Chapitre 7 : Programmation linéaire en nombres entiers

J.-F. Scheid

Plan du chapitre

I. Introduction et exemples

II. Solutions optimales à valeurs entières

III. Procédures de Séparation et d'Evaluation ("Branch and Bound")
   1. Programmation linéaire en variables binaires
   2. Programmation linéaire en nombres entiers (PLNE) à valeurs bornées

IV. Introduction aux méthodes des coupes pour les PLNE

I. Introduction et exemples

- PL en *nombres entiers* déjà rencontrés : pb d'affectation, pb de flot maximal, pb de production en nb entiers
- Le caractère **entier** de la solution résulte directement de la structure du programme et plus précisément des propriétés de la matrice A des contraintes ($A\mathbf{x} \leq \mathbf{b}$).
- Pour certains problèmes où on cherche une solution optimale entière (par ex. quantité entière de produit ...), il faut inclure la contrainte de nombres entiers dans le programme, sans quoi la solution optimale n'est pas entière (pb de sac à dos, pb de remplissage)

Exemple 1. Problème de sac à dos (knapsack)

Un randonneur emporte dans son sac à dos un poids limité à P. Chaque objet i qu'il peut emporter pèse un poids p_i et possède une utilité c_i pour la randonnée.

Quels objets le randonneur doit-il emporter pour maximiser l'utilité totale, sans dépasser le poids total permis ?

Modélisation du problème de sac à dos

- n objets de poids p_i et d'utilité c_i
- Variables $x_i = \begin{cases} 1 & \text{si le randonneur prend l'objet } i \\ 0 & \text{sinon} \end{cases}$

Problème de sac à dos

$$\begin{cases} \max\limits_{\mathbf{x}} \left[F(\mathbf{x}) = \quad \right] & \leftarrow \text{maximisation de l'utilité totale} \\ & \leftarrow \text{limitation du poids total} \\ x_i \in \{0, 1\} \end{cases}$$

Exemple 2. Problème de remplissage (généralisation)

On veut remplir la cale d'un bateau, d'un entrepôt ... limité en poids (P) et volume (V).

- n objets de poids p_i, de volume v_i et d'utilité c_i.
- Variables x_i = quantité de l'objet i avec $x_i \in \{0, \cdots, m\}$.

Problème de remplissage

$$\begin{cases} \max\limits_{\mathbf{x}} \left[F(\mathbf{x}) = \sum\limits_{i=1}^{n} c_i x_i \right] & \leftarrow \text{ maximisation de l'utilité totale} \\ \sum\limits_{i=1}^{n} p_i x_i \leq P & \leftarrow \text{ limitation du poids total} \\ & \leftarrow \text{ limitation du volume total} \\ x_i \in \{0, \cdots, m\} \end{cases}$$

Remarques.

- Pb de sac à dos et remplissage = pbs typiques de programmation linéaire en nombres entiers (PLNE).
- Mais, il ne s'agit plus strictement de programmation linéaire car l'ensemble des solutions réalisables *n'est plus un polyèdre* mais un **ensemble discret de points**.

II. Solutions optimales à valeurs entières

PL sous forme standard

$$(PL) \begin{cases} \max\limits_{\mathbf{x}\in\mathbb{R}^n} \left[F(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}\right] \\ A\mathbf{x} = \mathbf{b} \\ \mathbf{x} \geq 0 \end{cases}$$

où A matrice $m \times n$ à coefficients **entiers**. Les vecteurs $\mathbf{c}$ et $\mathbf{b}$ sont **entiers**.

En général, la solution optimale de (PL) n'est pas *entière*.

On cherche une condition suffisante sur la matrice A pour que la solution optimale soit **entière**.

Définition

Une matrice A de taille $m \times n$ est dite **totalement unimodulaire** si toute sous-matrice carrée de A a un déterminant qui vaut 0, $+1$ ou -1.

Une *sous-matrice* est une matrice obtenue à partir d'une matrice en ne gardant que certaines lignes ou colonnes.

Remarque. Toute matrice *totalement unimodulaire* est nécessairement composée de 0, $+1$ ou -1.

Théorème

Soit A une matrice **totalement unimodulaire** et **b** et **c** des vecteurs **entiers**. Si (PL) admet une solution optimale, il admet une solution optimale **entière**.

Remarque. Le résultat est encore vrai si (PL) est écrit sous forme canonique pure avec des contraintes inégalités de la forme $A\mathbf{x} \leq \mathbf{b}$ où A est *totalement unimodulaire*.

Comment reconnaître une matrice totalement unimodulaire ?

Proposition (Condition suffisante)

Soit A une matrice contenant seulement les éléments 0, $+1$ ou -1 et qui satisfait les 2 conditions suivantes :

1. Chaque colonne contient **au plus** 2 éléments non-nuls.
2. Les lignes de A peuvent être partitionnées en 2 sous-ensembles I_1 et I_2 tels que pour chaque colonne contenant 2 éléments non-nuls :
   - si les 2 éléments non-nuls ont *le même signe* alors l'un est dans I_1 et l'autre dans I_2.
   - si les 2 éléments non-nuls sont *de signes différents* alors ils sont tous les deux dans I_1 ou tous les deux dans I_2.

Alors A est **totalement unimodulaire**.

Applications :

⋆ Problème d'affectation : programmation linéaire (cf. Chapitre 6)

	C_1	C_2	C_3	C_4
L_1				
L_2				
L_3				

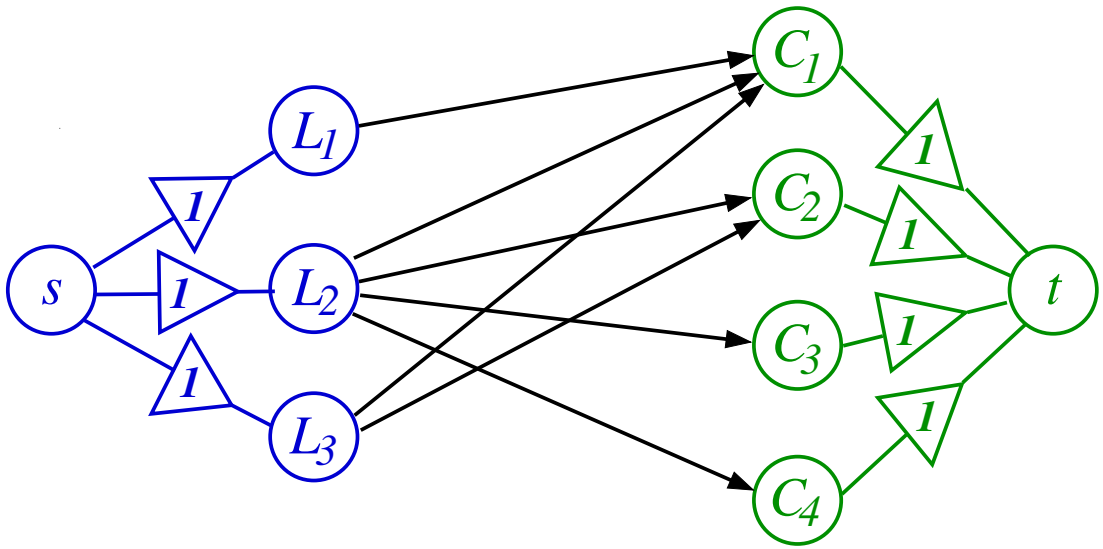
$$\max_{\mathbf{t}} F(\mathbf{t}) = \mathbf{c}^\top \mathbf{t}$$
$$\begin{cases} A\mathbf{t} \leq \mathbf{b} \\ \mathbf{t} \geq 0 \end{cases}$$

$$\mathbf{t} = \left(\; t_{11}, t_{21}, t_{22}, t_{23}, t_{24}, t_{31}, t_{32} \; \right)^\top \in \mathbb{R}^7$$
$$\mathbf{b} = \mathbf{c} = \left(\; 1, 1, 1, 1, 1, 1, 1 \; \right)^\top \in \mathbb{R}^7$$

$$A = \left(\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array} \right)$$

La matrice A est **totalement unimodulaire** $\Rightarrow$ _____.

⋆ Problème d'affectation : flot maximal dans le graphe biparti complété (cf. Chapitre 6)



$$\max_{\mathbf{f},v} [F_2 = v]$$
$$\begin{cases} A\mathbf{f} + \mathbf{v} = 0 \\ \mathbf{f} \leq 1 \\ \mathbf{f} \geq 0 \end{cases}$$

$$\mathbf{f} = \left(\; f_{s1}, f_{s2}, f_{s3} \,|\, f_{11}, f_{21}, f_{22}, f_{23}, f_{24}, f_{31}, f_{32} \,|\, f_{1t}, f_{2t}, f_{3t}, f_{4t} \; \right)^\top \in \mathbb{R}^{14}$$

$$\mathbf{v} = \left(\; -v, 0, 0, 0, 0, 0, 0, 0, +v \; \right)^\top \in \mathbb{R}^9 ;$$

Matrice A de taille 9×14 :

$$A = \left(\begin{array}{ccc|ccccccc|cccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \end{array}\right)$$

A_i : ligne i de A.

A est **totalement unimodulaire** $\Rightarrow$ _____.

Remarque. Dans les pb de sac-à-dos/remplissage, la solution optimale n'est pas entière en général.

$\Rightarrow$ il faut rajouter **la contrainte d'intégrité** dans le problème.

III. Procédures de Séparation et d'Evaluation (Branch and Bound)

1) Programmation linéaire en variables binaires

a) Cas des coefficients positifs

Problème de sac à dos

$$\begin{cases} \max\limits_{\mathbf{x}} \left[F(\mathbf{x}) = \sum\limits_{i=1}^{n} c_i x_i \right] \\ \sum\limits_{i=1}^{n} a_i x_i \leq d \\ x_i \in \{0, 1\} \end{cases}$$

avec a_i et c_i **positifs ou nuls**.

$\rightarrow$ 2^n cas à examiner a priori. On réduit ce nombre en procédant par "*Séparation et Evaluation*" (Branch and Bound).

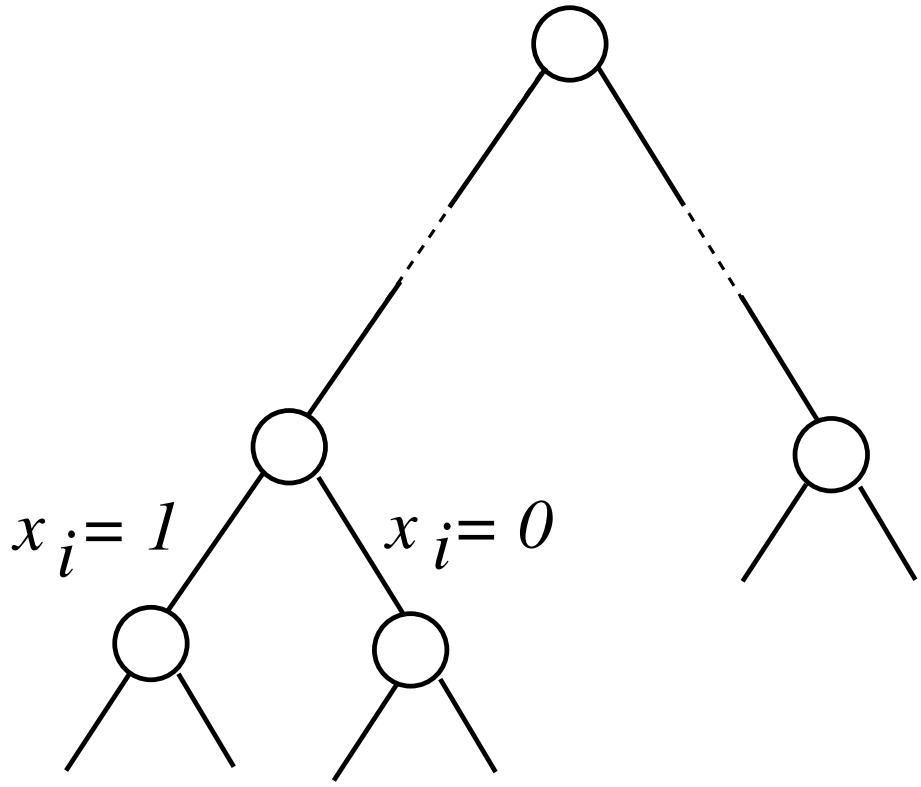
Principe des *Procédures de Séparation et Evaluation* (PSE)

⋆ Séparation

L'examen d'un objet (le prendre ou le laisser) **sépare** l'ensemble des solutions possibles en 2.

⇒ Développement d'une arborescence dont chaque sommet correspond à un sous-ensemble de solutions réalisables.

La racine de l'arborescence représente l'ensemble de toutes les solutions réalisables.


$x_i = 1$
$x_i = 0$

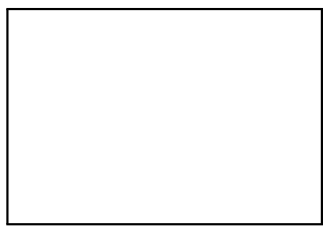
⋆ Evaluation

- Estimation principale.

  Pour chaque sommet S_k, on **évalue** une valeur b_k qui est un majorant de la meilleure solution correspondant à S_k (un minorant si on a un "min").

  • Si $b_k < \overline{F}$ où $\overline{F}$ est la valeur de F pour une solution réalisable connue, alors le sommet S_k n'est pas exploré : S_k ne peut pas contenir une meilleure solution que celle déjà obtenue ($b_k < \overline{F} \leq \max F$).

  • Initialement, pour le sommet S_0, on choisit

  (correspond à prendre $x_1 = x_2 = \cdots = x_n = 1$ dans F)

- Estimation secondaire.

  Pour éviter de construire des sommets S_k qui n'ont pas de solution réalisable, on évalue une valeur secondaire e_k qui est une variable d'écart associée à la contrainte et à S_k.

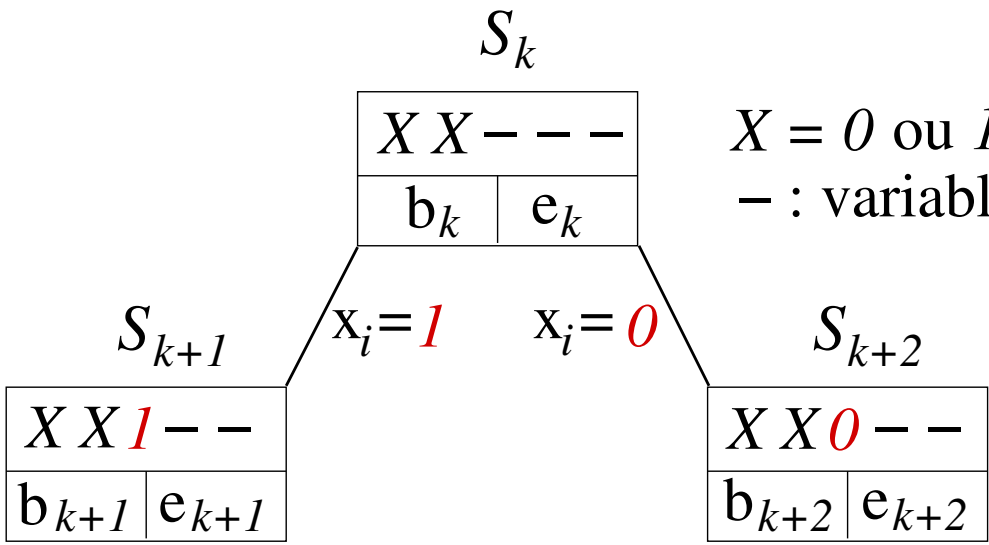
  - Si $e_k < 0$ alors le sommet S_k n'est pas construit.
  - Initialement, pour le sommet S_0, on choisit

    $\boxed{}$ (d : second membre)

  (correspond à prendre $x_1 = x_2 = \cdots = x_n = 0$ dans les contraintes)

⋆ Calcul des estimations b_k et e_k

Examen de la variable x_i correspondant au sommet S_k.



Mise à jour des estimations principale et secondaire

Stratégie de parcours : on explore en priorité le sommet S_k qui a la meilleure estimation principale b_k.

$\star$ Détermination de $\overline{F}$: valeur de F pour une solution réalisable connue.

Deux façons de construire une solution réalisable

1. On renumérote les variables par leurs coefficients dans F décroissants : $c_1 \geq c_2 \geq \cdots \geq c_n$.

   On regarde si on peut donner la valeur $x_1 = 1$ puis on examine x_2, etc ... $\rightarrow$ on obtient la valeur $\overline{F}_1$

2. On renumérote les variables par les coefficients $\frac{c_i}{a_i}$ décroissants : $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}$.

   On regarde si on peut donner la valeur $x_1 = 1$ puis on examine x_2, etc ... $\rightarrow$ on obtient la valeur $\overline{F}_2$

On choisit

$$\boxed{\overline{F} = \max(\overline{F}_1, \overline{F}_2)}$$

Exemple

$$\begin{cases} \max\limits_{\mathbf{x}} [F(\mathbf{x}) = 16x_1 + 18x_2 + 15x_3] \\ x_1 + 4x_2 + 3x_3 \leq 7 \\ x_i \in \{0, 1\} \end{cases}$$

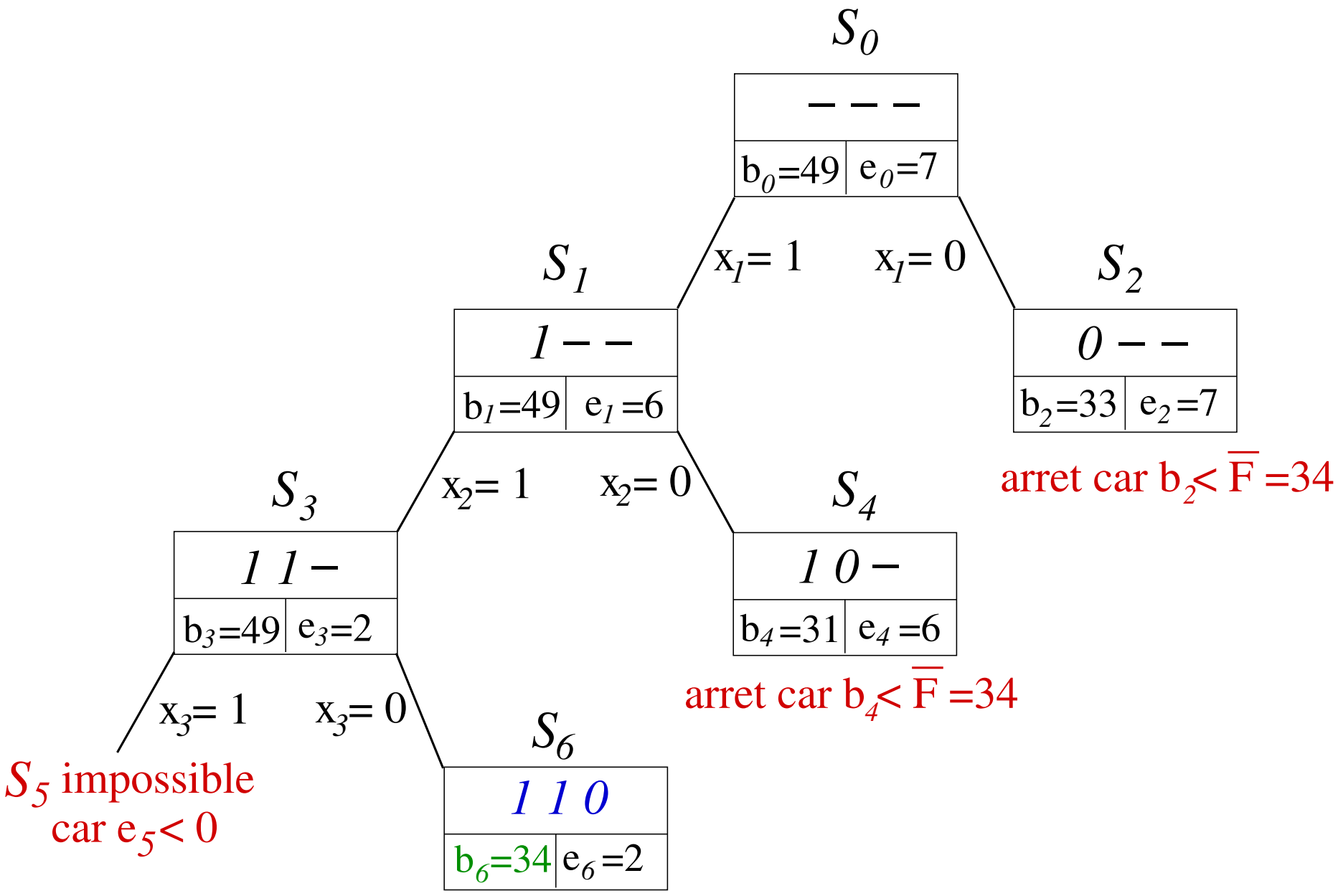
- Estimations initiales : $b_0 = c_1 + c_2 + c_3 = 49\,;\ e_0 = 7.$
- Détermination de $\overline{F}$

i	1	2	3
c_i	16	18	15
c_i/a_i	16	4,5	5

  - ⋆ Calcul de $\overline{F}_1$:

  - ⋆ Calcul de $\overline{F}_2$:

On choisit $\overline{F} = \max(\overline{F}_1, \overline{F}_2) =$



On obtient $\max F = 34$ avec $x_1^* = 1$, $x_2^* = 1$, $x_3^* = 0$.

b) Cas général : cas où les coefficients *ne sont plus nécessairement positifs*.

S'il existe des coefficients c_i négatifs, on se ramène au cas des coeff. positifs par un changement de variables :

Si $c_k < 0$ alors on pose

$$\boxed{y_k = 1 - x_k \in \{0, 1\}}$$

On obtient alors

$$F(\mathbf{x}) = \sum_{\substack{l \text{ tq} \\ c_l > 0}} c_l x_l + \sum_{\substack{k \text{ tq} \\ c_k < 0}} c_k x_k = \sum_{\substack{l \text{ tq} \\ c_l > 0}} c_l x_l + \sum_{\substack{k \text{ tq} \\ c_k < 0}} (-c_k) y_k + \sum_{\substack{k \text{ tq} \\ c_k < 0}} c_k$$

De même

$$\sum_{i=1}^{n} a_i x_i = \sum_{\substack{l \text{ tq} \\ c_l > 0}} a_l x_l + \sum_{\substack{k \text{ tq} \\ c_k < 0}} a_k x_k = \sum_{\substack{l \text{ tq} \\ c_l > 0}} a_l x_l + \sum_{\substack{k \text{ tq} \\ c_k < 0}} (-a_k) y_k + \sum_{\substack{k \text{ tq} \\ c_k < 0}} a_k$$

- Si $c_l > 0$, alors on pose

$$\boxed{c'_l = c_l > 0, \quad a'_l = a_l, \quad x'_l = x_l}$$

- Si $c_k < 0$, alors on pose

$$\boxed{c'_k = -c_k > 0, \quad a'_k = -a_k, \quad x'_k = y_k}$$

$$\Rightarrow \quad F(\mathbf{x}') = \sum_{i=1}^{n} c'_i x'_i + \sum_{\substack{k \text{ tq} \\ c_k < 0}} c_k$$

$$\sum_{i=1}^{n} a_i x_i = \sum_{i=1}^{n} a'_i x'_i + \sum_{\substack{k \text{ tq} \\ c_k < 0}} a_k$$

Le problème du sac à dos est équivalent à

$$\begin{cases} \max\limits_{\mathbf{x}'} \left[F'(\mathbf{x}') = \sum\limits_{i=1}^{n} c'_i x'_i \right] \\ \sum\limits_{i=1}^{n} a'_i x'_i \ \leq \ d' = d - \sum\limits_{\substack{k \text{ tq} \\ c_k < 0}} a_k \\ x'_i \in \{0, 1\} \end{cases}$$

On a $c'_i > 0$ mais a'_i **de signe quelconque**.

$\Rightarrow$ On procède donc comme pour le cas des coefficients positifs **sauf pour les estimations secondaires** e_k (variables d'écart).

On note I_+ (resp. I_-) l'ensemble des indices i tels que $a'_i > 0$ (resp. $a'_i < 0$). On a

$$e' = d' - \sum_{i=1}^{n} a'_i x'_i = \underbrace{d' - \sum_{i \in I_-} \underbrace{a'_i}_{<0} x'_i}_{=\tilde{e}'} - \sum_{j \in I_+} \underbrace{a'_j}_{>0} x'_j$$

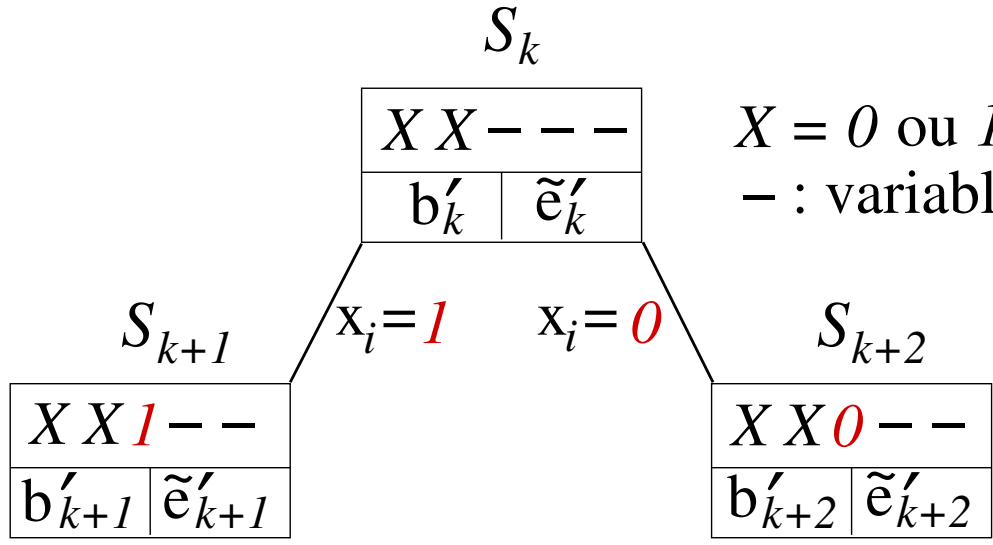
$$\Rightarrow \qquad \boxed{e' \leq \tilde{e}' = d' - \sum_{i \in I_-} a'_i x'_i}$$

On prend comme estimations secondaires les majorants $\tilde{e}'$ de la variable d'écart e' avec :

- $b'_0 = \sum_{i=1}^{n} c'_i$
- $\tilde{e}'_0 = d' - \sum_{i \in I_-} a'_i$

⋆ Calcul des estimations b'_k et $\tilde{e}'_k$

Examen de la variable x'_i correspondant au sommet S_k.



Mise à jour des estimations principale et secondaire

$$b'_{k+1} = b'_k$$
$$\tilde{e}'_{k+1} = \begin{cases} \tilde{e}'_k - a'_i & \text{si } a'_i > 0 \\ \tilde{e}'_k & \text{si } a'_i \leq 0 \end{cases}$$

$$b'_{k+2} = b'_k - c'_i$$
$$\tilde{e}'_{k+2} = \begin{cases} \tilde{e}'_k & \text{si } a'_i > 0 \\ \tilde{e}'_k + a'_i & \text{si } a'_i \leq 0 \end{cases}$$

Exemple

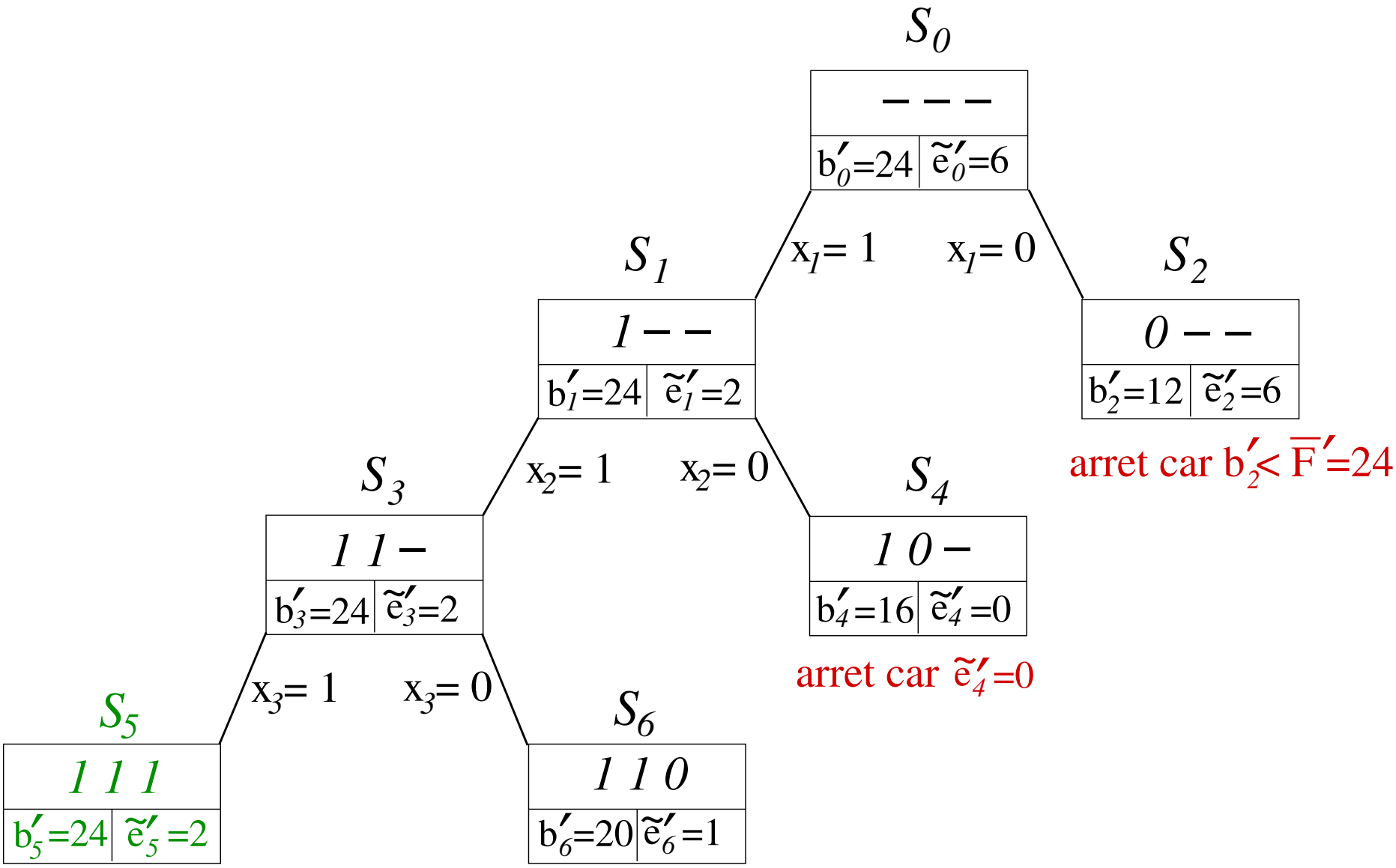
$$\begin{cases} \max\limits_{\mathbf{x}} [F(\mathbf{x}) = 12x_1 - 8x_2 + 4x_3] \\ 4x_1 + 2x_2 - x_3 \leq 5 \\ x_i \in \{0, 1\} \end{cases}$$

On pose $y_2 = x'_2 = 1 - x_2$.

Problème équivalent ($x'_1 = x_1$, $x'_3 = x_3$) :

$$\begin{cases} \max\limits_{\mathbf{x}'} \left[F'(\mathbf{x}') = 12x'_1 + 8x'_2 + 4x'_3\right] \\ 4x'_1 - 2x'_2 - x'_3 \leq 3 \\ x'_i \in \{0, 1\} \end{cases}$$

- Estimations initiales :
  $b'_0 = c'_1 + c'_2 + c'_3 = 12 + 8 + 4 = 24$;
  $\tilde{e}'_0 = d' - (a'_2 + a'_3) = 3 + 2 + 1 = 6 \quad (a'_2, a'_3 < 0)$
- Détermination de $\overline{F}'$ par coefficients c'_i décroissants :
  on examine d'abord x'_1 : $x'_1 = 1$; puis $x'_2 = 1$ et enfin $x'_3 = 1 \Rightarrow \overline{F}' = 24$



On obtient $\max F' = 24$ avec $x_1'^* = 1$, $x_2'^* = 1$, $x_3'^* = 1$.

Retour au pb initial : $x_1^* = 1$, $x_2^* = 0$, $x_3^* = 1$ et $\max F = 16$.

c) Problème de sac à dos généralisé (pb de remplissage)

$$\begin{cases} \max\limits_{\mathbf{x}} \left[F(\mathbf{x}) = \sum\limits_{j=1}^{n} c_j x_j \right] \\ \sum\limits_{j=1}^{n} a_{ij} x_j \leq d_i \quad \forall i = 1, \cdots, m \\ x_j \in \{0, 1\} \quad \forall j = 1, \cdots, n \end{cases}$$

Même façon de procéder que dans le cas d'une seule contrainte.

A chaque étape k, il y a maintenant m évaluations secondaires $e_{k,1}, \cdots, e_{k,m}$ correspondant à chacune des contraintes.

2) PL en nombres entiers à valeurs bornées.

On suppose que les variables sont à valeurs entières **bornées** et qu'on connait les bornes. Par exemple, les variables x_j du problème de sac à dos sont telles que

$$x_j \in \{0, 1, \cdots, m_j\}$$

On utilise les PSE en considérant des séparations possibles en $(m_j + 1)$ sous-ensembles pour l'examen de la variable x_j.

Remarque. On a intérêt à séparer en dernier les variables avec une valeur m_j grande.

Programmation linéaire en nombres entiers (cas général)

$$(PLNE)\begin{cases} \max\limits_{\mathbf{x}} \left[F(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}\right] \\ A\mathbf{x} \leq \mathbf{b} \\ \mathbf{x} \geq 0 \\ \mathbf{x} \text{ entier} \end{cases}$$

Principe général des méthodes de coupes.

Il s'agit de méthodes itératives. A chaque itération :

1. on résout un (PL) sans contrainte d'intégrité sur les variables (simplexe). On obtient une solution optimale qui n'est pas nécessairement **entière** (sinon, c'est gagné ...).
2. on rajoute une contrainte supplémentaire qui va éliminer la solution non-entière obtenue précédemment, mais sans éliminer de solution entière au problème de départ (PLNE). Cette contrainte supplémentaire s'appelle une coupe.
3. on recommence jusqu'à obtenir une solution entière.

Différentes coupes possibles.

1) Coupes entières

1. On commence par résoudre le problème **sans la contrainte** d'intégrité sur les variables : *relaxation linéaire* de $(PLNE)$. On note ce problème $(\mathcal{P}_0)$.

   On obtient (ou pas...) une solution optimale $\mathbf{x}^*$ qui n'est pas nécessairement **entière**.

2. Supposons qu'il existe une composante x_k^* **non-entière**.
   Pour obliger la variable x_k à être entière, on sépare l'ensemble des solutions en 2 sous-ensembles *disjoints* :

$$\{x_k \leq \lfloor x_k^* \rfloor\} \quad \text{et} \quad \{x_k \geq \lfloor x_k^* \rfloor + 1\}$$

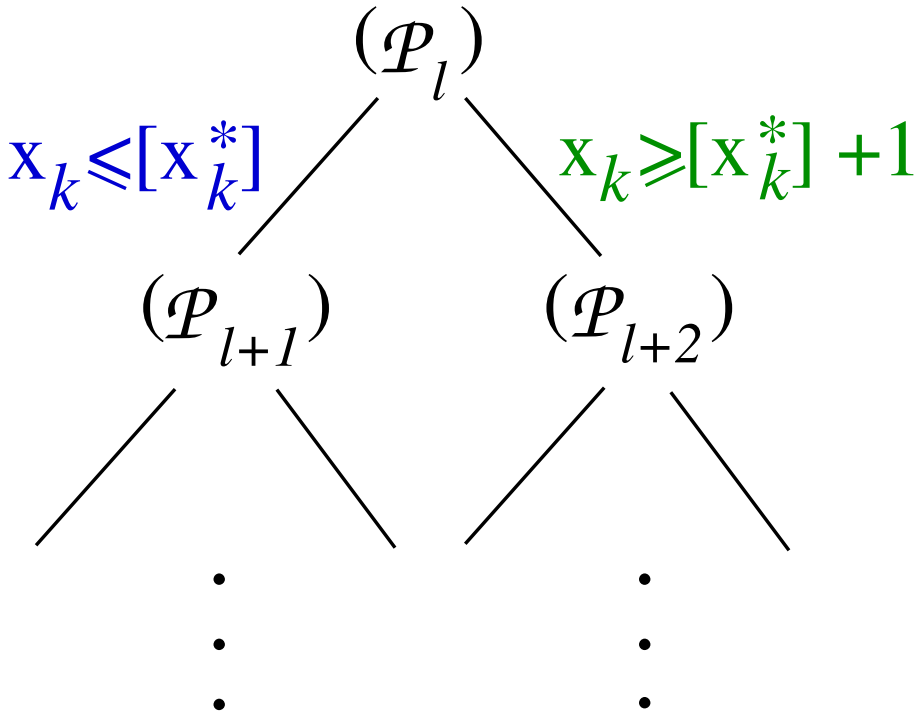
   $\lfloor \cdot \rfloor$: partie entière

3. On résout alors 2 problèmes $(\mathcal{P}_1)$ et $(\mathcal{P}_2)$ avec les contraintes **supplémentaires** issues de la séparation :

$$(\mathcal{P}_1) \begin{cases} \max\limits_{\mathbf{x}} \left[F(\mathbf{x}) = \mathbf{c}^\top \mathbf{x} \right] \\ A\mathbf{x} \leq \mathbf{b} \\ x_k \leq \lfloor x_k^* \rfloor \\ \mathbf{x} \geq 0 \end{cases}$$

   et même chose pour $(\mathcal{P}_2)$ avec la contrainte $x_k \geq \lfloor x_k^* \rfloor + 1$ à la place.

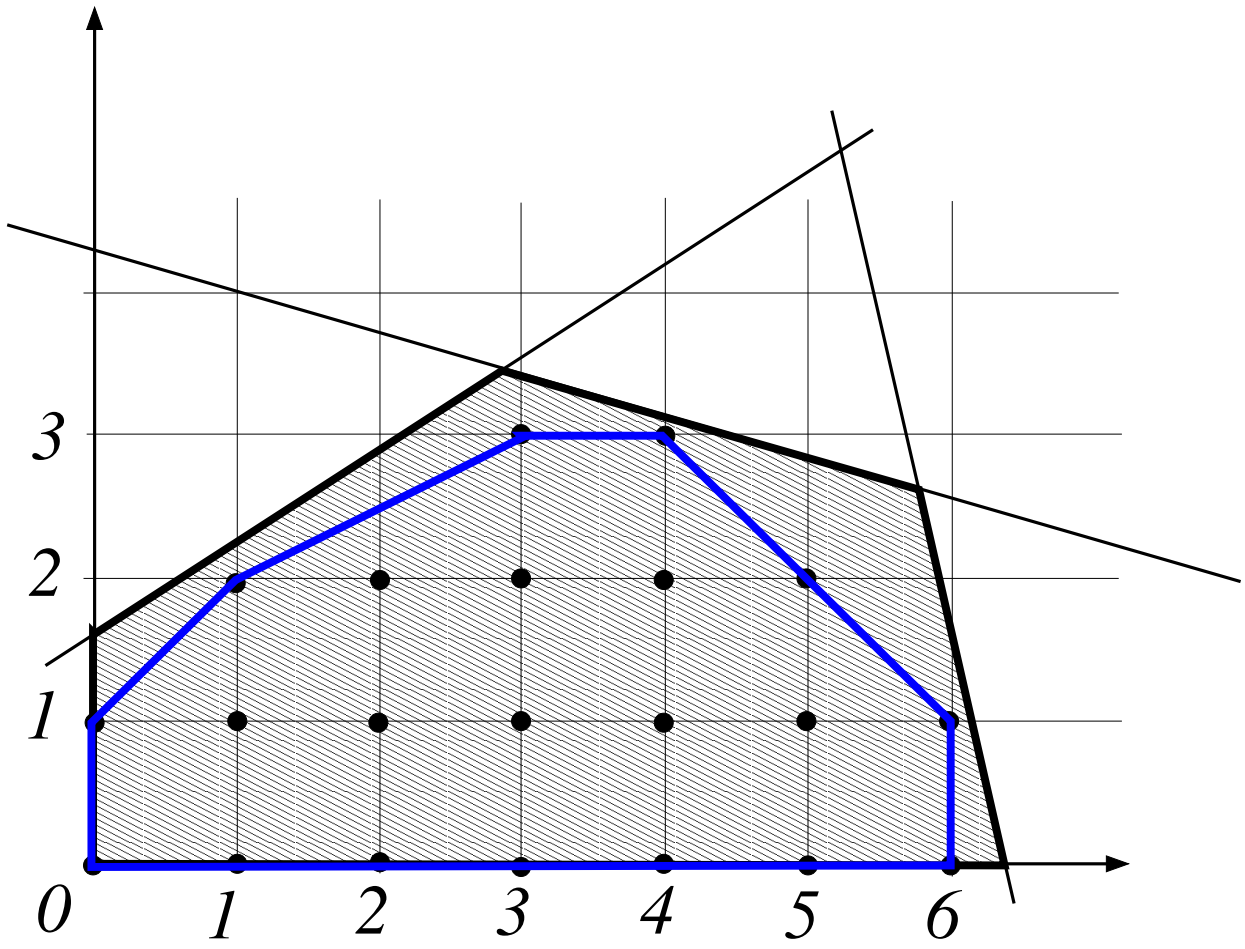
On itère jusqu'à obtenir des variables **entières**.



2) Coupes de Gomory

$$(PLNE)\begin{cases} \max\limits_{\mathbf{x}} \left[F(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}\right] \\ A\mathbf{x} \leq \mathbf{b} \\ \mathbf{x} \geq 0 \\ \mathbf{x} \text{ entier} \quad (\mathbf{x} \in \mathbb{N}^n) \end{cases}$$

Idée : On remplace les contraintes dans $(PLNE)$ par d'autres contraintes linéaires (les **coupes**) qui correspondent à l'**enveloppe convexe** des variables **entières** se trouvant à l'intérieur du polyèdre $\{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq 0\}$.


3
2
1
0
1
2
3
4
5
6

Remarque : On peut combiner l'ajout des contraintes (les coupes) et la séparation/évaluation (Branch-and-bound) des solutions. On obtient des méthodes de type *Branch-and-cut*.

La plupart des solveurs modernes utilisent des méthodes *Branch-and-cut*.

Quelques solveurs PLNE

- CPLEX (IBM ILOG). Code commercial mais licence éducation gratuite.
- GLPK (GNU Linear Programming Kit)
- COIN-OR (COmputational INfrastructure for Operations Research), Logiciel OpenSource.
- KNITRO. Code commercial.
- Xpress Optimizer. Code commercial, version 'bridée' gratuite.

Services en ligne

- AMPL (Automatic Mathematical Programming Language) ; version 'bridée' gratuite (nb variables ≤ 300)
- NEOS Solvers