Résumé du cours d'Algorithmique et Recherche Opérationelle *INFO-F310*

Théo Verhelst

5 mars 2017

Table des matières

Chapitre 1

Introduction

Parmi les nombreux domaines compris dans l'algorithmique et la recherche opérationelle, dans ce cours seront abordés la *programmation mathématique* et les *méthodes combinatoires dans les graphes*.

Chapitre 2

Programmation mathématique

2.1 Définition de la programmation mathématique

La programmation mathématique est une modélisation de problèmes (qui peuvent provenir d'une large gamme de domaines) ainsi que leur résolution.

Définition : Un problème de programmation mathématique est défini par un tuple (z, G), où

$$z : E^n \to E : (x_1, \ldots, x_n) \mapsto f(x_1, \ldots, x_n)$$

est appelée *fonction économique* (ou encore *fonction de coût*), et où

$$\star \in \{=, \geq, \leq\},\ G = \{(g_j(x_1, \ldots, x_n) \star b_j)\ \forall j \in [1, m]\}$$

sont appelées *contraintes*.

Notation : On notera

$$g_j(x_1, \ldots, x_n) \left\{ \begin{matrix} \leq \\ \geq \\ = \end{matrix} \right\} b_j \quad \forall j \in [1, m]$$

Définition : On classe les problèmes selon la nature de l'ensemble E :
- $E = \mathbb{R}$ correspond aux problèmes continus
- $E = \mathbb{R}$ correspond aux problèmes entiers
- $E = \{1, 0\}$ correspond aux problèmes booléens

Ces classes peuvent être mixées, si E est l'union de plusieurs de ces sous-ensembles.

Définition : Résoudre un problème de programmation mathématique consiste à trouver les valeurs $(x_1, \ldots, x_n)$ qui maximisent ou minimisent le plus possible d'une valeur donnée la fonction économique z, tout en vérifiant toutes les contraintes g_j.

Définition : Pour une solution donnée $(x_1, \ldots, x_n)$, on dit qu'une contrainte d'indice j est saturée quand :

$$g_j(x_1, \ldots, x_n) = b_j$$

Cette notion n'est intéressante que pour les contraintes à inégalités, et représente le cas où une ressource est utilisée à son maximum.

Définition : Quand les fonctions f et g_j sont linéaires en x_i, alors le problème est appelé problème de programmation linéaire.

2.2 Problèmes de programmation linéaire continus

On commencera par exprimer les problèmes de programmation linéaire continus sous forme matricielle :

$$\begin{cases} \text{Max } z = cx & z \in \mathbb{R},\ c \in \mathbb{R}^{1 \times n},\ x \in \mathbb{R}_+^{n \times 1} \\ Ax \leq b & A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^{m \times 1} \end{cases}$$

où x est le vecteur de variables, c est le vecteur de coefficients de la fonction économique z, A est la matrices de coefficients des contraintes, et b est le vecteur de termes indépendants des contraintes.

Note : On se passe des contraintes en $\geq$ et $=$, car on peut toujours reformuler ces contraintes avec d'autres contraintes en $\leq$:

$$a = b \Leftrightarrow a \leq b \wedge -a \leq -b$$

$$a \geq b \Leftrightarrow -a \leq -b$$

Note : On ne considère que les problèmes où les variables $(x_1, \ldots, x_n)$ sont non-strictement positives, car ces variables représentent souvent des quantités, et ne peuvent donc pas être négatives. Si toutefois une variable x_i, $i \in [1, n]$ peut être négative, on se ramène dans le cas non-strictement positif en posant

$$x_i = y_i - z_i$$

où y_i, z_i sont deux nouvelles variables dans $\mathbb{R}^+$.

2.3 Exemples de problèmes de programmation mathématique

Le problème du sac à dos Supposons que, dans l'optique d'une randonnée en montagne, l'on cherche à remplir au mieux un sac à dos avec des aliments ayant chacun un poids et une valeur énergétique, en sachant que l'on ne peut pas porter plus de 16 unités de poids :

	A	B	C
Énergie	5	4	1
Poids	4	2	1

On va représenter le problème comme suit :

$$f(x_A, x_B, x_C) = 5x_A + 4x_B + x_C$$

$$4x_A + 2x_B + x_C \leq 16$$

où x_A, x_B, x_C sont le nombre de fois que l'on mettra un objet A, B ou C respectivement dans le sac.
On cherche donc à maximiser f, car il faut maximiser la valeur nutritionelle totale des aliments dans le sac.
C'est un problème de programmation linéaire.

Le problème du voyageur de commerce Problème bien connu, passons directement à la modélisation en programmation mathématique :
Soit n villes, posons $x_{ij} = 1$ si le voyageur va de la ville i à la ville j, $x_{ij} = 0$ sinon.
Le cout du trajet entre chaque ville est décrit par la matrice $[c_{ij}]_{i,j \in [n]^2}$ Nous cherchons à minimiser la fonction de coût

$$z = \sum_i \sum_j x_{ij} c_{ij}$$

Les contraintes sont les suivantes :

— Il faut que chaque ville soit visitée une seule fois :

$$\sum_i x_{ij} = 1$$

— Il faut aussi que le voyageur parte aussi de chaque ville :

$$\sum_j x_{ij} = 1$$

— Mais il faut également que le graphe ainsi formé soit connexe ! Pour toute partition de l'ensemble V des villes en deux sous-ensembles S et $\bar{S}$ tels que $S \cup \bar{S} = V$ et $S \cap \bar{S} = \emptyset$, il faut qu'il existe un chemin reliant S et $\bar{S}$:

$$\sum_{s \in S} \sum_{\bar{s} \in \bar{S}} x_{s\bar{s}} \geq 1$$

pour $x_{s\bar{s}} = x_{ij}$ si s et $\bar{s}$ correspondent aux villes i et j respectivement.

Difficulté : L'énumération de toutes les possibilités de cette dernière contrainte prends beaucoup, beaucoup de temps. C'est ce qui explique la nature difficile de ce problème, et qui le rends impossible à résoudre en temps polynomial.

On pourrait être tenté de résoudre le problème en énumérant toutes les solutions possibles, et en prenant la meilleure. On peut trouver facilement que le nombre de solutions possibles est $n!$. Pour $n = 52$ (le problème tel que posé pour visiter tous les états des États-Unis), on a à une vache près 10^{69} solutions. Leur énumération est simplement impossible, même avec le plus puissant des ordinateur connus.

2.4 Respect d'un nombre paramétrique de contraintes

On peut étendre la définition de la programmation mathématique en permettant de ne respecter qu'un nombre m' de contraintes, avec $m' < m$. Pour cela, introduisons m variables booléennes δ_i qui indiqueront si la contrainte i est respectée. Introduisons également un très grand nombre M, qui est beaucoup plus grand que toutes les valeurs que peuvent prendre les variables et les paramètres. On peut alors réécrire les contraintes comme suit :

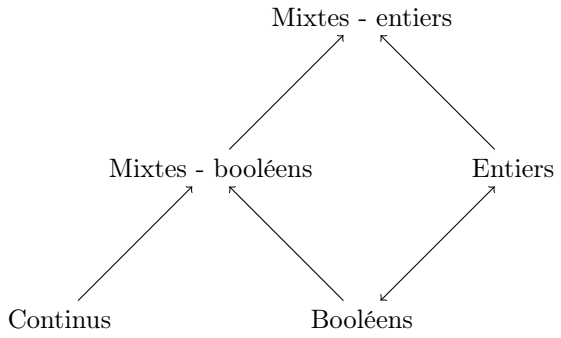
$$g_i(x_1, \ldots, x_n) - b_i \begin{Bmatrix} \leq \\ \geq \\ = \end{Bmatrix} M(1 - \delta_i)$$

et rajouter la contrainte suivante :

$$\sum_{i=1}^{m} \delta_i \geq m'$$

Le problème résultant reste un problème de programmation linéaire si f et g sont des fonctions linéaires.

2.5 Classification des problèmes de programmation linéaire



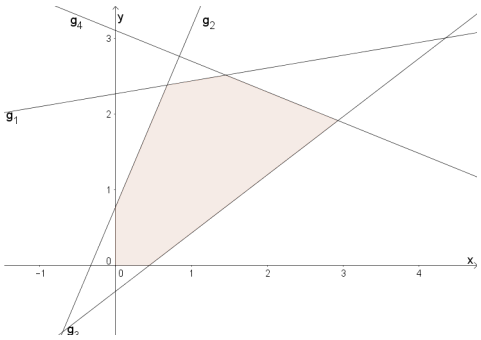
On peut classer les problèmes de programmation linéaire selon la nature de leur variable. Le sens des flèches dans le schéma indique qu'une méthode permettant de résoudre le problème à la destination de la flèche permet également de résoudre un problème à la base de la flèche. On peut donc en conclure qu'un solveur de problème mixant variables entières et continues permet de résoudre tout type de problème de programmation linéaire.

Note : Un problème à nombre entiers peut également être résolu par un solveur booléens : on pourrait imaginer convertir tous les variables entières en suites de variables booléennes grâce à la conversion binaire du nombre.

2.6 Algorithme du simplex

2.6.1 Intuition

L'algorithme du simplex résout des problèmes de programmation linéaire continus, et peut être facilement imaginé dans le cas d'un problème à deux variables continues (x, y). Représentons dans le plan chacune des contraintes comme des sous-ensembles du plan. L'ensemble des solutions admissibles (qui est l'intersection de toutes ces régions) est alors représenté par un polygone.



On peut prouver que la solution se trouve sur un des sommets du polygone. L'algorithme du simplex démarre sur l'un de ces sommets et passe de sommet en sommet vers la solution optimale, toujours en améliorant la solution courante.