreurs au niveau de la saisie, vous pouvez revenir à votre éditeur de texte, corriger puis recharger le fichier, en commençons par vider l'environnement :
`ample: reset;`
— Pour résoudre le problème, il suffit d'invoquer
`ample: solve;`
— `AMPL` affiche
`CPLEX 8.0.0: optimal solution; objective 17433.33333`
`2 simplex iterations (0 in phase I)`
— Pour afficher la solution, il suffit d'invoquer la fonction d'affichage `display` avec les variables à afficher
`ample: display PaintB, PaintG;`
— Vous pouvez rediriger la sortie vers un fichier
`ample: display PaintB, PaintG > examp1.out`

2.7.2 Exercice [Un modèle plus général]

Le problème que l'on vient de manipuler est figé ; cependant un nombre de problèmes économiques d'optimisation peuvent se formuler dans le même type de modèle mais se portant sur un nombre de produits, des coûts, et des volumes horaires quelconques. Soit ici ce modèle général :

Soient	n : le nombre de couleurs
	t : le temps total disponible
	p_i : le profit par gallon i, $i = 1..n$
	r_i : gallons par heure de la couleur i, $i = 1..n$
	m_i : la demande maximale de la couleur i, $i = 1..n$
Variables	x_i : le nombre de gallons de la couleur i, $i = 1..n$
Maximiser	$\sum_{i=1..n} p_i x_i$
Subject to	$\sum_{i=1..n}(1/r_i)x_i \leq t$
	$0 \leq x_i \leq m_i$ pour tout i, $i = 1..n$.

Ainsi, notre modèle simple se résume en le modèle général ci-dessus où $n = 2, t = 40, p_1 = 10, p_2 = 15, r_1 = 40, r_2 = 30, m_1 = 1000$ et $m_2 = 860$.

Justement `AMPL` permet de saisir des modèle généraux permettant de les paramétrer aisément. Le modèle général est toujours stocké dans un fichier à part, soit par exemple `model.mod`. Le paramétrage ou la fixation des paramètres sur une instance donnée est faite dans un fichier séparé, dit fichier des données, soit *model.dat*.

Dans notre cas, nous pouvons saisir le modèle général de la compagnie dans un fichier `examp1g.mod` comme suit :

```
param n;
param t;
param p{i in 1..n};
param r{i in 1..n};
param m{i in 1..n};
var paint{i in 1..n};

maximize profit: sum{i in 1..n} p[i]*paint[i];

subject to time: sum{i in 1..n} (1/r[i])*paint[i] <= t;
subject to capacity{i in 1..n}: 0 <= paint[i] <= m[i];
```

Soit le fichier de données `examp1g.dat` qui contient le jeu de données pour notre premier modèle simple :

```
## Example Two - Data
param n:= 2;
param t:= 40;

param p:= 1 10
          2 15;
param r:= 1 40
          2 30;
param m:= 1 1000
          2 860;
```

Chargeons maintenant ces fichiers pour résoudre encore le modèle simple dans cette nouvelle modélisation générale :

```
ample: reset
ample: model examp1g.mod;
ample: data examp1g.dat;
ample: solve;
```

Invoquons l'affichage de la solution :

```
ample: display paint;
```

2.7.3 Exercice [Une autre manière pour saisir les données]

On peut saisir tous les paramètres en une seule fois, avec :

```
## Example Two - Another way to write the data
param n:= 2;
param t:= 40;
param: p r m:=
  1 10 40 1000
  2 15 30 860;
```

2.7.4 Exercice [Les ensembles]

Dans la tâche de la modélisation, nous avons souvent besoin de garder la nomenclature de l'environnement dans lequel le problème a été posé. Soit par exemple la formulation

```
## Example Two - Model with sets
set P;
param t;
param p{i in P};
param r{i in P};
param m{i in P};
var paint{i in P};

maximize profit: sum{i in P} p[i]*paint[i];

subject to time: sum{i in P} (1/r[i])*paint[i] <= t;
subject to capacity{i in P}: 0 <= paint[i] <= m[i];
```

Ainsi, on voit que l'ensemble des couleurs est maintenant un ensemble que l'on peut énumérer en utilisant des noms réels :

```
## Example Two - Data with sets
set P:= blue gold;
param t:= 40;
param p:= blue 10 gold 15;
param r:= blue 40 gold 30;
param m:= blue 1000 gold 860;
```

Et aussi

```
## Example Two - Data with sets
set P:= blue gold;
param t:= 40;
param: p r m:=
blue 10 40 1000
gold 15 30 860;
```

2.7.5 Exercice [Des paramètres et des variables de deux dimensions]

Nous avons souvent besoin de faire appel à des modèles où les paramètres et les variables ont plusieurs dimensions, et tout particulièrement deux dimensions.

Soit le problème suivant :

La compagnie a fait une extension, et a maintenant trois dépôts pour stocker la peinture bleue. Dans une semaine, la peinture doit être acheminer à différents clients. Pour chaque dépôt avec chaque client, les coûts d'acheminement sont différents. Nous donnons ci-dessous ces coûts.

```
                Cust 1     Cust. 2     Cust. 3     Cust. 4
Warehouse 1     1           2          1           3
Warehouse 2     3           5          1           4
Warehouse 3     2           2          2           2
```

Note de traduction : Warehouse est le dépôt, Customer est le client.

Le nombre de gallons disponibles au niveau de chaque dépôt, et le nombre de gallons demandé par chaque client sont donnés respectivement comme suit :

```
Warehouse 1 250
Warehouse 2 800
Warehouse 3 760

Customer 1: 300
Customer 2: 320
Customer 3: 800
Customer 4: 390
```

Le modèle `AMPL` de ce problème est donné comme suit :

```
## Example Three - Model
param warehouse; # number of warehouses
param customer; # number of customers

#transportation cost from warehouse i
#to customer j
param cost{i in 1..warehouse, j in 1..customer};
param supply{i in 1..warehouse}; #supply at warehouse i
param demand{i in 1..customer}; #demand at customer j

var amount{i in 1..warehouse, j in 1..customer};

minimize Cost: sum{i in 1..warehouse, j in 1..customer}
cost[i,j]*amount[i,j];

subject to Supply {i in 1..warehouse}: sum{j in 1..customer} amount[i,j] = supply[i];
subject to Demand {j in 1..customer}: sum{i in 1..warehouse} amount[i,j] = demand[j];
subject to positive{i in 1..warehouse, j in 1..customer}: amount[i,j]>=0;
```

Stockons ce modèle dans un fichier `examp2.mod`.

Et le jeu de données qui suit dans `examp2.dat`.

```
## Example Three - Data
param warehouse:= 3;
param customer:= 4;
param cost: 1 2 3 4 :=
1 1 2 1 3
```

```
2 3 5 1 4
3 2 2 2 2;

param supply:=
1 250
2 800
3 760;

param demand:=
1 300
2 320
3 800
4 390;
```

On peut aussi saisir les noms propres des différents paramètres pour obtenir un modèle qui reprend les mêmes termes que ceux utilisés en pratique.

```
## Example Three - Model using sets
set Warehouses;
set Customers;

#transportation cost from warehouse i
#to customer j
param cost{i in Warehouses, j in Customers};
param supply{i in Warehouses}; #supply at warehouse i
param demand{j in Customers}; #demand at customer j
var amount{i in Warehouses, j in Customers};

minimize Cost: sum{i in Warehouses, j in Customers} cost[i,j]*amount[i,j];

subject to Supply {i in Warehouses}: sum{j in Customers} amount[i,j]=supply[i];
subject to Demand {j in Customers}: sum{i in Warehouses} amount[i,j]=demand[j];
subject to positive{i in Warehouses, j in Customers}: amount[i,j]>=0;
```

Et le jeu de données comme suit :

```
## Example Three - Data with sets
set Warehouses := Oakland San_Jose Albany;
set Customers:= Home_Depot K_mart Wal_mart Ace;

param cost: Home_Depot K_mart Wal_mart Ace :=
Oakland   1 2 1 3
San_Jose  3 5 1 4
Albany    2 2 2 2;

param supply:=
Oakland  250
```

```
San_Jose 800
Albany   760;

param demand:=
Home_Depot 300
K_mart     320
Wal_mart   800
Ace        390;
```

2.7.6 Exercice [Programmation en nombres entiers]

Souvent, dans les modèles mathématiques, certaines variables peuvent être entières. `AMPL` facilité la manipulation de ces variables via les mots clés `integer` et `binary` que l'on peut placer après le nom de la variable pour dire que la variable en question est respectivement entière ou prend ses valeurs dans le domaine binaire $\{0, 1\}$.

Soit une modification au niveau des capacités des dépôts :

```
Warehouse 1 550
Warehouse 2 1100
Warehouse 3 1060
```

Cette augmentation vient avec un coût donné ci-dessous

```
Warehouse 1 500
Warehouse 2 500
Warehouse 3 500
```

Nous voyons ici que la demande excède la produits disponibles.

Soit maintenant le modèle suivant qui prend en compte ces nouvelles spécificités :

```
## Example Four - Mixed-IP model file for the warehouse location
  problem
set Warehouses;
set Customers;
#transportation cost from warehouse i
#to customer j
param cost{i in Warehouses, j in Customers};
param supply{i in Warehouses}; #supply capacity at warehouse i
param demand{j in Customers}; #demand at customer j
param fixed_charge{i in Warehouses}; #cost of opening warehouse j

var amount{i in Warehouses, j in Customers};
var open{i in Warehouses} binary; # = 1 if warehouse i is opened, 0 otherwise

minimize Cost: sum{i in Warehouses, j in Customers}
                 cost[i,j]*amount[i,j] + sum{i in Warehouses} fixed_charge[i]*open[i]
```

```
subject to Supply {i in Warehouses}: sum{j in Customers} amount[i,j] <= supply[i]
subject to Demand {j in Customers}: sum{i in Warehouses} amount[i,j]=demand[j];
subject to positive{i in Warehouses, j in Customers}: amount[i,j]>=0;
```

Soit le jeu de données suivant :

```
## Example Four - Data file for the warehouse location problem
set Warehouses:= Oakland San_Jose Albany;
set Customers:= Home_Depot K_mart Wal_mart Ace;

param cost: Home_Depot K_mart Wal_mart Ace:=
Oakland  1 2 1 3
San_Jose 3 5 1 4
Albany   2 2 2 2;

param supply:=
Oakland  550
San_Jose 1100
Albany   1060;

param demand:=
Home_Depot 300
K_mart     320
Wal_mart   800
Ace        390;

param fixed_charge:=
Oakland  500
San_Jose 500
Albany   500;
```

Faites les tests de résolution !

2.7.7 Exercice [Mise en pratique des modèles étudiés]

1. Mettre en forme le problème de l'exemple 2 sous forme AMPL. Commentez !
2. Mettre en forme le problème de l'exemple 3 sous forme AMPL, en proposant des paramètres significatifs. Commentez !
3. Soit le problème de localisation de l'exemple 3 : modifier le modèle en supposant qu'un site a plusieurs capacités de production à décider !
4. Mettre en forme le problème de l'exemple d'un objectif avec coûts fixes sous forme AMPL. Commentez !
5. Mettre en forme le problème de l'exemple 7 sous forme AMPL, en mettant M à une très grande valeur. Comment se comporte le modèle si M est petit ? Commentez !
6. Mettre en forme le problème de l'exemple 8 sous forme AMPL. Commentez !

7. Mettre en forme le problème de l'exemple 9 sous forme AMPL, dans sa forme générique, en séparant entre le modèle et les données des paramètres.

2.7.8 Exercice [Séparation/Evaluation]

1. Déroulez à nouveau la résolution du cas 0-1 de l'algorithme par séparation/évaluation en exploitant le solveur AMPL sur le domaine réel.
2. Déroulez à nouveau la résolution du cas $I\!N$ de l'algorithme par séparation/évaluation en exploitant le solveur AMPL sur le domaine réel.

2.7.9 Exercice [Programmation nonlinéaire]

Assez souvent, on fait appel dans les modèles à des fonction nonlinéaires qui donnent lieu à des programmes nonlinéaires.

Première chose, il faut maintenant faire appel à un solveur qui sait manipuler ces problèmes. `Cplex` sait résoudre des programmes linéaires classiques et les programmes linéaire en nombres entiers ou mixte. Pour les modèles nonlinéaire, on fera appel au solveur `Minos`.

```
option solver minos;
```

Soit le problème suivant :

Supposons que nous avons plusieurs alternatives d'investissement A. Et nous savons évaluer le retour sur investissement pour certaines années T. A partir de ces données, on peut calculer la matrice des covariance des investissements.

On veut donc :

— *maximiser le retour sur investissement sous le maximum des risques potentiels.*
— *minimiser les risques sous le minimum de retour sur investissement.*

Le modèle suivant reprend les termes de ce problème d'une façon générale.

```
##Example 5 - Nonlinear Portfolio
set A; # asset categories
set T := {1984..1994}; # years
param s_max default 0.00305; # i.e., a 5.522 percent std. deviation on reward
param R {T,A};
param mean {j in A} := ( sum{i in T} R[i, j] - mean[j];
param Rtilde {i in T, j in A} := R[i,j] - mean[j];

var alloc{A} >=0;

minimize reward: - sum{j in A} mean[j]*alloc[j];

subject to risk_bound:
  sum{i in T} (sum{j in A} Rtilde[i,j]*alloc[j])^2 / card{T} <= s_max;

subject to tot_mass: sum{j in A} alloc[j] = 1;
```

Notons :

— Nous avons introduit la variable `alloc` pour indiquer le taux de ressources que l'on veut utiliser au niveau de chaque investissement.

— Le retour sur investissement est donné comme suit $\sum_{j \in A} mean_j * alloc_j$.

— Le risque encouru est donné par $\sum_{i \in T}(\sum_{j \in A} \tilde{R}_{ij} * alloc_j)^2 / card(T)$; où $\tilde{R}_{ij} = R_{ij} - mean_j$.

— On voit que la fonction objectif du problème est linéaire, alors que les contraintes sont quadratiques.

Pour enfin compléter le modèle, soit le jeu de données suivant :

```
set A := US_3-MONTH_T-BILLS US_GOVN_LONG_BONDS SP_500 WILSHIRE_5000;

param R:
US_3-MONTH_T-BILLS US_GOVN_LONG_BONDS SP_500 WILSHIRE_5000 :=
1984 1.103 1.159 1.061 1.030
1985 1.080 1.366 1.316 1.326
1986 1.063 1.309 1.186 1.161
1987 1.061 0.925 1.052 1.023
1988 1.071 1.086 1.165 1.179
1989 1.087 1.212 1.316 1.292
1990 1.080 1.054 0.968 0.938
1991 1.057 1.193 1.304 1.342
1992 1.036 1.079 1.076 1.090
1993 1.031 1.217 1.100 1.113
1994 1.045 0.889 1.012 0.999 ;
```

Comme nous l'avons vu dans le cours, ces problèmes sont difficiles à résoudre. La plupart des solveurs fournissent des solution locales, c'est-à-dire des solutions qui sont optimales localement à un voisinage. Justement, `Minos` fournit des solutions locales.

Il existe des solveurs qui trouvent des solutions globales, telles que `Baron`, `GlobSol`, etc. mais qui ne sont pas encore interfacés avec l'environnement `AMPL`.

2.8 Exercices compléments

2.8.1 Exercice

Etant donné deux variables x_1, x_2 binaires (i.e., $x_1 \in \{0,1\}$, $x_2 \in \{0,1\}$). Soit l'expression non linéaire de leur multiplication :

$$(P)\begin{cases} y = x_1 \times x_2 \\ x_1 \in \{0,1\}, x_2 \in \{0,1\}, y \in \{0,1\} \end{cases} \tag{2.9}$$

Nous proposons un système d'équations linéaires (Q)

$$(Q)\begin{cases} y \leq x_1 \\ y \leq x_2 \\ y \geq x_1 + x_2 - 1 \\ x_1 \in \{0,1\}, x_2 \in \{0,1\}, y \in \{0,1\} \end{cases} \tag{2.10}$$

Démontrer que (P) est équivalent à (Q) ?

2.8.2 Exercice

Soit le problème

$$(P)\begin{cases} \min \sum_{i=1..n} c_i x_i \\ \\ (\sum_{j=1..n} A_j x_j \leq b) \text{ OU } (\sum_{j=1..n} C_j x_j \leq d) \\ x_i \geq 0, i = 1..n \end{cases} \tag{2.11}$$

Le problème (P) n'est pas un PL car il contient la contrainte de disjonction "OU". Nous proposons un autre problème (Q)

$$(Q)\begin{cases} \min \sum_{i=1..n} c_i x_i \\ \sum_{j=1..n} A_j x_j \leq b + My \\ \sum_{j=1..n} C_j x_j \leq d + M(1-y) \\ x_i \geq 0, i = 1..n \\ y \in \{0,1\} \end{cases} \tag{2.12}$$

où M est un nombre très grand.
Démontrer que (P) est équivalent à (Q) ?

2.8.3 Exercice

Une firme *Penault* produit deux types de carrosseries C_1 et C_2. Nous nous intéressons à deux tâches indépendantes :

- *Peinture* : Cette firme ne pourra pas peindre C_2 à partir d'un maximum de 40 carrosseries C_1. Cette firme ne pourra pas aussi peindre C_1 à partir d'un maximum de 60 carrosseries C_2.
- *Fabrication* : Cette firme ne pourra pas produire C_2 à partir d'un maximum de 50 carrosseries C_1. Cette firme ne pourra pas aussi fabriquer C_1 à partir d'un maximum de 50 carrosseries C_2.

La carrosserie C_1 rapporte 300 DA et C_2 200 DA.

1. Modéliser sous forme d'un programme linéaire PL la recherche des quantités optimales x_1 de C_1 et x_2 de C_2 à produire pour maximiser le gain ? (N.B. Dans une tâche T donnée dans la firme, si la firme ne pourra pas produire C_2 à partir de k_1 C_1, et la firme ne pourra pas aussi produire C_1 à partir de k_2 C_2, la contrainte T peut être traduite en une seule contrainte linéaire $(x_1/k_1) + (x_2/k_2) \leq 1$.)
2. Tracer graphiquement le polyèdre décrivant le PL et extraire la solution optimale sans passer par l'algorithme du Simplexe ?
3. Soit la contrainte supplémentaire suivante : la firme *Penault* doit produire au moins 30 carrosseries C_1 et au moins 20 carrosseries C_2. Comment devient-il le PL ?

2.8.4 Exercice

Soit le programme en nombres entiers :

$$(P) \begin{cases} \min & \sum_{j=1..n} c_j x_j \\ \text{s.c.} & \sum_{j=1..n} A_{i,j} x_j \leq V_i, i = 1..m \\ & x_j \in \{0,1\}, j = 1..n \end{cases} \tag{2.13}$$

— Définir ce qu'est une relaxation du problème (P) ?
— Proposer une méthode utilisant la méthode du Simplexe qui permet la relaxation du problème (P) ?

2.8.5 Exercice

Soit le programme en nombres entiers relatif au problème du sac-à-dos :

$$(P) \begin{cases} \max & \sum_{j=1..n} c_j x_j \\ \text{s.c.} & \sum_{j=1..n} p_j x_j \leq V \\ & x_j \in \{0,1\}, j = 1..n \end{cases} \tag{2.14}$$

— Définir ce qu'est une relaxation du problème (P) ?
— Proposer une heuristique rapide de résolution approchée de (P) sans faire appel à la méthode du Simplexe ?

2.8.6 Exercice

Soit le programme en nombres entiers :

$$(P) \begin{cases} \min & \sum_{j=1..n} c_j x_j \\ \text{s.c.} & \sum_{j=1..n} A_{i,j} x_j \leq V_i, i = 1..m \\ & x_j \in \{0,1\}, j = 1..n \end{cases} \tag{2.15}$$

— Définir le principe d'évaluation (bounding) dans l'algorithme de branch& bound résolvant (P) ?

— Définir le principe de séparation (branching) dans l'algorithme de branch& bound résolvant (P) ?
— Comment sont combinés ces deux principes (i.e., séparation et évaluation) afin de résoudre (P) ?

2.8.7 Exercice

Soit le programme linéaire suivant :

$$\begin{array}{ll} \min & 2x_1 + 3x_2 \\ \text{s.c.} & \frac{1}{2}x_1 + x_2 \geq 1 \\ & \frac{2}{3}x_1 - x_2 \geq -2 \\ & x_1, x_2 \geq 0 \end{array} \tag{2.16}$$

1. Dessiner le polyèdre qui définit la forme géométrique des inéquations du problème (i.e. l'espace faisable) ?
2. Donner l'ensemble des sommets du polyèdre ?
3. Donner le sommet qui correspond à la solution optimale du problème ; tout en justifiant son optimalité ?

Chapitre 3

Introduction à l'optimisation continue

3.1 Préliminaires

3.1.1 Concepts de base d'optimisation continue

Définition 1 (problème d'optimisation continue (POC)). *Un problème général d'optimisation continue (POC) s'exprime comme suit : trouver des valeurs des variables de décision $x_1, ..., x_n$ telles que*

$$\begin{aligned} &minz = f(x_1, ..., x_n) \\ &g_i(x_1, ..., x_n) = 0, i = 1..m_e \\ &g_j(x_1, ..., x_n) \leq 0, j = m_e + 1..m. \end{aligned}$$

Définition 2 (région (ou espace) faisable). *L'ensemble des points $(x_1, ..., x_n)$ satisfaisant les m contraintes dans un POC est appelé espace faisable. Un point dans cet espace est dit point faisable, sinon on dit point infaisable.*

Définition 3 (minimum global et solution optimale). *Un point x^* de l'espace faisable pour lequel $f(x^*) \leq f(x)$ est vérifiés pour tous les points faisables x est dit minimum global (solution optimale) du POC.*

Définition 4 (minimum local et solution locale). *Un point faisable $x' = (x'_1, ..., x'_n)$ est un minimum local (solution locale) s'il existe un ϵ tel que $\forall x = (x_1, ..., x_n)$ avec*

$$\forall i, |x_i - x'_i| < \epsilon, f(x) \geq f(x').$$

Exemple : Si K unités de matières premières et L unités de travail sont utilisées, une entreprise peut produire KL unités d'un bien manufacturé. La matière première peut être acheté à 4 \$/unité et la main-d'oeuvre à 1 \$/unité. Un total de 8 \$ est disponible pour l'achat de la matière première et de main-d'oeuvre. Comment l'entreprise peut-elle maximiser la quantité de bien pouvant être fabriquée ?

$$\begin{aligned} &maxz = KL \\ &4K + L \leq 8 \\ &K, L \geq 0 \end{aligned}$$

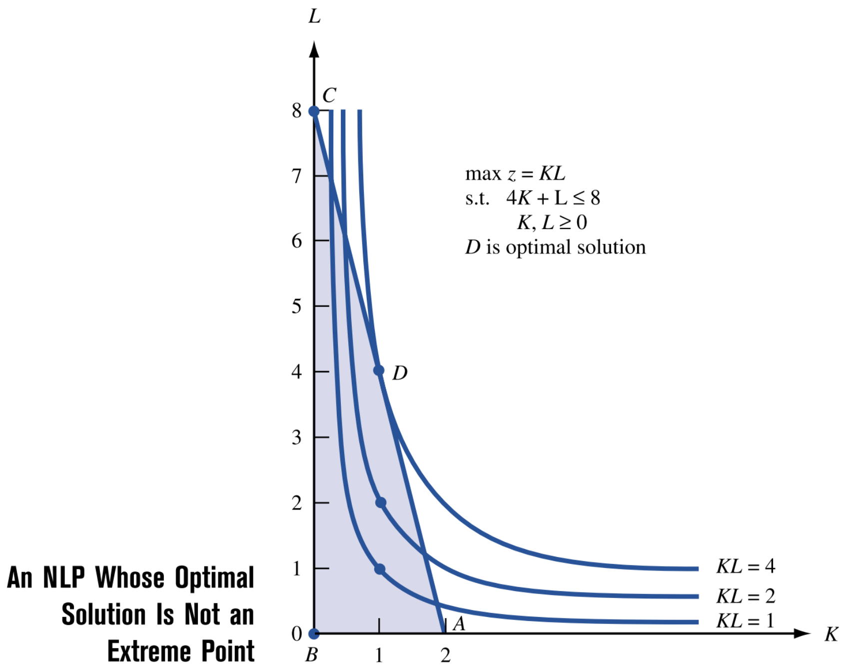


Figure 3.1 – Espace faisable et solution optimale.

3.1.2 Fonctions convexes

Définition 5 (ensemble convexe)**.** *Un ensemble s est convexe si $\forall x', x'' \in s$, tout point du segment liant x' à x'' appartient à s ; c'est-à-dire*

$$\forall c \in [0,1], cx' + (1-c)x'' \in s.$$

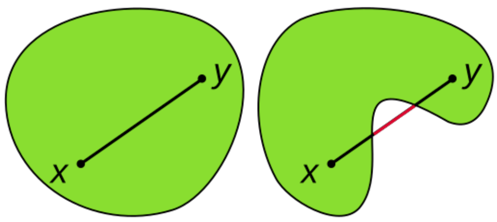


Figure 3.2 – Illustration d'un ensemble convexe vs non-convexe

Définition 6 (fonction convexe et fonction concave)**.** *Soit $f(x_1, ..., x_n)$ une fonction définie sur un ensemble convexe s est convexe (resp. concave)si $\forall x' \in s, x'' \in s$*

$$f(cx' + (1-c)x'') \leq cf(x') + (1-c)f(x'')$$

$$(resp.\ f(cx' + (1-c)x'') \geq cf(x') + (1-c)f(x''))$$

pour tout $c \in [0,1]$.

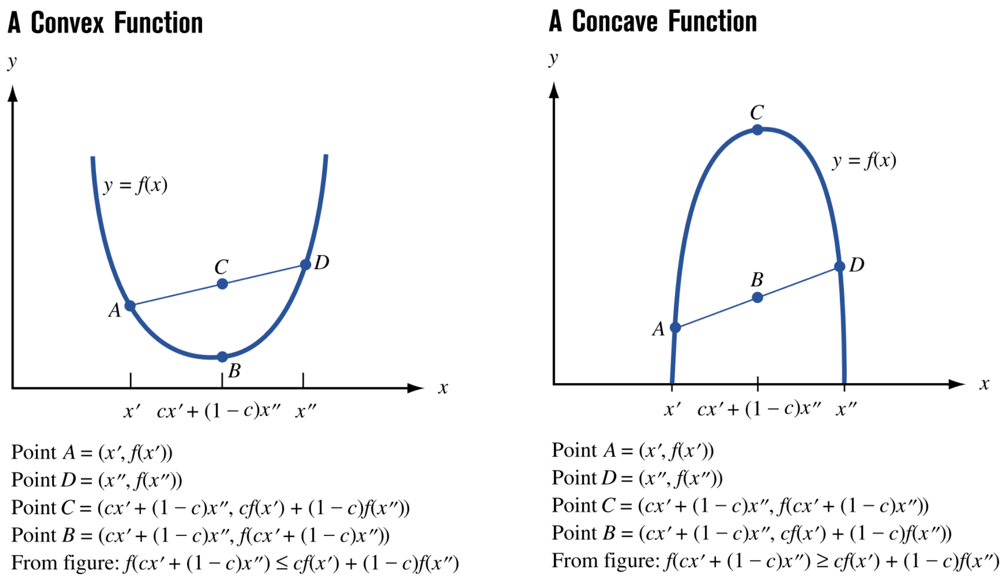


FIGURE 3.3 – Fonctions convexes et fonctions concaves.

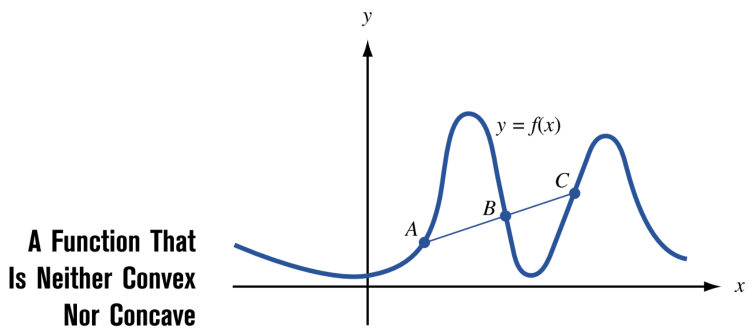


FIGURE 3.4 – Fonctions convexes et fonctions concaves.

Théorème 1 (optimum d'un POC convexe)**.** *Supposons que l'espace faisable s d'un POC est convexe. Si $f(x)$ est convexe dans s, alors tout minimum local du POC est un minimum global.*

Preuve : exercice !

Théorème 2 (convexité et concavité)**.** *Supposons que $f''(x)$ existe pour tout x dans un espace convexe, alors $f(x)$ est convexe (resp. concave) dans s ssi $\forall x, f''(x) \geq 0$ (resp. $f''(x) \leq 0$).*

Définition 7 (matrice hessienne)**.** *La matrice hessienne de la fonction $f : x = (x_1, ..., x_n) \to f(x) = f(x_1, ..., x_n)$ est la matrice carrée $n \times n$ où la (i, j)ième entrée est définie par $\frac{\partial^2}{\partial x_i \partial x_j}$.*

Par exemple, soit $f(x_1, x_2) = x_1^2 + 2x_1x_2 + x_2^2$, alors $H_1(x_1, x_2) = 6x_1$ et $H_2(x_1, x_2) = 12x_1 - 4$.

3.1.3 Illustration : résoudre un POC sans contraintes avec une seule variable

Soit le POC suivant

$$min_{x\in[a,b]}f(x).$$

Naïvement, on peut calculer l'ensemble de tous les minimums locaux puis on prend le plus petit. Nous avons trois cas :

1. points x où $a < x < b$ et $f'(x) = 0$ (appelés points stationnaires)
2. points x où $f'(x)$ n'existe pas
3. les points extremums a et b de $[a, b]$

Parmi les points stationnaires, on prendrait seulement ceux qui sont des minimums locaux, car on peut avoir $f'(x) = 0$ sans que le point soit un minimum local : $f'(x) = 0$ est une condition nécessaire mais insuffisante pour que le point x soit un minimum local. Nous avons donc besoin de propriétés mathématiques pour localiser les minimums locaux. Et en plus, nous devons aussi considérés les valeurs de f sur les points extremums a et b et aux points où les dérivées ne sont pas disponibles car ces derniers peuvent contenir le minimum global.

Voici un théorème qui donne quelques propriétés pour localiser un minimum local.

Théorème 3 (localisation des minimums locaux)**.** *Si $f'(x_0) = 0$ et $f''(x_0) > 0$ alors x_0 est un minimum local.*

Si $f'(x_0) = 0$ et $f''(x_0) < 0$ alors x_0 est un maximum local.

On veut résoudre le POC unidimensionnel sans contraintes

$$max_{a\leq x\leq b}f(x)$$

en se privant d'utiliser la dérivée $f'(x)$.

Dans cette section, on décrit comment résoudre ce POC si la fonction est unimodulaire.

Définition 8 (fonction unimodulaire)**.** *La fonction $f : x \to f(x)$ est unimodulaire dans $[c, d]$ si $\exists x^* \in [a, b]$ où $f(x)$ est strictement croissante dans $[a, x^*)$ et décroissante dans $[x^*, b]$.*

Ce problème peut être résolu simplement par dichotomie. La procédure de résolution se présente comme suit :

1. Soient x_1 et x_2 des points distinct dans $[a, b]$.
2. Si $f(x_1) = f(x_2)$ alors nécessairement le maximum est dans l'intervalle $[a, x_2]$.
3. Si $f(x_1) < f(x_2)$ alors nécessairement le maximum est dans l'intervalle $[x_1, b]$.
4. Si $f(x_1) > f(x_2)$ alors nécessairement le maximum est dans l'intervalle $[a, x_2]$.
5. Relancer ce traitement sur l'intervalle restant jusqu'à ce que sa largeur soit inférieure à la précision exigée par l'utilisateur.

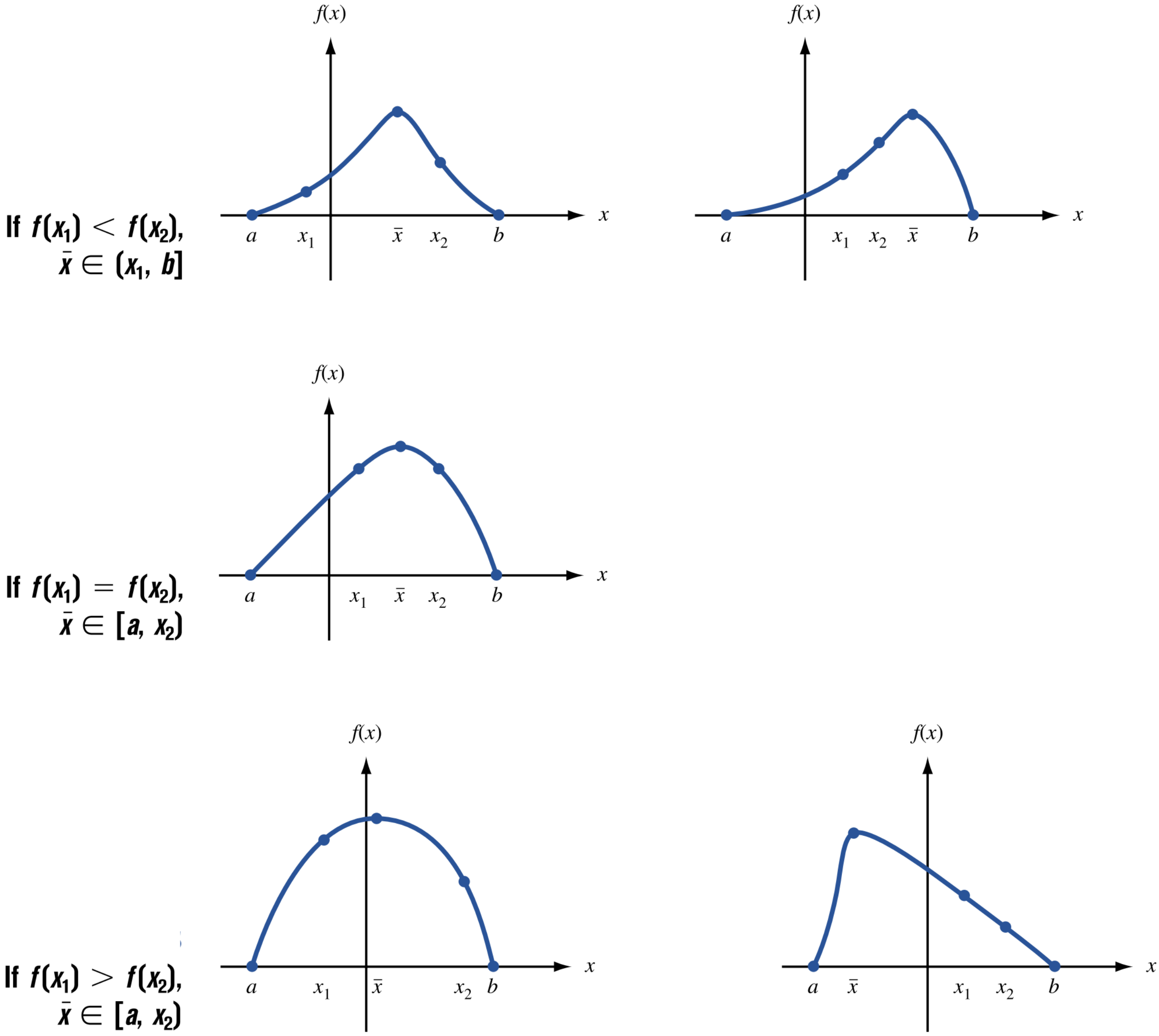


FIGURE 3.5 – Illustration des 3 cas d'optimisation/unimodulaire.

Cette recherche est dite méthode du nombre d'or si à chaque itération on prend $x_2 = a + (b - a)0.618$ et $x1 = b - (b - a)0.618$.

0.618 provient du nombre d'or (ou section dorée, proportion dorée), qui est une proportion définie initialement en géométrie comme l'unique rapport a/b entre deux longueurs a et b telles que le rapport de la somme $a + b$ des deux longueurs sur la plus grande (a) soit égal à celui de la plus grande (a) sur la plus petite (b), ce qui s'écrit :

$$\frac{a+b}{a} = \frac{a}{b}$$

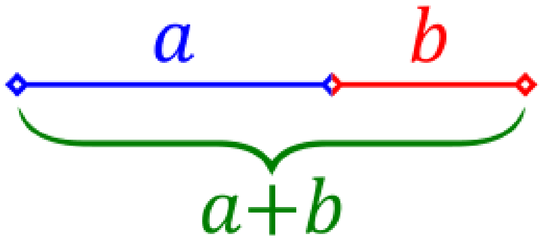


FIGURE 3.6 – Illustration du nombre d'or.

On a donc

$$\frac{a}{b} = r = 1 + \frac{1}{r}.$$

La seule solution de ce problème $r^2 = r + 1$ est $r = \frac{5^{1/2}+1}{2} = 1.6180339887$.
TP : mettre en oeuvre cet algo !

3.1.4 Algorithmes et convergence

Définition 9 (vitesse de convergence)**.** *1. si $lim_{k\to\infty}\frac{\|u^{k+1}-u^*\|}{\|u^k-u^*\|} = \alpha < 1$, on dit que la convergence est linéaire et α est la taux de convergence associé.*

2. si $lim_{k\to\infty}\frac{\|u^{k+1}-u^*\|}{\|u^k-u^*\|} = 0$, on dit que la convergence est superlinéaire.

3. si $\exists\gamma, lim_{k\to\infty}sup\frac{\|u^{k+1}-u^*\|}{\|u^k-u^*\|^\gamma} = \alpha < 1$, on dit que la convergence est d'ordre γ. En particulier si $\gamma = 2$, on parle de vitesse de convergence quadratique ($\gamma = 3$, cubique).

On peut définir

$$d_k = -\log_{10}|u^k - u^*|,$$

cette quantité, à une approximation près, représente le nombre de chiffres décimaux exacts de u_k.

Exemple 1. *Soit la suite :*

$$u_k = 12 + \frac{1}{n}$$

qui converge vers 12*. Nous avons :*

$$\begin{array}{rclrcl} u_{100} & = & 12.01 & d_{100} & = & 2 \\ u_{1000} & = & 12.001 & d_{1000} & = & 3 \\ u_{10000} & = & 12.0001 & d_{10000} & = & 4 \end{array}$$

Lorsque k est suffisamment grand, on peut trouver une constante C telle que :

$$|u_{n+1} - u^*| \approx C|u_k - u^*|^r$$

Nous avons donc $d_{n+1} = r \times d_n - \log_{10}(C)$. C'est-à-dire qu'à chaque itération, nous multiplions par environ r le nombre de chiffres exacts et nous ajoutons $R = -\log_{10}(C)$. Si $r = 1$, nous ne faisons qu'ajouter R chiffres décimaux par itération.

Exemple 2. *Si $C = 0.999$ alors $R = 1/2500$. Cela signifie que l'on obtient un nouveau chiffre exact toutes les 2500 itérations. Par contre, si $r = 1.01$, on multiplie par 2 le nombre de chiffres exacts toutes les 70 itérations ; car pour avoir $2 = 1.01^n$, on obtient $n = log(2)/log(1.01) = 70$. Si $r = 2$, on multiplie par 2 le nombre de chiffre significatifs à chaque itération $n = 1$.*

Ces exemples montrent bien l'intérêt d'utiliser les suites d'un ordre supérieur à 1. C'est pour cette raison que la convergence d'ordre deux ou *quadratique* est caractérisée de convergence *rapide*.

3.2 Optimisation sans contraintes

Dans cette section, nous nous intéressons au problème suivant

$$min_{x \in I\!R^n} f(x) \tag{3.1}$$

où $x \in I\!R^n$ est un vecteur réel avec $n \geq 1$ et $f : I\!R^n \to I\!R$ est une fonction continue et continûment dérivable.

Nous supposons que les premières et secondes dérivées de $f(x_1, x_2, ..., x_n)$ existent et sont continues sur tous les points. Soit $\frac{\partial f}{\partial \tilde{x}_i}$ la dérivée partielle de $f(x_1, x_2, ..., x_n)$ relativement à x_i, évaluée au point $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$.

Une condition nécessaire pour que $\tilde{x} = \tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}$ soit un minimum local de (3.1) est donnée par le théorème suivant :

Théorème 4. *Si $\tilde{x}$ est un minimum local de (3.1), alors $\frac{\partial f(\tilde{x})}{\partial x_i} = 0$.*

La preuve de ce théorème vient du fait que si $\tilde{x}$ est un minimum local alors il a un voisinnage où la fonction objectif prend son min en ce point. Supposons que le point en question n'annule pas la dérivée de la fonction objectif, alors nécessairement la fonction objectif est monotone en ce point faisant ainsi diminuer la fonction objectif ; ce qui contredit que le point est optimal dans son voisinnage.

Définition 10. *Un point $\tilde{x}$ ayant $\frac{\partial f}{\partial x_i} = 0$ pour tout $i = 1, 2, ..., n$ est dit un point stationnaire de f.*

Si un point stationnaire n'est pas un minimum local, il est dit un point d'inflexion (saddle point).

Nous pouvons aussi exploiter les propriétés de convexité et de concavité de la fonction objectif pour détecter des minimums globaux (voir la section 3.1.2).

Nous allons tout d'abord commencer par voir les méthodes locales de descente.

Les méthodes les plus efficaces d'optimisation continue sans contraintes utilisent le gradient de la fonction objectif. La méthode la plus simple est celle de la plus grande descente et les plus raffinées et performantes sont celles de quasi-Newton et Newton (secant methods).

Toutes ces méthodes sont basées sur la stratégie de recherche linéaire que nous introduisons ci-dessous.

FIGURE 3.7 – Recherche linéaire

```
λk := 1 ;
while (f(xk + λk/2dk) < f(xk + λdk)) do
    λk := λk/2 ;
end while
```

FIGURE 3.8 – Schéma des algorithmes de stratégie linéaire

```
Initialisation de x0 et d0 ;
k := 0 ;
do
    Déterminer λk tel que minf(xk + λk dk)
    Calculer le nouveau point xk+1 := xk + λk dk
    Calculer la nouvelle direction dk+1 ;
while (|xk+1 − xk| < ε)
```

3.2.1 Recherche linéaire

L'algorithme choisi une direction d_k et cherche au long de cette direction à partir de l'itéré courant x_k un nouveau itéréx_{k+1} minimisant la fonction objectif. La structure générale de cet algorithme est donnée dans la Figure 3.2.1.

Nous voulons résoudre $minf(x_k + \lambda_k d_k)$ avec $\lambda_k \in [0, 1]$.

Une solution approchée de ce problème peut être obtenue avec la procédure donnée dans la Figure 3.2.1.

Cette recherche est de convergence linéaire. Elle peut être raffinée en construisant un modèle quadratique de $f(x_k + \lambda_k d_k)$ en fonction de λ_k. Ce modèle quadratique est de la forme

$$P(\lambda) = f_1 + h_1\lambda + h_3\lambda(\lambda - \lambda_2)$$

qui interpole la fonction au 3 points $\alpha = 0, \alpha = \alpha_2, \alpha = \alpha_3$; où

$$\begin{aligned} f_i &= f(x_k + \lambda_i d_k) \\ h_1 &= (f_2 - f_1)\lambda_2 \\ h_2 &= (f_3 - f_2)/(\lambda_3 - \lambda_2) \\ \lambda_2 &= \lambda_3/2 \\ h_3 &= (h_2 - h_1)/\lambda_3 \end{aligned}$$

On sait que la solution de ce modèle est $\lambda_3 = 0.5(\lambda_2 - h_1/h_3)$. Cette dernière solution peut donner une meilleure solution que que le choix naïf $\lambda/2$. Quand cette méthode est utilisée dans un quasi-Newton, on doit prendre en considération dans le choix de λ le fait que H doit rester définie positive.

3.2.2 Méthode de la plus grande descente

Ce sont les méthodes les plus simples. La recherche de la direction est faite suivant la décroissance de f qui est dans l'opposé de $\Delta f(x)$

$$x_{k+1} := x_k - \lambda_k \Delta f(x_k).$$

où $\Delta f(x) = [\frac{\partial f(x)}{\partial x_1}, ..., \frac{\partial f(x)}{\partial x_n}]$ est le vecteur gradient de $f(x)$.

On peut se poser la question : quel est la grandeur d'influence de λ_k ?

$$\frac{\partial f(x_{k+1})}{\partial \lambda_k} = \frac{\partial f(x_k - \lambda_k \Delta(f(x_k)))}{\partial \lambda_k} = 0$$

$$\Rightarrow -\Delta f(x_{k+1}) \partial f(x_k) = 0$$

λ_k est donc choisie telle que $\Delta f(x_{k+1})$ et $\Delta f(x_k)$ soient orthogonales.

Cette méthode est de convergence linéaire [**?**] ; mais elle a de bonne propriétés de convergence globale.

3.2.3 Méthode de quasi-Newton

La méthode la plus efficace et celle des méthodes de quasi-Newton. Ces méthodes construisent une approximation quadratique :

$$min_x \frac{1}{2} x^t H x + C^T x + b$$

où la matrice H est semi-définie positive.

Rappel 1 (matrice semi-définie positive (SDP)). *Une matrice $A \in I\!R^{n \times n}$ est dite semi-définie postive si*

$$\forall x \in I\!R^n \Rightarrow x^T A x \geq 0.$$

Elle est dite définie positive si

$$\forall x \in I\!R^n \Rightarrow x^T A x > 0.$$

Une condition nécessaire de l'optimalité d'une solution x^* de cette approximation quadratique est que les dérivées partielles soient toutes égales à zéro

$$\Delta f(x^*) = H x^* + C = 0.$$

Les méthodes de type Newton calculent H directement, ce qui est très couteux si le nombre de variables est important.

Plusieurs méthodes ont ainsi été développées pour approximer directement l'inverse de H avec seulement les valeurs du gradient. Deux formules ont été proposées, celle de DFP (Davidon, Fletcher et Powell) et BFGS (Broyden, Fletcher, Goldfard et Shanno).

La formule de BFGS est

$$H_{k+1} := H_k + \frac{q_k q_k^T}{q_k s_k} + \frac{H_k^T S_k^T S_k H_k}{S_k^T H_k s_k}$$

où $s_k := x_{k+1} - x_k$ et $q_k := \Delta f(x_{k+1}) - \Delta f(x_k)$.

DFP est similaire à BFGS en interchangant q_k et s_k. H_0 est fixée à n'importe quelle matrice symétrique semi-définie positive, par exemple la matrice identité. BFGS a les mêmes propriétés théoriques que DFP. Mais en pratique $BFGS$ est préférée à DFP car elle est plus stable numériquement (moins sensible aux erreurs d'arrondi).

L'algorithme général de quasi-Newton est celui de la stratégie linéaire où :

$$d_{k+1} := \; - H_{k+1}\Delta f(x_k)$$

Il a été prouvé que quasi-Newton est de convergence superlinéaire ; mais elle est moins bonne au niveau convergence globale que les méthodes de descente du gradient.

3.2.4 Moindres carrés

La stratégie de recherche linéaire de quasi-Newton peut être utilisée pour résoudre le problème d'optimisation particulier des moindres carrés

$$LS \;\; minf(x) = \frac{1}{2}\|F(x)\| = \frac{1}{2}\sum F_i(x)^2.$$

Ce problème apparaît souvent dans plein d'applications pratiques particulières dans l'interpolation à partir de données expérimentales. Ce problèmes peut être résolu simplement avec un quasi-Newton. Les propriétés mathématiques de ce problème permettent une meilleure résolution.

La méthode qui tire profit de ces propriétés et qui est la plus utilisée sur les moindres carrés est celle de Levenberg-Marquardt (LM).

LM prend comme direction de recherche la solution d_k de l'équation

$$[J(x_k)^T J(x_k) + \lambda_k I]d_k = -J(x_k)F(x_k).$$

La solution recherche de (LS) est atteinte quand $F_i(x) = O, i = 1..m$.

La formule de Taylor permet de le ramener à :

$$F_i(x_k) + J(x_k)x = 0$$
$$\Rightarrow J_i(x_k).x = -F_i(x_k)$$

Puisque le système n'est pas carré, en le rend carré en le multipliant par la transposée

$$\Rightarrow J_i(x_k)^T J_i(x_k)x = -J_i(x_k)^T F_i(x_k).$$

Résoudre ce dernier système permet d'avoir la direction de quasi-Newton car $J_i(x_k)^T J_i(x_k)$ est le hessien de $\sum F_i(x)^2$.

En d'autres termes, plus λ_k augmente, LM prend comme direction de recherche la solution d_k de l'équation

$$[J(x_k)^T J(x_k) + \lambda_k I]d_k = -J(x_k)F(x_k).$$

plus on se rapproche de la méthode de descente et ainsi obtenir une meilleure convergence globale.

Donc quand on n'a pas la convergence de quasi-Newton, on augmente λ_k pour converger avec la méthode de la plus grande descente.

3.3 Optimisation avec des contraintes et une fonction objectif : le cas général

Un problème général d'optimisation continue (POC) s'exprime comme suit : trouver des valeurs des variables de décision $x_1, ..., x_n$ telles que :

$$\begin{array}{l} minz = f(x_1, ..., x_n) \\ g_i(x_1, ..., x_n) = b_i, i = 1..m_e \\ g_j(x_1, ..., x_n) \leq b_j, j = m_e + 1..m. \end{array} \tag{3.2}$$

3.3.1 Les multiplicateurs de Lagrange

Les multiplicateurs de Lagrange peuvent être utilisés quand les contraintes sont des égalités. Nous considérons donc le problème d'optimisation (POC) suivant :

$$\begin{array}{l} minz = f(x_1, ..., x_n) \\ c_i : g_i(x_1, ..., x_n) = b_i, i = 1..m_e \end{array} \tag{3.3}$$

Pour résoudre (3.3), on associe à chaque contrainte c_i un multiplicateur λ_i, et on forme ainsi le Lagrangien :

$$L(x_1, x_2, ..., x_n, \lambda_1, \lambda_2, ..., \lambda_m) = f(x_1, x_2, ..., x_n) + \sum_{i=1}^{m} \lambda_i [b_i - g_i(x_1, x_2, ..., x_n)] \tag{3.4}$$

Nous cherchons à trouver un point $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}, \tilde{\lambda_1}, , \tilde{\lambda_2}, ..., \tilde{\lambda_m})$ qui minimise $L(x_1, x_2, ..., x_n, \lambda_1, \lambda_2, ..., \lambda_m)$. Souvent $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}, \tilde{\lambda_1}, , \tilde{\lambda_2}, ..., \tilde{\lambda_m})$ résout aussi (3.3). Si $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}, \tilde{\lambda_1}, , \tilde{\lambda_2}, ..., \tilde{\lambda_m})$ minimise L, alors

$$\frac{\partial L}{\partial \lambda_i} = b_i - g_i(x_1, x_2, ..., x_n) = 0$$

Ici $\frac{\partial L}{\partial \lambda_i}$ est la dérivée partielle de L par rapport à λ_i. Ceci montre bien que le point en question satisfait d'une façon optimale(3.3). Pour montrer que $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$ résout (3.3), soit un point quelconque de l'espace faisable de (3.3). Puisque $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}, \tilde{\lambda_1}, \tilde{\lambda_2}, ..., \tilde{\lambda_m}$ minimise L, alors pour tout $\lambda'_i, i = 1..m$, nous avons :

$$L(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}, \tilde{\lambda_1}, , \tilde{\lambda_2}, ..., \tilde{\lambda_m}) \leq L(x'_1, x'_2, ..., x'_n, \lambda'_1, , \lambda'_2, ..., \lambda'_m)$$

Puisque $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$ et $(x'_1, x'_2, ..., x'_n)$ sont tous les deux faisables, alors tous les facteurs des multiplicateurs sont nuls. On obtiens ainsi $f(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}) \leq f(x'_1, x'_2, ..., x'_n)$. Ceci montre que $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$ est bien la solution de (3.3). En bref, si $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}, \tilde{\lambda_1}, \tilde{\lambda_2}, ..., \tilde{\lambda_m})$ résout le problème d'optimisation sans contraintes

$$minL(x_1, x_2, ..., x_n, \lambda_1, \lambda_2, ..., \lambda_m) \tag{3.5}$$

alors $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$ résout aussi (3.3).

A partir du chapitre **??**, nous savons qu'une condition nécessaire pour que $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}, \tilde{\lambda_1}, \tilde{\lambda_2}, ..., \tilde{\lambda_m})$ résolve (3.5) est qu'au point $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n}, \tilde{\lambda_1}, \tilde{\lambda_2}, ..., \tilde{\lambda_m})$ on doit avoir

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial x_2} = ... = \frac{\partial L}{\partial x_n} = \frac{\partial L}{\partial \lambda_1} = \frac{\partial L}{\partial \lambda_2} = ... = \frac{\partial L}{\partial \lambda_n} = 0 \tag{3.6}$$

Le théorème suivant donne des conditions suffisantes pour qu'un tel point soit un minimum global.

Théorème 5. *Si $f(x_1, x_2, ..., x_n)$ est une fonction convexe et chaque $g_i(x_1, ..., x_n)$ est une fonction linéaire, alors tout point $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$ satisfaisant (3.6) est une solution optimale.*

Les variables λ_i ont une interprétation intéressante comme étant les coûts marginaux (shadow prices) associés à chacune des des contraintes. Si la partie gauche d'une contrainte $g_i(x_1, x_2, ..., x_n) = b$ croit de δ_i, alors la valeur de la fonction objectif croit de $\delta_i \lambda_i$.

3.3.2 Les conditions de Kuhn-Tucker

Nous discutons dans cette section des conditions nécessaires et suffisantes pour que $(\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$ soit une solution optimale d'un POC avec des contraintes d'inégalité :

$$\begin{array}{l} minz = f(x_1, ..., x_n) \\ c_i : g_i(x_1, ..., x_n) \leq b_i, i = 1..m \end{array} \tag{3.7}$$

Pour que les résultats de cette section soient applicable, il qu'on ait seulement des contraintes d'inégalité inférieure. Les contraintes d'égalité ou d'inégalité supérieure peuvent être aisément ramenées en contraintes d'inégalité inférieure.

Le théorème suivant donne des conditions nécessaires pour qu'un point $\tilde{x} = (\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_n})$ soit optimal pour le POC (3.7).

Pour que ce théorème soit applicable il faut que les fonctions g_i satisfassent des conditions de régularité (voir [?]). Quand les contraintes sont linéaires, ces conditions sont toujours satisfaites. Nous supposons dans la suite que les contraintes satisfassent toujours les conditions de régularité.

Théorème 6.

Comme au niveau des multiplicateurs de Lagrange de la section précédente, les multiplicateurs λ_i associés au conditions KT correspondent aux coûts marginaux des contraintes.

3.3.3 SQP : Sequential Quadratique Programming

3.3.4 TP

3.4 Résolution des équations non-linéaires

3.4.1 Algorithme de Newton

Les méthodes locales s'intéressent au problème suivant :

Définition 11 (local zero (LZ))**.** *Soit $f : I\!R^n \to I\!R^n$, trouver x^* une approximation locale du zéro de la fonction f ($y \in I\!R^n$ est un zéro de f si $f(y) = 0$).*

La méthode de Newton pour résoudre des systèmes d'équations consiste à appliquer le procédé itératif suivant en partant d'un point initial x^0 :

$$x^k = x^{k-1} - J(x^{k-1})^{-1} f(x^{k-1})$$

où $J(x)$ est la matrice jacobienne $n \times n$: $[\frac{\partial f_i(x)}{\partial x_j}, i = 1..m, j = 1..n]$.

Cette méthode est de convergence locale quadratique ; mais elle n'a pas de bonnes propriétés de convergence globale. C'est pour cette raison que l'on transforme le système d'équations en un problème d'optimisation pour bénéficier de ses méthodes de convergence globale. C'est l'objet de la sous-section suivante.

3.4.2 Transformation en un problèmes d'optimisation

Le système d'équations non-linéaires $f(x) = 0$ est transformé en un problème d'optimisation aux moindres carrés

$$min_x \sum_{i=1..m} f_i(x_1, ..., x_n)^2$$

Ainsi, on pourrait appliquer aisément les méthodes des moindres carrés du chapitre précédent. En raison de la particularité de ce problème, une méthode dédiée a été développée, l'algorithme de Levenberg-Marquardt-Method (LMM), pour exploiter ses propriétés.

LMM utilise la direction de recherche d_k qui est la solution du système linéaire :

$$(J(x_k)^T J(x) + \lambda_k I)d_k = -J(x_k)F(x_k)$$

où $J(x)$ est la matrice jacobiennem $m \times n$: $[\frac{\partial f_i(x)}{\partial x_j}, i = 1..m, j = 1..n]$. Le scalaire λ_k contrôle la magnitude et la direction de d_k.

La valeur de λ_k est choisie pour garantir la convergence de l'ensemble du procédé. Quand λ_k est égale à zéro, l'algorithme se comporte comme un quasi-newton et bénéficie ainsi de sa bonne convergence locale. Plus λ_k est grand, plus l'algorithme se comporte comme l'algorithme de la plus grande descente et bénéficie ainsi de sa convergence globale. λ_k est initialisée à zéro, et si l'algorithme ne converge pas, λ_k est augmenté progressivement pour pouvoir ainsi converger globalement ; par la suite λ_k est diminuer pour converger rapidement une fois que les itérations précédentes ont permis de se rapprocher suffisamment de la solution. Une fois λ_k fixée, le vecteur d_k est obtenu en résolvant le système linéaire ci-dessus. Finalement, une stratégie de recherche linéaire (line-search strategy) est utilisée pour fixer α_k pour que $x_{k+1} = x_k + \alpha_k d_k$ fasse décroître la fonction objectif (la fonction aux moindres carrés).

Chapitre 4

Méthodes approchées et métaheuristiques

4.1 Méthodes de descente

4.2 Le recuit simulé

4.3 La méthode Tabou

4.4 Algorithmes génétiques

4.5 Méthodes d'optimisation bio-inspirée

Chapitre 5

Méthodes d'optimisation pour l'IA et le deep-learning

5.1 Introduction aux architectures neuronales

5.2 Problématique d'optimisation dans l'apprentissage automatique

Bibliographie

[Aribi, 2014] Aribi, N. (2014). Contribution à l'élicitation des paramètres en optimisation multicritère. Technical report, Thèse Docteur en Sciences, Université Oran1, Université de Nice-Sophia Antipolis.

[Bastin, 2010] Bastin, F. (2010). Modèles de recherche opérationnelle. Technical report, Support de cours, Département d'Informatique et de Recherche Opérationnelle https://www.iro.umontreal.ca/~bastin/Cours/IFT1575/IFT1575.pdf.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In Harrison, M. A., Banerji, R. B., and Ullman, J. D., editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM.

[Cori et al., 2001] Cori, R., Hanrot, G., and Steyaert, J.-M. (2001). Conception et analyse des algorithmes. Technical report, Ecole polytechnique.

[Cormen et al., 1994] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1994). *Introduction à l'algorithmique (traduit par Xavier Cazin)*. Dunod.

[Cormen et al., 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms, Second Edition*. The MIT Press.

[Gaudel et al., 1990] Gaudel, M.-C., Froidevaux, C., and Soria, M. (1990). *Types de données et algorithmes*. McGraw-Hill, Paris.

[Levin, 1973] Levin, L. A. (1973). Universal search problems (??????????????????????????). *Problems of Information Transmission (??????????????????????????)*, 9(3).

[Papadimitriou et al., 2006] Papadimitriou, C. H., Dasgupta, S., and Vazirani, U. (2006). *Algorithms*. McGraw Hill.

[Prins, 1994] Prins, C. (1994). *Algorithmes de graphes*. Eyrolles.

[R. Faure, 1995] R. Faure, B. Lemaire, C. P. (1995). *Précis de recherche opérationnelle : Méthodes et exercices d'application*. Eyrolles.