RAIRO. RECHERCHE OPÉRATIONNELLE

CATHERINE ROUCAIROL
PIERRE HANSEN

Problème de la bipartition minimale d'un graphe

RAIRO. Recherche opérationnelle, tome 21, n° 4 (1987), p. 325-348

<http://www.numdam.org/item?id=RO_1987__21_4_325_0>

R.A.I.R.O. Recherche opérationnelle/Operations Research
(vol. 21, n° 4, novembre 1987, p. 325 à 348)

PROBLÈME DE LA BIPARTITION MINIMALE D'UN GRAPHE (*)

par Catherine ROUCAIROL (1) et Pierre HANSEN (2)

Résumé. — *Le problème posé consiste à chercher une partition d'un graphe valué en deux parties de taille fixée de façon à minimiser la somme des coûts des arcs « à cheval » (ayant une extrémité dans chacune des parties).*

Après avoir formulé ce problème comme un programme quadratique en variables 0,1 *sous contrainte, nous montrons comment calculer une borne inférieure par relaxation lagrangienne; le multiplicateur de Lagrange optimal est trouvé en, au plus,* $(n-1)$ *applications d'un algorithme de flot maximal dans un réseau à* $n+2$ *sommets et* $n+m$ *arcs,* n *nombre de sommets et* m *d'arcs du graphe initial. Cette borne est utilisée dans un algorithme « Branch and Bound ». Les résultats obtenus montrent que cet algorithme fournit une solution optimale en des temps raisonnables, très inférieurs à ceux des méthodes exactes connues.*

Mots clés : Problème NP complet; théorie des graphes; programmation quadratique en variables 0,1; coupe minimale; relaxation lagrangienne; procédure par séparation et évaluation.

Abstract. — *The problem considered is that of partitioning a arc-weighted graph into two parts, each of which constrained in size, in order to minimize the sum of the costs of the cut arcs.*

After formulating this problem as a 0.1 *quadratic program, we calculated a lower bound by using Lagrangian relaxation. The best Lagrangian multiplier is determined in at most* $(n-1)$ *applications of a maximum flow algorithm to a network with* $n+2$ *vertices and* $n+m$ *arcs, where* n *and* m *denote the number of vertices and arcs in the initial graph. A Branch and Bound algorithm using this result is presented and computational experience shows that an optimal solution is obtained within quite reasonable times which are much less than for the exact methods known up to now.*

Keywords : NP complete problem; graph theory; quadratic 0.1 programming; minimal cut; Lagrangian relaxation; Branch and Bound.

(*) Reçu janvier 1987.
(1) Université Paris-VI et I.N.R.I.A., Domaine de Voluceau, B.P. n° 105, 78153 Le Chesnay Cedex.
(2) RUTCOR, Rutgers University, New Brunswick, New Jersey, U.S.A.

R.A.I.R.O. Recherche opérationnelle/Operations Research, 0399-0559/87/04 325 24/$ 4.40

I. INTRODUCTION

Soit un graphe $G=(X, U)$ non orienté, dont les arêtes ont des valuations entières non négatives [c_{ij} coût de l'arête (i, j), $c_{ij} \in \mathbb{N}$]. Une partition de X en deux sous ensembles disjoints X_0, X_1 permet de définir une *partition de G* en deux sous-graphes G_0 et G_1 avec

$$G_0=(X_0, U_0) \qquad \text{et} \qquad G_1=(X_1, U_1)$$
$$X_0 \cap X_1=\varnothing, \qquad X_0 \cup X_1=X.$$

Une partition est dite *contrainte* si la taille des sous-graphes créés est bornée : $|X_1| \leqq p$, $p \leqq n$ où $|X|=n$. L'ensemble X_1 contient p sommets.

Le *coût de la partition* (G_0, G_1) est mesurée par la somme des coûts des arêtes ayant une extrémité dans X_0 et l'autre dans X_1.

$$C(G_0/G_1)=\sum_{\substack{i \in X_0 \\ j \in X_1}} c_{ij}$$

Le problème dit de la *partition minimale d'un graphe* consiste alors à chercher la partition d'un graphe respectant la contrainte de taille et de coût minimal.

Ce problème NP-difficile (Minimal cut into bounded sets dans Garey *et al.* [4]) se pose dans de nombreuses applications pratiques : segmentation de programme, classification (*cf.* exemples cités dans [3]), en particulier, CAO des circuits électroniques [13] : les points à répartir sont alors des composants qui doivent être placés sur l'une ou l'autre des deux faces d'un circuit, de manière telle que le nombre des équipotentielles communes aux deux faces soit minimal et que les encombrements des faces soient comparables.

De nombreuses méthodes de recherche de solutions approchées ont été proposées pour ce problème : partant d'une configuration initiale où les sommets sont répartis aléatoirement entre les deux sous ensembles X_0 et X_1, on procède dans la plupart d'entre eux à des échanges élémentaires (méthode k-optimale de Lin et Kerninghan, méthode de recuit simulé [13]). D'autres méthodes comme celle de E. Barnes [1] sont basées sur l'équivalence du problème de partitionnement d'un graphe et celui du problème de l'approximation de la matrice C; en effet, si $P=(p_{ij})$ est la matrice P où p_{ij} vaut 1 si les sommets i et j sont dans le même sous-ensemble, 0 sinon et si $\|\,.\,\|$ est la norme de Frobenius :

$$\|C-P\|^2-\|C\|^2-2\sum_{i,j} c_{ij}p_{ij}+\|P\|^2$$
$$=\|C\|^2-2\Big(\sum_{i,j} c_{ij}-\text{coût des arêtes à cheval}\Big)+(p^2+(n-p)^2)$$

Plus P est proche de C, plus le coût des arêtes « à cheval » est faible. Barnes cherche alors une solution approchée de ce problème en résolvant un programme linéaire.

Très peu de méthodes exactes (Christofidès et Brooker [3]) ont à notre connaissance été proposées, c'est pourquoi, il nous a paru intéressant d'en concevoir une et de tester ses possibilités.

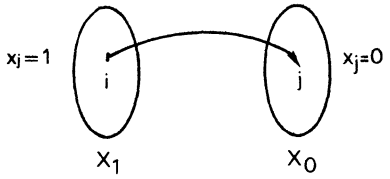
II. PROGRAMME QUADRATIQUE EN VARIABLES 0,1 ÉQUIVALENT

Soit une partition $\mathscr{P}=(G_0, G_1)$ d'un graphe $G=(X,U)$ répondant à une contrainte de taille p.

$$|X_1|=p, \qquad X=X_0 \cup X_1, \qquad |X|=n \qquad (p<n)$$

Associons à chaque sommet de G une variable $x_j \in \{0,1\}$:

$$x_j \begin{cases} =1 \text{ si le sommet } j \text{ de } G \text{ appartient à } X_1, \\ =0 \text{ sinon} \end{cases}$$



Alors le problème de la partition minimale sous contrainte peut s'écrire :

$$x_j \in \{0,1\}, j=1, \ldots, n \tag{1}$$

$$P \qquad \sum_{j=1}^{n} x_j = p \tag{2}$$

$$\text{Min } z = \sum_{i=1}^{n} \sum_{j>i}^{n} c_{ij}[x_i(1-x_j)+x_j(1-x_i)]$$

z s'écrit aussi en développant et en notant : $C_i = \sum_{j<i} c_{ji} + \sum_{j>i} c_{ij}$

$$z = \sum_{i=1}^{n} C_i x_i - 2 \sum_{i=1}^{n} \sum_{j>i} c_{ij} x_i x_j$$

z est donc une fonction quadratique en variables 0,1.

Les coefficients des termes quadratiques étant négatifs, le problème sans la contrainte (2) est équivalent au problème de la coupe minimale (ou flot maximal) dans un graphe particulier [5, 10, 11] : problème polynomial.

C'est pourquoi le problème de la partition minimale d'un graphe est aussi appelé problème de la coupe minimale sous contrainte.

Tirant parti de cette remarque, nous allons définir un problème relaxé où la contrainte (2) sera introduite dans la fonction économique avec une pénalité λ (méthode de relaxation lagrangienne).

III. CALCUL D'UNE BORNE INFÉRIEURE DE Z PAR RELAXATION LAGRANGIENNE

Soit λ le multiplicateur associé à (2) et la fonction lagrangienne :

$$L(x, \lambda) = \sum_{i=1}^{n} \sum_{j>i} c_{ij}[x_i(1-x_j)+x_j(1-x_i)] + \lambda\left(\sum_{j=1}^{n} x_j - p\right)$$

Soit le problème relaxé : $\omega(\lambda) = \min_{x \in (1)} L(x, \lambda)$.

1. $\forall x \in (1)$, $\omega(\lambda) \leqq z^*$

On retrouve les propriétés connues de la relaxation lagrangienne :

$$\omega(\lambda) \leqq L(x^*, \lambda) = z^* + \lambda\left(\sum_{j=1}^{n} x_j^* - p\right) = z^*$$

Mais, puisque $\omega(\lambda) = \min_{x \in (1)} L(x, \lambda)$

$$\omega(\lambda) \leqq \min [L(O, \lambda), L(I, \lambda)]$$

où O et I sont les vecteurs $(0, \ldots, 0)$ et $(1, \ldots, 1)$.

D'où :

$$\omega(\lambda) \leqq \min [-\lambda p, \lambda(n-p)] = -\lambda p + \min [o, \lambda n]$$

si

$$\lambda > 0, \qquad \omega(\lambda) \leqq -\lambda p < 0$$

$$\lambda < 0, \qquad \omega(\lambda) \leqq \lambda(n-p) < 0 \quad \text{car } p < n$$

Cette borne n'est donc pas intéressante car toujours négative.

Par contre, le problème P peut se décomposer en deux sous-problèmes si on fixe la valeur d'une variable quelconque x_s

$$P_s \begin{cases} P \\ \text{avec } x_s = 1 \end{cases} \quad \text{et} \quad \overline{P_s} \begin{cases} P \\ \text{avec } x_s = 0 \end{cases}$$

$$z^* = \min(z_s^*, \bar{z}_s^*)$$

où z_s^* est la valeur de la solution optimale de P_s et $\bar{z}_s^*$ celle de $\bar{P}_s$.

Explicitons P_s; il revient à chercher :

$$x_j \in \{0,1\}\, j \neq s,\ j = 1, \ldots, n \tag{1}'$$

$$\sum_{\substack{j=1 \\ j \neq s}}^{n} x_j = p - 1 \tag{2}'$$

P_s

$$\text{Min } z_s = C_s + \sum_{\substack{i=1 \\ i \neq s}}^{n} (C_i - 2c_{is})\, x_i - 2 \sum_{\substack{i=1 \\ i \neq s}}^{n} \sum_{\substack{j>i \\ j \neq s}} c_{ij} x_i x_j$$

Le programme $\overline{P_s}$, si on pose $\bar{x}_j = 1 - x_j$ et $\bar{p} = n - p$ se ramène à un problème en variables 0,1 semblable à P_s avec des variables $\bar{x}_j$, et où le second membre de (2)′ est $n - p - 1$.

Cherchons alors une borne inférieure de la valeur de z_s^* en utilisant à nouveau la relaxation lagrangienne [contrainte (2)′ avec multiplicateur λ]. Cette fois-ci, la fonction lagrangienne vaut :

$$L_s(x, \lambda) = C_s - (p-1)\lambda + \sum_{i \neq s} [C_i - 2c_{is} + \lambda]\, x_i - \sum_{\substack{i \\ i \neq s}} \sum_{\substack{j>i \\ j \neq s}} 2 c_{ij} x_i x_j$$

$$\omega_s(\lambda) = \min_{x \in (1)'} L_s(x, \lambda)$$

et de même :

$$\omega_s(\lambda) \leqq \min [L_s(O', \lambda),\ L_s(I, \lambda)]$$

avec $O' = (0, \ldots 0, 1, 0, \ldots 0)$: la s^e coordonnée vaut 1.

D'où :

$$\omega_s(\lambda) \leqq \min [C_s - \lambda(p-1), \lambda(n-p)]$$

Le maximum du second terme de cette inéquation est atteint pour

$$\lambda_0 = \frac{C_s}{n-1}.$$

D'où :

$$\forall \lambda, \quad \omega_s(\lambda) \leqq -\frac{C_s}{n-1}(p-1) + C_s = \frac{n-p}{n-1} C_s = (n-p)\lambda_0.$$

La borne calculée par relaxation lagrangienne peut être positive. Elle sera au plus égale à $(n-p)$ fois la valeur moyenne d'une arête reliant s aux autres sommets.

La meilleure borne inférieure sera obtenue par résolution du problème dual :

$$\max_{\lambda} \omega_s(\lambda) = \omega_s(\lambda^*).$$

2. *Propriétés de* $\omega_s(\lambda)$

Pour plus de facilité nous démontrerons ces propriétés sur $\omega(\lambda)$.

PROPRIÉTÉS 1 ET 2 : $\omega(\lambda)$ est

(P1) une fonction concave,

(P2) linéaire par morceaux en λ.

Pour x^0 fixé vecteur booléen, $L(x^0, \lambda)$ est une fonction linéaire en λ. $\omega(\lambda)$ est l'enveloppe inférieure de ces fonctions linéaires. $\omega(\lambda)$ est concave (*cf.* propriétés relaxation lagrangienne dans M. Minoux [7]).

PROPRIÉTÉ 3 : $\omega(\lambda)$ possède la propriété d'inclusion.

Soient : $\lambda_1 < \lambda_2$ et

$$\omega(\lambda_1) = \operatorname*{Min}_{x \in (1)} L(x, \lambda_1) = L(x^1, \lambda_1)$$

$$\omega(\lambda_2) = \operatorname*{Min}_{x \in (1)} L(x, \lambda_2) = L(x^2, \lambda_2)$$

alors $x^1 \geqq x^2$ (i. e. $\forall j = 1, \ldots, n, \; x_j^1 \geqq x_j^2$).

La démonstration de cette propriété est identique à celle faite par Picard et Queyranne dans [10] pour des fonctions pseudo-booléennes hyperboliques

et reprise par Chaillou, Hansen, Mahieu [2] lors du calcul d'une borne inférieure d'un problème de sac à dos quadratique par relaxation lagrangienne.

PROPRIÉTÉ 4 : Pour $\lambda<0$, $\omega_s(\lambda)=L(I, \lambda)=\lambda(n-p)$.

En effet pour $\lambda=0$, $L_s(x, 0)=z_s(x)$. D'où :

$$\omega_s(0)=\min_x L_s(x, 0)=L_s(I, 0)=0=\min_x z_s(x).$$

Pour $\lambda<0$, $\omega_s(\lambda)=\underset{x}{\text{Min}}\, L(x, \lambda)=L(x^k, \lambda)$.

D'après la propriété d'inclusion, $x^k \geqq I$. Comme $x^k \in \{0, 1\}^n$, $x^k=1$ et pour $\lambda<0$, $\omega_s(\lambda)=L_s(I, \lambda)=\lambda(n-p)$. □

COROLLAIRE 4 : $\omega(\lambda)$ *est composée au plus de* $n-1$ *tronçons de droite.*

Ceci se déduit directement de la propriété d'inclusion.

3. *Calcul de* $\omega_s(\lambda)$

Pour λ donné, $L_s(\lambda)$ est une fonction pseudo-booléenne quadratique dont les coefficients des termes quadratiques sont négatifs. En effet, $L_s(\lambda)$ peut s'écrire :

$$L_s(\lambda)=l+\sum_{j\in J_s} q_j x_j-\sum_{J_s}\sum_{J_s} q_{ij} x_i x_j$$

avec $l=C_s-(p-1)\lambda$ et en notant $J_s=\{1, 2, \ldots, s-1, s+1, \ldots, n\}$

$$q_j=C_j-2c_{js}+\lambda$$

$$q_{ij}=2c_{ij}$$

D'après le théorème de Picard et Ratliff [11] :

« le problème qui consiste à minimiser $l+\sum_{j\in J} q_j x_j-\sum_J\sum_J q_{ij} x_i x_j$ avec $q_{ij}\geqq 0$, $x_j\in\{0, 1\}$, $J=\{1, 2, \ldots, n\}$ est équivalent au problème de la coupe minimale dans un réseau (source S et puits P) où les capacités sont :

$$c_{ij}=q_{ij}$$

$$c_{Sj}=\max\left(\sum_J c_{ji}-q_j, 0\right),$$

$$c_{jP}=\max\left(q_j-\sum_j c_{ji}, 0\right);$$

le minimum de la fonction pseudo-booléenne est alors égal à :

$$l-\sum_{J} c_{\mathbf{S}_j}+\Psi$$

où Ψ est le flot maximal dans le réseau précédemment défini ».

$\omega_s(\lambda)$ qui est le minimum de $L_s(\lambda)$ est donc égal au flot maximal dans le réseau de transport $N_\lambda=(V, E, \gamma_\lambda)$ où V est un ensemble de sommets correspondant à J_s auquel on adjoint une source $\boldsymbol{S}$ et un puits $\boldsymbol{P}$: $V=\{\boldsymbol{S}, 1, 2, \ldots, s-1, s+1, \ldots, n, P\}$.

E un ensemble d'arcs définis ainsi :

$$E=\{(\mathbf{S}, j)\,|\,j\in J_s\}\cup\{(i, j)\,|\,j>i,\ i\in J_s,\ j\in J_s\}\cup\{c_j, \mathbf{P})\,|\,j\in J_s\}$$

avec des arcs γ_λ de capacité :

$$\gamma_\lambda(ij)=2c_{ij} \qquad \text{avec} \quad j>i, \quad j\in J_s, \quad i\in J_s$$

$$\gamma_\lambda(\boldsymbol{S}j)=\max\Big(0, \sum_{\substack{i\in J_s\\ i>j}} \gamma_\lambda(ji)-q_j\Big)$$

$$\gamma_\lambda(j\,\mathbf{P})=\max\Big(0, q_j-\sum_{\substack{i\in J_s\\ i>j}} \gamma_\lambda(ji)\Big).$$

Remarquons que pour un arc quelconque $\gamma_\lambda(ij)$ est indépendante de λ. Seules les capacités des arcs reliés à la source ou au puits en dépendent. En effet, pour $j\in J_s$:

$$\sum_{\substack{i\in J_s\\ i>j}} \gamma_\lambda(ji)-q_j=\sum_{\substack{i\in J_s\\ i>j}} 2c_{ji}-C_j+2c_{js}-\lambda$$

$$=\sum_{\substack{i\in J_s\\ i>j}} c_{ji}-\sum_{\substack{i\in J_s\\ i<j}} c_{ji}+c_{js}-\lambda$$

Posons $U_j=\displaystyle\sum_{\substack{i\in J_s\\ i>j}} c_{ji}-\sum_{\substack{i\in J_s\\ i<j}} c_{ji}+c_{js}$.

Remarque : Il est facile de vérifier que $\displaystyle\sum_{j\in J_s} U_j=C_s$. On a alors :

$$\gamma_\lambda(\boldsymbol{S}j)=\max(0, U_j-\lambda)$$

$$\gamma_\lambda(j\,\mathbf{P})=\max(0, \lambda-U_j).$$

PROPRIÉTÉ 5 : Pour λ donné,

$$\omega_s(\lambda) = C_s - (p-1)\lambda + \Psi(\lambda) - \sum_{j \in J_s} \gamma_\lambda(\boldsymbol{S}j)$$

où $\Psi(\lambda)$ est le flot maximal dans le réseau $N=(V, E, \gamma_\lambda)$.

Ceci découle directement du théorème de Picard et Ratliff et de ce qui précède.

PROPRIÉTÉ 6 : Posons :

$$\lambda_{\max} = \max_{j \in J_s} U_j;$$

$$\lambda_{\min} = \min_{j \in J_s} U_j; \qquad \lambda_0 = \frac{C_s}{n-1}.$$

(i) Si $\lambda > \lambda_{\max}$ alors $\omega_s(\lambda) = C_s - (p-1)\lambda = L_s(O', \lambda)$.

(ii) Si $\lambda < \lambda_{\min}$ alors $\omega_s(\lambda) = (n-p)\lambda = L_s(1, \lambda)$.

(iii) Pour tout λ, $\omega_s(\lambda) \leqq (n-p)\lambda_0$.

Preuve :

Cas (i) : $\forall \lambda$, $U_j - \lambda < 0 \Rightarrow \gamma_\lambda(\boldsymbol{S}j) = 0 \; \forall j \in J_s$, aucun sommet n'est relié à la source.

$\Psi(\lambda) = 0$. La coupe correspondante ne contient que les arcs issus de $\boldsymbol{S}$, d'où la solution en x : $x_j = 0 \; \forall j \in J_s$.

Cas (ii) : aucun sommet n'est relié au puits, par contre ils sont tous reliés à la source : $\forall \lambda$, $U_j - \lambda > 0$

$$\Psi(\lambda) = 0, \qquad \gamma_\lambda(\boldsymbol{S}_j) = U_j - \lambda$$

$$\omega_s(\lambda) = C_s - (p-1)\lambda + 0 - \sum_{j \in J_s} (U_j - \lambda)$$

$$= C_s - (p-1)\lambda - C_s + (n-1)\lambda$$

$$= (n-p)\lambda.$$

La solution correspondante en X est $x_j = 1 \; \forall j \in J_s$ donc le vecteur 1.

Cas (iii) : nous l'avons démontré de façon plus générale au paragraphe 1 : $\forall \lambda$, $\omega_s(\lambda) \leqq (n-p)\lambda_0$.

D'où : $\omega_s(\lambda_0) \leqq (n-p)\lambda_0$.

Nous le retrouvons ici :

$$\omega_s(\lambda_0) = C_s - \frac{(p-1)\,C_s}{n-1} + \Psi(\lambda_0) - \sum_{J_s} \gamma_{\lambda_0}(Sj)$$

$$= \frac{n-p}{n-1} C_s + \Psi(\lambda_0) - \sum_{J_s} \gamma_{\lambda_0}(Sj).$$

Par définition de la valeur d'un flot :

$$\Psi(\lambda_0) - \sum_{j \in J_s} \gamma_{\lambda_0}(Sj) \leqq 0$$

D'où : $\omega_s(\lambda_0) \leqq \dfrac{n-p}{n-1} C_s$. □

De plus, si le flot maximal dans $N_{\lambda_0} = (V, E, \gamma_{\lambda_0})$ sature les arcs entrants, alors :

$$\Psi(\lambda_0) = \sum_{j \in J_s} \gamma(Sj)$$

et

$$\omega_s(\lambda_0) = C_s - (p-1)\lambda_0 = \frac{n-p}{n-1} C_s$$

Comme

$$\omega_s(\lambda) \leqq \omega_s(\lambda_0), \qquad \forall \lambda$$

alors

$$\omega_s(\lambda_0) = \max_{\lambda} \omega_s(\lambda) = \omega_s(\lambda^*)$$

λ_0 est alors la meilleure valeur de λ, celle donnant la meilleure borne inférieure. D'où :

PROPRIÉTÉ 7 : Si le flot maximal dans $N_\lambda = (V, E, \gamma_\lambda)$ sature les arcs entrants, alors :

$$\omega_s(\lambda^*) = \max_{\lambda} \omega_s(\lambda) = \frac{n-p}{n-1} C_s$$

et

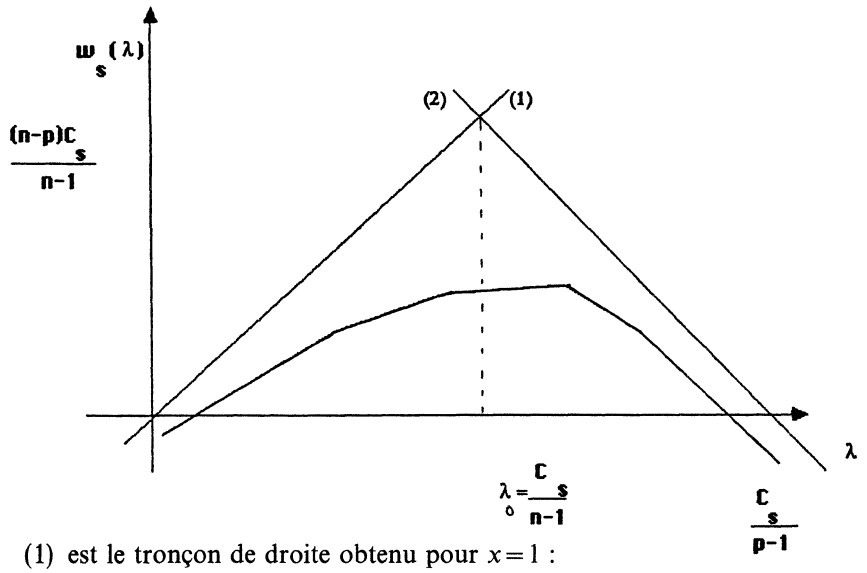
$$\lambda^* = \frac{C_s}{n-1} = \lambda_0.$$

Ce cas particulier est intéressant puisque dans ce cas la meilleure borne inférieure est obtenue très facilement !

4. *Graphe de* $\omega_s(\lambda)$

Nous pouvons ainsi tracer le graphe représentatif de $\omega_s(\lambda)$ en fonction de λ : enveloppe inférieure de fonctions linéaires en λ.

Pour x^i donné, tel que $L(x^i, \lambda) = \min_{x \in (1)'} L(x, \lambda)$, $\omega_s(\lambda)$ est un tronçon de droite que nous écrirons par la suite : $\alpha\lambda + \beta$.



(1) est le tronçon de droite obtenu pour $x = 1$:

$$L_s(1, \lambda) = \lambda(n-p) = \alpha_1 \lambda + \beta_1$$
$$\alpha_1 = n - p > 0, \qquad \beta_1 = 0.$$

(2) est le tronçon obtenu pour $x = O'$;

$$L_s(O', \lambda) = C_s - (p-1)\lambda = \alpha_2 \lambda + \beta_2$$
$$\alpha_2 = -p + 1 < 0, \qquad \beta_2 = C_s$$

(1) et (2) se coupent en $\lambda_0 = C_s/(n-1)$.

5. MEILLEURE BORNE INFÉRIEURE

La meilleure borne inférieure est obtenue par résolution du problème dual

$$\max_{\lambda} \omega_s(\lambda) = \omega_s(\lambda^*)$$

en cherchant le meilleur multiplicateur de Lagrange λ^*.

Comme pour chaque valeur de λ donnée, $\omega_s(\lambda)$ est obtenu par recherche d'un flot maximal dans un réseau de transport N (propriété 5), d'après le corollaire 4, λ^* est calculé en au plus $n-1$ itérations puisque $\omega_s(\lambda)$ possède au $n-1$ tronçons.

D'où le *théorème* :

La complexité du calcul de la meilleure borne inférieure du problème de la partition minimale d'un graphe par relaxation lagrangienne est $O(n^4)$ (pire des cas).

La recherche d'un flot maximal dans le réseau N qui possède $n+2$ sommets et $n+m$ arcs (dans le graphe initial, $|X|=n$ et $|U|=m$) étant dans le pire des cas en $O(n^3)$ si on utilise l'algorithme de Karzanov [6] ou celui « des Trois Indiens » [8].

6. ALGORITHME POUR LA RECHERCHE DE LA MEILLEURE BORNE INFÉRIEURE (algorithme 1)

Nous allons voir que la meilleure valeur de λ peut être obtenue très simplement.

(a) Initialisation

$$k=0$$

$$\lambda = \lambda_0 \text{ point d'intersection des tronçons (1) et (2)}$$

$$= \frac{\beta_1-\beta_2}{\alpha_1-\alpha_2} = \frac{C_s}{n-1}$$

avec

$$\alpha_2 = 1-p, \qquad \beta_2 = C_s$$

$$\alpha_1 = n-p, \qquad \beta_1 = 0.$$

(b) Itération k

Recherche de $\omega_s(\lambda_k) = \min_x L_s(x, \lambda_k)$.

Soit $L_s(x^k, \lambda_k) = \min_{x \in (1)'} L_s(x, \lambda_k) = \omega(\lambda_k)$; la solution qui minimise $L_s(x, \lambda_k)$, que nous appellerons x^k, est trouvée en cherchant dans un réseau particulier N un flot de valeur maximale (propriété 5).

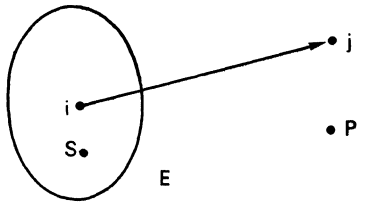
Il faut donc :

- construire N_{λ_k};
- chercher un flot maximal entre S et P;
- soit C_k le sous-ensemble d'arcs de la coupe minimale correspondant au flot maximal. C_k est associée à un ensemble de sommets E tel que :

$$S \in E,\ P \notin E \text{ et } C_k = \{(i, j) \mid i \in E,\ j \notin E\}$$

X^k a donc pour coordonnées :

$$\begin{cases} x_i^k = 1 & \text{si } i \in E \\ x_i^k = 0 & \text{sinon} \end{cases}$$

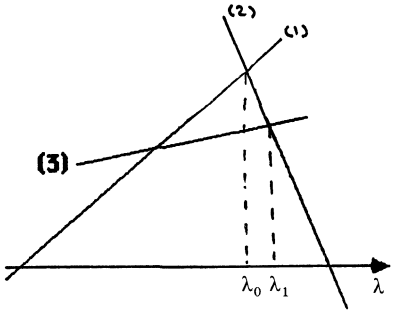


$L_s(x^k, \lambda)$ est un tronçon de droite ω_k que nous noterons :

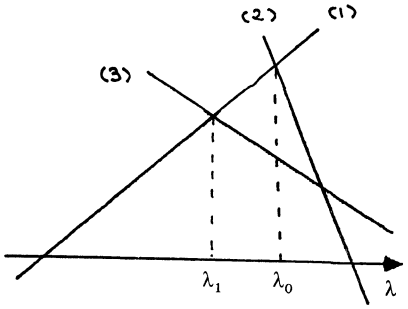
$$\omega_k = \alpha_3 \lambda + \beta_3.$$

(c) Nouvelle valeur de λ : calcul de λ_{k+1}

La suite de cet algorithme va prouver qu'une simple résolution graphique est nécessaire pour trouver λ^*. Le tronçon de droite correspondant à ω_k coupe les tronçons (1) et (2).

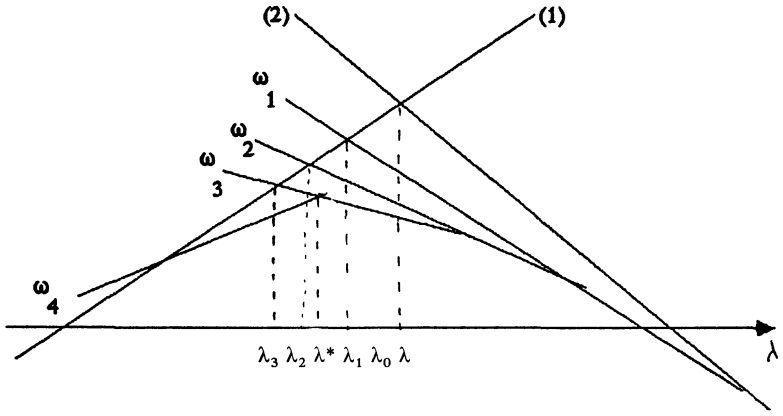


Si $\alpha_3 > 0$, le nouveau λ est pris à l'intersection de (3) et (2)



Si $\alpha_3 < 0$, le nouveau λ est pris à l'intersection de (3) et (1)

(d) Test d'arrêt



Si le signe de la pente du tronçon (3) change de pente par rapport à l'itération précédente, le dernier $\lambda(\lambda_k)$ est optimal : $\omega(\lambda^*) = \max_\lambda \omega(\lambda) = \omega(\lambda_k)$.

FIN.

Sinon on recommence en (*b*) avec la nouvelle valeur de λ

$k = k + 1$

si $\alpha_3 > 0$, le rôle de la droite (1) est jouée par (3),

si $\alpha_3 < 0$, le rôle de la droite (2) est jouée par (3).

Le fait que $\omega_s(\lambda)$ soit composée au plus de n tronçons assure qu'il ne faille pas plus de $(n-1)$ itérations pour trouver la meilleure borne inférieure.

Nous donnons en annexe cet algorithme dans un cas général.

IV. ALGORITHME « BRANCH AND BOUND »

Possédant maintenant une fonction d'évaluation pour z, avant de mettre en œuvre une procédure B & B, nous allons donner une méthode pour trouver une solution réalisable [vérifiant la contrainte (2) de P] à partir d'une solution quelconque $x^k \in \{0, 1\}^n$.

1. Procédure gourmande de recherche d'une solution réalisable [appelée Sol (x^k)]

Soit $x^k \in \{0, 1\}^n$ de valeur z^k

$$\sum_{j=1}^{n} x_j^k = p^k \qquad \text{avec} \quad p^k \neq p \tag{2}$$

Si $p^k < p$, il faut faire passer de 0 à 1 $(p - p^k)$ composantes de x^k pour obtenir une solution réalisable [vérifiant (2)]. Elles vont être choisies de la façon ci-dessous :

si $x_l^k = 0$, poser $x_l^k = 1$ provoque une augmentation de z, Δz_l égale à :

$$\sum_{j \neq l} c_{lj}(1 - x_j) - \sum_{j \neq l} c_{lj} x_j = C_l - 2 \sum_{j \neq l} c_{lj} x_j$$

Soit $\Delta z_h = \min_l \Delta z_l$. Poser $x_h^k = 1$, est donc le changement qui provoque la plus faible augmentation de z. D'où la méthode :

Tant que $p^k < p$, chercher $\Delta z_h = \min_{l \in \{ l \mid x_l^h = 0 \}} \Delta z_l$ poser $x_l^k = 1$.

De façon symétrique, si $p^k > p$ on fixera à 0 une composante x_h^k qui vaut 1 dans x^k et ainsi de suite.

2. Cas $p = n/2$

Nous allons maintenant construire une procédure par séparation et évaluation (B & B) pour le problème de la partition minimale :

– en fixant progressivement la valeur des variables à 0 ou 1,

– en se servant pour l'évaluation de l'algorithme précédemment défini (algorithme 1).

2.1. *Racine de l'arborescence*

— Chercher le sommet s que l'on va fixer à 1 :

s est choisi tel que $C_s = \max_i C_i$ avec $C_i = \sum_{j=1}^{n} c_{ij}$;

— poser $x_s = 1$,

z_{opt} valeur de la meilleure solution réalisable connue $= +\infty$;

— chercher la meilleure borne inférieure du problème P_s par l'algorithme 1.

Le sommet racine de l'arborescence $T^0 = \{ x \mid x \in \{0, 1\}^n$, et x vérifiant (2), $x_s = 1)$. Son évaluation $v(T^0) = L(x^k, \lambda^*)$.

Soit x^k la solution associée à la coupe minimale. Elle peut être rendue réalisable par la méthode Sol :

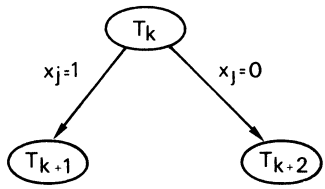
poser $z_{opt} = \mathrm{Sol}(x^k)$.

2.2. *Principe de séparation*

Nous avons tester deux principes : le premier est une « dichotomie » (arborescence DICO), le second une « polytomie » (arborescence POLY).

— *dichotomie*

Lorsqu'un sommet de l'arborescence T_k est séparé, deux successeurs T_{k+1} et T_{k+2} sont créés.



$$T_{k+1} = \{ x \mid x \in T_k \text{ et } x_j = 1 \}$$

$$T_{k+2} = \{ x \mid x \in T_k \text{ et } x_j = 0 \}.$$

Le critère de choix de x_j est celui de la plus forte augmentation Δz_j pour les variables non encore fixées (on veut éliminer rapidement les plus mauvaises solutions).

— *polytomie*

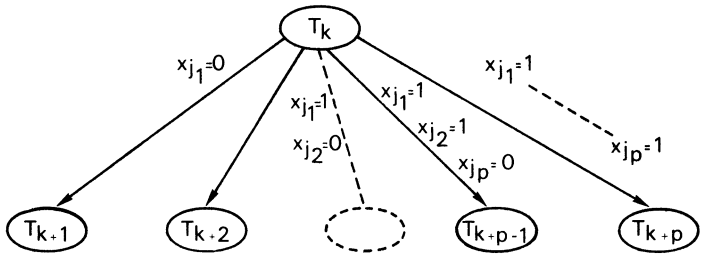
Soit x^h la solution réalisable obtenue par application de Sol sur x^k solution trouvée lors du calcul de la borne inférieure au sommet T_k.

Classons les p variables à 1 dans cette solution suivant le critère :

$$\Delta z_j \text{ décroissant}$$

(les plus « mauvaises » solutions sont en tête...).

Soient alors $x_{j1}, x_{j2}, \ldots, x_{jp}$ les p variables de tête. La séparation de T_k va consister à créer p successeurs :



L'évaluation de T_{k+p} est connue : $v(T_{k+p})=z_k$, celle de T_{k+p-1} également $v(T_{k+p-1})>z_h$ d'après la procédure gourmande Sol utilisée. Finalement, $(p-2)$ successeurs seulement seront créés et évalués.

Remarque : Dès que $(p-1)$ variables sont fixées, la p-ième variable peut être trouvée en une itération de la procédure gourmande : c'est alors la meilleure solution réalisable avec p variables à 1.

2.3. *Fonction d'évaluation*

En un sommet T_k de l'arborescence, nous utilisons l'algorithme 1 pour calculer la meilleure borne inférieure des solutions appartenant à ce sommet.

Remarques :

– Le premier multiplicateur de Lagrange λ_0 en un sommet T_k peut être pris égal à la valeur du point d'intersection des tronçons de droite (1) et (2) (toutes variables non encore fixées à 0, puis 1) ou à la meilleure valeur trouvée pour le calcul de la borne inférieure du sommet père.

– Au cours de l'algorithme 1, dès que $\omega(\lambda_i)>z_{opt}$, on arrête son déroulement, les solutions appartenant au sommet dont on cherche la meilleure évaluation ne pourront évidemment donner une solution de meilleure valeur que celle de l'actuel z_{opt}.

– L'algorithme utilisé pour la recherche d'un flot maximal dans $N(\lambda)$ est l'algorithme de Karzanov. Comme les capacités des arcs reliés à la source ou au puits ne sont pas forcément entières, λ étant réel, il a du être adapté à la recherche d'un flot maximal dans un réseau à capacités réelles.

2.4. *Stratégie*

« meilleur d'abord » dans les deux arborescences.

3. Cas $p \neq n/2$

Le choix de l'appartenance de x_s à X_1 n'est pas indifférent comme nous l'avons vu au paragraphe III. 1. Le problème P se décompose en deux sous-problèmes selon que le sommet s est du côté des p sommets ou des $(n-p)$ sommets.

Pour le problème P_s, la fonction lagrangienne $L'(x, \lambda)$ ne varie que d'une constante.

$$L'(x, \lambda) = \sum \sum \ldots + \lambda \left(\sum_{j \neq s} x_j - n + p + 1 \right)$$

$$= L(x, \lambda) + \lambda (2p - n)$$

Soit

$$\omega'(\lambda) = \min_{x \in (1)'} L'(x, \lambda)$$

$$= \lambda (2p - n) + \min_{x \in (1)'} L(x, \lambda)$$

$$= \lambda (2p - n) + \omega(\lambda).$$

Les points anguleux de la fonction $\omega'(\lambda)$ sont les mêmes que ceux de $\omega(\lambda)$: $\lambda_0, \lambda_1, \ldots \lambda_i$.

Par contre, l'optimum n'est pas forcément le même. Mais, pour λ^*, $\omega' = \omega^* + \lambda^*(2p - n)$ fournit une borne inférieure pour le problème $\bar{P}$.

Ainsi, après avoir cherché la solution optimale du problème P, on calculera cette borne inférieure : $\omega' = \omega + \lambda^*(2p - n)$ (λ^* meilleur multiplicateur) pour tous les sommets pendants de l'arborescence.

Ne seront par la suite, candidats à l'exploration, que des sommets pendants d'évaluation inférieure strictement à la valeur de la solution optimale de P.

VI. TESTS ET COMPARAISONS

Le programme a été écrit ([3]) en Pascal. Il est interactif et permet de préciser, lors d'une session, le type de graphe : (graphe particulier, généré au hasard, complet), sa taille (n), le nombre de sommets de la coupe (p), éventuellement le degré moyen d'un sommet du graphe (*cf.* VI. 2), le principe de séparation dans l'arborescence (dichotomie ou polytomie).

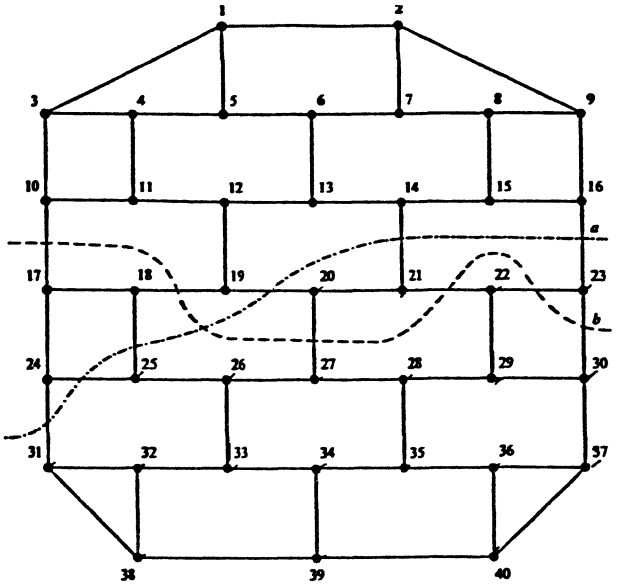
([3]) Projet de Maîtrise d'Informatique : D. Cofais, et D. Marchand.

Il utilise un sous programme de recherche de flot maximal basé sur l'algorithme de Karzanov [9] écrit en Fortran qui a dû être adapté aux réseaux à capacités réelles.

1. Exemples de la littérature

Nous avons utilisé les exemples de Barnes [1] et Christofidès et Brooker qui ont proposé [3] un algorithme B & B pour résoudre ce problème. Les bornes inférieures sont recherchées par résolution de problèmes de flot maximal, le principe de séparation se fonde sur la recherche d'un arbre de coût maximal.

Un des exemples donné est le suivant :



graphe. test.

Arête (i, j)	Coût c_{ij} Graphe A	Graphe B	Arête	Coût G_A	G_B
(1.2)	10	7	(19.20)	1	1
(1.3)	4	8	(20.21)	4	8
(1.5)	2	7	(20.27)	8	2
(2.7)	6	5	(21.22)	9	4
(2.9)	8	1	(22.23)	9	1
(3.4)	5	2	(22.29)	9	7
(3.10)	2	8	(23.30)	3	7
(4.5)	9	7	(24.25)	8	8
(4.11)	8	6	(23.31)	2	8
(5.6)	1	8	(25.26)	7	10
(6.7)	4	10	(26.27)	3	10
(6.13)	10	8	(26.33)	8	6
(7.8)	3	9	(27.28)	9	5
(8.9)	5	4	(28.29)	2	4
(8.15)	2	6	(28.35)	7	5
(9.16)	4	5	(29.30)	4	9
(10.11)	8	1	(30.37)	9	8
(10.17)	6	3	(31.32)	5	5
(11.12)	3	3	(31.38)	3	10
(12.13)	2	7	(32.33)	9	4
(12.19)	3	4	(32.38)	3	2
(13.14)	7	10	(33.34)	9	2
(14.15)	5	5	(34.35)	1	2
(14.21)	3	3	(34.39)	5	2
(15.16)	4	10	(35.36)	3	8
(16.23)	3	6	(36.37)	3	3
(17.18)	4	5	(36.40)	9	10
(17.24)	6	5	(37.40)	8	9
(18.19)	6	2	(38.39)	6	4
(18.25)	2	3	(39.40)	5	9

Ces exemples ($n=40$) sont résolus en 22,8 et 9,11 secondes sur DPS8 sous Multics. L'optimalité des solutions trouvées par Barnes ($n=20$) est prouvée en 2,5 et 1,7 s.

Les tests suivants permettront de mieux comparer nos résultats avec ceux de Christofides et Brooker.

2. Graphes irréguliers de degré moyen d

Un générateur de graphe construit ces graphes de la façon suivante : il génère aléatoirement un arbre de n sommets, puis lui ajoute des arêtes jusqu'à obtenir un degré moyen $d=\sum_i d_i/n$ où d_i est le degré d'un sommet. Le coût des arêtes est uniformément réparti entre 1 et 10.

Pour chaque type de graphe, un temps moyen de l'algorithme pour 10 passages est donné (temps CPU, Bull DPS8 sous système Multics).

Temps moyen
(en secondes)

	ROUCAIROL HANSEN (Bull DPS 8)				CHRISTOFIDES BROOKER (CDC 6600)			
	Graphe de degré moyen d=3							
n	$\overline{T}$	T	I	NS	$\overline{T}$	T	I	NS
12	0.09	0.30	0.59	7.2	0.08	0.27	0.71	462
20	0.36	1.05	2.27	8.3	0.38	3.86	15.48	429
30	2.87	6.096	12.73	22.7	12.21	26.42	35.92	1093
40	6.9	23.55	47.31	37.0	60.12	108.3	300.0	3809
50	40.19	63.40	132.46	97.7				
					pas de résultats au delà			

n nombre de sommets, $p=n/2$, *T* temps moyen;
$\overline{T}$ meilleur temps, *I* plus mauvais temps;
NS nombre moyen de sommets créés et évalués dans l'arborescence (flot maximal résolus) pour les deux méthodes.

Temps moyen
(en secondes)

	ROUCAIROL HANSEN (Bull DPS8)				CHRISTOFIDES BROOKER (CDC 6600)			
	Graphe de degré moyen d=4							
n	$\overline{T}$	T	I	NS	$\overline{T}$	T	I	NS
12	0.11	0.412	1.0	7.1	0.93	1.73	4.28	250
20	0.54	1.75	3.61	12.7	9.31	19.61	127.4	1182
30	1.62	8.56	13.99	31.1		+233.5		12504
40	4.89	35.28	65.34	48.8				
50	30.56	71.56	110.74	103.8				

Graphe complet

n	$\overline{T}$	T	$\underline{T}$	NS	n	$\overline{T}$	T	$\underline{T}$	NS
12	0.74	1.09	1.46	20.6	10	3.68	5.41	8.89	1327
20	142.84	65.9	300.34	1532.3	16		+293.1		6845

\+ un seul essai;
n nombre de sommets, $p=n/2$, T temps moyen;
$\overline{T}$ meilleur temps, $\underline{T}$ plus mauvais temps;
NS nombre moyen de sommets créés et évalués dans l'arborescence (flot maximal résolus) pour les deux méthodes.

Pour comparer les temps obtenus par notre algorithme à ceux de Christofidès et Brooker, il faudrait les diviser par un facteur entre 1,5 et 2 (le CDC 6600 étant plus rapide). Ceci n'aurait donc pour effet que d'améliorer les temps qui étaient déjà à chaque fois, meilleurs.

Ceci s'explique par le fait que l'arborescence créée par « Branch and Bound » dans notre méthode possède peu de sommets (colonne *NS*); ainsi le nombre de problèmes de flot maximal résolus est petit (400 fois moins pour une taille de 30 sommets) ce qui nous permet de résoudre des problèmes de grande taille.

Les temps croissent également moins rapidement avec la densité du graphe dans le cas de notre algorithme.

BIBLIOGRAPHIE

1. E. R. Barnes, *An Algorithm for Partitioning the Notes of a Graph*, S.I.A.M. J. Alg. Disc. Meth., vol. , n° 4, 1982, p. 541-550.
2. P. Chaillou, P. Hansen et Y. Mahieu, *Best Network Flow Bounds for the Quadratic Knapsack Problem*, Presented at NETFLOW 83 International Workshop, Pisa, Italy, 1983.
3. N. Christofidès et P. Brooker, *The Optimal Partitioning of Graphs*, S.I.A.M. J. Apl. Math., vol. 30, n° 1, 1976, p. 55-70.
4. M. Garey, E. Johnson et Stockmeyer, *Some Simplified NP-Complete Problems*, Theoretical Computer Science, vol. 1, 1976, p. 237-267.
5. P. L. Hammer (Ivanescu), *Some Network Flow Problems Solved by Pseudo-Boolean Programming*, Operations Research, vol. 13, 1965, p. 388-299.
6. A. V. Karzanov, *Determining the Maximal Flow in at Network by the Method of Preflows*, Soviet. Math. Dokl., vol. 15, 1974, p. 434-437.

7. M. MINOUX, *Programmation Mathématique*, t. 1, C.N.E.T., E.N.S.T., Dunod, Paris, 1983.
8. V. M. MALHORTA, M. P. KUMAR et S. N. MAHESHWARI, *An Algorithm for Finding Maximum Flow in Networks*, Information Processing Letters, vol. 7-6, 1978, p. 277-278.
9. A. NIJENHUIS et H. S. WILF, *Combinatorial Algorithms*, Second Edition, Academic Press, New York, 1978.
10. J. C. PICARD et M. QUEYRANNE, *Networks, Graphs and Some Non Linear* 0.1 *Programming Problems*, Tech. Rep. EP77-R-32, École Polytechnique de Montréal, Canada, 1977.
11. J. C. PICARD et H. D. RATLIFF, *Minimum Cuts and Related Problems*, Networks, vol. 5, 1975, p. 357-370.
12. J. C. PICARD et H. D. RATLIFF, *A Cut Approach to a Class of Quadratic Integer Programming Problems*, Networks, vol. 10, 1980, p. 363-370.
13. P. SIARRY, *La méthode du Recuit Simulé : application à la conception de circuits électroniques*, Thèse de Doctorat de l'Université Paris-VI, Spécialités : Sciences Physiques, Paris, novembre 1986.

ANNEXE

Recherche de la meilleure borne inférieure (algorithme 1)

(1) Calculer α_1 et β_1 coefficients directeurs de la droite en λ obtenue lorsque toutes les variables libres sont fixées à 1 dans $L(x, \lambda)$; calculer α_2 et β_2 coefficients directeurs de la droite en λ obtenue lorsque toutes les variables libres sont fixées à 0 dans $L(x, \lambda)$; si $\beta_2 < \beta_1$, alors $\lambda_0 = 0$ aller en (4).

(2) $k = 0$.

(3) $\lambda_k = (\beta_2 - \beta_1)/(\alpha_1 - \alpha_2)$.

(4) Construire le réseau de transport $N(\lambda_k)$.

Chercher le flot maximal dans $N(\lambda_k)$ en déduire x^k.

$$\min_{x \in (1)'} L(x, \lambda_k) = L(x^k, \lambda_k).$$

(5) $\omega_k = L(x^k, \lambda) = \alpha_3 \lambda + \beta_3$.

Calculer α_3 et β_3.

(6) Si $\alpha_3 \lambda + \beta_3 = \alpha_1 \lambda + \beta_1$ alors $\lambda_k = \lambda^*$

$\omega_k(\lambda_k) = \omega(\lambda^*)$

FIN

sinon

si $\alpha_3 < 0$ *alors poser* $\alpha_2 = \alpha_3$, $\beta_2 = \beta_3$

sinon poser $\alpha_1 = \alpha_3$, $\beta_1 = \beta_3$

$k = k + 1$
aller en (3).