### 10.3.3 Algorithme du simplexe

J'ai volontairement décidé d'abandonner une notation cryptique du type $f(a, b, c, d, ...)$ pour un appel de fonction. J'écris directement un problème de programmation linéaire de la façon suivante :

$$\textbf{maximiser } v +^t \overrightarrow{c}\,\overrightarrow{x}_N \quad \begin{cases} x_B = \overrightarrow{b} - A\overrightarrow{x}_N \\ \overrightarrow{x} \geq 0 \end{cases}$$

où $v$ est la valeur déjà atteinte, $N$ sont les indices des variables hors base ('nulles') et $B$ sont les indices des variables dans la base. Un tel problème de programmation linéaire est passé en argument des fonctions *pivot* et *simplexe* décrites ci-dessous.

Tout d'abord, on considère la fonction *pivot* (algorithme 41) qui transforme un problème de programme linéaire en pivant les variables de la manière suivante : $i_0$ sort de la base $B$ et entre dans l'ensemble $N'$ alors que $j_0$ sort de l'ensemble $N$ et entre dans la nouvelle base $B'$. Intuitivement, $x_{i_0}$ va devenir nulle et $x_{j_0}$ prendra une valeur positive, à savoir $\frac{b_{i_0}}{a_{i_0,j_0}}$. Cela aura pour effet d'augmenter la valeur de la fonction objectif. Le pivot correspond à se promener d'un sommet vers un sommet voisin sur le polyèdre en empruntant une arête.

---

**Algorithme 41 :** $pivot\left(\textbf{maximiser } v +^t \overrightarrow{c}\,\overrightarrow{x}_N \quad \begin{cases} x_B = \overrightarrow{b} - A\overrightarrow{x}_N \\ \overrightarrow{x} \geq 0 \end{cases}, i_0, j_0\right)$

---

**Entrées** : Un problème de programmation linéaire, $i_0$, $j_0$ des indices de variables avec $i_0 \in B$ et $j_0 \in N$

**Sorties** : Le problème de programmation linéaire dans lequel $i_0$ sort de la base $B$ et $j_0$ y entre

1 **retourner** $\textbf{maximiser } v' +^t \overrightarrow{c}'\overrightarrow{x}_{N'} \quad \begin{cases} x_{B'} = \overrightarrow{b}' - A'\overrightarrow{x}_{N'} \\ \overrightarrow{x} \geq 0 \end{cases}$ où pour tout $i \in B$, $j \in N$ :

- $N' = \{i_0\} \cup N \setminus \{j\}$ ;
- $B' = \{j_0\} \cup B \setminus \{i\}$ ;
- $v' = v + \frac{b_{i_0} c_{j_0}}{a_{i_0,j_0}}$
- $c'_j = c_j - c_{j_0} \frac{a_{i_0,j}}{a_{i_0,j_0}}$
- $c'_{i_0} = \frac{-c_{j_0}}{a_{i_0,j_0}}$
- $b'_i = b_i - a_{i,j_0} \frac{b_{i_0}}{a_{i_0,j_0}}$
- $b'_{j_0} = \frac{b_{i_0}}{a_{i_0,j_0}}$
- $a'_{i,j} = a_{i,j} - \frac{a_{i,j_0} a_{i_0,j}}{a_{i_0,j_0}}$
- $a'_{i,i_0} = \frac{a_{i,j_0}}{a_{i_0,j}}$
- $a'_{j_0,i_0} = \frac{1}{a_{i_0,j_0}}$
- $a'_{j_0,j} = \frac{-a_{i_0,j}}{a_{i_0,j_0}}$.

---

La fonction *simplexe* (algorithme 42) prend en entrée un problème de la programmation linéaire et retourne la valeur maximale (potentiellement $+\infty$ si la fonction linéaire à maximiser n'est pas borné sur le polyèdre).

**Remarque 10.** *Aux lignes 2 et 4, on choisit à chaque fois des indices les plus petits afin d'éviter les bouclages.*

Le test à la ligne 1 vérifie si on peut effectivement améliorer la valeur actuelle de la fonction linéaire. S'il n'est pas possible d'améliorer la valeur actuelle $v$, on retourne $v$. Sinon, on choisit une variable à augmenter $x_{j_0}$ à augmenter. Le test à la ligne 3 vérifie s'il existe une contrainte qui limite l'augmentation de $x_{j_0}$. S'il n'y a pas de telles contraintes, on peut augmenter $x_{j_0}$ autant

**Algorithme 42 :** *simplexe* $\left( \begin{cases} \textbf{maximiser } v + {}^t\overrightarrow{c}\,\overrightarrow{x}_N \\ \begin{cases} x_B = \overrightarrow{b} - A\overrightarrow{x}_N \\ \overrightarrow{x} \geq 0 \end{cases} \end{cases} \right)$

**Entrées** : Un problème de programmation linéaire
**Sorties** : La valeur maximale que prend la fonction linéaire à maximiser

```
1 si il existe j_0 ∈ N tel que c_{j_0} > 0 alors
2     choisir j_0 ∈ N le plus petit tel que c_{j_0} > 0
3     si il existe i_0 ∈ B tel que −a_{i_0,j_0} < 0 alors
4         choisir i_0 ∈ B le plus petit tel que −a_{i_0,j_0} < 0 et b_{i_0}/a_{i_0,j_0} minimal
5         retourner simplexe( pivot( { maximiser v + ᵗc x_N ; { x_B = b − A x_N ; x ≥ 0 } } , i_0, j_0 ) )
6     sinon
7         retourner +∞
8 sinon
9     retourner v
```

qu'on veut pour augmenter la fonction linéaire à maximiser : on retourne donc $+\infty$. Sinon, ligne 4, on choisit une des contraintes les plus fortes. Il s'agit de la contrainte $x_{i_0} = \dots$. Puis on appelle récursivement *simplexe* sur le problème de programmation linéaire dans lequel $i_0$ est sorti de la base et $j_0$ est entré dans la base.