

Résumé de mathématiques discrètes

Auteur : Samuel Robyr¹, 26 février 2006. Dernière version sur http://epfl.neoch.net/

1 Programmation linéaire

Forme canonique

Max z = \sum_{j=1}^n c_j x_j \quad \text{problème de maximisation}

s.c. \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m \quad \text{contraintes "≤"}

x_j \geq 0, \quad j = 1, \dots, n \quad \text{variables ≥ 0}

Forme standard

Max z = \sum_{j=1}^n c_j x_j \quad \text{problème de maximisation}

s.c. \sum_{j=1}^n a_{ij} x_j = b_i, \quad i = 1, \dots, m \quad \text{contraintes "="}

x_j \geq 0, \quad j = 1, \dots, n \quad \text{variables ≥ 0}

On passe de la forme canonique à la forme standard en rajoutant des variables d’écart.

1.1 Règles de transformation

Minimisation ⇔ Maximisation min(z) = max(−z)

Inéquations ax ≥ b ⇔ (−a)x ≤ (−b)

Équation ⇒ Inéquation ax = b ⇔ { ax ≤ b, −ax ≤ −b }

Inéquation ⇒ Équation ax ≤ b ⇔ ax + s = b, s ≥ 0

ax ≥ b ⇔ −ax + s = −b, s ≥ 0

Variable bornée inférieurement x ≥ b ⇔ { x' = x − b, x' ≥ 0 }

Variable réelle ⇒ variable non négative

x ∈ ℝ ⇒ x = x⁺ − x[−], avec x⁺, x[−] ≥ 0

Maximum / Minimum

min z = max{c₁x + d₁, ..., c_nx + d_n} ⇔ { min z = t, s.c. c_ix + d_i − t ≤ 0, t ∈ ℝ }

Valeurs absolues (fonction objectif)

min z = |x| ⇔ { min z = t, s.c. x − t ≤ 0, −x − t ≤ 0, t ≥ 0 }

Valeurs absolues (s.c.) |x| + a ≤ b ⇔ { x + a ≤ b, −x + a ≤ b }

Attention : |x| ≥ b n’est pas linéaire !

1.2 Règles de dualisation

Problème de maximisation	↔	Problème de minimisation
Variable x _j ≥ 0	↔	j ^e contrainte de type ≥
Variable x _j ∈ ℝ	↔	j ^e contrainte de type =
Variable x _j ≤ 0	↔	j ^e contrainte de type ≤
i ^e contrainte de type ≤	↔	Variable y _i ≥ 0
i ^e contrainte de type =	↔	Variable y _i ∈ ℝ
i ^e contrainte de type ≥	↔	Variable y _i ≤ 0

Si un programme linéaire primal a n variables et m contraintes, alors le programme linéaire dual a m variables et n contraintes. Les variables basiques du programme linéaire dual sont associées aux variables hors base du programme linéaire primal.

1.3 Théorème des écarts complémentaires

Soit x une solution admissible d’un PL et λ celle de son dual. Ces deux solutions sont optimales si :

λ_i (b_i − ∑_j a_{ij} x_j) = 0 ∀ i et (∑_i λ_i a_{ij} − c_j) x_j = 0 ∀ j

1.4 Dictionnaires (aussi appelés tableaux)

On construit le dictionnaire en partant du P.L. standard :

max z = c_D x_D

s.c. Ax_D + Ix_E = b

x_D, x_E ≥ 0

z =

0	−x _D
−c _D	

x_E =

b	A
---	---

Le dictionnaire associé à une base B de Ax = b est

z =

z ₀ = c _B B ^{−1} b	−x _N
β = B ^{−1} b	−γ _N = c _B B ^{−1} N − c _N

x_B =

β = B ^{−1} b	α = B ^{−1} N
-----------------------	-----------------------

Avec z₀ = c_BB^{−1}b, −γ_N = c_BB^{−1}N − c_N, β = B^{−1}b et α = B^{−1}N

1.5 Algorithme du Simplexe

Tout d’abord, il faut formuler le problème de départ sous forme canonique, puis standard. De la forme standard, l’on construit le tableau initial correspondant.

Si le tableau initial est admissible, on passe directement à la phase II ; autrement il faut effectuer la phase I pour le rendre admissible avant de passer à la phase II.

Règle de Bland. Lorsque plusieurs candidats sont susceptibles d’entrer ou de sortir de la base, les départager en choisissant toujours la variable x_k ayant le plus petit indice k.

1.5.1 Pivotage dans un dictionnaire

Pivotage autour de a :

- 1. Remplacer le pivot a(≠ 0) par 1/a ;
- 2. Diviser les autres éléments de la ligne du pivot par a et ceux de la colonne du pivot par −a ;
- 3. Pour les autres éléments utiliser la règle du rectangle ;

- 4. Echanger la variable associée à la ligne du pivot avec celle associée à la colonnes du pivot ;

x_j

c	−x _i
a	a
f	d

→

x_i

c	x _j
a	1/a
f − cd/a	−d/a

1.5.2 Algorithme du Simplexe (Phase I)

Donnée : Un tableau non-admissible.

Résultat : Un tableau admissible ou non-borné.

- 1. Construction d’un problème auxiliaire :
 - (a) Introduire la variable artificielle x₀ dans le tableau valant −1 pour toutes les contraintes vérifiant β_i < 0 (0 sinon).
 - (b) Ajouter la fonction objectif auxiliaire (à maximiser) z’ = −x₀.
 - (c) Faire entrer x₀ dans la base en pivotant dans une ligne r ∈ { i | b_i = min_{k|b_k<0} (b_k) }
- 2. Résoudre le P.L. auxiliaire à l’aide de la phase II de l’algorithme du simplexe et en utilisant la règle de Bland. Après résolution :
 - Si z’ = 0, supprimer la colonne de x₀ et la ligne de z’. Le tableau initial est maintenant admissible.
 - Si z’ < 0, le problème de départ n’admet pas de solutions admissible.

1.5.3 Algorithme du Simplexe (Phase II)

Donnée : Tableau primal admissible.

Résultat : Tableau optimal ou non-borné.

- 1. Choix du pivot :
 - Colonne j : −γ_j négatif (+ règle de Bland si le tableau est dégénéré).
 - S’il n’existe pas de colonne ayant un coût marginal négatif : STOP, le tableau est optimal.
 - Ligne r : min (β_i/α_{ij}) pour α_{ij} > 0 (quotient caractéristique ; en cas d’égalité, on prend le max(α_{ij})).
 - Si une telle ligne n’existe pas : STOP, le tableau est non-borné.
- 2. Pivoter autour de α_{rj} et retourner au point 1.

¹Ce résumé est une version modifiée de Recherche Opérationnelle écrit par François Bochatay & Christina Hauenstein. Il contient aussi des parties de Formulaire de Recherche Opérationnelle écrit par Arnaud Zuffery. Un grand merci à tous pour avoir écrit et distribué ces documents. Page 1.

1.5.4 Algorithme Dual du Simplexe (Phase II)

Donnée : Tableau dual admissible.

Résultat : Tableau optimal ou certificat d'absence de solution admissibles.

1. Choix du pivot :
Ligne r : $\beta_i < 0$ et règle de Bland. Si une telle ligne n'existe pas : STOP, la tableau est optimal.
Colonne j : $\max \left(\frac{-\gamma_i}{\alpha_{ij}} \right)$ pour $\alpha_{ij} < 0$ (quotient caractéristique dual) Si une telle colonne n'existe pas : STOP, le tableau est non-borné et le primal sans solution admissible.
2. Pivoter autour de α_{rj} et retourner au point 1.

1.6 Analyse de sensibilité

1.6.1 Sensibilité du membre de droite

Dans quel intervalle δ_i peut varier le $i^{\text{ème}}$ coefficient de β sans que la base optimale ne change ?

$$\delta_i \vec{B_i}^{-1} \geq -\vec{\beta}$$

où $\vec{B_i}^{-1}$ est la colonne correspondant à la $i^{\text{ème}}$ variable d'écart du système.
La nouvelle solution optimale en fonction de $b_i \in [\beta_i \pm \delta_i]$ est donnée par

$$\vec{b} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix} \qquad x_B^* = \begin{pmatrix} x_1^* \\ \vdots \\ x_n^* \end{pmatrix} = B^{-1} \vec{b} \qquad z' = \sum_{i=1}^n \lambda_i b_i \Rightarrow z' \in [z^* + \delta_i \lambda_i]$$

où B^{-1} est la matrice correspondant aux variables d'écart.

Exemple : Soit le PL standard et dictionnaire optimal :

$$\begin{array}{ll} \max & z = 400x_1 + 1200x_2 \\ \text{s.c.} & 10x_1 + 20x_2 + x_3 = 1100 \\ & x_1 + 4x_2 + x_4 = 160 \\ & x_1 + x_2 + x_5 = 100 \\ & x_i \geq 0, \forall i \end{array}$$

	$-x_3$	$-x_4$
$z =$	54000	20 200
$x_1 =$	60	1/5 -1
$x_2 =$	25	-1/20 1/2
$x_5 =$	60	-3/20 1/2

Ainsi la base B reste optimale tant que la première composante de b reste dans l'intervalle $[1100 - 300; 1100 + 100] = [800; 1200]$.

1.6.2 Sensibilité de la fonction objectif

Dans quelle mesure peut-on modifier c_i le $i^{\text{ème}}$ coefficient de la fonction objectif sans que la base optimale ne change ?

$$-\gamma_N = c_B B^{-1} N - c_N \geq 0 \quad \Leftrightarrow \quad \delta y_{ri} \geq -(z_i - c_i) \quad \text{et} \quad z' = z^* + \delta y_{r0}$$

où H est la matrice des variables hors base.

Exemple (suite) : la variable x_1 est la première variable dans la base. Si on modifie son coefficient dans la fonction objectif (actuellement 400) d'une quantité δ , la base B reste optimale tant que

$$\begin{cases} \delta y_{13} \geq -(z_3 - c_3) \\ \delta y_{15} \geq -(z_4 - c_4) \end{cases} \Leftrightarrow \begin{cases} \delta(1/5) \geq -20 \\ \delta(-1) \geq -200 \end{cases} \Leftrightarrow \delta \in [-100; 200]$$

Ainsi B reste optimale tant que le coefficient de x_1 dans la fonction objectif reste dans l'intervalle $[400 - 100; 400 + 200] = [300; 600]$.
Pour tout point dans cet intervalle, la solution optimale ne change pas, mais la valeur de cette solution optimale est $z' = z^* + \delta y_{r0}$.
Comme ici $r = 1$ (x_1 est la 1ère variable de base), on a $z' = z^* + \delta y_{10} = 54000 + 60\delta$.

²On peut retrouver B^{-1} visuellement dans le dictionnaire en prenant les colonnes des variables d'écarts (ici x_3, x_4 et x_5) :
– Si la variable est hors-base (ici x_3 et x_4) on recopie la colonne correspondante du dictionnaire ;
– Si elle est dans la base (ici x_5) on lui associe la colonne de la matrice identité correspondante à sa place dans la base (ici $(0 \ 0 \ 1)^T$ car x_5 est la 3^{ème} colonne).

1.6.3 Ajout de contrainte

Pour ajouter une contrainte dans un tableau optimal, il faut l'exprimer en fonction des variables dans la base optimale. Il faut de plus faire entrer la nouvelle variable d'écart dans la base. Enfin, il faut ré-appliquer l'algorithme du simplexe si nécessaire pour obtenir le nouveau tableau optimal.

Exemple (suite et fin) : si on veut rajouter la contrainte

$$3x_1 + 8x_2 \leq 360$$

On introduit une nouvelle variable d'écart x_6 , pour obtenir

$$3x_1 + 8x_2 + x_6 = 360$$

Cette équation correspond à la nouvelle ligne dans le dictionnaire optimal. Maintenant il faut exprimer x_6 en fonction des variables hors base. Du dictionnaire optimal on tire :

$$x_1 = 60 - \frac{1}{5}x_3 + x_4 \quad \text{et} \quad x_2 = 25 + \frac{1}{20}x_3 - \frac{1}{2}x_4$$

2 Théories des graphes

- Graphe $G = (V, E, \phi)$: (sommets, arêtes/arcs, fonction d'incidence).
- Graphe orienté (réseau) => arcs ; Graphe non orienté => arêtes.
- Arêtes/Aracs incidents à v : toutes les arêtes/arcs qui sont connectés à v .
- Graphe simple = sans boucles (sur lui-même \odot) ni arêtes/arcs multiples.
- Multigraphe = avec boucles et/ou arêtes/arcs multiples.
- Degré³ : $\deg(v)$ = nombre d'arêtes/arcs incidents à v .
 - degré extérieur : $\deg_+(v) (\leftarrow \bullet \rightarrow)$
 - degré intérieur : $\deg_-(v) (\rightarrow \bullet \leftarrow)$

2.1 Matrice d'adjacence et d'incidence d'un graphe ayant n sommets et m arêtes

2.1.1 Graphe simple non orienté

Matrice d'adjacence sommets-sommets $B \ (n \times n)$:

$$b_{ij} = \begin{cases} 1 & \text{si } v_i \text{ et } v_j \text{ sont adjacents} \\ 0 & \text{sinon} \end{cases}$$

Matrice d'incidence sommets-arêtes $A \ (n \times m)$:

$$a_{ij} = \begin{cases} 1 & \text{si } v_i \text{ est incident à } e_j \\ 0 & \text{sinon} \end{cases}$$

2.2 Problème de l'arbre recouvrant de poids minimum

2.2.1 Algorithme de Kruskal

Donnée : Un graphe connexe avec pondération ($\in \mathbb{R}$) des arêtes
Résultat : Un arbre recouvrant de poids total minimal (ou max).

1. Numéroter les arêtes par ordre croissant (resp. décroissant) en fonction de leurs poids.
2. Partir d'un arbre solution vide.
3. Si e_k ne forme pas de cycle, l'ajouter à l'arbre solution.

³Si un sommet possède une ou plusieurs boucles, chacune apporte une contribution de 2 dans le calcul du degré de ce sommet.

Ainsi

$$x_6 = 360 - 3x_1 - 8x_2 = -20 + \frac{1}{5}x_3 + x_4$$

On rajoute la contrainte au dictionnaire :

	$-x_3$	$-x_4$
$z =$	54000	20 200
$x_1 =$	60	1/5 -1
$x_2 =$	25	-1/20 1/2
$x_5 =$	60	-3/20 1/2
$x_6 =$	-20	-1/5 -1

Le nouveau tableau n'est plus optimal mais reste dual-admissible !

Déterminer le nouvel optimum avec l'algo dual du simplexe.

- Chaîne : suite alternée sommets/arêtes.
- Élémentaire : chaque sommet apparaît au plus une fois
- Simple : chaque arête apparaît au plus une fois
- Chemin : suite alternée sommets/arcs
- Cycle : chaîne dont les extrémités sont confondues
- Arbre : multigraphe non orienté sans cycle et connexe
- Forêt : ensemble d'arbres non reliés entre eux
- Connexité : on peut atteindre n'importe quel sommet
- Connexité forte : connexité + les deux sens de circulation existent.
- Nombre cyclomatique = $m - n + p$ (m arêtes, n sommets, p composantes connexes)

2.1.2 Graphe simple orienté

Matrice d'adjacence sommets-sommets $B \ (n \times n)$:

$$b_{ij} = \begin{cases} 1 & \text{si } (v_i, v_j) \in E \\ 0 & \text{sinon} \end{cases}$$

Matrice d'incidence sommets-arêtes $A \ (n \times m)$:

$$a_{ij} = \begin{cases} -1 & \text{si } v_i \text{ est l'extrémité initiale de } e_j \\ 1 & \text{si } v_i \text{ est l'extrémité finale de } e_j \\ 0 & \text{sinon} \end{cases}$$

Exemple :

k	e_k	$c(e_k)$	$\in T$
1	$\{v_1, v_3\}$	1	✓
2	$\{v_2, v_3\}$	1	✓
3	$\{v_1, v_2\}$	2	✗
...

Pour obtenir l'arbre recouvrant, trier dans l'ordre décroissant.

2.2.2 Algorithme de Prim

Donnée : Un graphe connexe avec pondération ($\in \mathbb{R}$) des arêtes.

Résultat : Un arbre recouvrant de poids total minimal.

1. Partir d'un arbre solution vide.
2. Choisir arbitrairement un sommet.
3. Choisir l'arête la plus économique pour rejoindre un sommet pas encore visité depuis un sommet déjà présent dans l'arbre solution. Ajouter cette arête à l'arbre solution.

2.3 Problème de plus courts chemins

Remarque : la longueur d'un chemin \neq nombre d'arcs, mais = somme des poids des arcs !!!

Principe d'optimalité de Bellman. Un plus court chemin est formé de plus courts chemins.

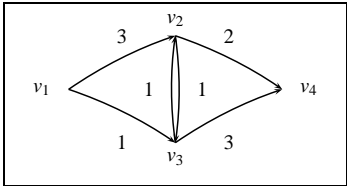
2.3.1 Algorithme générique

Algorithme générique du plus court chemin d'un sommet-source (v_s) à tous les autres :

1. On part d'un vecteur λ initial : $\lambda_s = 0$ et $\lambda_i = \infty \forall i \neq s$
2. On tient à jour une liste de candidats L , initialement $L = \{v_s\}$
3. Tant que $L \neq \emptyset$ on retire le sommet i de L et on teste ses successeurs j :
si $\lambda_j > \lambda_i + c_{ij}$, on pose $\lambda_j = \lambda_i + c_{ij}$ et on introduit j dans L

Exemple (sens de travail : \leftarrow) :

Itération	Candidats L	Étiquettes λ	Sommet retiré de L
0	$\{v_1\}$	$(0, \infty, \infty, \infty)$	-
1	$\{v_2, v_3\}$	$(0, 3, 1, \infty)$	v_1
2	$\{v_3, v_4\}$	$(0, 3, 1, 5)$	v_2
\vdots	\vdots	\vdots	\vdots
5	\emptyset	$(0, 2, 1, 4)$	v_2



2.3.2 Algorithme de Dijkstra

Donnée : Réseau connexe avec pondération non-négative des arcs.

Résultat : Plus court chemin de v_s à v_i ainsi que le prédécesseur p_i du sommet i .

1. $\lambda_s = 0$, $\lambda_i = \infty \forall i \neq s$, $p_i = \text{NULL} \forall i$, $T = V$
2. Tant que $T \neq \emptyset$ faire :
 - (a) Soit i le sommet de T de plus petite étiquette λ_i .
 - (b) Si un tel sommet n'existe pas ($\lambda_i = \infty \forall i \in T$) : STOP (les sommets restants ne sont pas atteignables depuis s).
 - (c) Sinon retirer i de T et pour tout successeurs j de i encore dans T :
si $\lambda_j > \lambda_i + c_{ij}$, on pose $\lambda_j = \lambda_i + c_{ij}$ et $p(j) = i$

Exemple (même graphe que précédent) :

Itér.	i min	Étiquette λ_i / prédécesseur $p(i)$				T
0	-	0/null	∞ /null	∞ /null	∞ /null	$\{v_1, v_2, v_3, v_4\}$
1	v_1	0/null	3/ v_1	1/ v_1	∞ /null	$\{v_2, v_3, v_4\}$
2	v_3		2/ v_3	1/ v_1	4/ v_3	$\{v_2, v_4\}$
3	v_2		2/ v_3		4/ v_3	$\{v_4\}$
4	v_4				4/ v_3	\emptyset

2.3.3 Graphes acycliques (sans-circuit)

Un graphe acyclique (sans-circuit) possède au moins un sommet sans prédécesseur, et au moins un sommet sans successeur. Un graphe est acyclique si et seulement si on peut attribuer à chaque sommet i un nombre $r(i)$, appelé rang, tel que pour tout arc reliant i à j on ait $r(i)$ plus petit que $r(j)$.

2.3.4 Algorithme du rang

Donnée : Un graphe orienté acyclique.

Résultat : \forall sommet i un rang $r(i)$ minimal.

1. $k = 1$.
2. On prend les sommets sans prédécesseurs : rang = k .
3. On élimine tous les sommets sans prédécesseurs.
4. $k = k + 1$; retourner à 2.

2.3.6 Plus courts chemins dans les réseaux sans circuit

1. Tri topologique du graphe : sommet $1 \dots n$
2. $\lambda_i = 0$ et pour $k = 1 \dots n$ on calcule $\lambda_k = \min\{\lambda_j + c_{jk} \mid j \in \text{Pred}(k)\}$

2.3.7 Application à la gestion de projet (méthode du chemin critique)

On considère un projet complexe (tâches et ordre défini), on cherche une planification optimale minimisant la durée totale.

Le problème doit être modélisé par un réseau : les sommets représentent les tâches, les arcs les contraintes de précedence entre deux tâches. Le poids de l'arc sortant équivaut à la durée de la tâche. On ajoute des sommets α (de durée nulle et sans prédécesseur) représentant le début des travaux, et ω (de durée nulle et sans successeur) représentant la fin des travaux.

On cherche maintenant le plus long chemin entre α et ω , on utilise pour ce faire l'algorithme du chemin critique.

Algorithme du chemin critique

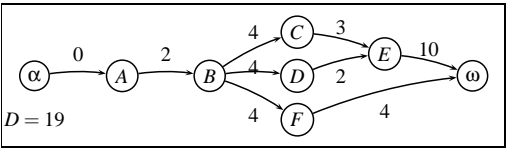
Donnée : Réseau associé au projet trié topologiquement

Résultat : Durée minimale D du projet et pour chaque tâche i sa date δ_i de début au plus tôt et sa date ϕ_i de début au plus tard.

1. Calcul des dates de début au plus tôt (phase avançant)
 - (a) Poser $\delta_1 = 0$
 - (b) Pour $j = 2 \dots n$ poser $\delta_j = \max_{i \in \text{Pred}(j)} (\delta_i + c_{ij}) = \max_{i \in \text{Pred}(j)} (\delta_i + d_i)$
 - (c) Poser $D = \delta_n$
2. Calcul des dates de début au plus tard (phase arrière)
 - (a) Poser $\phi_n = D$
 - (b) Pour $j = n - 1 \dots 1$ poser $\phi_j = \min_{k \in \text{Succ}(j)} (\phi_k - c_{jk}) = \min_{k \in \text{Succ}(j)} (\phi_k - d_j) = \min_{k \in \text{Succ}(j)} (\phi_k) - d_j$

Exemple :

Tâche	Description	Durée	Préd.
A	Choix stations	2	-
B	Accord administratif	4	A
C	Commande décodeurs	3	B
D	Installation antennes	2	B
E	Installation décodeurs	10	C,D
F	Modification facturation	4	B



Tâche	α	A	B	C	D	E	F	ω
N° k	1	2	3	4	5	6	7	8
Durée d_k	0	2	4	3	2	10	4	0
Préd.(k)	-	α	A	B	B	C,D	B	E,F
$\delta_k \rightarrow$	0	0	2	6	6	9	6	19 (=D)
Succ.(k)	A	B	C,D,F	E	E	ω	ω	-
$\phi_k \leftarrow$	0	0	2	6	7	9	15	19

- Une tâche est critique si $\phi_i = \delta_i$ (un retard dans la tâche implique un retard du projet).
- Le chemin critique ($\alpha \rightarrow \omega$) est le chemin composé de tâches critiques.
- La longueur du chemin critique est la durée minimale du projet.

2.4 Problème du transbordement

2.4.1 Définitions

- Soit un graphe orienté connexe $G = (V, E)$. Soit b une pondération des sommets représentant les offres (sommet source, $-$) et les demandes (sommet puits, $+$). Soit c une pondération des arcs représentant les coûts unitaires d'utilisation des arcs. La solution basique linéaire du problèmes est l'arbre maximal du réseau.
- Un graphe $G = (V, E)$ est biparti s'il existe une partition de ses sommets en deux ensembles V_1 et V_2 telle que toute arête (i, j) de E ait une extrémité dans V_1 et l'autre dans V_2 .
- Lorsque G est un graphe biparti complet, chaque sommet source étant relié à tous les sommets puits, on parle de problème de transport.
- Si, dans un problème de transport, il y a autant de sommets source que de sommets puits et que toutes les offres et les demandes sont égales à 1, on parle de problème d'affectation.

2.4.2 L'algorithme graphique du Simplexe dans les réseaux (phase II primale)

Donnée : Un réseau connexe $R = (V, E, b, c)$ avec une solution-arbre admissible $T = (V, E_T)$.

Résultat : Un flot x de coût total minimum ou la preuve qu'un tel flot n'existe pas.

1. Calcul des solutions primale x et duale y associées à T .
2. Recherche d'un arc entrant, s'il n'en existe pas => STOP : solution optimale.
3. Recherche d'un arc sortant, s'il n'en existe pas => STOP : pas d'optimum fini.
4. Mise à jour de la solution-arbre et retour au point 1.

2.4.3 Calcul d'une solution-arbre admissible (phase I)

Lorsque qu'une solution-arbre initiale admissible n'est pas connue ou facile à déterminer, il faut recouvrir à une phase I. On va définir un problème auxiliaire

- possédant toujours des solutions admissibles.
- possédant toujours un optimum fini.
- possédant un optimum fini si et seulement si le problème de départ possède au moins une solution admissible.

2.4.4 Construction du problème auxiliaire et d'une solution initiale admissible

Donnée : Un réseau $R = (V, E, b, c)$ connexe.

Résultat : Un réseau $R' = (V, E', b, c)$ et une solution-arbre $T' = (V, E'_T)$ admissible pour R'

1. Poser $c'_{ij} = 0$ pour tout $(i, j) \in E$.
2. Choisir un sommet source, disons k .
3. Relier chaque sommet source i ($i \neq k$) à k par un arc artificiel (i, k) de poids $c'_{ik} = 1$ (si il existe déjà un arc de i à k dans R ne pas le rajouter).
4. Relier k à chaque sommet puits j par un arc artificiel (k, j) de poids $c'_{kj} = 1$ (si il existe déjà un arc de k à j dans R ne pas le rajouter).
5. Poser $E'_T = \{(i, k) | i \text{ sommet source}\} \cup \{(k, j) | j \text{ sommet puits}\}$ et compléter E'_T jusqu'à obtenir un arbre maximal (si nécessaire).

2.4.5 Calcul de la solution primale associée à $T = (V, E_T)$

Donnée : Un réseau connexe $R = (V, E, b, c)$ (b = pond.des sommets, c = pond.des arcs) et une solution-arbre $T = (V, E_T)$.

Résultat : Le flot $x : E \rightarrow \mathbb{R}_+$ associé à T (et $z = \sum c_{ij}x_{ij}$).

1. Tant que $|E_T| > 1$ faire
 - (a) Trouver un sommet pendant j de T . Soit e le seul arc incident avec j dans T et i l'autre extrémité de e .
 - (b) Si $e = (i, j)$ poser $x_{ij} = b_j$, sinon poser $x_{ji} = -b_j$.
 - (c) Poser $b_i = b_i + b_j$.
 - (d) Retirer e de $E_T : E_T = E_T \setminus \{e\}$.
2. Il reste un seul arc dans E_T , disons (i, j) , poser $x_{ij} = b_j$.

2.4.6 Calcul de la solution duale associée à $T = (V, E_T)$

Donnée : Un réseau connexe $R = (V, E, b, c)$ et une solution-arbre $T = (V, E_T)$.

Résultat : Les prix duaux $y : V \rightarrow \mathbb{R}$ associés à T et $w = \sum b_i y_i = z$.

1. Choisir arbitrairement $i \in V$ et poser $y_i = 0$ et $W = V \setminus \{i\}$.
2. Tant que $W \neq \emptyset$ faire
 - (a) Trouver un arc $e \in E_T$ avec une extrémité j dans W et une extrémité i dans $\bar{W} = V \setminus W$.
 - (b) Si $e = (i, j)$ poser $y_j = y_i + c_{ij}$, sinon poser $y_j = y_i - c_{ij}$.
 - (c) Retirer j de $W : W = W \setminus \{j\}$.

2.4.7 Recherche d'un arc entrant

Donnée : Une solution-arbre admissible $T = (V, E_T)$ ainsi que les solutions primale (x) et duale (y) associées.

Résultat : Un arc entrant dans la base.

1. On passe en revue les arcs $(i, j) \notin E_T$ (hors base) dans l'ordre lexicographique et on teste pour chacun d'eux si la contrainte duale associée $y_j - y_i \leq c_{ij}$ est satisfaite ou non.
2. Si une contrainte violée est trouvée STOP : l'arc $e = (i, j)$ va entrer dans la base.
3. Si toutes les contraintes sont satisfaites, la solution actuelle est optimale.

2.4.8 Recherche d'un arc sortant

Si on ajoute à la solution-arbre actuelle l'arc entrant $e = (i, j)$, on forme un cycle unique C . Utilisant l'orientation de (i, j) pour définir le sens de parcours de C , on divise les arcs de C en 2 ensembles disjoints C^+ (même orientation de (i, j)) et C^- .

1. Si $C^- = \emptyset$ STOP : il n'y a pas d'optimum fini.
2. Sinon soit $\Delta = \min\{x_{kl} | (k, l) \in C^-\}$ et s un arc pour lequel le minimum est atteint. L'arc s quitte la base.

2.4.9 Mise à jour des solutions

1. La nouvelle solution-arbre est donnée par $E_T = E_T \cup \{e\} \setminus \{s\}$.
2. La solution basique primale ne change que sur les arcs de C :

$$x_{ij} = \begin{cases} x_{ij} = x_{ij} + \Delta & \text{si } (i, j) \in C^+ \\ x_{ij} = x_{ij} - \Delta & \text{si } (i, j) \in C^- \\ x_{ij} = x_{ij} & \text{si } (i, j) \notin C \end{cases}$$

3. La solution basique duale ne change que de la manière suivante.

$$y_d = y_d + \epsilon$$

où d est le sommet commun entre les arcs e et s , et

$$\epsilon = \begin{cases} -y_d + y_a + c_{ad} & \text{si } e = (a, d) \text{ et } s = (b, d) \\ y_a - y_d - c_{da} & \text{si } e = (d, a) \text{ et } s = (d, b) \end{cases}$$

2.4.10 Dégénérescence

Si la solution basique primale est dégénéré, c-à-d s'il existe des arcs dans E_T le long desquels des quantités nulles sont transportées, il y a risque de cyclage. Pour éviter cela, recourons à la règle de Bland graphique :

- Tester les arcs dans l'ordre lexicographique et faire entrer le premier arc dont la contrainte duale associée est violée.
- Si la quantité Δ est transportée le long de plusieurs arcs de C^- , faire sortir l'arc le plus petit dans l'ordre lexicographique.

2.5 Remarques

- Le coût de l'arc (i, j) peut varier dans l'intervalle $c_{ij} \geq y_j - y_i$ sans que la base optimale change.
- Si on modifie le coût de l'arc (i, j) , pour avoir la nouvelle solution il faut calculer les nouvelles valeurs duales y_k et vérifier que les contraintes duales ne soit pas violées.