UNIVERSITÉ NANCY 1 U.E.R. SCIENCES MATHÉMATIQUES

ADAPTATION ET EXTENSION DE LA MÉTHODE DE LITTLE POUR UNE APPLICATION SUR ORDINATEUR

SOLUTION OPTIMALE ET SOLUTIONS SOUS-OPTIMALES

THÈSE

pour l'obtention du

Doctorat de Spécialité en Informatique

soutenue le 28 Novembre 1975

par M. GOEPP Dominique

JURY : M. LEGRAS président

Mme ROLLAND

M. BOYER

UNIVERSITÉ NANCY 1 U.E.R. SCIENCES MATHÉMATIQUES

ADAPTATION ET EXTENSION DE LA MÉTHODE DE LITTLE POUR UNE APPLICATION SUR ORDINATEUR SOLUTION OPTIMALE ET SOLUTIONS SOUS-OPTIMALES

THÈSE

pour l'obtention du

Doctorat de Spécialité en Informatique

soutenue le 28 Novembre 1975

par *M. GOEPP Dominique*

JURY : M. LEGRAS président

Mme ROLLAND

M. BOYER

Je remercie vivement Monsieur le Professeur LEGRAS de l'honneur qu'il me fait de présider le Jury de cette thèse et je lui exprime ma gratitude d'avoir bien voulu diriger mes travaux de recherches pour lesquels il m'a prodigué de précieux conseils.

Je suis également reconnaissant à Madame ROLLAND et Monsieur BOYER qui ont bien voulu juger ce travail.

Mes remerciements vont aussi à Monsieur STROSSER, grâce à qui nos travaux sur ordinateur ont été beaucoup facilité ainsi qu'à Madame RUCH et Monsieur LITT pour la part qu'ils ont pris dans la réalisation pratique de cette thèse.

P L A N Page

I N T R O D U C T I O N

On se propose d'étudier dans ce travail le problème connu sous le nom de "Problème du voyageur de commerce", en anglais "Traweling salesman problem" ; en termes mathématiques on le désigne comme la recherche des circuits hamiltoniens à coût minimal.

Rappelons l'énoncé de problème : un voyageur de commerce part d'une ville quelconque d'un réseau de villes et doit passer par chaque ville une et une seule fois, revenir à son point de départ en parcourant une distance minimale.

Pour résoudre de façon effective ce problème, BELLMANN a proposé un algorithme qui a ses justifications dans la programmation dynamique, FORD JONHSSON et FULKERSON avaient élaboré auparavant une solution qui s'inspirait de la programmation linéaire avec l'application d'un algorithme proche du simplex et KRUKSAL une solution faisant appel à des éléments de la théorie des graphes.

Nous nous intéresserons ici plus particulièrement à l'utilisation de méthodes du type Branch and Bound qui sont par ailleurs les plus récentes.

La méthode Branch and Bound la plus classique pour la résolution du problème du voyageur de commerce est l'algorithme de LITTLE. Cependant, on constate que son application devient impossible quand le nombre de points du réseau dépasse 20 même sur un ordinateur de grande taille.

On a donc essayé d'abord d'adapter l'algorithme de LITTLE de façon à diminuer l'emplacement mémoire et ceci par des méthodes de calcul qui permettront le retour à la situation à l'étape p du développement de l'arborescence quand on est à la situation $p + j$.

L'écriture du problème du voyageur de commerce sous la forme d'un programme linéaire en variables booléennes nous permettra auparavant de justifier le choix de la variable de séparation qu'a fait LITTLE.

....

Mais cette adaptation n'est pas suffisante et l'algorithme de LITTLE ainsi modifié ne permet pas d'obtenir une convergence vers une solution optimale quand on a un réseau dont le nombre de points dépasse 40 ou 50.

On a donc essayé d'envisager un objectif différent pour ce type de graphe toujours à l'aide d'une méthode Branch and Bound, mais dans laquelle le mode de construction et de cheminement dans l'arborescence sont différents on calcule des circuits sous-optimaux. De plus cette deuxième méthode permet à tout moment d'obtenir un encadrement de l'optimum. On peut ainsi se fixer un temps de calcul et l'algorithme fournira quand ce temps sera écoulé le dernier circuit sous-optimal trouvé et une estimation de la différence par rapport au vrai coût minimal.

La comparaison et la programmation de ces deux algorithmes fourniront la matière de la troisième partie et permettront de définir des critères d'application de l'un ou de l'autre selon les dimensions du graphe.

o o o

o o

o

Principes de la Méthode Branch and Bound

On considère l'ensemble des solutions réalisables, c'est-à-dire l'ensemble des solutions qui satisfont les contraintes d'inégalité et d'égalité ; on essaie de diviser cet ensemble en deux ou plusieurs sous-ensembles disjoints, sans les expliciter de façon exhaustive, par des critères qui dépendront du problème à traîter; parmi ces sous-ensembles on va chercher à voir lequel a le plus de chances, à priori, de contenir la solution optimale; on verra plus loin comment peut se faire ce choix.

Ce sous-ensemble étant choisi, on le divise selon les mêmes critères en deux ou plusieurs sous-ensembles parmi lesquels s'opérera à nouveau un choix pour la prochaine division ou séparation qui est le terme employé dans la bibliographie française.

Quand on itère ce procédé, on arrive au bout d'un nombre fini de séparations sur un sous-ensemble qui n'a plus qu'un seul élément ; cet élément sera la solution optimale ou plutôt une des solutions optimales, puisqu'il n'existe aucune théorème d'unicité.

On constate donc que pour tout problème destiné à être résolu par une méthode Branch and Bound, il faudra préciser deux types de renseignements : les critères permettant d'opérer une séparation et les critères permettant d'effectuer un choix sur les sous-ensembles.

Critères de séparation

Dans la pluspart des cas en programmation linéaire en variables entières ou booléennes, c'est un critère qui consiste à fixer une valeur pour une variable jusqu'alors indéterminée; le nombre de sous-ensembles résultant de la séparation sera de 2 si la variable est booléenne et égal au nombre de valeurs du domaine de définition de la variable si la variable est entière.

Critères de choix d'un sous-ensemble

Définition d'un sous-ensemble pendant : un sous-ensemble est dit pendant s'il n'a pas encore subi de séparation .

Si on appelle E l'ensemble qui contient toutes les solutions réalisables et $E_1^{(p)}$, $E_2^{(p)}$, ... $E_m^{(p)}$ les sous-ensembles issus à l'itération p de la séparation de $E_k^{(p-1)}$ on aura

$$E_k^{(p-1)} = E_1^{(p)} \cup E_2^{(p)} \ldots \cup E_{mp}^{(p)}$$

Remarque : Le critère de choix doit se faire sur l'ensemble des bouts pendants qui résultent de toutes les séparations et non pas uniquement sur

$$E_1^{(p)} , E_2^{(p)} , \ldots\ldots E_{mp}^{(p)}$$

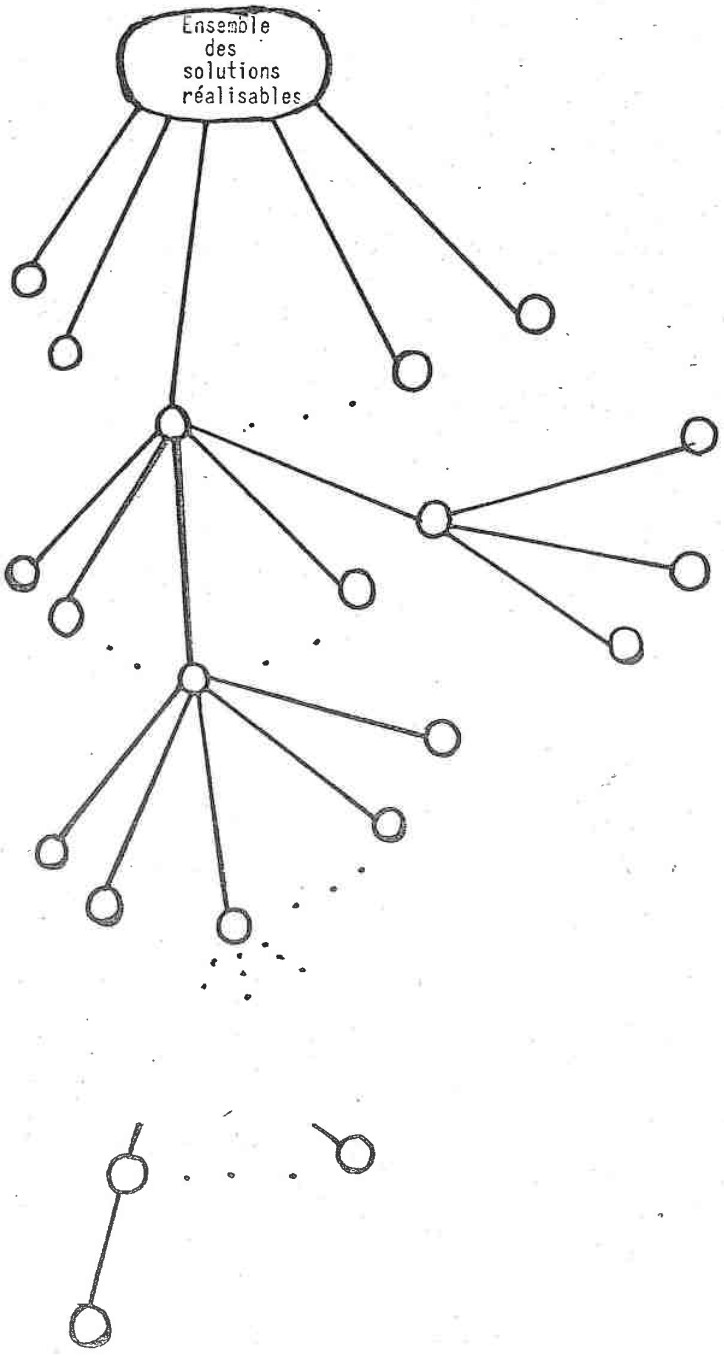
....

Pour choisir un sous-ensemble pour la $p^{ième}$ séparation, il faut associer à chaque sous-ensemble pendant une valeur minorante ou majorante de la fonction objectif qu'on appellera la borne du sous-ensemble (en anglais Bound).

On choisira donc le sous-ensemble $E^{(p-e)}$ pendant qui aura la borne la plus petite ou la plus grande selon que le problème traité sera un problème de minimisation ou de maximisation.

Remarque : le meilleur minorant d'une fonction objectif est évidemment le plus grand et le meilleur majorant est évidemment le plus petit.

Représentation par arborescence de cette méthode



1ère méthode de progression dans l'arborescence

Après une itération donnée, c'est-à-dire après avoir effectué une séparation supplémentaire et affecté une borne à chaque sous-ensemble issu de la séparation, on choisit celui parmi tous les sous-ensembles du graphe qui n'ont pas encore été divisés, qui a la borne la plus favorable, c'est-à-dire la plus petite s'il s'agit d'une minimisation et la plus grande s'il s'agit d'une maximisation.

Pour ce qui est du choix de la propriété séparatrice et de la méthode de calcul de la borne, ils sont étroitement liés au problème traité et il faut se référer à l'exemple traité pour le concevoir clairement. On peut voir quelle est la façon dont on peut représenter les sous-ensembles et leurs liaisons entre-eux, grâce à un graphe arborescent dans lequel la source représente l'ensemble de toutes les solutions réalisables du problème. un sommet quelconque l'un des sous-ensembles intermédiaires, un arc la liaison d'un sous-ensemble avec le sous-ensemble dans lequel il est inclus.

Deuxième méthode de progression dans l'arborescence

Au lieu de prendre après chaque nouvelle itération le sous-ensemble choisi entre tous les sous-ensembles pendants qui a la borne la plus favorable afin de lui appliquer la technique de séparation , on peut choisir celui des

$E_k^{(p)}$ pour lequel $g_k^{(p)} \leqslant g_i^{(p)}$ $i \neq p$ $i = 1 \ldots m_p$

m_p étant le nombre de sous-ensembles résultant de la séparation de $E_{j\,p-1}^{(p-1)}$

On arrive ainsi plus rapidement sur un sous-ensemble à un seul élément, mais dont l'élément n'est pas nécessairement la solution optimale. On continue alors le processus de séparation à partir du bout pendant $E_j^{(i)}$ qui a les propriétés suivantes :

(1) $g_j^{(i)} < g^f$ g^f étant la borne associée au dernier sous-ensemble terminal trouvé

(2) $i \geqslant k$ $\forall k$ / $g^{(k)} < g^f$

Il peut alors se présenter deux possibilités : soit après un nombre fini de séparations on détermine un autre sous-ensemble terminal avec une borne $g_1^f < g^f$ et dans ce cas on recommence la recherche d'un bout pendant vérifiant les conditions (1) et (2) énoncées plus haut, soit après un nombre fini de séparations la borne la plus petite des sous-ensembles issus de la dernière séparation est supérieure ou égale à g^f ; dans ce cas on cherche un sous-ensemble pendant qui vérifie les conditions (1) et (2), c'est-à-dire on prend le sous-ensemble pendant dont la borne est inférieure à g^f et qui a été déterminé dans une séparation la plus éloignée de la source possible.

On obtiendra une solution optimale lorsqu'il n'existe plus dans l'arborescence de bouts pendants, dont la borne soit inférieure à la borne du dernier sous-ensemble terminal trouvé. La solution optimale sera l'élément de ce sous-ensemble.

....

Intérêt théorique comparé des deux méthodes de progression

On suppose dans ce paragraphe qu'on est confronté avec un problème de minimisation.

La grande particularité de la deuxième méthode par rapport à la première est qu'au bout d'un nombre de séparations qu'on connaît, on obtient une solution sous-optimale à laquelle est associée une borne que l'on peut considérer comme un majorant de la fonction objectif; donc, contrairement à ce qui se passe dans la première méthode où l'on améliore, selon un certain procédé propre au problème, les minorants de la fonction objectif jusqu'à ce qu'on arrive sur un sous-ensemble terminal qui dans ce cas contient tout de suite une solution optimale, on améliore dans la deuxième méthode des majorants de la fonction objectif en calculant des sous-optimaux.

Le point qui peut rester commun dans les deux méthodes pour un problème donné, c'est la méthode de calcul de la borne associée à chaque sous-ensemble.

Nous verrons plus loin que ces différences au niveau théorique se répercutent au niveau pratique quand on traitera le problème du voyageur de commerce pour des nombres de variables qui rendent le traitement sans calculateur impossible.

Etude et programmation du problème du voyageur de commerce à l'aide d'une méthode Branch and Bound

Présentation du problème réel

Imaginons un réseau de villes et un voyageur de commerce qui doit partir de l'une des villes, visiter chacune des villes une et une seule fois et revenir à son point de départ.

Etant donné que le trajet d'une ville i à une ville j entraîne un certain coût , le problème qui se pose au voyageur de commerce est de trouver un circuit qui minimise la somme de ses dépenses sur les différents trajets.

C'est bien un problème d'optimisation linéaire avec contraintes, puisqu'il s'agit de minimiser la somme des coûts occasionnés par les différents trajets en respectant la règle que le voyageur de commerce ne peut et ne doit rentrer qu'une et une seule fois dans chaque ville et revenir à son point de départ.

Remarque concernant les coûts

Le coût pour aller de la ville i à la ville j n'est pas nécessairement égal au coût de la ville j à la ville i . Il suffit pour cela d'imaginer que l'on prend pour coûts, soit le temps t_{ij} , soit la consommation d'essence et que la ville j soit située à une altitude plus élevée que i ; on aura alors ;

$t_{ij} > t_{ji}$ ou si c_{ij} est la consommation d'essence

$c_{ij} > c_{ji}$

....

Ce qu'il est important de constater c'est qu'il n'est pas indifférent dans ce cas là de prendre un trajet passant par (i,j) et un trajet passant par (j,i).

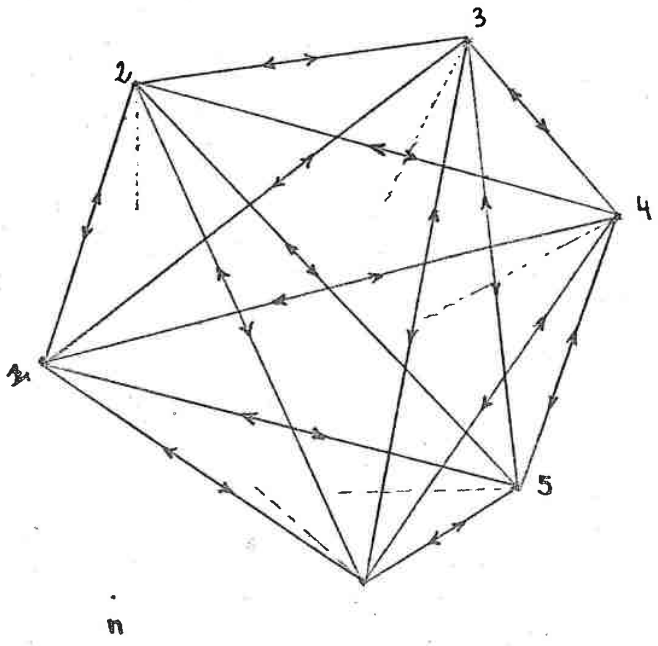
Au contraire, lorsque les coûts sont exprimés en terme de distances séparant les différentes villes alors on a

$$d_{ij} = d_{ji}$$

Ce cas particulier pourra nécessiter un algorithme de résolution plus spécifique qui assurera une convergence plus rapide.

Interprétation du problème en termes de programme linéaire

Représentation sous forme de graphe



Remarque : Chaque sommet représente une ville et chaque arc un trajet.

....

Il est clair qu'il existe des solutions réalisables sur un tel graphe.

Dénombrement du nombre de solutions réalisables pour un graphe symétrique (c.à d. un graphe auquel correspond une matrice des coûts symétrique)

Nombre de solutions réalisables dans un réseau à n villes = $\frac{(n-1)!}{2}$

pour un graphe non symétrique $N = (n-1)!$

La solution théorique brutale du problème du voyageur de commerce consisterait à énumérer toutes les solutions réalisables (c'est-à-dire à rechercher tous les circuits hamiltioniens du graphe) à calculer leur coût et choisir celle qui correspond au coût minimal.

En fait ce qui nous intéresse ici n'est pas de savoir qu'il existe une façon théorique de trouver une solution optimale, mais de calculer cette solution de manière effective quand la dimension du graphe le permet. Or on voit bien que même pour un graphe de petite taille de dimension n = 10 le nombre de solutions réalisables est égal à 9 != 362 880 solutions.

Formalisation en langage de programme linéaire

Associons à chaque arc(i,j) du réseau une variable xij qui prendra la valeur 0 si l'arc (i,j) n'appartient pas au circuit solution et la valeur 1 si l'arc (i,j) appartient au circuit solution.

xij = 1 l'arc (i,j) appartient au circuit

xij = 0 l'arc (i,j) n'appartient pas au circuit

Fonction objectif

$$\text{Min} \quad F(x) = \sum_{i,j} d_{ij} \; x_{ij}$$

où d_{ij} est le coût associé à l'arc (i,j)

Contraintes

Contraintes unilatérales

Dans cette catégorie de contraintes entrent celles qui expriment le fait que les circuits de longueur inférieure à n sont interdits.

L'énoncé complet de toutes ces contraintes n'apporte pas de relations déterminantes : on verra comment au niveau de l'algorithle on peut vérifier ces contraintes au moment où elles apparaissent.

....

Les plus simples parmi ces contraintes sont toutes celles du type

$$x_{ji} + x_{ij} \leqslant 1$$

pour un graphe de dimension n elles donnent naissance à $\binom{2}{n}$ relations (nombres de couples parmi n), c'est-à-dire $\frac{n(n-1)}{2}$ relations.

En ignorant les variables duales associées à ces contraintes, nous allons énoncer des résultats pour un problème qui contient le problème du voyageur de commerce; ces résultats seront formulés en supposant implicitement que les contraintes unilatérales sont toujours vérifiées.

Contraintes bilatérales

Traduisons le fait que de chaque ville i le voyageur de commerce ne part qu'une et une seule fois

$$\sum_j x_{ij} = 1 \qquad i = 1, \ldots n \tag{3}$$

et le fait qu'en chaque ville il rentre une et une seule fois

$$\sum_i x_{ij} = 1 \qquad j = 1, \ldots n \tag{4}$$

Conditions de signe

$$x_{ij} \geqslant 0 \qquad i,j = 1, \ldots n$$

Construisons le problème dual

Si à la série de conditions (3) on associe pour tout i la variable duale α_i et à la série de conditions (4) on associe pour tout j la variable duale β_j le problème dual reviendra à optimiser.

$$\text{Max} \left(\sum_i \alpha_i + \sum_j \beta_j \right) \tag{5}$$

avec comme contraintes

$$\alpha_i + \beta_j \leqslant d_{ij} \tag{6}$$

à chaque contrainte (6) est associée une variable x_{ij} sur laquelle on a la condition de signe $x_{ij} \geqslant 0$

....

On en déduit une condition d'optimalité intéressante pour notre problème, à savoir que

$$\text{ou } x_{ij} = 0 \qquad \text{où } \alpha_i + \beta_j = d_{ij}$$

ce qui peut encore s'écrire

$$\alpha_i + \beta_j < d_{ij} \Longrightarrow x_{ij} = 0$$

$$\text{ou } d_{ij} - \alpha_i - \beta_j > 0 \Rightarrow x_{ij} = 0 \qquad (1)$$

ce qui veut dire que si α_i et β_j vérifient les conditions (6) on ne devra admettre un arc (i,j) dans le circuit qu'à la condition

$$d_{ij} - \alpha_i - \beta_j = 0$$

Il faut remarquer que dans ce raisonnement nous n'avons pas supposé que les x_{ij} étaient des variables booléennes et que par conséquent les théorèmes relatifs au Primal - Dual étaient applicables .

De l'implication (2) on peut construire un algorithme de résolution du problème du voyageur de commerce en s'appuyant en outre sur une méthode Branch and Bound.

Interprétation en termes de modifications des coûts

L'analyse au problème du voyageur de commerce peut se faire de façon plus directe en ne passant pas par la dualité.

Pour cela considérons la matrice des coûts associée au réseau;

d_{ij} = coût de l'arc (i,j)

....

Par convention pour interdire un arc (i,j) on posera $d_{ij} = \infty$ ce qui rend le poids de la variable x_{ij} prohibitif.

Comme dans la solution finale on n'accepte aucun des arcs (i, i) on pose donc

$$d(i,i) = \infty \qquad \forall_i = 1, n$$

On raisonne alors de la façon suivante : comme on a la contrainte

$$\sum_j x_{ij} = 1 \qquad i = 1, \ldots n$$ cette contrainte entraîne de toute façon un coût minimum pour chaque arc (i, k_i) où $d_{i\,ki} = \min_j (d_{ij})$

Ceci permet de ramener le problème du voyageur de commerce au problème équivalent suivant :

Appelons g_o la somme des $d_{i\,ki}$ pour $i = 1, \ldots n$

$$g_o = \sum_{i=1} d_{i\,ki} = \sum_i \min_j (d_{ij})$$

et considérons la matrice des coûts $d_{ij}^{(1)} = d_{ij} - d_{iki}$

c'est-à-dire

$$\begin{pmatrix} \infty & d_{12} - d_{1k1} & d_{13} - d_{1k1} & \cdots & d_{1n} - d_{1k1} \\ d_{21} - d_{2k2} & \infty & d_{23} - d_{2k2} & \cdots & d_{2n} - d_{2k2} \\ \vdots & & & & \vdots \\ d_{n1} - d_{nkn} & \cdots & & \cdots & \infty \end{pmatrix}$$

Alors le problème du voyageur de commerce peut s'écrire :

$$\text{Min} \sum d_{ij}^{(1)} x_{ij} + g_o$$

$$\text{avec} \sum_j x_{ij} = 1 \qquad i = 1, \ldots n$$

$$\sum_i x_{ij} = 1 \qquad j = 1, \ldots n$$

....

Il s'agit maintenant de rédiger ce problème en procédant de la façon suivante :

Dans la matrice $d_{ij}^{(1)}$ on calcule

$$d_j\, p_j = \min_i \left(d_{ij}^{(1)} \right)$$

on pose $g_1 = \sum_j \min \left(d_{ij}^{(1)} \right)$

et on considère la matrice $d_{ij}^{(2)} = d_{ij}^{(1)} - d_j\, pj$

c'est-à-dire

$$\begin{pmatrix} \infty & d_{12}^{(1)} - d_{2p2} & \cdots & d_{1n}^{(1)} - d_{1pn} \\ d_{21}^{(1)} - d_{1p1} & \infty & & \\ d_{31}^{(1)} - d_{1p1} & d_{32}^{(1)} - d_{2p2} & \infty & \\ \vdots & & & \\ d_{n1}^{(n)} - d_{1p1} & & \cdots & \infty \end{pmatrix}$$

On en revient alors à résoudre le problème suivant :

$$\text{Min} \sum d_{ij}^{(2)}\, x_{ij} + g_1 + g_o$$

$$\text{avec} \sum_j x_{ij} = 1 \qquad i = 1, \ldots n$$

$$\sum_i x_{ij} = 1 \qquad j = 1, \ldots n$$

Interprétation : on constate que quel que soit le circuit hamiltonien qu'on puisse trouver le coût minimal du circuit sera égal à $g_1 + g_o$

donc quel que soit l'ensemble x_{ij} qui sera optimal il lui correspondra nécessairement et au minimum le coût $g_1 + g_o$

Remarque : La matrice des coûts $d_{ij}^{(2)}$ contient nécessairement un zéro pour chaque ligne et un zéro pour chaque colonne.

....

Algorithme de LITTLE pour la résolution du problème du voyageur de commerce

Revenons à l'analyse Primal-Dual du problème; on avait établi les propositions suivantes :

trouver $\quad \text{Min} \sum_{i,j} d_{ij} \; x_{ij} \qquad (1)$

avec $\quad \sum_{i} x_{ij} = 1 \quad j = 1, \ldots n \qquad (2)$

$$\sum_{j} x_{ij} = 1 \quad i = 1, \ldots n \qquad (3)$$

et les contraintes de non bouclage équivaut à trouver

$$\text{Max} \sum_{i} \alpha_i + \sum_{j} \beta_j \qquad (4)$$

avec $\quad \alpha_i + \beta_j \leq d_{ij} \qquad (5)$

et que $d_{ij} - \alpha_i - \beta_j > 0 \Longrightarrow x_{ij} = 0$

Appelons case admissible tout élément de la matrice

$d_{ij} - \alpha_i - \beta_j$ qui est nul.

On a la relation suivante

$$\alpha_i + \beta_j = d_{ij} \Longrightarrow \begin{cases} x_{ij} = 0 \\ x_{ij} = 1 \end{cases}$$

$$\alpha_i + \beta_j < d_{ij} \Longrightarrow x_{ij} = 0$$

Ceci veut dire qu'on n'admettra une variable x_{ij} dans le circuit que si la relation $\alpha_i + \beta_j = d_{ij}$ est vérifiée.

....

Si on pose

$$\alpha_i = \min_j (d_{ij})$$

et

$$\beta_j = \min_i (d_{ij} - \alpha_i)$$

la relation (5) est vérifiée $\forall$ i et j = 1, ... n

On construit donc un tableau à partir des coûts où on enlève le minimum sur chaque ligne à la ligne et cette opération étant achevée on recommence sur le tableau obtenu la même opération sur les colonnes.

Dans ce tableau on a donc sur chaque ligne et sur chaque colonne au moins un zéro. Ce sont autant de cases admissibles, donc de couples (i, j) pour lesquels les variables x_{ij} peuvent prendre des valeurs égales à 1.

Il s'agit maintenant de construire à partir de ces cases admissibles et à partir de la borne minimale $g_1 + g_o$ établie dans l'analyse directe un circuit hamiltonien de coût minimal.

C'est à ce niveau que se justifie l'emploi de la méthode Branch and Bound exposée dans le chapitre précédent et avec les particularités spécifiques au problème suivantes.

On prendra comme ensemble source E l'ensemble de tous les circuits hamiltoniens de longueur n .

Borne associée à l'ensemble source E

$$g_o^o = \sum_i \alpha_i^o + \sum_j \beta_j^o$$

α_i^o étant les $\alpha_i = \min_j (d_{ij}^{(o)}) = \min_j (d_{ij})$

β_j^o étant les $\beta_j = \min_i (d_{ij}^{(o)} - \alpha_i^o) = \min_j (d_{ij} - \alpha_i^o)$

la démonstration a été faite dans l'analyse directe où on avait

$$g_o = \sum_i \alpha_i^o \quad \text{et} \quad g_1 = \sum_j \beta_j^o$$

Choix de la propriété séparatrice

Quand parmi les variables admissibles, à une itération donnée k , on aura choisi de la façon qu'on verra dans le paragraphe suivant celle qui paraît la plus favorable on aura déterminé par la même occasion un arc qu'on note i_k j_k

....

Cet arc servira à déterminer deux sous-ensembles $E_1^{(k)}$ et $E_2^{(k)}$ tels que

$$E_1^{(k)} \cup E_2^{(k)} = E^{(k^1)}$$

$E^{(k^1)}$ étant le sous-ensemble pendant auquel est associé la borne minimale avant l'itération k (c'est le sous-ensemble père).

Si on note $\overline{(i_k, j_k)}$ le fait que l'arc (i_k, j_k) n'appartienne pas à un ensemble, $E_1^{(k)}$ et $E_2^{(k)}$ seront définis de la façon suivante :

$$E_1^{(k)} = E^{(k^1)} \cap \left\{ \overline{(i_k, j_k)} \right\}$$

$$E_2^{(k)} = E^{(k^1)} \cap \left\{ (i_k, j_k) \right\}$$

La notation $\left\{(i_k, j_k)\right\}$ signifiera ensemble des circuits hamiltoniens contenant l'arc (i_k, j_k)

Calcul des bornes associées à chaque sous-ensemble $E_1^{(k)}$ et $E_2^{(k)}$

Les indices 1 et 2 assignés aux deux sous-ensembles sont conformes aux deux relations dans lesquelles ils sont définis : l'indice 1 sera pour des sous-ensembles qui contiennent un arc supplémentaire, l'indice 2 pour des sous-ensembles qui ne contiennent pas un arc donné.

Borne de $E_1^{(k)}$

Un arc (i_k, j_k) étant choisi et g^{k^1} étant la borne de l'ensemble $E^{(k^1)}$, dans la matrice des coûts correspondant au sous-ensemble $E^{(k^1)}$ qu'on note $d_{ij}^{(k^1)}$, on barre la ligne i_k et la colonne j_k et on interdit l'arc associé à l'arc (i_k, j_k) qui créerait un cycle de longueur inférieure ou égal à $n - 1$. (On verra plus tard que cet arc est unique).

On calcule alors les quantités

$\alpha_i^k = \min_j \left(d_{ij}^{(k^1)} \right)$, $\quad iEI =$ ensemble des lignes non barrées

et $\beta_j^k = \min_i \left(d_{ij}^{(k^1)} - \alpha_i^k \right)$ $\quad jE J =$ ensemble des colonnes non barrées

d'où on déduit la nouvelle borne g_1^k de $E_1^{(k)}$

$$g_1 = g^{k^1} + \sum_{iEI} \alpha_i^k + \sum_{jEJ} \beta_j^k$$

....

Pour montrer qu'il s'agit bien d'une minorante pour les circuits définis par l'ensemble $E^{(k)}$, on peut se reporter à la page 15 avec cette différence qu'à lieu que toutes les variables x_{ij} aient des valeurs indéfinies, il n'y en ait qu'une partie, c'est-à-dire qu'on a pour tout sous-ensemble E^k , l'équivalence suivante

Trouver $\boxed{\text{Min} \sum_{i,j \in I,J} d_{ij}^{(k^1)} \; x_{ij} + g^{k^1}}$

équivaut à trouver

$$\boxed{\text{Min} \sum_{\substack{i \in I - \{ik\} \\ j \in J - \{jk\}}} d_{ij}(k) \; x_{ij} + g^{k^1} + \sum_i \alpha_i^{(k)} + \sum_j \beta_j^{(k)}}$$

$$\text{avec } d_{ij}^{(k)} = d_{ij}^{(k^1)} - \alpha_i^{(k)} - \beta_j^{(k)}$$

Borne de $E_2^{(k)}$

Nous verrons dans cette section que le calcul de cette borne est lié au choix de la variable de séparation

$$x_{ik\ jk}$$

En effet, pour toute case admissible de la matrice $d_{ij}^{(k^1)}$ calculons les quantités suivantes.

Si x et y sont les coordonnées d'une case admissible quelconque, on pose

$$\gamma(x,y) = \min_{x,y/d_{xy}^{(k1)} = 0} \left(d_{1y}^{(k^1)} \right) + \min_{\substack{1 \neq x \\ 1 \neq y}} \left(d x_1^{(k^1)} \right)$$

Interprétation physique de la quantité $\gamma(x,y)$

$\gamma(x,y)$ est la quantité minimale qu'il faudra dépenser de toutes façons si on n'admet pas l'arc (x,y) ; si on ne prend pas l'arc (x,y) on prendra nécessairement un autre arc partant de x et qui dans le cas le plus favorable entraînera un coût égal à $\min_{1 \neq y} (d_{x1})$

De même si on ne vient pas en y en partant de x il faudra y venir d'un autre point et dans le meilleur des cas cela entraînera une augmentation du coût égal à $\min_{1 \neq x} (d_{1y})$.

soit $\gamma(i_k, j_k) = \max_{i,j} (\gamma(i,j))$

...

i_k et j_k sont les coordonnées d'une case admissible et représentent par conséquent un arc dont la défection serait plus coûteuse que n'importe quelle autre des cases admissibles à l'itération k .

On se sert de ce critère pour le choix de la variable qui définit la propriété séparatrice.

On choisira pour élément de séparation un arc (i_k , j_k) tel que

$$\gamma(i_k , j_k) = \max_{\substack{i \in I \\ j \in J}} (\gamma(i , j))$$

On établit aussi le calcul de la borne de $E_2^{(k)}$ de la façon suivante

$$g_2^k = g^{k^1} + \gamma(i_k , j_k)$$

Démontrons que cette borne g_2^k réalise bien une minorante pour le sous-ensemble $E_2^{(k)}$

On cherchait au niveau du sous-ensemble $E^{(k^1)}$

$$\min \sum_{\substack{i \in I \\ j \in J}} x_{ij} \, d_{ij}^{(k^1)} + g^{k^1}$$

où I = ensemble des lignes non barrées

J = ensemble des colonnes non barrées

si $\gamma(i_k , j_k)$ est l'arc déterminant la séparation, la quantité

$\gamma(i_k , j_k)$ peut être écrite sous la forme

$$\gamma(i_k , j_k) = \gamma_1 + \gamma_2$$

où $\gamma_1 = \min_{j \in J} (d_{i_k j}^{(k^1)})$

et $\gamma_2 = \min_{i \in I} (d_{i j_k}^{(k^1)})$

....

si on pose $d_{ij}^{(k)}{}_{k} = d_{i_{kj}}^{(k^1)} - \gamma_1 \quad \forall\, j \in J - \{jk\}$

$$d_{ijk}^{(k)} = d_{ijk}^{(k^1)} - \gamma_2 \quad \forall\, i \in I - \{ik\}$$

$$d_{jk\ i_k}^{(k)} = \infty$$

$$d_{ij}^{(k)} = d_{ij}^{(k^1)} \quad \text{pour les autres éléments}$$

la relation (1) équivaut à

$$\text{Min} \sum_{\substack{i \in I \\ j \in J}} x_{ij}\ d_{ij}^{(k)} + g^{k^1} + \gamma_1 + \gamma_2$$

qui équivaut à écrire

$$\text{Min} \sum_{\substack{i \in I \\ j \in J}} x_{ij}\ d_{ij}^{(k)} + g_2^{(k)} \qquad \text{avec } x_{ik\ jk} = 0$$

On a ainsi mis en place des relations de récurence qui donnent une borne minorante de plus en plus grande, car $\gamma(i_k, j_k) \geqslant 0$ et $\sum_{i \in I} \alpha_i^{(k)} + \sum_j \beta_j^{(k)} \geqslant 0$

Ce procédé converge nécessairement au bout d'un nombre fini d'itérations; le problème est de savoir si on obtient bien l'optimum de la fonction objectif quand on a barré toutes les lignes et toutes les colonnes de $d_{ij}^{(k)}$ ce qui équivaut à dire qu'on a satisfait toutes les contraintes unilatérales du Primal.

Problème de l'optimum

Supposons qu'on soit arrivé au problème suivant au bout d'un nombre fini d'itérations :

$$\underline{\text{Min}} \sum_{\substack{i = i_n \\ j = j_n}} x_{ij}\ d_{ij}^{(f-1)} + g^{f-1}$$

$$= \text{Min}\ x_{in\ jn} \cdot d_{in\ jn}^{(f-1)} + g^{f-1}$$

or, comme la dernière case de la matrice doit être une case admissible on a $d_{i_n\ j_n}^{(f-1)} = 0$. Pour le choix de la variable séparatrice il ne reste que $x_{in\ jn}$ d'où $x_{in\ jn} = 1$.

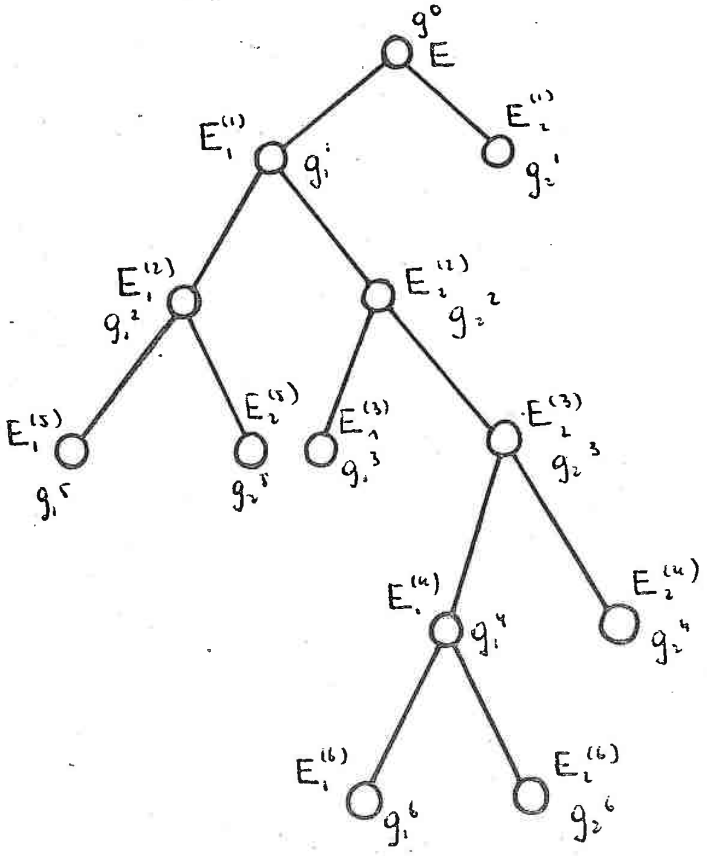
....

et on a bien

$$\text{Min} \sum_{i,j} d_{ij}^{(0)} \; x_{ij} = g^{f-1} = g^{f}$$

(g^f : borne finale).

On a ainsi défini tous les éléments pour calculer la solution par une méthode Branch and Bound : la technique de séparation et les méthodes de calcul de bornes minorantes. Pour se faciliter la représentation des différents sous-ensembles on a associé à la méthode une arborescence qui a l'allure suivante



C'est un arbre binaire ou chaque sommet représente un sous-ensemble muni d'une borne et chaque arc un arc du réseau employé en tant que séparateur.

....

Développons cet algorithme sur un petit exemple; supposons qu'on ait un graphe avec $n = 6$ sommets et dont la matrice des coûts est la suivante :

$d_{ij}^{(o)}$ =

	1	2	3	4	5	6	$\alpha_i^{(o)}$
1	∞	5	9	6	3	5	3
2	8	∞	9	8	5	9	5
3	6	9	∞	2	6	7	2
4	7	11	4	∞	4	2	2
5	4	6	3	2	∞	7	2
6	5	2	2	8	4	∞	2
$\beta_j^{(o)}$	2	0	0	0	0	0	

d'où la matrice $d_{ij}^{(1)} = d_{ij}^{(o)} - \alpha_i^{(o)} - \beta_j^{(o)}$ pour tout i,j

	1	2	3	4	5	6	$\alpha_i^{(1)}$
1	∞	2	6	3	0	2	0
2	1	∞	4	3	0	4	0
3	2	7	∞	0	4	5	0
4	3	9	2	∞	2	0	
5	0	4	1	0	∞	5	0
6	1	0	0	6	2	∞	0
$\beta_j^{(1)}$	0	0	0	0	0		

Itération 1

On calcule pour chaque case admissible la quantité

Par exemple pour la case (4 6)

$$\gamma(4,6) = \min_{\substack{j\neq 4 \\ j\neq 6}} \left(d_{4j}^{(1)} \right) + \min_{\substack{i\neq 6 \\ i\neq 4}} \left(d_{i6}^{(1)} \right)$$

....

d'où $\gamma(4,6) = 2 + 2 = 4$

On obtient de la même façon

$\gamma(1,5) = 2$
$\gamma(2,5) = 1$
$\gamma(3,4) = 2$
$\gamma(5,1) = 1$
$\gamma(5,4) = 0$
$\gamma(6,2) = 2$
$\gamma(6,3) = 1$

D'où $\underset{i,j \,/\, d_{ij}^{(1)} = 0}{\text{Max}} (\gamma(i,j)) = \gamma(4,6) = 4$

On choisit donc l'arc (4,6) pour définir la propriété séparatrice.

Auparavant on aura pu calculer la borne g_o de l'ensemble de tous les circuits hamiltoniens.

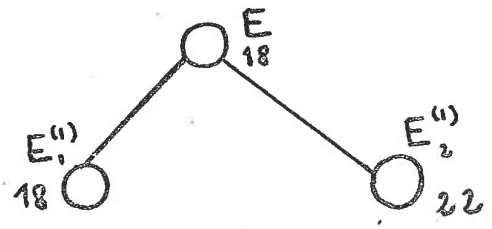
$$g_o = \sum_i \alpha_i^o + \sum_j \beta_j^{(o)} = 16 + 2 = 18$$

calculs des bornes de $E_1^{(1)}$ et $E_2^{(1)}$

$$g_o = g_o + \sum_{iEI} \alpha_i^{(1)} + \sum_{jEJ} \beta_j^{(1)} = g_o + o + o = 18$$

$$g_2^{(1)} = g_o + \text{Max}(\gamma(i,j)) = 18 + 4 = 22$$

Le début de l'arbosrescence peut donc se représenter de la façon suivante



Le sous-ensemble pendant ayant la borne minimale est le sous-ensemble $E_1^{(1)}$; on va donc étudier les ensembles de circuits hamiltoniens qui contiennent l'arc (4,6).

Il n'est pas apparu de nouvelles cases admissibles au cours de l'itération 1 puisque $\alpha_i^{(1)} = o \;\forall\, i \in I$ et $\beta_j^{(1)} = o \;\forall\, j \in J$;

....

Cependant, il faut recalculer toutes les quantités $(i, j) / d_{ij}^{(1)} = 0$, car il est possible qu'en barrant la ligne (4) et la colonne (6) on ait supprimé des éléments qui sont les minimums sur la ligne 4 ou la colonne 6.

Itération 2

Après calcul on a

$\gamma(1,5) = 2$ qui est maximum

	1	2	3	4	5	6	$\alpha_i^{(2)}$
1							
2	1	∞	4	3			1
3	2	7	∞	0			0
4							
5	∞	4	1	0			0
6	1	0	0	∞			0
$\beta_j^{(2)}$							

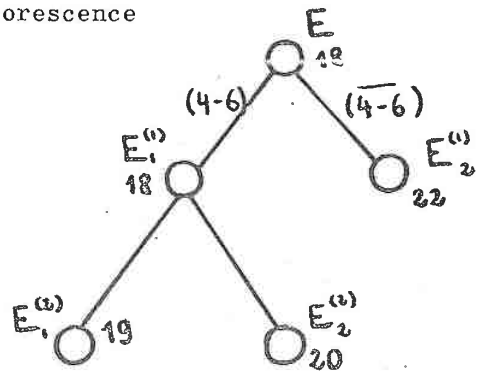
$$g_1^{(2)} = g_1^{(1)} + \sum_{i \in I} \alpha_i^{(2)} + \sum_{i \in J} \beta_j^{(2)} \qquad I = \{2, 3, 5, 6\} \quad J = \{1, 2, 3, 4\}$$

$$= 18 + 1 = 19$$

$$g_2^{(2)} = g_1^{(1)} + \gamma(1,5) = 20$$

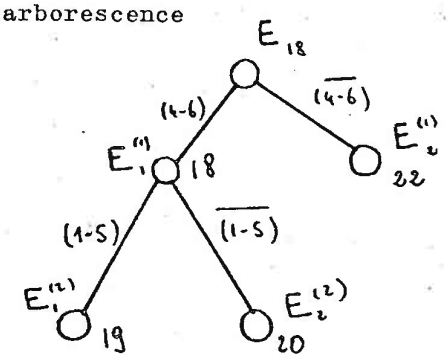
Au cours de cette itération il faut de plus interdire l'arc (2–4) car $\begin{Bmatrix} 4 & 6 & 2 \\ 6 & 2 & 4 \end{Bmatrix}$ formerait un cycle de longueur $3 < 6$.

Etat de l'arborescence



....

Etat de l'arborescence



Itération 3

							$\alpha_i^{(3)}$
0		3	2				0
2		∞	0				0
∞		1	0				0
$\beta_j^{(3)}$ 0		1	0				

Max ($\gamma(i,j)$) = $\gamma(6,2)$ = 4

$i,j/d_{ij}^{(3)}$ = 0

$g_2^{(3)} = g_1^{(2)} + \gamma(6,2) = 23$

$g_1^{(3)} = g_1^{(2)} + \beta_3^{(3)} = 19 + 1 = 20$

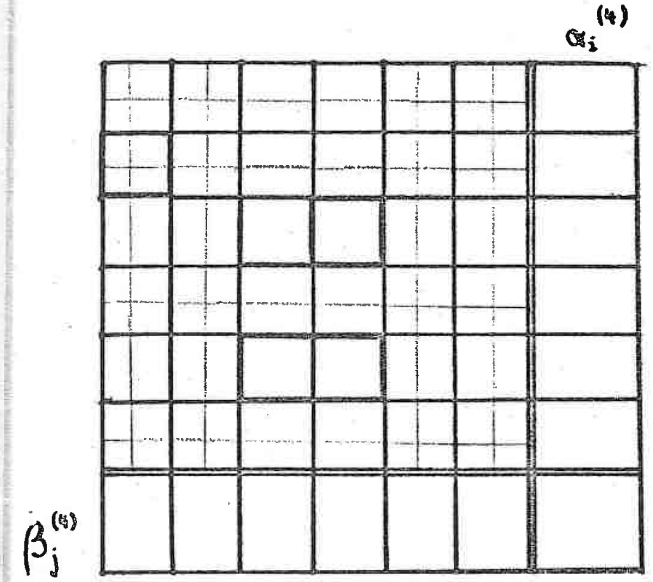
Au cours de cette itération il faut de plus interdire l'arc (2-4)

car (4 6 2) formerait un cycle de longueur 3 < 6
(6 2 4)

....

Le sous-ensemble ayant la borne minimale est $E_1^{(3)}$ mais il n'est pas unique; en effet $E_2^{(2)}$ a la même borne minorante; cependant on choisira $E_1^{(3)}$ pour sous-ensemble devant subir la séparation suivante, car c'est lui qui contient le plus grand nombre d'arcs déjà admis et qui est par conséquent de cardinal inférieur à celui de $E_2^{(2)}$.

Itération 4



L'arc choisi pour la séparation sera l'arc (2,1)

car $\underset{i,j/\, d_{ij}^{(4)} = 0}{\text{Max}} (\gamma(i,j)) = \gamma(2,1) = 4$

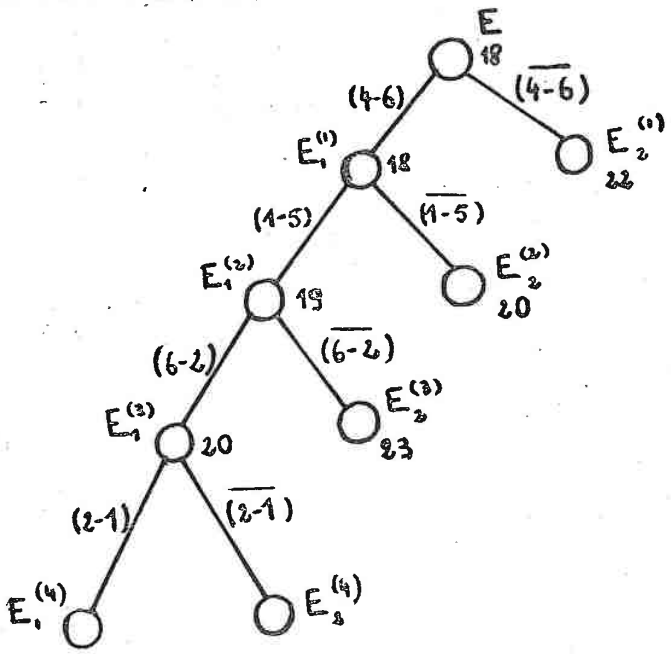
Calcul des bornes

$g_1^{(4)} = g_1^{(3)} = 20$ car $\alpha_i = 0 \quad \forall\, i \in I$

$g_2^{(4)} = g_1^{(3)} + \gamma(2,1) = 24$

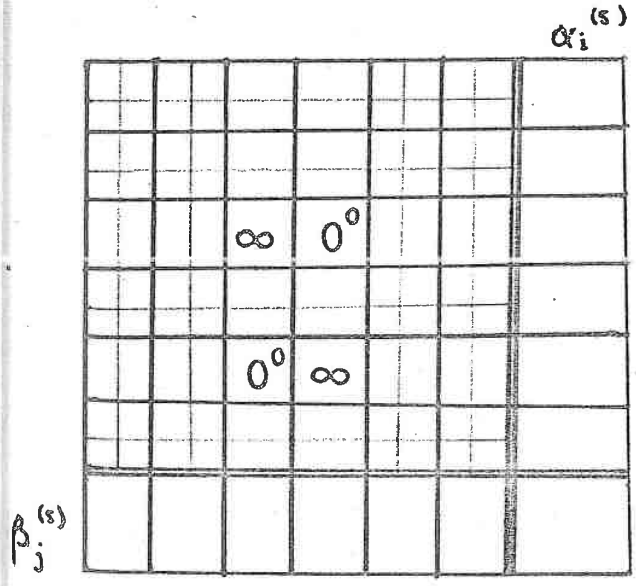
....

d'où état de l'arbre



De plus, au cours de cette itération, il faut interdire l'arc (5–4) (en posant $d_{54}^{(4)} = \infty$) car $\begin{Bmatrix} 4 & 6 & 2 & 1 & 5 \\ 6 & 2 & 1 & 5 & 4 \end{Bmatrix}$ formerait un circuit de longueur 5 qui est inférieure à la dimension du graphe.

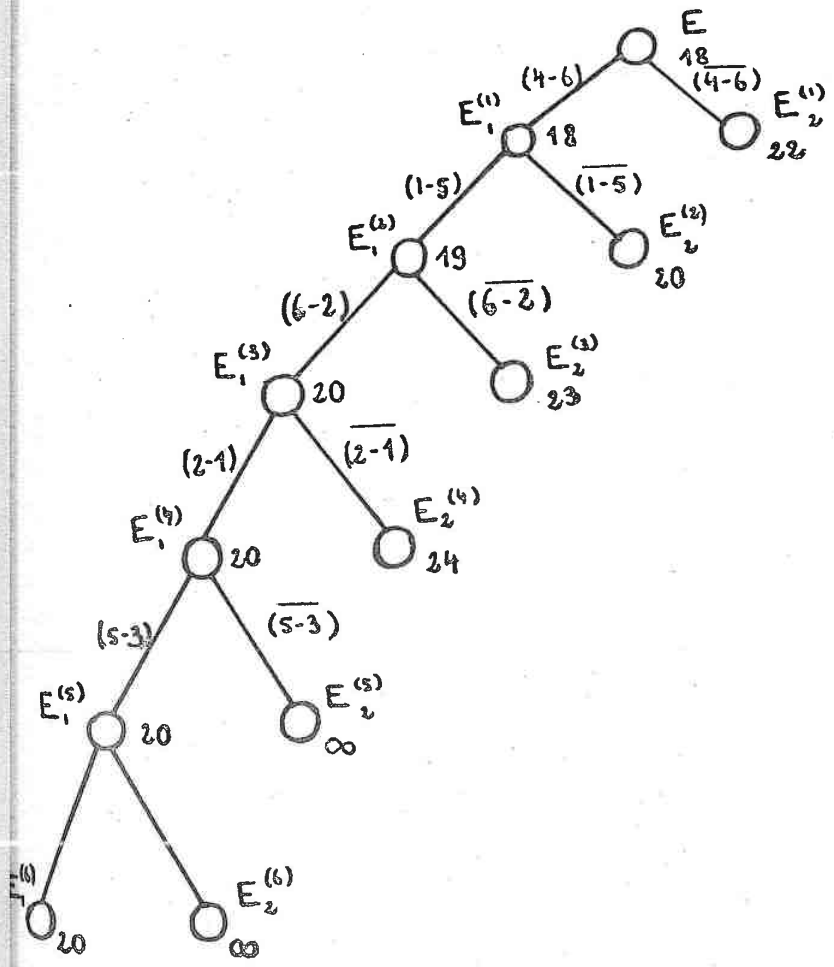
Itération 5



....

On choisit donc l'un ou l'autre des arcs si on applique l'algorithme de façon rigoureuse; mais en fait on peut admettre les deux puisqu'en les admettant aucun des α_i ou β_j encore incrémentables n'augmente et qu'en plus si on admet (3,4) il ne reste plus comme seule case admissible la case (5,3).

Nous voyons donc qu'on est arrivé sur une solution optimale qui correspond au circuit (4 6 2 1 5 3 4) et qui nécessite un coût g = 20. Il est à remarquer que sur cet exemple simple l'algorithme converge vers une solution optimale avec un nombre d'itérations minimal, c'est-à-dire un nombre d'itérations égal à la dimension du graphe (dans la mesure où la dernière admission est considérée comme une véritable itération); c'est évidemment un cas particulier favorable parce qu'à chaque itération k on a $k^1 = k - 1$; autrement dit on progresse toujours dans la même direction dans l'arborescence comme le montre le graphe ci-dessous qui représente l'arborescence quand on est à un optimum.



....

En fait, dans le cas général il est toujours possible qu'après une itération k on soit obligé de revenir sur un bout pendant déterminé à une itération k - p (p entier $>$ 1). Il faut remarquer en outre qu'après k itérations on a nécessairement k bouts pendants; si on se contente donc de l'algorithme tel qu'on l'a exposé, il faut donc avoir au minimum pour un réseau de n vill n matrices (n . n) à disposition.

Cette remarque nous suggère deux problèmes : pour un réseau où n $>$ 20 le calcul à la main peut être extrêmement long et on songe à une application sur calculateur électronique; mais pour cette application se pose alors le problèm de la conservation à tout moment des matrices correspondant à tous les bouts pendants.

Ces deux problèmes feront l'objet de l'étude qui suit.

....

Application sur calculateur électronique de l'algorithme de LITTLE

Comme nous l'avons énoncé plus haut le grand problème qui se posera à nous sera celui du stockage de toutes les matrices $d_{ij}^{(k)}$ correspondant à des sous-ensembles pendants de l'arborescence.

La solution adoptée n'est pas à proprement parler une méthode de stockage : c'est un processus qui permet de retrouver une matrice correspondant à un sous-ensemble pendant quelconque en ne conservant en mémoire qu'une seule matrice que nous appellerons la matrice"actuelle" qui sera celle correspondant à l'itération en cours.

Ce processus est construit à partir d'un certain nombre de renseignements découlant de l'exécution de chaque itération qui permettent de passer d'une matrice correspondant à un sommet (c. à d. un sous-ensemble quelconque) à la matrice correspondant au sommet dont il est le fils.

Dans ce qui suit on essaye de montrer que ces renseignements, ces informations peuvent être rendus minimums grâce à quelques considérations d'ordre mathématique qui découlent de l'algorithme.

La structure de ces renseignements permettra une opération inverse de celle qui nous fait passer d'une matrice $d_{ij}^{(k^1)}$ à la matrice $d_{ij}^{(k)}$, $d_{ij}^{(k)}$ et $d_{ij}^{(k^1)}$ étant les notations du chapitre précédent.

Quelques démonstrations

Si a une itération donnée on désigne par I l'ensemble des indices i qui correspondent à des lignes non barrées et par J l'ensemble des indices j qui correspondent à des colonnes non barrées de la matrice $d_{ij}^{(k)}$;

si $\gamma(i,j)$ désigne la quantité décrite dans le chapitre précédent

$i,j \;/\; d_{ij}^{(k)} = 0$

alors si $\underset{i,j}{\text{Max}} \; (\gamma(i,j))$ est différent de 0 on a les résultats suivants :

....

$$\text{Max}_{i,j/d_{ij}^{(k)}=0} (\gamma(i,j)) = \gamma(i_m, j_m) \neq 0$$

soit $\alpha_i = 0 \quad \forall\, i \in I - \{i_m, j_m\}$

et $\alpha_{jd} \geqslant 0$

$\beta_j \geqslant 0 \quad \forall\, j \in J$

soit $\beta_j = 0 \quad \forall\, j \in J - \{j_m, i_m\}$

et $\beta_{id} \geqslant 0$

$\alpha_i \geqslant 0 \quad \forall\, i \in I$

ou l'arc (i_d, j_d) est l'arc interdit correspondant à (i_m, j_m)

Autrement dit on affirme que, après avoir barré la ligne i_m et la colonne j_m au cours d'une itération k, ce sont soit des lignes qui devront être transformées soit des colonnes (selon que ce soit les β_j qui sont tous nuls ou les α_i) mais jamais les deux types de rangées au cours d'une même itération sauf éventuellement pour la ligne j_m et la colonne i_m si elles ne sont pas encore barrées.

Preuve

Rappelons que $d_{ij}^{(k)}$ est une notation pour la matrice des coûts après k itérations et que

$$\gamma(i,j) = \min_{1 \in I - \{i\}} (d_{1j}^{(k)}) + \min_{1 \in J - \{j\}} (d_{i1}^{(k)})$$
$$i,j\,/\,d_{ij}^{(k)} = 0$$

si $\text{Max}_{i,j} (\gamma(i,j)) = \gamma(i_m, j_m)$

$\gamma(i_m, j_m) \neq 0$ alors

soit $\min_{1 \neq i_m} (d_{1\,j_m}^{(k)}) \neq 0$ et $\min_{1 \neq j_m} (d_{i_m 1}^{(k)}) = 0$ (1)

soit $\min_{1 \neq i_m} (d_{1\,j_m}^{(k)}) = 0$ et $\min_{1 \neq j_m} (d_{i_m 1}^{(k)}) \neq 0$ (2)

soit $\min_{1 \neq i_m} (d_{1\,j_m}^{(k)}) \neq 0$ et $\min_{1 \neq j_m} (i_{m1}^{(k)}) \neq 0$ (3)

....

Dans les trois cas envisagés, si (i_d, j_d) est l'arc interdit qui créerait un cycle de longueur inférieure à n, si $d^{(k)}_{i_d j_d} = 0$

et si on pose $d^{(k+1)}_{i_d j_d} = \infty$ pour interdire (i_d, j_d)

si $\min_l (d^{(k)}_{i_d l}) \neq 0$ alors $\alpha_{i_d} = \min_{l \neq j_m} (d^{(k)}_{i_d l}) > 0$

si $\min_l (d^{(k)}_{l j_d}) \neq 0$ alors $\beta_{j_d} = \min_{l \neq j_m} (d^{(k)}_{l j_d}) > 0$

La ligne i_d et la colonne j_d ont donc des propriétés particulières; le cas le plus simple qui puisse se présenter est celui où on interdit un cycle de longueur 2 c'est-à-dire où $i_d = j_m$

et $j_d = i_m$

Prenons la démonstration

dans le cas (3): il est évident qu'on ne supprimera aucun zéro; c'est-à-dire aucune case admissible, de la matrice $d^{(k)}_{ij}$ en barrant la ligne i_m et la colonne j_m par conséquent on aura

$$\min_i (d^{(k)}_{ij}) = \underline{\alpha_i^{(k)} = 0} \qquad \forall\, i \in I - \{ i_m, i_d \}$$

$$\min_j (d^{(k)}_{ij}) = \underline{\beta_j^{(k)} = 0} \qquad \forall\, j \in J - \{ j_m, j_d \}$$

dans le cas (2): on ne supprimera pas de case admissible en barrant la ligne i_m puisque la seule de la ligne est la case (i_m, j_m) et que le minimum sur la ligne i_m est supposé différent de zéro.

Donc $\underline{\beta_j = 0 \quad \forall\, j \in J - \{ j_d, j_m \}}$.

par contre le minimum sur la colonne j_m est nul; donc en barrant la colonne j_m on risque de faire apparaître une ou plusieurs lignes telles que

$$\min (d_i^{(k)}) > 0$$

d'où $\underline{\alpha_i \geq 0 \qquad \forall\, i \in I}$

....

Dans le cas (1) la démonstration est symétrique et on aura

$$\alpha_i = 0 \qquad \forall\, i \in I - \{ i_d, i_m \}$$

$$\beta_j \geqslant 0 \qquad \forall\, j \in I$$

Intérêt de la proposition : il existe un nombre maximum de rangées à modifier quand on passe d'une matrice $d_{ij}^{(k)}$ à $d_{ij}^{(k+1)}$;

le nombre maximum est égal à $n - 1$.

2ème proposition :

On veut montrer que si $\text{Max}\ (\gamma(i, j)) = 0$ alors deux

$$i,j / d_{ij}^{(k)} = 0$$

rangées au plus sont susceptibles d'être modifiées dans la matrice $d_{ij}^{(k)}$ pour obtenir $d_{ij}^{(k+1)}$

Autrement dit

si $\text{Max}\ (\gamma(i, j)) = \gamma(i_{m1}, j_{m1}) = \gamma(im_2, jm_2) = \ldots \gamma(im_p, jm_p) = 0$
$i, j / d_{ij}^{(k)} = 0$

alors

$$\alpha_i = 0 \quad \forall\, i \in I - \{ i_{d1} \} \text{ ou } I - \{ i_{d2} \} \text{ ou } \ldots \quad I - \{ i_{dp} \}$$

$$\beta_j = 0 \quad \forall\, j \in J - \{ j_{d1} \} \text{ ou } J - \{ j_{d2} \} \ldots \text{ ou } J - \{ j_{dp} \}$$

où les couples $\{ i_{dl}, j_{dl} \}$ représentent les arcs associés aux arcs $\{i_{m1}, j_{m1}\}$ .... $\{im_p, jm_p\}$ et qui sont interdits pour les itérations ultérieures.

Preuve

Raisonnons par l'absurde.

Supposons qu'on ait choisi parmi les p couples (i,j) le couple (i_{m1}, j_{m1}) pour déterminer la séparation.

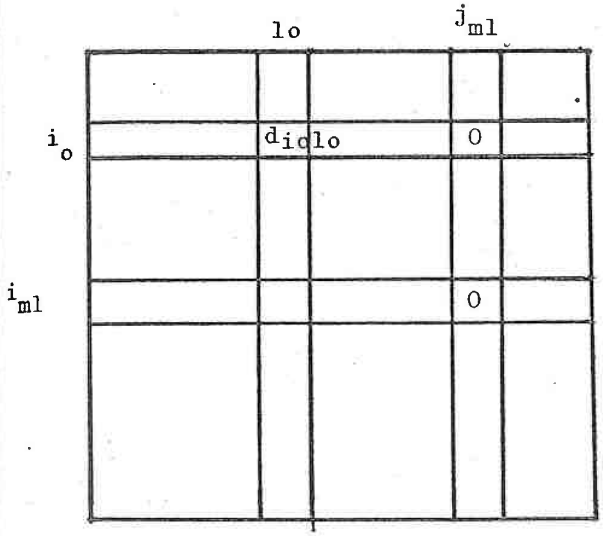
Supposons qu'il existe une rangée de la matrice $d_{ij}^{(k)}$ telle que après avoir barré la ligne i_{m1} et la colonne j_{m1} le minimum sur cette rangée soit différent de zéro.

....

Remarque : Cette rangée ne sera pas la ligne i_{dl} ou la colonne j_{dl} où il est toujours possible si $d^{(k)}_{i_{dl} j_{dl}} = 0$ que $\alpha_{i_{dl}} > 0$ et $\beta_{j_{dl}} > 0$

Appelons i_o la rangée (on fait la même démonstration si c'est une ligne ou une colonne) telle que

$$\min_{l \in J - \{j_d, j_m\}} (d^{(k)}_{i_o l}) > 0 \quad \text{avec} \quad \min (d^{(k)}_{i_{ol}}) = d^{(k)}_{i_o l_o}$$



mais dans ce cas on aurait eu

$$\gamma(i_o, j_{ml}) = d^{(k)}_{i_o l_o} > 0$$

ce qui est contraire à l'hypothèse $\max_{i, j / d_i j^{(k)} = 0} (\gamma(i, j)) = 0$

Cette proposition complète la première où on avait étudié le cas $\max (\gamma(i, j)) > 0$.

Dans les deux cas les seules rangées qui peuvent toujours être modifiées sont la ligne i_d et la colonne j_d

....

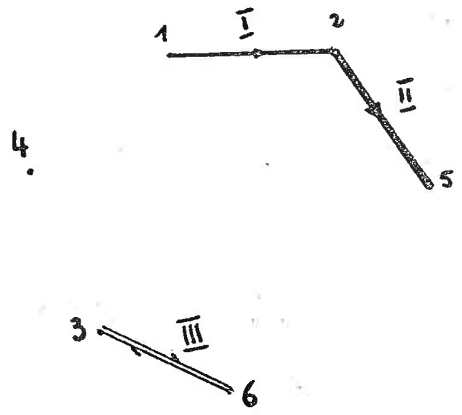
Etude des cycles de longueur inférieure à n qui apparaissent après chaque itération.

Dans le paragraphe précédent nous avons déjà utilisé la propriété principale qui est la suivante :

Règle : à chaque itération où on admet un arc supplémentaire pour définir un sous-ensemble $E_1^{(k)}$ il faut et il suffit d'interdire un et un seul arc pour empêcher la formation de cycles de longueur inférieure à n

Pour la démonstration il faut envisager quatre cas :

. au cours d'itérations précédentes ni l'origine ni l'extrémité de l'arc admis à l'itération k n'ont été acceptés



sur le graphe ci-dessus : on suppose que l'arc (1-2) a été admis à l'itération (I) et que l'arc (2-5) a été admis à l'itération (II) et que l'itération "actuelle" est l'itération (III) où on a admis l'arc (6-3).

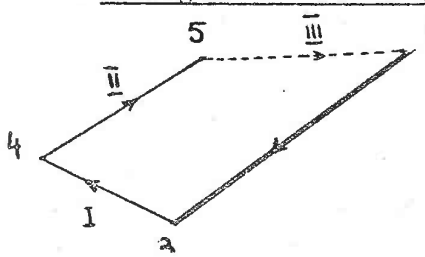
Ce qu'il est important de remarquer, pour la suite comme pour ce cas-là, c'est qu'on ne s'occupe pas des cycles interdits qui se sont révélés à des étapes antérieures; ce qui nous intéresse c'est celui associé à l'itération en cours.

Dans le cas étudié le seul circuit de longueur inférieure à 7 est le circuit (6-3) (3-6) de longueur 2 . Il faut donc interdire l'arc (3-6) pour tous les sous-ensembles inclus dans le sous-ensemble $E_1^{(III)}$ en posant

$$d_{36}^{(III)} = \infty$$

.. envisageons le deuxième cas : l'arc qu'on vient d'admettre dans le sous-ensemble $E_1^{(k)}$ a l'une ou l'autre de ses extrémités qui figure dans un arc admis dans l'un des sous-ensembles qui contient $E_1^{(k)}$, à une itération précédente.

. l'origine de l'arc figure déjà

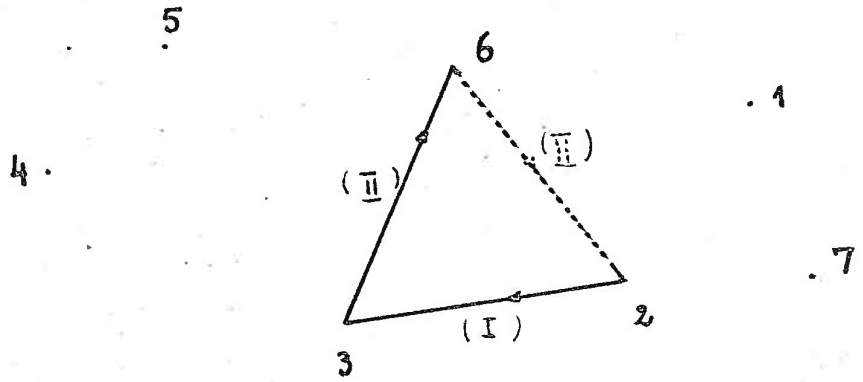


....

On a admis (3-4) à l'itération (I), (4-5) à l'itération (II) et (5-6) à l'itération actuelle (III); dans ce cas il faut interdire l'arc (6-3) car (3 4 5 6 3) formerait un cycle de longueur 4 inférieure à la dimension du graphe.

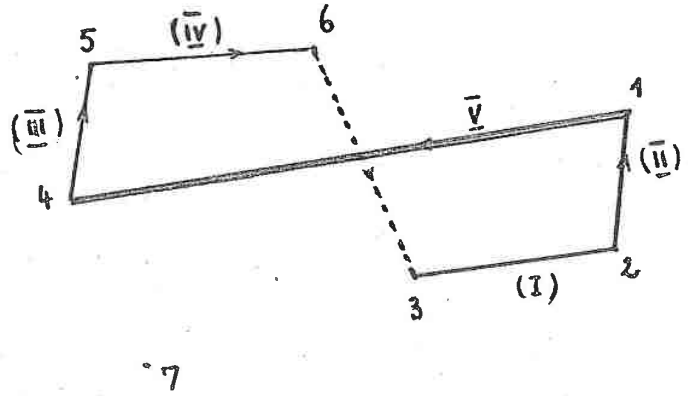
Il est tout à fait inutile d'interdir l'arc (6-4) ou (6-5), car les colonnes 4 et 5 sont barrées au cours des itérations précédentes.

. l'extrémité de l'arc figure déjà



le cycle (6-2) (2-3) (3-6) est un cycle de longueur 3 donc interdit.

. . . le dernier cas est celui où dans l'itération k on admet un arc dont à la fois l'origine et l'extrémité sont représentées dans des arcs admis à des itérations antérieures et qui définissent des sous-ensembles contenant le sous-ensemble $E_1^{(k)}$



Les arcs (3-2), (2-1), (4-5), (5-6) ont été admis au cours d'itérations précédentes; si on admet l'arc actuel (6-3) il faut interdire l'arc (1-4) car (4 5 6 3 2 1 4) est un cycle de longueur 6 , longueur qui est inférieure à la dimension du graphe.

....

Remarque : Chacun de ces cas se produit à l'exclusion des autres dans la mesure par exemple où dans le cas 4 il est inutile d'interdire l'arc (3-6) étant donné que la colonne 6 et la ligne 3 sont barrées au cours d'itérations précédentes.

On a ainsi défini les coordonnées de l'arc qu'on avait noté (i_d , j_d) dans les paragraphes précédents.

Nous verrons comment s'effectue la recherche de l'arc (i_d , j_d) dans la partie informatique; le principe est de grouper les arcs admis à mesure qu'ils apparaissent sous forme de portions de circuits orientés; il est alors facile de déterminer l'arc (i_d , j_d).

Nous avons ainsi défini tous les éléments nécessaires à la mise en application sur ordinateur de l'algorithme.

Dans un chapitre précédent nous avions noté qu'après k itérations il fallait conserver k matrices de coûts correspondant aux k bouts pendants. Il faut noter en plus que le nombre d'itérations, pour arriver sur une solution optimale, évolue d'une façon qui n'est pas du tout linéaire avec la dimension du graphe.

Ainsi si on a $n = 20$ et que le nombre d'itérations est de 200, il faudrait conserver en mémoire 200 matrices (20,20) c'est-à-dire 80 000 places mémoires.

Un des buts essentiels de l'application sur ordinateur est de créer une structure permettant de ne conserver qu'une seule matrice des coûts en mémoire et de passer d'une matrice $d_{ij}^{(k)}$ à une matrice $d_{ij}^{(k1)}$.

Pour effectuer le passage d'une matrice actuelle $d_{ij}^{(k)}$ à une matrice $d_{ij}^{(k1)}$ correspondant à un bout pendant quelconque, il y a une opération de base qui est le passage de $d_{ij}^{(k)}$ à $d_{ij}^{père(k)}$

Passage de $d_{ij}^{(k)}$ à $d_{ij}^{père(k)}$

Informations nécessaires :

numéro du sommet d'origine et du sommet extrémité de l'arc ayant défini la séparation, origine, extrémité et valeur du coût de l'arc interdit associé, liste des $\alpha_i^{(k)} \neq 0$ et $\beta_j^{(k)} \neq 0$ qu'on a enlevé respectivement aux lignes i et colonnes j de la matrice $d_{ij}^{père(k)}$ pour obtenir $d_{ij}^{(k)}$.

Opérations à effectuer

ajouter aux lignes et aux colonnes de $d_{ij}^{(k)}$ les valeurs $\alpha_i^{(k)}$ et $\beta_j^{(k)}$ qu'on leur avait enlevées, remettre le coût de l'arc interdit à sa valeur initiale, faire en sorte que la ligne représentant l'origine de l'arc et la colonne représentant l'extrémité ne soient plus marquées.

....

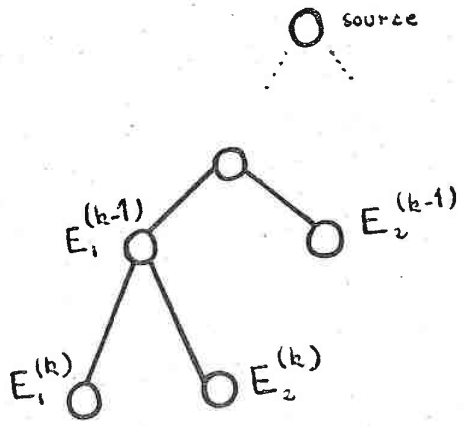
Remarque 1 : Cette opération peut s'effectuer dans les deux sens une fois que toutes les informations nécessaires ont été répertoriées.

Remarque 2 : Dans le cas où la matrice $d_{ij}^{(k)}$ est issue d'une transformation du type ' on n'accepte pas un arc' le processus est le même : si (i_m, j_m) est l'arc de la séparation en posant $d_{im\ jm} = \infty$ on détermine $\alpha_{im}^{(k)}$ et $\beta_{jm}^{(k)}$ à soustraire à la ligne i_m et à la colonne j_m .

Essai de formalisation de l'algorithme en langage algébrique

Pour cela on convient de noter $f\ (d_{ij}^{père(k)})$ l'opération qui associe à la matrice $d_{ij}^{père(k)}$ la matrice $d_{ij}^{(k)}$ et $f^{-1}\ (d_{ij}^{(k)})$ l'opération inverse qui associe à la matrice $d_{ij}^{(k)}$ la matrice $d_{ij}^{père(k)}$. Ces opérations et les paramètres qui leurs sont associés ont été définies ci-dessus.

1er cas : Le bout pendant minimal est un des deux sous-ensembles issus de l'itération précédente.



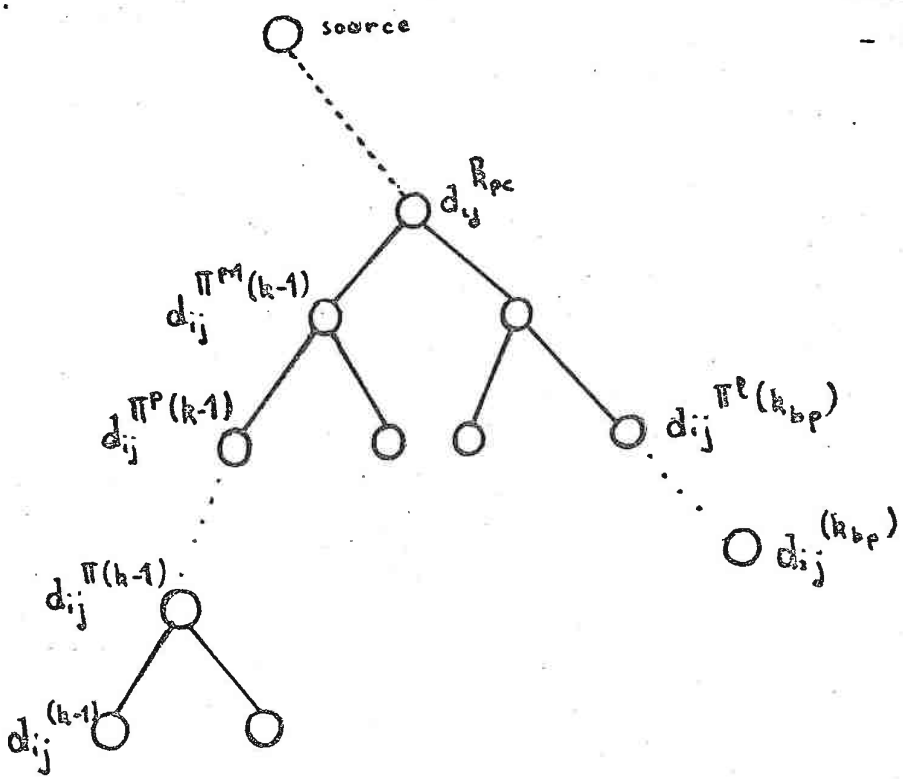
Dans ce cas il n'y a pas de problème il faut appliquer l'opérateur f à la matrice actuelle $d_{ij}^{(k-1)}$ pour obtenir $d_{ij}^{(k)}$

$$d_{ij}^{(k)} = f\ (d_{ij}^{(k-1)})$$

2ème cas : Le bout pendant minimal est un sous-ensemble pendant quelconque de l'arborescence.

Le problème est alors de reconstituer la matrice correspondant à ce bout pendant à partir de la matrice actuelle $d_{ij}^{(k-1)}$

....



La fonction Π qui apparaît sur le graphe est une fonction qui à chaque numéro d'itération associe le numéro de l'itération qui la précède sur la branche du graphe.

On définit une composition de cette application Π de la façon suivante :

$\Pi^n(k)$ donne le $n^{ième}$ prédecesseur de l'itération k

convention $\Pi^o(k) = k$

et N la fonction qui a tout sous-ensemble associe son niveau par rapport à la source, c'est-à-dire le nombre minimal de séparations qui permettent de passer de la source à ce sous-ensemble.

<u>1ère étape</u> : Déterminer la matrice correspondant au sous-ensemble

$E^{(kpc)}$ (pc point critique)

Elle doit vérifier les relations suivantes

Si on note f^n l'opérateur qui effectue $\underbrace{fo\ fo\ \ldots\ of}_{n\ fois}$

et on pose $m = N(k-1) - N(kpc)$

$$d_{ij}^{(k-1)} = f^m\left(d_{ij}^{(kpc)}\right) \qquad (1)$$

ceci veut dire qu'on applique m fois la transformation f à la matrice $d_{ij}^{(kpc)}$ pour obtenir $d_{ij}^{(k-1)}$ étant entendu que f dépend chaque fois de l'indice supérieur des d_{ij} qui est le numéro de l'itération à laquelle elle a été obtenue et que c'est une écriture formelle puisqu'on ne connaît pas effectivement $d_{ij}^{(kpc)}$

....

si on pose $m^1 = N(kbp) - N(kpc)$

$$d_{ij}^{(kbp)} = f^{m1} \left(d_{ij}^{(kpc)} \right) \qquad (2)$$

L'interprétation est la même que celle plus haut et de même que la formule (1) la formule (2) est une convention d'écriture.
Cependant en considérant qu'il existe l'opérateur f^{-1} on va pouvoir déduire la matrice $d_{ij}^{(kbp)}$ à partir de la seule matrice $d_{ij}^{(k-1)}$

(1) $$f^{-1} \left(d_{ij}^{(k-1)} \right) = f^{-1} \circ f \circ f^{m-1} \left(d_{ij}^{(kpc)} \right)$$

$$f^{-1} \left(d_{ij}^{(k-1)} \right) = f^{m-1} \left(d_{ij}^{(kpc)} \right)$$

en itérant m fois cette opération on obtient

$$d_{ij}^{(k_{pc})} = f^{-m} \left(d_{ij}^{(k-1)} \right)$$

en remplaçant $d_{ij}^{(k_{pc})}$ dans (2) on aura

$$\boxed{d_{ij}^{(k_{bp})} = f^{m1} \left(f^{-m} \left(d_{ij}^{(k-1)} \right) \right)} \qquad (3)$$

Donc connaissant les paramètres associés à toutes les fonctions f aux itérations précédentes et la matrice des coûts de l'itération k - 1 on peut passer à l'itération k quand on connaît $d_{ij}^{(k_{bp})}$.

Détermination du point critique kpc

posons $r = m - m1 = (N(k-1) - N(kpc)) - (N(k_{bp}) - N(k_{pc}))$

$$= N(k-1) - N(k_{bp})$$

et représente la différence de niveau entre k-1 et k_{bp} dans l'arborescence représentée sur le graphe page 33.

si $r \leqslant 0$
on calcule $k_{bp1} = \prod^{|r|} (k_{bp})$
on a alors $N(k_{bp1}) = N(k-1)$
on cherche alors le plus petit l tel que $\prod^{|r|+1} (k_{bp}) = \prod^{l} (k-1)$
si l vérifie cette relation on a

$$k_{pc} = \prod^{|r|+1} (k_{bp}) = \prod^{l} (k-1)$$

....

si $r > 0$

on calcule $k^1 = \prod^r (k-1)$

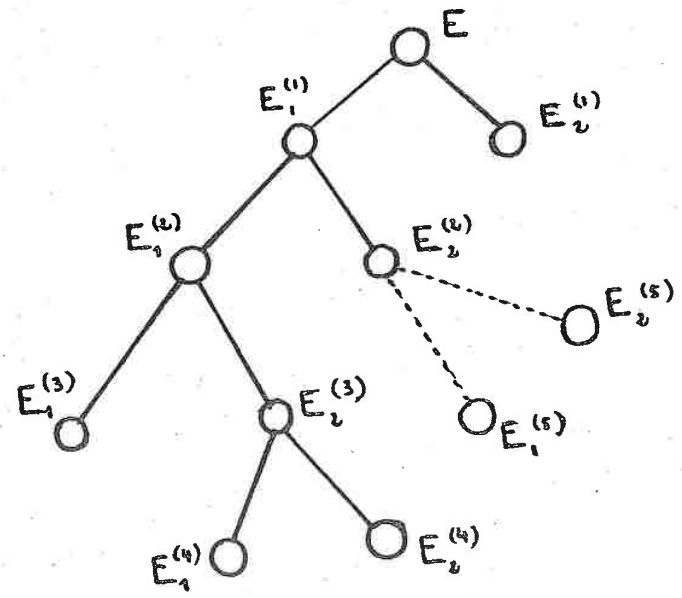
et on a alors k^1 tel que $N(k^1) = N(k_{bp})$

on cherche alors le plus petit entier p tel que $\prod^{r+p} (k-1) = \prod^{p} (k_{bp})$

si p vérifie cette relation on a

$$k_{pc} = \prod^{r+p} (k-1) = \prod^{p} (k_{bp})$$

Pour mieux saisir cette formalisation appliquons la sur un exemple



on suppose qu'on a $k-1 = 4$ et $k_{bp} = 2$

Recherche du point critique : $r = N(4) - N(2) = 2$

$\prod^r (4) = 2$

$\prod^{r+1} (4) = \prod^1 (2) = 1$

ensuite

$f^{-1} (d_{ij}^{(4)}) = d_{ij}^{(3)}$

$f^{-1} (d_{ij}^{(3)}) = d_{ij}^{(2)}$

$f^{-1} (d_{ij}^{(2)}) = d_{ij}^{(1)} = d_{ij}^{(k_{pc})}$

....

puis on applique la fonction f correspondant au sous-ensemble $E_2^{(2)}$

$$f_{(2,2)} \left(d_{ij}^{(1)} \right) = d_{ij}^{(2)}$$

et on peut alors effectuer l'itération $k = 5$

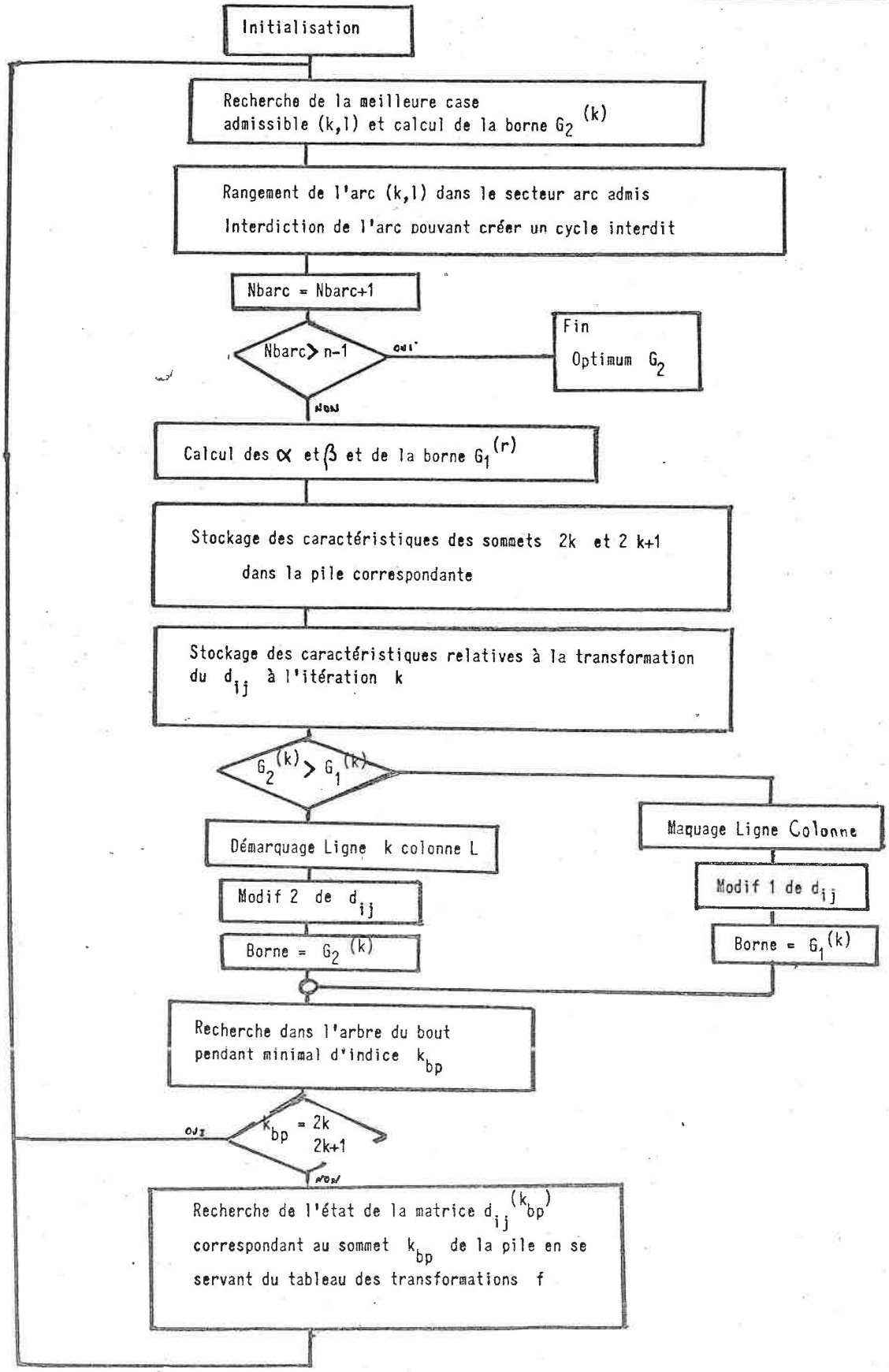
$$f \left(d_{ij}^{(2)} \right) = d_{ij}^{(5)}$$

On a ainsi défini tous les éléments nécessaires à la programmation ; on verra que les applications f seront des sous-programmes et leurs paramètres des tableaux dont l'accès se fera avec le numéro de l'itération et que les fonctions $\prod$ et N seront utilisées sous forme de pile qui sera décrite plus loin.

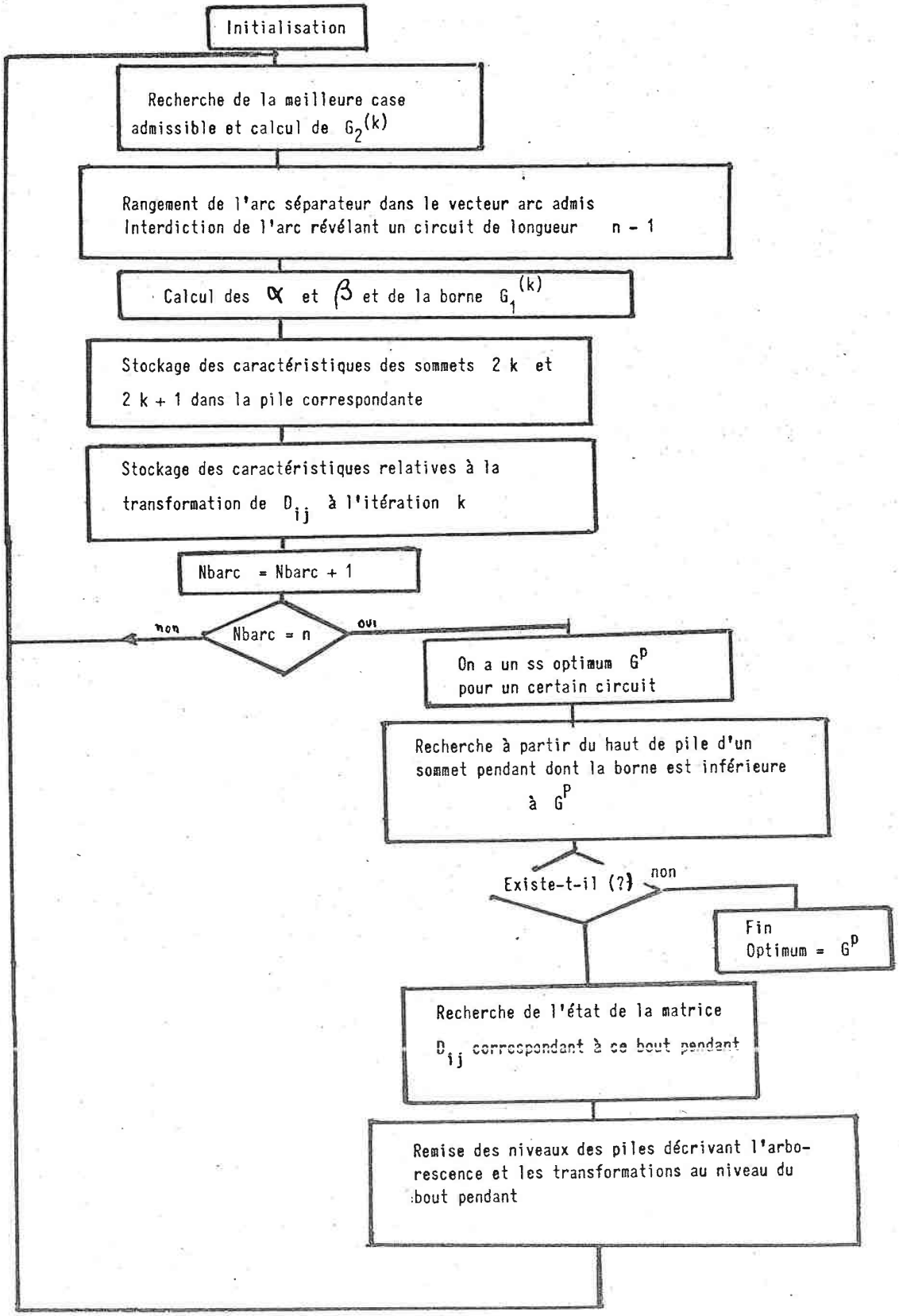
- - - - - - -

....

Organigramme général du programme de calcul dans le cas où on choisit la méthode de progression classique dans l'arborescence.



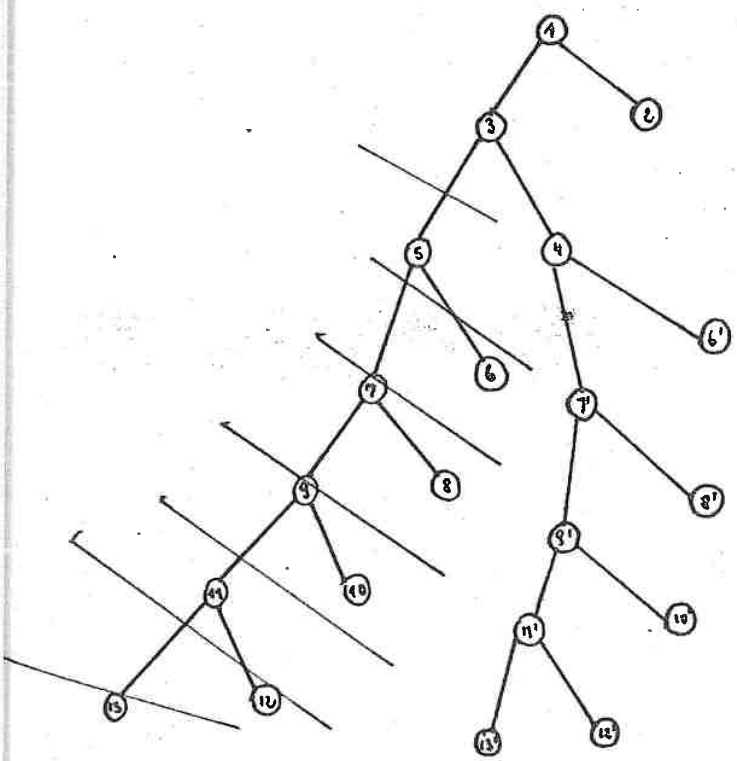
Organigramme général dans le cas où on choisit la deuxième méthode de progression dans l'arborescence



On utilise à quelques détails près les mêmes sous-procédures dans le premier algorithme et dans la deuxième. Elles seront étudiées de façon plus précise dans le chapitre "Programmation".

Evolution de l'arborescence

supposons un exemple avec n = 6



La zone hachurée est la partie de l'arborescence qui est devenue inutile : les bornes associées aux sommets 12, 10, 8 sont supérieures à celles de 13.

6 est le numéro du 1er sommet pendant dont la borne soit inférieure à celle du sommet 13 . A partir de 6 on redéveloppe l'algorithme.

On constate donc qu'on n'aura donc jamais qu'une branche principale dans l'arborescence.

....

Discussions, résultats, commentaires

Dans ce chapitre nous allons aborder les performances comparées des deux programmes, les améliorations qu'on peut apporter dans l'une ou l'autre phase pour accélérer la convergence et étudier le cas où les matrices des coûts sont symétriques.

Comparaison des deux algorithmes

1. Par rapport aux contraintes du matériel

Comme toujours il y a deux aspects : celui du temps de calcul et celui de l'occupation mémoire.

Pour le premier algorithme on ne contrôle pas l'occupation mémoire; l'extension de l'arborescence est une fonction qui croît très vite quand le nombre de points du réseau augmente et il faut conserver en mémoire tous les sommets et toutes les transformations associées.

En effet, lors d'un saut d'une branche de l'arbre à une autre il peut arriver que le premier prédecesseur commun aux deux sommets concernés par le saut soit la source.

Pour ce qui est du temps de calcul, il n'a pas pu être testé avec un exemple de grande dimension ($n > 40$) car la place mémoire limitée interdisait la convergence de l'algorithme. Si on avait choisi de conserver des parties de piles sur des mémoires auxiliaires à accès rapide, cela aurait considérablement augmenté encore ce temps du fait des nombreuses opérations de lecture écriture sur bande ou sur disque.

Pour ce qui est du deuxième algorithme on a l'avantage de connaître la place mémoire occupée; en effet, comme on l'a vu précédemment, l'arborescence se résume, tout au long du déroulement de l'algorithme, à une seule branche principale, les autres branches ayant toutes des longueurs égales à 1 et débouchant sur des sommets pendants.

La longueur de cette branche sera égale à $n + x$, si on appelle x le nombre de branches de l'arborescence qui représentent des propriétés "ne passe pas par" sur la branche principale. On sait que l'on ne prend en compte une telle propriété dans le deuxième algorithme que lorsqu'on exécute un saut du sommet extrémité de la branche principale vers le bout pendant, ayant une borne inférieure, le plus proche.

Ceci entraîne que la branche principale aura une longueur maximale à l'optimum et qu'elle n'excédera pas $n + x$

x étant le nombre de variables x_{ij} définissant le circuit il restera dans l'ordre de grandeur de n car la propriété "ne passe pas par" est moins significative pour la définition d'un circuit que la propriété "passse par" .

Pour ce qui est du temps d'exécution il semble, du moins pour les exemples qui ont convergé pour les deux algorithmes, qu'il soit sensiblement plus grand que pour la 1ère méthode de progression.

....

2. Par rapport aux utilisateurs éventuels des programmes

Si la dimension de la matrice des coûts n'excède pas 30 il semble préférable d'utiliser, quand on dispose d'un calculateur avec une mémoire centrale de 64 K mots, le premier programme car on est assuré que la mémoire centrale suffira pour que l'algorithme converge et que le temps d'exécution est plus petit que pour le deuxième programme.

Pour des problèmes pour lesquels la dimension est inférieure à 30 , dans les cas où on ne peut affirmer que pour un temps donné de calcul assez grand l'algorithme convergera, il est préférable de choisir le deuxième programme.

En effet, on peut imaginer que l'utilisateur ne veuille pas avoir l'optimum rigoureux, mais un circuit dont le coût est voisin de l'optimum et ceci avec un temps de calcul qu'il s'est fixé à l'avance; le deuxième programme fournit, après un temps qui doit tout de même être supérieur au moins au temps pour trouver le premier circuit sous-optimal (il est très faible), une borne majorante du coût optimal et un circuit complet associé. Il fournit aussi une borne minorante : parmi tous les bouts pendants à un moment donné du développement de l'algorithme il en existe un qui ait une borne minimale qui peut être considérée comme une minorante pour le coût de tout circuit.

A tout moment le deuxième algorithme nous permet donc de disposer d'un encadrement de la fonction coût.

Si j'appelle $g_M^{(p)}$ la borne majorante associée au dernier circuit sous optimal trouvé et $g_m^{(p)}$ la borne minorante décrite ci-dessus on aura don

$$g_m^{(p)} < C < g_M^{(p)}$$

on aura aussi les relations

$$g_M^{(p+1)} < g_M^{(p)}$$

$$g_m^{(p+1)} \geqslant g_m^{(p)}$$

Intérêt de cet encadrement

Il donne une estimation de l'erreur qu'on fait par rapport à la vraie valeur du coût optimal.

Ceci nous amène à penser que l'utilisateur peut vouloir un résultat non plus en se fixant comme limite un temps de calcul, mais en se donnant une marge d'erreur : on peut alors concevoir comme test d'arrêt du programme le suivant

appelons $e = \dfrac{g_M^{(p)} - g_m^{(p)}}{2\, g_M^{(p)}}$ l'erreur admissible

....

et supposons qu'on se soit fixé un sous-optimum à 5 %

si $e > 5\ \%$ on continue l'algorithme

si $e < 5\ \%$ on arrête le calcul et on imprime le dernier circuit sous-optimal trouvé avec sa borne.

Remarque :

$$\lim \; g_M^{(p)} - g_m^{(p)} = 0 \Longrightarrow C = G_M^{(p)} = g_m^{(p)}$$

Remarque concernant $g_m^{(p)}$

Le deuxième algorithme laisse $g_m^{(p)}$ invariant pour les premiers circuits sous-optimaux trouvés; ce n'est que lorsqu'on n'est proche de l'optimum que l'on a $g_m^{(p+1)} > g_m^{(p)}$.

Il serait par conséquent intéressant d'étudier une combinaison des deux algorithmes l'un tendant à trouver le meilleur majorant, l'autre le meilleur minorant. Ce programme n'a pas été écrit.

Discussion sur le mode de recherche de la meilleure case admissible

Le choix de la meilleure case admissible ne doit pas nécessairement se faire comme on le fait dans l'algorithme de LITTLE. En effet, au lieu de prendre celle qui correspond à $\text{Max}\ (\gamma(i, j))$ avec

$$i,j \,/ d_{ij}^{(k)} = 0$$

$$\gamma(i,j) = \min_{l \in J - \{j\}} (d_{il}^{(k)}) + \min_{l \in I - \{i\}} (d_{lj}^{(k)})$$

on pourrait songer à prendre celle qui correspond à

$$\text{Max}_{i,j \,/ d_{ij} = 0} \left(\gamma(i,j) - \sum_{l \in I - \{i\}} \alpha_l^{(k)} - \sum_{p \in J - \{j\}} \beta_p^{(k)} \right)$$

c'est-à-dire celle qui entraîne la différence la plus grande entre les bornes des ensembles $E_1^{(k)}$ et $E_2^{(k)}$.

Ce choix éviterait de définir beaucoup d'ensembles du type $E_2^{(k)}$ dans la mesure où les conditions suffisantes pour que l'on trouve une case admissible de ce type sont souvent vérifiées.

....

En effet on a l'équivalence

$$g_1^{(k)} \leqslant g_2^{(k)} \Longleftrightarrow e\ (i_m, j_m) = \text{Max}\ \left(\gamma(i, j) - \sum_{l \in I - \{i\}} \alpha_l^{(k)} - \sum \beta_p^{(k)} \right) \geqslant 0$$

Condition suffisante pour que $e(i_m, j_m) \geqslant 0$

Une condition suffisante pour que $e(i_m, j_m)$ soit positif ou nul est que

1) $\min_{l \in j_m} (d i_{ml}) = \alpha' i_m > 0$ et $\min_{l \neq i_m} (d j_{ml}) = \alpha_{jm} > 0$

et

2) $\min_{l \neq i_m} (d_{ljm}) = \beta'_{jm} > 0$ et $\min_{l \neq jm} (d_{li_m}) = \beta\, i_m > 0$

Preuve :

Supposons dans une première partie que $d(jm, im) \neq 0$; dans ce cas en barrant la ligne i_m et la colonne j_m de la matrice d_{ij} on ne supprime pas de cases admissibles par conséquent on a

$$\alpha_i = 0 \quad \forall\, i \in I - \{ j_m \}$$

$$\beta_j = 0 \quad \forall\, j \in I - \{ i_m \}$$

De plus comme on suppose $d(j_m, i_m) \neq 0$ on ne supprime pas de case admissible en posant $d(j_m, i_m) = \infty$ on aura donc

$$\alpha\, j_m = 0 \Rightarrow \alpha\, i = 0 \quad \forall\, i \in I$$

$$\beta\, i_m = 0 \Rightarrow \beta_j = 0 \quad \forall\, j \in J$$

Supposons maintenant que $d(j_m, i_m) = 0$

on a alors

$$e(i_m, j_m) = \gamma(i_m, j_m) - \alpha\, j_m - \beta\, i_m \qquad (1)$$

où $\alpha\, j_m \geqslant 0$ et $\beta\, i_m \geqslant 0$

et $e(j_m, i_m) = \gamma(j_m, i_m) - (\alpha'_{im} + \beta'_{jm}) - \begin{cases} \sum_p \beta_p \\ \text{ou} \\ \sum_p \alpha_l \end{cases}$ (2)

....

A ce niveau on se rend compte qu'il faut ajouter une condition supplémentaire :

$\alpha_{jm} > 0$ et $\beta_{im} > 0$

$\alpha_{jm} > 0 \Longrightarrow \sum_p \beta_p = 0$ car $\beta_p = 0 \ \forall \ p$

$\beta_{im} > 0 \Longrightarrow \sum_l \alpha_l = 0$ car alors $\alpha_l = 0 \ \forall \ l$

et on a $\gamma(j_m, i_m) = \alpha_{j_m} + \beta_{i_m}$

les relations (1) et (2) deviennent

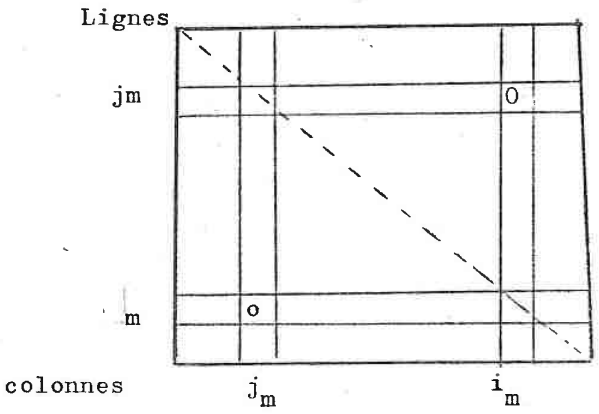
$$e(i_m, j_m) = \gamma(i_m, j_m) - \gamma(j_m, i_m) \qquad (3)$$

$$e(j_m, i_m) = \gamma(j_m, i_m) - \gamma(i_m, j_m) \qquad (4)$$

d'où comme c'est l'arc (i_m, j_m) qui a été admis et que par conséquent on a $e(i_m, j_m) \geqslant e(j_m, i_m)$ d'après (3) et (4) on a

$$\gamma(i_m, j_m) \geqslant \gamma(j_m, i_m)$$

et $\qquad e(i_m, j_m) \geqslant 0$



Pour cerner une condition nécessaire la tâche semble beaucoup plus difficile et on se contente dans la partie pratique de cette conditions suffisante.

Cette manière de choisir de meilleure case admissible semble d'ailleurs mieux convenir au deuxième algorithme, car on a intérêt à obtenir une borne majorante le plus rapidement possible, c'est-à-dire à définir un sous-ensemble terminal dans le moins d'étapes possibles.

....

Deuxième amélioration possible du système de recherche de case admissible

Que l'on prenne l'un ou l'autre des critères de recherche, on peut considérer que la sélection donne de bon résultats, c'est-à-dire désigne un arc qui figurera de façon définitive dans un circuit optimal, surtout pour les arcs qu sont obtenus à un niveau assez proche de la source.

Si on désigne par variables de base, dans le deuxième algorithme, les p premières variables admises selon les critères précédents, les applications montrent qu'il y a un intérêt certain à choisir les p + 1, p + 2, ... arcs suivants avec un critère différent.

Critère de sélection N° 3

Quand on a déterminé les p variables de base on se propose de choisir ensuite parmi les cases admissibles celle qui répondra, en plus de l'un ou de l'autre des deux premiers critères, à la condition suivante : l'origine ou l'extrémité de l'arc qu'elle représente figure déjà parmi les arcs admis. On peut justifier ce choix en disant que de toutes façons il existe des arcs qui partent ou arrivent sur des sommets déjà admis.

Remarque concernant le nombre p de variables de bases

Il dépend bien entendu de la dimension du réseau sur lequel on traite le problème. Il n'y a pas de règle précise pour le fixer d'autant plus que les applications montrent que pour différentes valeurs de p , après un même temps de calcul on aboutit sur les mêmes circuits sous-optimaux.

Par exemple pour le réseau où n = 42 on obtient au même circuit sous-optimal pour m = 6, 8, 10, 11 pour un temps de calcul donné. Par contre, pour des valeurs de m supérieures à celles de l'intervalle (6, 11) les sous-optimaux obtenus après le même temps de calcul sont supérieurs, c'est-à-dire moins bons.

De même pour les valeurs de m = 1, 2, 3, 4, 5 .

Remarque : pour m = n = 42 C = 950 pour m E(6, 11) C = 780

Cas des matrices de coût symétriques

La programmation de plusieurs exemples montre que, quelle que soit la méthode de progression choisie, le fait que la matrice des coûts soit symétrique diminue sensiblement la rapidité de convergence.

Par exemple on a programmé le cas d'un réseau où n = 10 et la matrice non symétrique : le nombre d'itérations nécessaires est égal à 17.

En reprenant des coûts du même ordre de grandeur, mais en rendant la matrice des coûts symétrique, le nombre d'itérations nécessaires s'est élevé à 85.

....

Causes de l'augmentation du nombre d'itérations

On aboutit à des arborescences qui ont l'allure suivante :

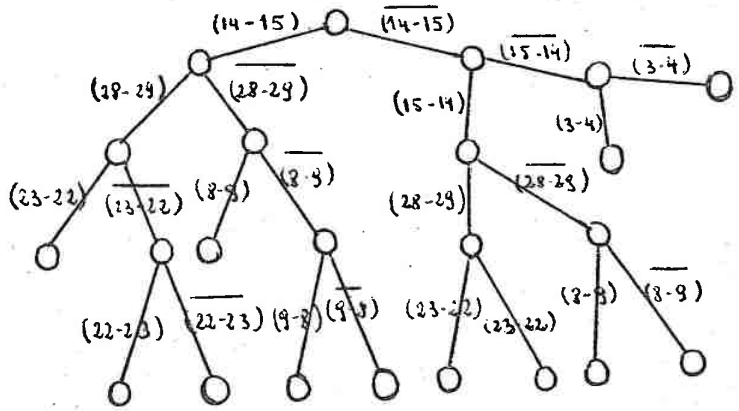
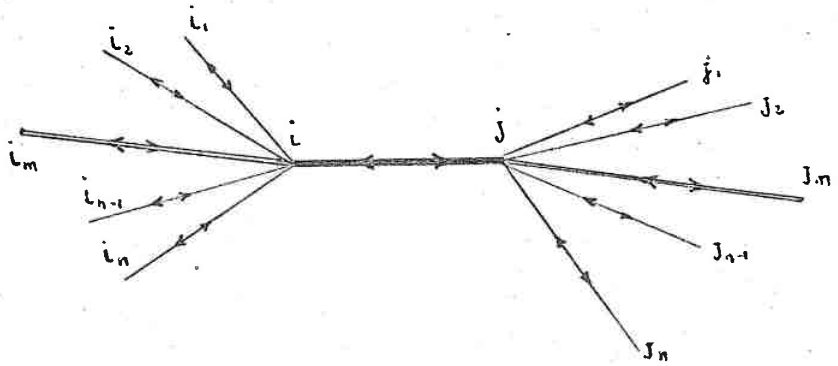


Fig 5

Les bornes des sous-ensemble $E_1^{(k)}$ et $E_2^{(k)}$ ne sont pas très différentes et la figure suivante s'explique en partie



que l'on admette l'arc (i, j) ou l'arc (j, i) le coût supplémentaire entraîné par cette admission est le même; le sous-ensemble contenant l'arc (i, j) et celui contenant l'arc (j, i) sont donc équivalents.

Mais l'algorithme de LTTLE ne fait pas apparaître ce phénomène au niveau de l'arborescence étant donné que l'on considère le problème en termes de circuits orientés, alors que pour un graphe dont les coûts sont symétriques l'orientation est superflue.

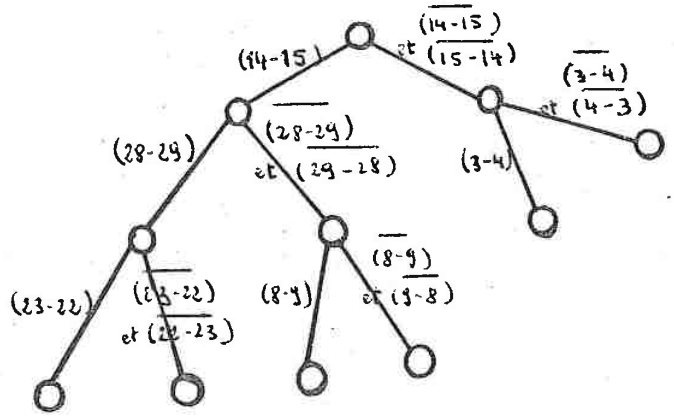
....

Amélioration proposée

Au moment où on admet un arc on définira les deux sous-ensembles de la façon suivante : $E_1^{(k)}$ de la façon habituelle mais $E_2^{(k)}$ comme ne contenant ni l'arc admis, ni l'arc symétrique à l'arc admis.

Le premier bénéfice de cette opération se situe au niveau de l'arborescence où le nombre de sommets diminue, le deuxième avantage est qu'en supprimant deux cases nulles à la fois dans $d_{ij}^{(k)}$ en les mettant à l' ∞ on augmente la borne associée à $E_2^{(k)}$.

Avec le principe exposé ci-dessus l'arborescence de la figure 5 devient



Pour ce qui est de la programmation de cette amélioration elle est activée ou non grâce à une carte paramètre où une variable SYM

SYM = 0 indique que la matrice des coûts n'est pas symétrique et

SYM = 1 indique que la matrice des coûts est symétrique et que l'amélioration proposée est active.

L'exemple dont nous parlions avant donne

N= 10 84 itérations avec SYM = 0

29 itérations avec SYM = 1

....

PROGRAMMATION

Implantation de l'arborescence en mémoire

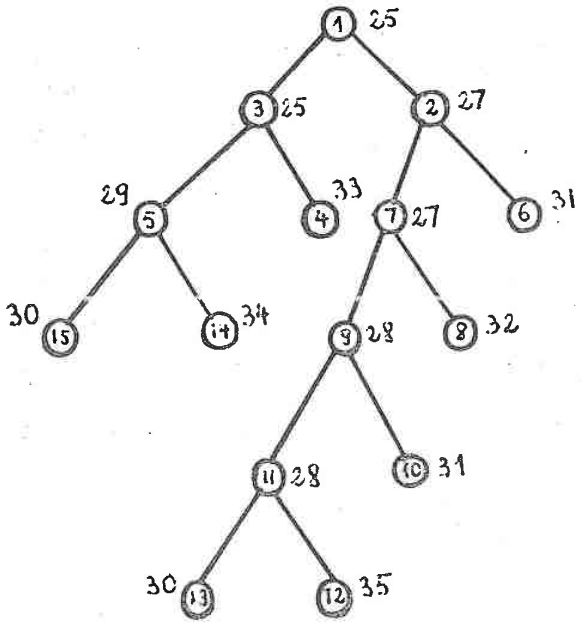
Chaque sommet de l'arbre peut être défini par quatre données : le numéro du sommet père, la borne du sous-ensemble associé, le niveau par rapport à la racine et le numéro du sommet fils.

Numérotation des sommets

En premier lieu la racine à laquelle on affecte le numéro 1
Puis à l'itération k on associe à l'ensemble $E_2^{(k)}$ le numéro 2k et à l'ensemble $E_1^{(k)}$ le numéro 2k + 1.
Cette numérotation permet de savoir à quelle itération k le sommet a été obtenu, c'est-à-dire de retrouver avec quelles transformations on a passé de $d_{ij}^{(k_{bp})}$ à $d_{ij}^{(k)}$

De plus les transformations relatives à un sommet de numéro pair étant différentes de celles relatives à un sommet impair la parité du numéro du sommet déterminera le type de transformations à appliquer à la matrice d_{ij} associée.

Exemple :
soit l'arbre suivant :



....

il sera décrit sur le calculateur à l'aide de la pile suivante :

	Π père	ℓ borne	N niveau	fils
1	0	25	0	3
2	1	27	1	7
3	1	25	1	5
4	3	33	2	0
5	3	29	2	15
6	2	31	2	0
7	2	27	2	9
8	7	32	3	0
9	7	28	3	11
10	9	31	4	0
11	9	28	4	13
12	11	35	5	0
13	11	30.	5	0
14	5	34	3	0
15	5	30	3	0

Représentation de la transformation f_k où k est le numéro de l'itération dans laquelle elle intervient.

Ce qu'il faut retenir quand on passe d'une matrice des coûts $d_{ij}^{(k_{bp})}$ à $d_{ij}^{(k+1)}$. ce sont les paramètres de la transformation qui seront stockés dans le tableau suivant en mémoire centrale

numéro itération	1	2	3	4	5	6	7	8	9	10
					arc interdit					
1										
2										
i										
m										

....

Description des informations contenues dans chaque case du tableau pour une ligne donnée

1ère case : origine de l'arc opérant la séparation

2ème case : extrémité de l'arc opérant la séparation

3ème case : index de base donnant le début d'une zone où sont répertoriées les lignes et les colonnes à modifier

4ème case : longueur de la zone dont le début est donné dans la case 3

5ème, 6ème, 7ème cases : donnent les coordonnées de l'arc interdit et la valeur du coût actuel (au cours de l'itération) de cet arc.

8ème case : valeur du coût de l'arc opposé (l_i, k_i). En fait celle-ci est superflue comme on le verra plus loin.

9ème et 10ème cases : concernent les cas où on ne prend pas l'arc $(k_i - l_i)$ La 9ème contient le minimum sur la ligne k_i et la 10ème le minimum sur la colonne l_i

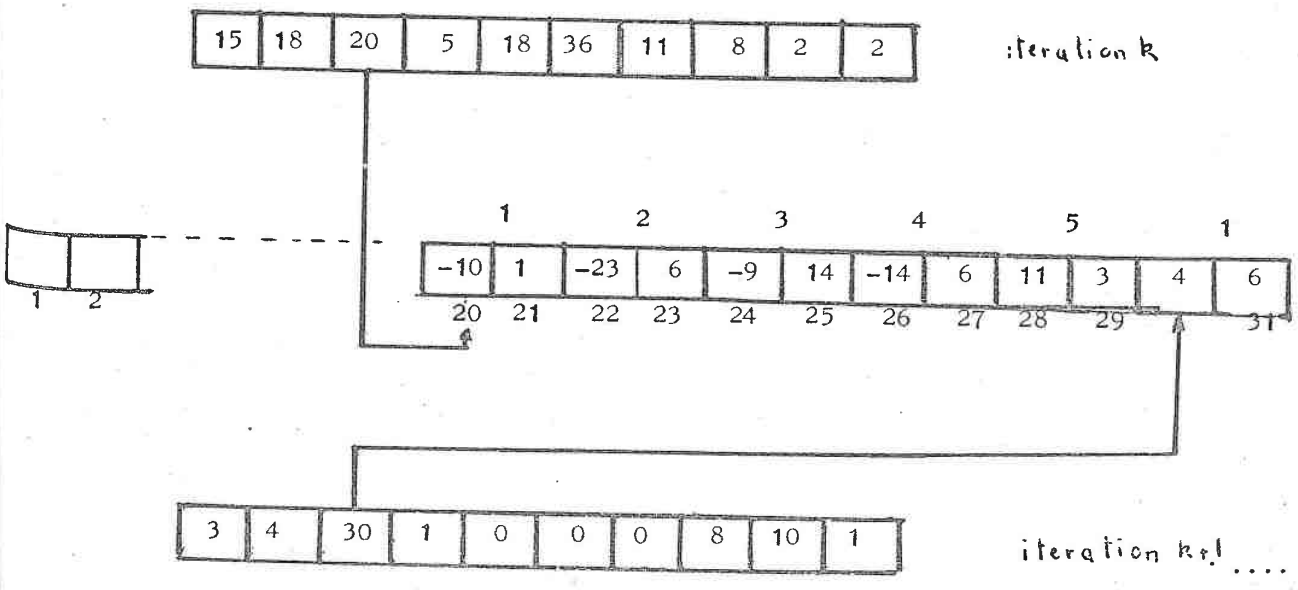
Nous avions vu dans le chapitre précédent que ces informations étaient suffisantes pour reconstituer la matrice des coûts associée à n'importe quel sommet de l'arbre.

Description de la zone contenant les numéros des rangées à modifier

Précisons d'abord qu'on a choisi d'affecter d'un signe moins les colonnes qu'on veut modifier pour les distinguer des lignes qui elles sont repérées par leur numéro positif.

Cette zone sera pour une itération donnée un ensemble de couples (r_i , v_i) où r_i sera le numéro de la ligne à modifier si $r_i > 0$ et $|r_i|$ sera la colonne à modifier si $r_i < 0$, v_i étant la valeur de la modification.

Exemple :



Cet exemple illustre pourquoi on n'a pas choisi de répertorier les rangées à modifier dans la tableau principal, dans l'itération k il aurait fallu 10 places mémoires pour répertorier les rangées à modifier dans le second il n'en faut que 2 .

Rappelons que le nombre maximal de rangées qui sont susceptibles d'être modifiées est n-1 si n est la dimension du réseau; par conséquent on aurait été obligé de dimensionner le tableau principal à (n + 8) • nombre d'itérations estimé.

Comme le nombre de rangées à modifier est variable selon les itérations, on a choisi de créer une zone indexée où on peut trouver ces informations.

Nous avons ainsi défini les éléments nécessaires pour appliquer les transformations f afin de retrouver avant chaque itération l'état de la matrice des coûts correspondant au sommet associé à la borne minimale.

....

Description de toutes les procédures intervenant dans l'organigramme

Notons tout de suite que la plupart de ces procédures intervenant comme des sous-programmes elles sont utilisées sans modification quand nous aborderons l'organigramme du programme décrivant l'autre méthode de progression dans l'arborescence.

1. Recherche de la meilleure case admissible

Une case admissible est une case du tableau des coûts D_{ij} telle que $D_{ij} = 0$. Parmi toutes les cases qui vérifient cette relation il faudra en choisir une pour déterminer la séparation. On choisira celle dont on estimera lors de l'itération qu'elle a le plus de chance d'appartenir à un circuit optimal.

Dans le programme décrit plus haut on a pris comme critère de choix la case admissible pour laquelle on a

$$\min_{jEL} (D_{kj}) + \min_{iEK} (D_{il}) \text{ maximum}$$

Ce critère est celui choisi par LITTLE dans son algorithme. On peut en prendre d'autres comme on le verra plus loin.

Cette opération est exécutée par le sous-programme MAX GAM

2. Rangement de l'arc (K, L) dans l'ensemble des arcs admis et recherche de l'arc pouvant créer dans les itérations suivantes des cycles interdits : cette opération s'effectue avec le sous-programme CYCINT.

N'envisageons pas le cas trival où l'arc interdit est l'arc (L,K) auquel cas il suffit de poser $D_{LK} = \infty$ et considérons sur des exemples les trois cas possibles.

1er cas

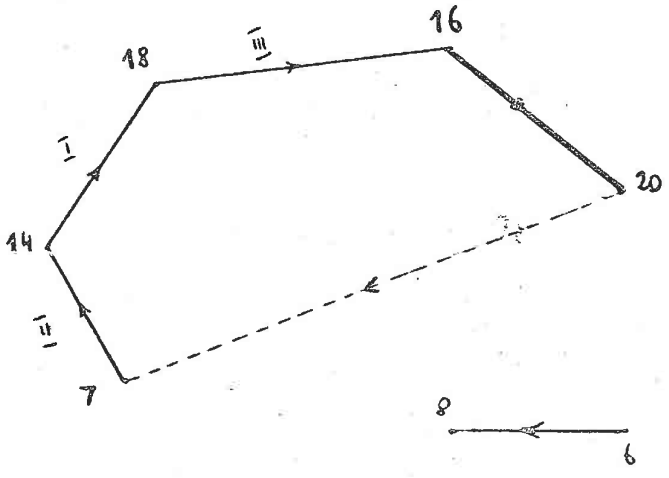


Fig I

....

L'ensemble des arcs admis sera représenté par une matrice à deux lignes et N colonnes; la première ligne donnant les origines des arcs, la deuxième les extrémités.

Ainsi pour la figure I l'état de cette matrice sera le suivant avant l'admission de l'arc (V)

7	14	18	8	0	0	
14	18	16	6	0	0	
1	2	3	4	5	6	7

Les numéros (I), (II), (III), (IV), (V) indiquent l'ordre d'apparition des arcs. Le programme transforme ce tableau après l'admission de l'arc (V) dans le suivant :

7	14	18	16	8		
14	18	16	20	6		
1	2	3	4	5	6	7

qui donne les coordonnées de l'arc à interdire : 20 pour l'origine, en l'occurence et 7 pour l'extrémité : on stocke ces valeurs respectivement dans les 5ème et 6ème cases de la ligne décrivant l'itération (V) dans le tableau des transformations; la valeur actuelle des coûts de l'arc (20, 7) étant stockée dans la 7ème case.

2ème cas

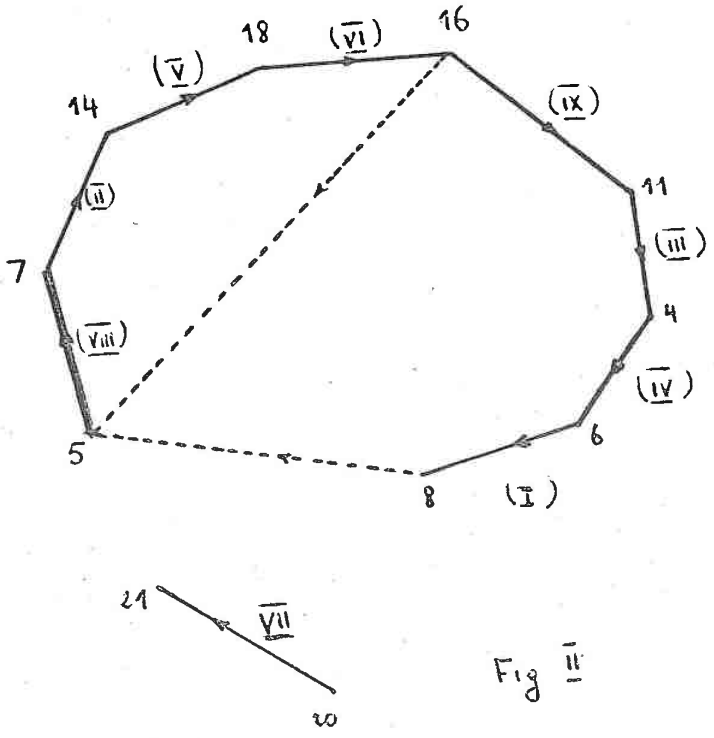


Fig II

....

Etat avant l'admission de (5-7)

11	4	6	7	14	18	21	0	0
4	6	8	14	18	16	20	0	0
1	2	3	4	5	6	7	8	9

après admission de (5-7)

11	4	6	5	7	14	18	21	0
4	6	8	7	14	18	16	20	0
1	2	3	4	5	6	7	8	9

Dans ce cas origine de l'arc interdit 16

extrémité de l'arc interdit 5

Comme précédemment on stocke ces deux valeurs dans les emplacements correspondants du tableau des transformations.

3ème cas

Reprenons la figure II et supposons que le neuvième arc soit (16-11) l'état avant l'admission de (16-11) est celui indiqué plus haut avec l'admission de (16-11) on obtient le tableau suivant :

5	7	14	18	16	11	4	6	21	
7	14	18	16	11	4	6	8	20	
1	2	3	4	5	6	7	8	9	10

origine de l'arc interdit pour IX : 8

extrémité de l'arc interdit : 5

Une telle opération est décrite et exécutée par le sous-programme appelé (CYCINT).

Chaque appel de CYCINT occasionne une incrémentation du nombre d'arcs admis; si ce nombre devient égal à n-1 on a trouvé une solution optimale et on peut se débrancher sur la fin du programme.

En effet, pour le n terme arc du circuit il n'y a plus de choix possible puisqu'il ne reste plus qu'une seule case admissible et il n'y a plus d'arc à interdire.

3. Calcul des α_i , β_j et de $G_1^{(k)}$

Est effectué par le sous-programme appelé MINLC.

Les valeurs de α_i figurent dans la première ligne d'un tableau DELTA les β_j sur la deuxième ligne de ce tableau à la sortie du sous-programme MINLC.

....

Un paramètre booléen BOOL qui est affecté dans le programme principal à 0 ou 1 indiquera si ces valeurs i et j il faut les retrancher de suite aux lignes i et colonnes j correspondantes (c'est ce que fera le sous-programme pour effectuer l'opération MODIF 1) ou uniquement les transférer dans DELTA.

Il est clair que le calcul de $G_1^{(k)}$ s'effectuera de la façon suivante :

$$G_1^{(k)} = \sum_i \alpha_i + \sum_j \beta_j \qquad i \in I, \quad j \in J$$

Rappelons que I = ensemble des lignes non encore barrées

J = ensemble des colonnes non encore barrées

Donc avant d'appeler le sous-programme MINLC il faut barrer les rangées K et L

c'est-à-dire procéder à l'opération $I = I - \{ K \}$

$J = J - \{ L \}$

La façon de procéder est décrite plus loin dans les phases MARQUAGE et DEMARQUAGE.

4. Stockage des sommets N° 2k et N° 2k+1 dans PILE

PILE (i, 4) est le tableau décrivant l'arborescence.

Si k_{bp} est l'indice du bout pendant actuel on aura

PILE (2k,1) = k_{bp}

PILE (2k+1, 1) = k_{bp}

les deux sommets ont le même père.

Pour le niveau

PILE (2k,3) = PILE (2k+1,3) = PILE (k_{bp}, 3) + 1

Pour les bornes

PILE (2k, 2) = PILE (k_{bp}, 2) + $G_2^{(k)}$

PILE (2k+1,2)= PILE (k_{bp}, 2) + $G_1^{(k)}$

....

5. Stockage des caractéristiques relatives à l'itération k

FONC est le nom du tableau qui les contient.

FONC (k, 1) = K

FONC (k, 2) = L

FONC (k, 3) = FONC (k-1,3) + (nbre rangées à modifier) * 2

FONC (k, 4) = nbre de rangées à modifier ($= \Sigma i + \Sigma j \ / \alpha_i \neq 0, \beta_j \neq 0$)

FONC (k, 5), FONC (k, 6), FONC (k, 7) ont déjà été affectés dans le sous-programme CYCINT.

FONC (k, 9) = $\min_{jEJ-\{L\}} (d_{Kj})$

FONC (k,10) = $\min_{iEI-\{K\}} (d_{iL})$

6. Marquage ou démarquage de la ligne k et de la colonne l

Le marquage est simulé grâce à un tableau appelé LCSUPR (pour lignes colonnes supprimées). C'est un tableau à 2 colonnes : qui ne contient que des 1 et des 0 et qui est régi de la façon suivante : A l'itération k si $G_2^{(k)} < G_1^{(k)}$, c'est-à-dire si on accepte l'arc (k,l), pour barrer la ligne k et le colonne l il suffit de mettre un masque, c'est-à-dire faire

LCSUPR (k,1) = 1 $\Longleftrightarrow$ I = I - $\{ k \}$

LCSUPR (1,2) = 1 $\Longleftrightarrow$ J = J - $\{ l \}$

Si $G_2^{(k)} < G_1^{(k)}$ il faut intégrer la ligne k et la colonne l dans l'ensemble des rangées non barrées en faisant

LCSUPR (k,1) = 0

LCSUPR (1,2) = 0

7. On explicite modif 1 de $D_{ij}^{(k)}$

Il suffit d'appeler le sous-programme MINLC (____,BOOL ,____) où sera un paramètre positionné à 1 pour que les α_i et β_j soient retranchées des rangées i et j correspondantes.

8. modif 2 de $D_{ij}^{(k)}$

De même que pour modif 1 il suffit d'appeler le sous-programme MAXGAM (BOOL,) où BOOL = 1 indique qu'il faut soustraire le minimum sur la ligne k et le minimum sur la colonne l et qu'on fasse $D_{kl} = \infty$

....

9. Recherche du bout pendant minimal d'indice k_{bp}

Pour l'expliquer reprenons l'exemple de la page 50 et supposons qu'on soit à l'itération 6 ; ce qui veut dire qu'on a défini les sommets 12 et 13 sur l'arborescence et dans la pile associée.

On veut chercher le bout pendant minimal associé à la 7ème itération.

On recherche donc en commençant par le sommet 2 parmi tous les sommets qui n'ont pas encore de successeur, c'est-à-dire tel que PILE (LL,4) = 0, celui qui correspond à la quantité PILE (LL,2) minimale et qui se trouve le plus loin possible de la source.

On est assuré par ce processus de trouver un sommet qui représente un ensemble de variables x_{ij} admises le plus grand possible.

Quand on a trouvé ce sommet dont l'indice est représenté par la variable K B P il faut indiquer quel sera son successeur en posant

PILE (K B P, 4) = 2 x (k+1) + 1

c'est-à-dire dans l'exemple que nous considérons

PILE (5, 4) = 15

On effectue ensuite un test qui indiquera si on peut continuer sur la même branche de l'arborescence pour l'itération suivante ou si au contraire on est obligé de modifier la matrice actuelle des coûts $D_{ij}^{(k)}$ (nom symbolique dans le programme COUT (I,J) .

Si k_{bp} est égal à 2k ou 2k+1 c'est que le bout pendant minimal correspond au dernier ou à l'avant dernier sommet défini sur l'arbre; on peut par conséquent continuer par la phase de recherche de la meilleure case admissible suivante en gardant la même matrice des coûts (à laquelle on a enlevé selon le cas la ligne k et la colonne 1) en incrémentant le compteur d'itérations d' 1 .

10. Si k_{bp} n'est pas égal à 2k ou 2k+1

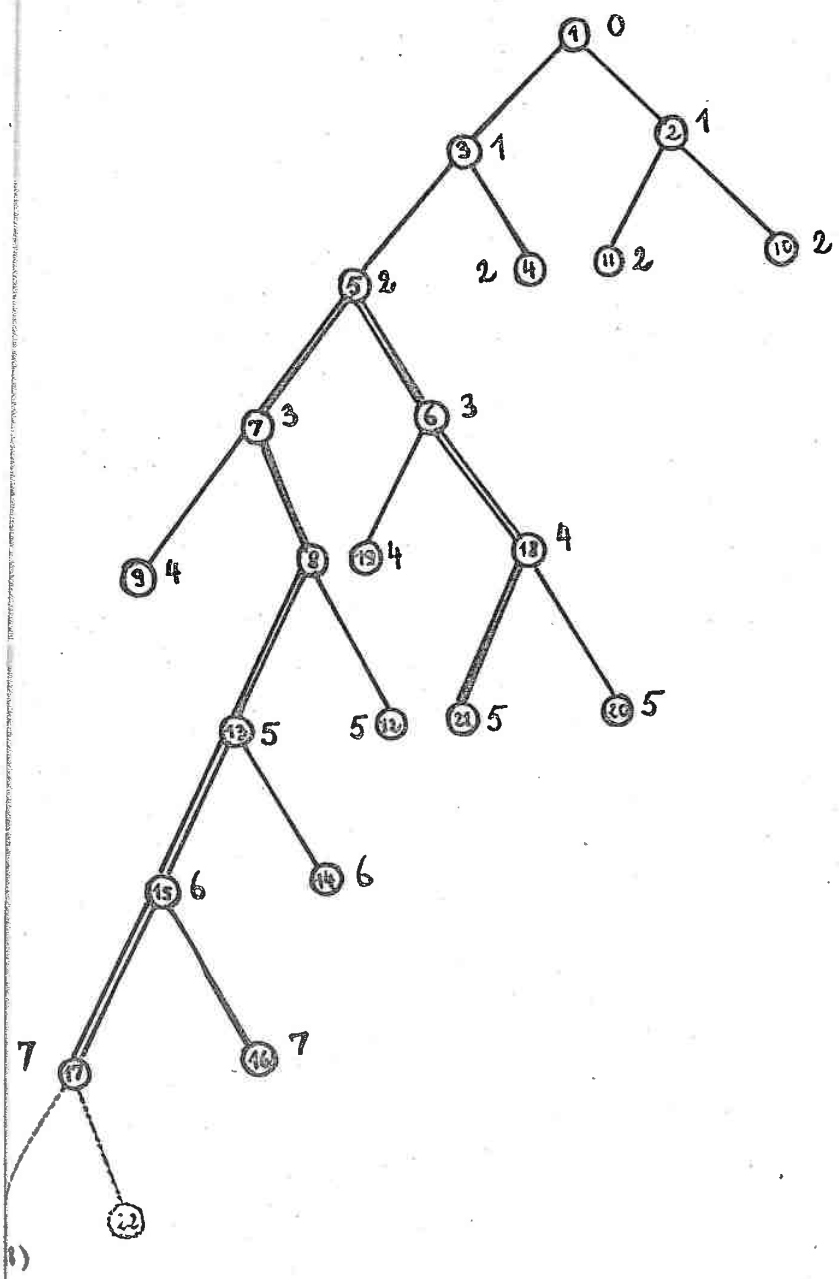
Recherche de l'état de la matrice $D_{ij}^{(k_{bp})}$ correspondant au sommet k_{bp}

C'est la partie la plus intéressante du programme : celle qui évite de conserver en mémoire toutes les matrices correspondant à des bouts pendants.

C'est dans cette partie qu'apparaît l'opportunité de la 3ème colonne de PILE qui donne, rappelons-le, le nombre d'arcs depuis la source (sommet 1) au sommet considéré.

....

Il s'agit de décrire dans le programme la formalisation exposée page quand $k_{bp} \neq 2k$ et $k_{bp} \neq 2k+1$.



Sur la figure on a représenté chaque sommet avec son numéro et son degré. On suppose que l'itération actuelle est l'itération 10, c'est-à-dire qu'on se trouve aux sommets 21 ou 20 et que le bout pendant soit 17.

Il s'agit dans une première étape de retrouver ce qu'on a appelé le point critique : en effet on voit qu'il est inutile de remonter jusqu'à la source pour pouvoir calculer $D_{ij}^{(8)}$

....

Il faut donc remonter jusqu'à un certain point dans l'arbre : quel est ce point (ce sommet) ? Pour cela on considère le niveau du sommet actuel (5 sur la figure) et le niveau (ou degré) du sommet pendant (7 sur la figure). On calcule la différence de niveau qu'on affecte dans le programme à la variable DNIV (= 2 sur l'exemple)

$$DNIV = PILE(k_{bp}, 3) - PILE\left(\begin{matrix}2k+1\\ \text{ou}\\ 2k\end{matrix}, 3\right)$$

$$= PILE(17,3) - PILE(21,3) = 2$$

Si DNIV est positif , ce qui signifie que le bout pendant minimal est "plus loin" de la source ; on remonte DNIV fois le chaînage PERE (c'est la fonction Π dont on a parlé dans le chapitre précédent) à partir de k_{bp} avec la séquence suivante $k_{bp}1 = k_{bp}$

pour ll = 1 jusqu'à DNIV faire
$k_{bp}1 = PILE(k_{bp}, 1)$

ce qui donne à la sortie de la boucle, sur l'exemple traité

$k_{bp}1 = 13$

Puis on remonte le chaînage PERE sur les deux branches en partant pour l'une des branches (celle qui commence avec le sommet k_{bp}) du sommet $k_{bp}1$ et pour l'autre du sommet actuel 2k ou 2k+1 jusqu'à ce qu'on trouve le premie prédécesseur commun; ce prédécesseur sera le point critique (sur la figure ce sera en l'occurence le sommet 5).

Si le DNIV est négatif : le bout pendant minimal est "plus près" de la source on remonte alors le chaînage PERE sur une longueur égale à (DNIV) à partir du sommet 2k ou 2k+1 et à partir du sommet $k_{bp}1$ obtenu, on applique la même procédure que précédemment pour obtenir le point critique.

Le point critique étant déterminé, il faut appliquer kes formules qui permettent de passer de D_{ij}^{k-1} à $D_{ij}^{k_{bp}}$. Pour cela on dispose de deux types d'opération : le rebroussement qui correspond à l'application d'une fonction f^{-1} et la progression qui correspond à l'application d'une fonction f à un sommet déjà répertorié.
Pour ces deux opérations on se sert pour la première du sous-programme REBROU et pour la seconde du sous-programme PROGR.

....

Le sous-programme REBROU

décrit l'opération élémentaire qui doit remettre la matrice $D_{ij}^{(k-1)}$ à l'état $D_{ij}^{\prod (k-1)}$ si $\prod (k-1)$ donne le numéro du sommet qui est le père de k-1 . Dans le cas trivial $\prod (k-1) = k-2$.

Pour cela il s'agit de savoir si k-1 est pair ou impair : en effet si k-1 est pair on sait qu'on a atteint le sommet k-1 par une opération de type "ne passe pas par" qui correspond à un type de transformations précis de $D_{ij}^{(k-1)}$.

Si k-1 est impair $D_{ij}^{(k-1)}$ a été atteint par une opération du type "passe par" à laquelle correspond aussi un type de transformations de $D_{ij}^{(k-1)}$.

On teste donc en premier lieu k-1 en posant

$X = (k-1) - (k-1)/2 \times 2$ ou / est la division entière.

si X = 0 k-1 est pair

X = 1 k-1 est impair.

cas X = 1 opération du type "passe par" correspondant à l'itération N° (k-1)/2. Les paramètres du sous-programme REBROU seront les huit premières positions de FONC((k-1) /2).

Ces paramètres sont ceux qui permettent de réadditionner aux rangées décrites dans la zone définie par les positions 3 et 4 les valeurs qui en ont été enlevées, de remettre la ligne indiquée en case 1 et la colonne indiquée en case 2 parmi l'ensemble de rangées non encore barrées, d'extraire l'arc (case 1 - case 2) de l'ensemble des arcs admis, de remettre l'arc interdit à sa valeur avant l'itération (k-1)/2 (cette valeur est contenue dans case 7).

si X = 0 nous sommes en présence d'une opération du type "ne passe pas par" On utilise les seuls paramètres FONC (1), FONC (2), FONC (9), FONC (10) et on effectue les trois opérations suivantes :

- on additionne la quantité contenue dans FONC (3) à tous les éléments de la ligne FONC (1) (ceux bien entendu qui n'appartiennent pas à des colonnes marquées)
- on additionne la quantité contenue dans FONC (10) à tous les éléments de la colonne indiquée dans FONC (2) (à l'exclusion des éléments qui appartiennent à des lignes marquées)
- et on pose D (FONC (1), FONC (2)) = 0.

....

Le sous-programme PROGR

C'est en fait un sous-programme où on simule l'admission d'un arc ou l'exclusion d'un arc, c'est-à-dire qu'on refait toutes les opérations décrites dans le programme principal, mais pour lesquelles il est inutile de calculer les paramètres qui figurent dans le tableau FONC.

La structure du sous-programme est la même que celle du sous-programme REBROU dont il est en quelque sorte l'inverse.

Si p est le numéro du sommet actuel et si p_1 = fils(p) il s'agit d'effectuer l'opération $D_{ij}^{p/2} \longrightarrow D_{ij}^{p1/2}$

on pose pour cela

$$p_o = p_1 - (p_1/2) \times 2$$

si $p_o = 1$ transformation du type "passe par" qu'on a étudié dans le programme principal : on la répète en se servant des paramètres contenus dans FONC ($(p_1/2)$)

si $p_o = 0$ transformation du type "ne passe pas par" déjà décrite aussi dans le programme principal.

....

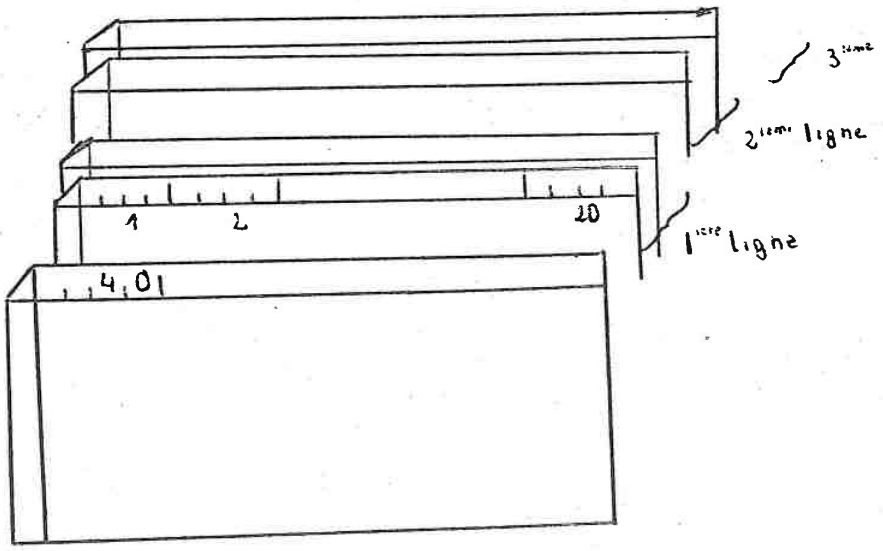
Notice d'utilisation des deux programmes

Il s'agit de voir quelle est la façon de rentrer les données du réseau en mémoire.

Remarquons tout de suite qu'elle est tout à fait identique dans les deux programmes.

Les coûts du réseau figurent sur un fichier carte à raison de 20 données par carte et ligne par ligne.
La première carte donnée, qui précède le fichier carte, indique la dimension du réseau. Elle est codée sur les quatre premières colonnes.

Image de la configuration



```
      IMPLICIT INTEGER (A-Z)
      DIMENSION  COUT(50,50),LCSUPR(50,2),DELTA(50,2),PILE(200,4)
      DIMENSION FONC(200,10),CCLIN(2,50),CCLIN1(2,50),DEBORD(200)
C LECTURE DES DONNEES
      READ(5,100) N
  100 FORMAT(I4)
      WRITE(6,104)N
  104 FORMAT('1',10X,'DIMENSION DU GRAPHE',I4)
      DO 15 I=1,N
   15 READ(5,101)(COUT(I,J),J=1,N)
  101 FORMAT(24I3)
      DO 17 I=1,N
      DO 18 J=1,2
   18 LCSUPR(I,J)=0
      COUT(I,I)=2000000
   17 CONTINUE
      OPTI=2000000
      LLANT=-1
      NIV=1
      KBP=1
      KNIV=0
      PILE(NIV,1)=0
      PILE(NIV,3)=0
      MINO=2000000
C     INITIALISATION CALCUL DE LA BORNE MINORANTE ABSOLUE
      DO 19 LL=1,N
      DO 20 KK=1,N
   20 MINO=MINO(MINO,COUT(LL,KK))
      BZERO=BZERO+MINO
      DO 21 KK=1,N
   21 COUT(LL,KK)=COUT(LL,KK)-MINO
      MINO=2000000
   19 CONTINUE
      DO 22 KK=1,N
      DO 24 LL=1,N
   24 MINO=MINO(MINO,COUT(LL,KK))
      BZERO=BZERO+MINO
      DO 25 LL=1,N
   25 COUT(LL,KK)=COUT(LL,KK)-MINO
      MINO=2000000
   22 CONTINUE
      PILE(NIV,2)=BZERO
   23 IF(KNIV.NE.(N-1))GO TO 224
      CCLIN(1,KNIV+1)=CCLIN(2,KNIV)
      CCLIN(2,KNIV+1)=CCLIN(1,1)
      K=CCLIN(2,KNIV)
      L=CCLIN(1,1)
      KNIV=N
      NBIT=NBIT+1
      FONC(NBIT,8)=COUT(L,K)
      GO TO 336
  224 NBIT=NBIT+1
C RECHERCHE DE LA MEILLEURE CASE ADMISSIBLE
      CALL MAXBAM(COUT,LCSUPR,N,K,L,BBAR,MK,ML,KNIV,CCLIN,FONC,NBIT,BI
     *DELTA,IND)
      K1=IND
      IF(K1.EQ.0)IND=1
      IF(K1.EQ.1)IND=0
  111 FORMAT(5X,30I4/)
C CONSTRUCTION DE L'ARBORESCENCE ET DE LA FONCTION F
  336 FONC(NBIT,1)=K
      FONC(NBIT,2)=L
      NIV=NIV+1
      PILE(NIV,1)=KBP
      PILE(NIV,2)=PILE(KBP,2)+BBAR
      PILE(NIV,3)=PILE(KBP,3)+1
      NIV=NIV+1
```

```
      PILE(NIV,1)=KBP
      PILE(NIV,2)=PILE(KBP,2)+B1
      PILE(NIV,3)=PILE(KBP,3)+1
      PILE(KBP,4)=NIV
      IF((KBP-(KBP/2)*2).EQ.0)GO TO 959
      PILE(KBP-1,4)=0
  959 IF(KNIV.EQ.1)GO TO 223
      FONC(NBIT,9)=KK
      FONC(NBIT,10)=ML
      KBP=NIV
      FONC(NBIT,3)=0
      FONC(NBIT,4)=LLANT
      IF(B1.EQ.0)GO TO 23
      LL=0
C DEFINITION DE LA ZONE CONTENANT LES VALEURS DES ALPHA ET BETA
      DO 125 KK=1,N
      IF(DELTA(KK,1).EQ.0)GO TO 126
      LL=LL+1
      DEBORD(LLANT+2*LL)=KK
      DEBORD(LLANT+2*LL+1)=DELTA(KK,1)
  126 IF(DELTA(KK,2).EQ.0)GO TO 125
      LL=LL+1
      DEBORD(LLANT+2*LL)=-KK
      DEBORD(LLANT+2*LL+1)=DELTA(KK,2)
  125 CONTINUE
      FONC(NBIT,3)=LL
      FONC(NBIT,4)=LLANT
      LLANT=LLANT+2*LL
      IF(PILE(NIV,2).LT.OPTI)GO TO 23
C RECHERCHE DU BOUT PENDANT MINIMAL LE PLUS LOIN DE LA SOURCE
  934 DO 935 LL=2,NIV
      IF(PILE(NIV-LL+2,2).GE.OPTI)GO TO 935
      IF(PILE(NIV-LL+2,4).NE.0)GO TO 935
      INDIC=1
      KBP1=NIV-LL+2
  935 CONTINUE
C     RECHERCHE DU CHEMIN MENANT DU SOMMET ACTUEL AU SOMMET PENDANT
      IF(INDIC.EQ.0)GO TO 33
      INDIC=0
  936 KBP=KBP1
      DNIV=PILE(NIV,3)-PILE(KBP,3)+1
      NBIT=NBIT-DNIV+1
      K1=KBP
   40 IF((PILE(K1,1)-(PILE(K1,1)/2)*2).EQ.0)GO TO 39
      NBT=K1/2
      GO TO 41
   39 K1=PILE(K1,1)
      GO TO 40
   41 LLANT=FONC(NBT,4)
      K1=NIV
      DO 938 LM=1,DNIV
      NBT=K1/2
      K2=PILE(K1,1)
      CALL REBROU(COUT,LCSUPR,N,FONC,K1,CCLIN,KNIV,DEBORD)
  938 K1=K2
      NIV=KBP+1
      CALL PROGR(COUT,LCSUPR,N,KBP,FONC)
      GO TO 23
  223 OPTI=PILE(NIV,2)
      DO 939 KK=1,N
      CCLIN1(1,KK)=CCLIN(1,KK)
  939 CCLIN1(2,KK)=CCLIN(2,KK)
      WRITE(6,107)PILE(NIV,2)
  107 FORMAT(10X,//10X'COUT INTERMEDIAIRE OPTIMAL',2I4)
      WRITE(6,109)
  109 FORMAT(15X,//'ENSEMBLE DES ARCS ADMIS A CE NIVEAU'/)
      WRITE(6,110)(CCLIN1(1,KK),KK=1,N)
```

```
  106 FORMAT(10X,40I3)
      GO TO 934
   33 WRITE(6,108)OPTI
  108 FORMAT(2X,////10X,'COUT FINAL OPTIMAL',I5)
      WRITE(6,110)
  110 FORMAT(10X,//10X,'CORRESPONDANT AU CIRCUIT HAMILTANIEN SUIVANT'/
      WRITE(6,106)(CCLIN1(1,KK),KK=1,N)
      WRITE(6,106)(CCLIN1(2,KK),KK=1,N)
      STOP
      END



C     CALCUL DE LA MATRICE D(FILS(K))
      SUBROUTINE PROGR(A,B,C,F,D)
      DIMENSION A(50,50),B(50,2),D(200,10)
      IMPLICIT INTEGER (A-Z)
      F1=F/2
      ALPHA=D(F1,1)
      BETA=D(F1,2)
      DO 1 KK=1,C
      IF(B(KK,1).NE.0)GO TO 1
      A(KK,BETA)=A(KK,BETA)-D(F1,10)
    1 CONTINUE
      DO 2 KK=1,C
      IF(B(KK,2).NE.0)GO TO 2
      A(ALPHA,KK)=A(ALPHA,KK)-D(F1,9)
    2 CONTINUE
      A(ALPHA,BETA)=2000000
      RETURN
      END



C     SOUS PROGRAMME DE CALCUL DE LA MATRICE D(PERE(K))
      SUBROUTINE REBROU(A,B,C,D,F,CCL,NIVAU,DEB)
      IMPLICIT INTEGER (A-Z)
      DIMENSION A(50,50),B(50,2),D(200,10),CCL(2,50),DEB(200)
      F1=F/2
      R=F-F1*2
      IF(R.EQ.0)GO TO 1
      ALPHA=D(F1,1)
      BETA=D(F1,2)
      C1=D(F1,3)
      BAS=D(F1,4)
      IF(C1.EQ.0)GO TO 38
      DO 35 LL=1,C1
      IF(DEB(BAS+2*LL).LT.0)GO TO 39
      I=DEB(BAS+2*LL)
      DO 36 KK=1,C
      IF(B(KK,2).NE.0)GO TO 36
      A(I,KK)=A(I,KK)+DEB(BAS+2*LL+1)
   36 CONTINUE
      GO TO 35
   39 J=IABS(DEB(BAS+2*LL))
      DO 37 KK=1,C
      IF(B(KK,1).NE.0)GO TO 37
      A(KK,J)=A(KK,J)+DEB(BAS+2*LL+1)
   37 CONTINUE
   35 CONTINUE
   38 IF(D(F1,5).EQ.0)GO TO 13
      I=D(F1,5)
      J=D(F1,6)
      A(I,J)=D(F1,7)
      D(F1,5)=0
      D(F1,6)=0
      D(F1,7)=0
```

```
      DO 49 KK=1,NIVAU
      IF((ALPHA.NE.CCL(1,KK)).AND.(BETA.NE.CCL(2,KK)))GO TO 49
      K1=KK
      DO 50 LL=K1,NIVAU
      CCL(1,LL)=CCL(1,LL+1)
   50 CCL(2,LL)=CCL(2,LL+1)
      NIVAU=NIVAU-1
      GO TO 59
   49 CONTINUE
   59 B(ALPHA,1)=0
      B(BETA,2)=0
      A(BETA,ALPHA)=D(F1,8)
      GO TO 11
    1 ALPHA=D(F1,1)
      BETA=D(F1,2)
      IF(D(F1,10).EQ.0)GO TO 10
      DO 8 KK=1,C
      IF(B(KK,1).NE.0)GO TO 8
      A(KK,BETA)=A(KK,BETA)+D(F1,10)
    8 CONTINUE
   10 IF(D(F1,9).EQ.0)GO TO 501
      DO 9 KK=1,C
      IF(B(KK,2).NE.0)GO TO 9
      A(ALPHA,KK)=A(ALPHA,KK)+D(F1,9)
    9 CONTINUE
  501 A(ALPHA,BETA)=0
      A(BETA,ALPHA)=D(F1,8)
   11 RETURN
      END


C     SOUS PROGRAMME DETERMINANT L'ARC INTERDIT ET GERANT LES PORTIONS D
      SUBROUTINE CYCIN(IND1,IND2,FCT,TAB,NBRIT,NIVO,COST,BOL3)   CIRCUIT
      IMPLICIT INTEGER (A-Z)
      DIMENSION TAB(2,50),FCT(200,10),COST(50,50)
      NIVO=NIVO+1
      TAB(1,NIVO)=IND1
      TAB(2,NIVO)=IND2
      IF(NIVO.EQ.1)GO TO 153
      NIVO1=NIVO-1
      DO 25 KK=1,NIVO1
      IF(TAB(1,NIVO).NE.TAB(2,KK))GO TO 15
      K1=KK+1
      LARC=1
      LARC1=2
      IF((NIVO-K1).EQ.0)GO TO 57
      R=TAB(1,K1)
      P=TAB(2,K1)
      TAB(1,K1)=TAB(1,NIVO)
      TAB(2,K1)=TAB(2,NIVO)
      NIVO2=NIVO-K1-1
      DO 16 LL=1,NIVO2
      TAB(1,NIVO-LL+1)=TAB(1,NIVO-LL)
   16 TAB(2,NIVO-LL+1)=TAB(2,NIVO-LL)
      TAB(1,K1+1)=R
      TAB(2,K1+1)=P
      GO TO 17
   15 IF(TAB(2,NIVO).NE.TAB(1,KK))GO TO 25
      K2=KK
      R=TAB(1,K2)
      P=TAB(2,K2)
      TAB(1,K2)=TAB(1,NIVO)
      TAB(2,K2)=TAB(2,NIVO)
      IF((NIVO-K2).EQ.1)GO TO 118
      NIVO2=NIVO-K2-1
      DO 26 LL=1,NIVO2
```

```
      TAB(1,NIVO-LL+1)=TAB(1,NIVO-LL)
   26 TAB(2,NIVO-LL+1)=TAB(2,NIVO-LL)
  118 TAB(1,K2+1)=R
      TAB(2,K2+1)=P
      GO TO 19
   25 CONTINUE
      GO TO 153
   17 LARC1=2
      IF(K1.EQ.2)GO TO 59
   57 K5=K1-2
      DO 58 LL=1,K5
      IF(TAB(1,K1-LL).NE.TAB(2,K1-LL-1))GO TO 59
      LARC1=LARC1+1
   58 CONTINUE
   59 IX1=TAB(2,K1)
      IY1=TAB(1,K1-LARC1+1)
      IF(BOL3.EQ.1)GO TO 63
      FCT(NBRIT,5)=IX1
      FCT(NBRIT,6)=IY1
      FCT(NBRIT,7)=COST(IX1,IY1)
   63 K5=K1+1
      K3=0
      DO 27 LL=K5,NIVO
      IF(TAB(1,LL).NE.TAB(2,K1))GO TO 27
      K3=LL
      GO TO 229
   27 CONTINUE
      IF(K3.EQ.0)GO TO 18
  229 DO 29 LL=K3,NIVO1
      IF(TAB(2,LL).NE.TAB(1,LL+1))GO TO 30
      LARC=LARC+1
   29 CONTINUE
   30 K5=K3-K1-1
      IF((K3-K1).EQ.1)GO TO 302
      DO 300 LL=1,K5
      R=TAB(1,K3-LL)
      P=TAB(2,K3-LL)
      DO 301 KK=1,LARC
      TAB(1,K3-LL+KK-1)=TAB(1,K3-LL+KK)
  301 TAB(2,K3-LL+KK-1)=TAB(2,K3-LL+KK)
      TAB(1,K3+LARC-LL)=R
  300 TAB(2,K3+LARC-LL)=P
  302 IX=TAB(2,K1+LARC)
      IY=TAB(1,K1-LARC1+1)
      IF(BOL3.EQ.1)GO TO 188
      FCT(NBRIT,5)=IX
      FCT(NBRIT,6)=IY
      FCT(NBRIT,7)=COST(IX,IY)
  188 IF(NIVO.EQ.BOL3)GO TO 153
      COST(IX,IY)=2000000
      GO TO 153
   19 IY1=TAB(1,K2)
      LARC=2
      DO 89 LL=2,NIVO
      IF(TAB(2,K2+LL-1).NE.TAB(1,K2+LL))GO TO 119
   89 LARC=LARC+1
  119 IX1=TAB(2,K2+LARC-1)
      IF(BOL3.EQ.1)GO TO 339
      FCT(NBRIT,5)=IX1
      FCT(NBRIT,6)=IY1
      FCT(NBRIT,7)=COST(IX1,IY1)
  339 K5=K2+LARC
      K4=0
      DO 39 LL=K5,NIVO
      IF(TAB(1,K2).NE.TAB(2,LL))GO TO 39
      K4=LL
   39 CONTINUE
      IF(K4.EQ.0)GO TO 18
```

```
      K5=K4-K2-LARC+1
      DO 159 LL=1,K5
      IF(TAB(1,K4-LL+1).NE.TAB(2,K4-LL))GO TO 161
  159 LARC1=LARC1+1
  161 DO 162 LL=1,LARC1
      R=TAB(1,K2)
      P=TAB(2,K2)
      TAB(1,K2)=TAB(1,K4)
      TAB(2,K2)=TAB(2,K4)
      K5=K4-K2-1
      DO 163 KK=1,K5
      TAB(1,K4-KK+1)=TAB(1,K4-KK)
  163 TAB(2,K4-KK+1)=TAB(2,K4-KK)
      TAB(1,K2+1)=R
  162 TAB(2,K2+1)=P
      IX=TAB(2,K2+LARC1+LARC-1)
      IY=TAB(1,K2)
      IF(BOL3.EQ.1)GO TO 182
      I1=5
      FCT(NBRIT,I1)=IX
      FCT(NBRIT,I1+1)=IY
      FCT(NBRIT,I1+2)=COST(IX,IY)
  182 IF(NIVO.EQ.BOL3)GO TO 153
      COST(IX,IY)=2000000
      GO TO 153
   18 IF(NIVO.NE.BOL3)COST(IX1,IY1)=2000000
  153 RETURN
      END
```

```
C     ON RETIRE UN ARC ADMIS A UNE ITERATION QUELCONQUE
      SUBROUTINE ENLEV(IND,CCLI,NIVA)
      INTEGER IND,CCLI
      DIMENSION CCLI(2,50)
      DO 14 KK=1,NIVA
      IF(CCLI(1,KK).NE.IND)GO TO 14
      LL=KK
      DO 15 MM=LL,NIVA
      CCLI(1,MM)=CCLI(1,MM+1)
   15 CCLI(2,MM)=CCLI(2,MM+1)
      NIVA=NIVA-1
      GO TO 16
   14 CONTINUE
   16 RETURN
      END
```

```
C    RECHERCHE DE LA MEILLEURE CASE ADMISSIBLE
      SUBROUTINE MAXGAM(A,B,M1,K1,K2,P,MK1,MK2,NIVO,CCL,D,F1,BZ,GAMA1,
     *IND1)
      IMPLICIT INTEGER (A-Z)
      DIMENSION A(50,50),B(50,2),CCL(2,50),GAMMA(50,2),D(200,10)
      DIMENSION GAMA1(50,2)
      LAMBDA=-2000000
      P=0
      BZ1=2000000
      IGAM1=2000000
      IGAM2=2000000
      DO 11 I=1,M1
      IF(B(I,1).NE.0)GO TO 11
      DO 21 J=1,M1
      IF(B(J,2).NE.0)GO TO 21
      IF(A(I,J).NE.0)GO TO 21
      IF(NIVO.LT.9)GO TO 19
      DO 20 KK=1,NIVO
      IF(J.NE.CCL(1,KK))GO TO 20
      GO TO 19
   20 CONTINUE
      GO TO 21
   19 DO 31 K=1,M1
      IF((B(K,2).NE.0).OR.(K.EQ.J)) GO TO 31
      IGAM1=MIN0(A(I,K),IGAM1)
   31 CONTINUE
      DO 41 K=1,M1
      IF((B(K,1).NE.0).OR.(K.EQ.I)) GO TO 41
      IGAM2=MIN0(A(K,J),IGAM2)
   41 CONTINJE
      P=IGAM1+IGAM2
      B(I,1)=1
      B(J,2)=1
      RET=A(J,I)
      A(J,I)=2000000
      INDIC=M1-1
      CALL CYCIN(I,J,D,CCL,F1,NIVO,A,INDIC)
      INT=2000000
      DO 34 II=1,M1
      GAMMA(II,1)=0
   34 GAMMA(II,2)=0
      BZ=0
      DO 1 II=1,M1
      IF(B(II,1).NE.0)GO TO 1
      DO 2 JJ=1,M1
      IF(B(JJ,2).NE.0)GO TO 2
      INT=MIN0(A(II,JJ),INT)
    2 CONTINUE
      GAMMA(II,1)=INT
      DO 3 K=1,M1
      IF(A(K,2).NE.0)GO TO 3
      A(II,K)=A(II,K)-GAMMA(II,1)
    3 CONTINUE
      BZ=BZ+INT
      INT=2000000
    1 CONTINUE
      DO 4 JJ=1,M1
      IF(B(JJ,2).NE.0)GO TO 4
      DO 5 II=1,M1
      IF(B(II,1).NE.0)GO TO 5
      INT=MIN0(A(II,JJ),INT)
    5 CONTINUE
      GAMMA(JJ,2)=INT
      BZ=BZ+INT
      INT=2000000
    4 CONTINUE
```

```
      JJ=D(F1,6)
      A(IU,JJ)=D(F1,7)
      D(F1,5)=0
      D(F1,6)=0
      D(F1,7)=0
      IF(BZ.GT.BZ1)GO TO 22
      IF(BZ.EQ.BZ1)GO TO 222
  223 BZ1=BZ
      DO 51 KK=1,M1
      GAMA1(KK,1)=GAMMA(KK,1)
   51 GAMA1(KK,2)=GAMMA(KK,2)
      K1=I
      K2=J
      MK1=IGAM1
      MK2=IGAM2
      LAMBDA=P-BZ
      GO TO 22
  222 IF((P-BZ).GT.LAMBDA)GO TO 223
   22 IGAM1=2000000
      IGAM2=2000000
   21 CONTINUE
   11 CONTINUE
      D(F1,8)=A(K2,K1)
      A(K2,K1)=2000000
      INDIC=M1-1
      CALL CYCIN(K1,K2,D,CCL,F1,NIVO,A,INDIC)
      B(K1,1)=1
      B(K2,2)=1
      DO 49 KK=1,M1
      IF(B(KK,2).NE.0)GO TO 49
      DO 50 LM=1,M1
      IF(B(LM,1).NE.0)GO TO 50
      A(LM,KK)=A(LM,KK)-GAMA1(KK,2)
   50 CONTINUE
   49 CONTINUE
      DO 59 KK=1,M1
      IF(B(KK,1).NE.0)GO TO 59
      DO 60 LM=1,M1
      IF(B(LM,2).NE.0)GO TO 60
      A(KK,LM)=A(KK,LM)-GAMA1(KK,1)
   60 CONTINUE
   59 CONTINUE
      P=LAMBDA+BZ1
      BZ=BZ1
      RETURN
      END
```

```
 DIMENSION DU GRAPHE  10

COUT INTERMEDIAIRE OPTIMAL  88

CIRCUIT HAMILTONIEN SOUS- OPTIMAL
  3  2  9  8  4  1  5  6  7 10
  2  9  8  4  1  5  6  7 10  3

COUT INTERMEDIAIRE OPTIMAL  85

CIRCUIT HAMILTONIEN SOUS- OPTIMAL
  4  8  3 10  7  1  5  6  2  9
  8  3 10  7  1  5  6  2  9  4

COUT FINAL OPTIMAL   85

CORRESPONDANT AU CIRCUIT HAMILTONIEN SUIVANT

  4  8  3 10  7  1  5  6  2  9
  8  3 10  7  1  5  6  2  9  4

 DIMENSION DU GRAPHE  20

COUT INTERMEDIAIRE OPTIMAL 458

CIRCUIT HAMILTONIEN SOUS- OPTIMAL
  5 15 16  1  7 10  3 12 11  8  2 18 20  4  6 17 14 13 19  9
 15 16  1  7 10  3 12 11  8  2 18 20  4  6 17 14 13 19  9  5

COUT FINAL OPTIMAL  458

CORRESPONDANT AU CIRCUIT HAMILTONIEN SUIVANT

  5 15 16  1  7 10  3 12 11  8  2 18 20  4  6 17 14 13 19  9
 15 16  1  7 10  3 12 11  8  2 18 20  4  6 17 14 13 19  9  5
```

```
C   PROGRAMME PRINCIPAL DANS LE PREMIER ALGORITHME
      IMPLICIT INTEGER (A-Z)
      DIMENSION FONC(1000,10),AVANC(40),CCLIN(2,43),FONC1(1000,2)
      DIMENSION DEBORD(4000)
      DIMENSION COUT(43,43),LCSUPR(43,2),DELTA(43,2),PILE(2000,4)
C   LECTURE DES DONNEES
      READ(5,100) N
  100 FORMAT(I4)
      DO 15 I=1,N
   15 READ(5,101)(COUT(I,J),J=1,N)
  101 FORMAT(24I3)
      DO 17 I=1,N
   17 COUT(I,I)=2000000
      LLANT=-1
      NIV=1
      KBP=1
      KNIV=0
      SYM=1
      PILE(NIV,1)=0
      PILE(NIV,3)=0
      BOL2=N-1
      BOOL=0
      CALL MINLC(COUT,LCSUPR,N,BZERO,BOOL,DELTA)
      PILE(NIV,2)=BZERO
      DO 16 I=1,N
   16 WRITE(6,102)(COUT(I,J),J=1,N)
C   TEST DE FIN DE L' ALGORITHME
   23 IF(KNIV.NE.(N-1))GO TO 223
      CCLIN(1,KNIV+1)=CCLIN(2,KNIV)
      CCLIN(2,KNIV+1)=CCLIN(1,1)
      KNIV=N
      GO TO 33
  223 IF(NBIT.GT.900)GO TO 33
  224 BOOL=1
C   RECHERCHE DE LA MEILLEURE CASE ADMISSIBLE
      CALL MAXGAM(COUT,LCSUPR,N,K,L,BBAR,BOOL,MK,ML)
      LCSUPR(K,1)=1
      LCSUPR(L,2)=1
      NBIT=NBIT+1
      FONC(NBIT,8)=COUT(L,K)
      FONC(NBIT,1)=K
      FONC(NBIT,2)=L
      COUT(L,K)=2000000
C   GESTION DES PORTIONS DE CIRCUITS ADMISES ET RECHERCHE DE L'ARC IN
      CALL CYCIN(K,L,FONC,CCLIN,NBIT,KNIV,COUT,BOL2)
      CALL MINLC(COUT,LCSUPR,N,B1,BOOL,DELTA)
      IF(B1.GE.0)GO TO 999
      WRITE(6,150)NBIT
  150 FORMAT(10X,'B1 GT 0',I3)
  999 NIV=NIV+1
      PILE(NIV,1)=KBP
      IF((SYM.NE.1).OR.(FONC(NBIT,8).NE.0))GO TO 333
      BBAR=BBAR+DELTA(L,1)+DELTA(K,2)
  333 PILE(NIV,2)=PILE(KBP,2)+BBAR
      PILE(NIV,3)=PILE(KBP,3)+1
      NIV=NIV+1
      PILE(NIV,1)=KBP
      PILE(NIV,2)=PILE(KBP,2)+B1
      PILE(NIV,3)=PILE(KBP,3)+1
      FONC(NBIT,9)=DELTA(L,1)
      FONC(NBIT,10)=DELTA(K,2)
      IF((BBAR.NE.0).OR.(B1.NE.0))GO TO 123
C MARQUAGE DE LA LIGNE K ET DE LA COLONNE L
  124 LCSUPR(K,1)=1
      LCSUPR(L,2)=1
```

```
C CONSTRUCTION DE L'ARBORESCENCE ET DE LA FONCTION F
      FONC(NBIT,3)=0
      FONC1(NBIT,1)=MK
      FONC1(NBIT,2)=ML
      PILE(KBP,4)=NIV
      KBP=NIV
      GO TO 23
  123 IF(B1.EQ.0)GO TO 124
      IF(BBAR-B1)19,21,21
   19 PILE(KBP,4)=NIV-1
      LCSUPR(K,1)=0
      LCSUPR(L,2)=0
      BOOL=0
      COUT(L,K)=FONC(NBIT,8)
      CALL MAXGAM(COUT,LCSUPR,N,K,L,BBAR,BOOL,MK,ML)
      IF((SYM.NE.1).OR.((DELTA(L,1)+DELTA(K,2)).NE.0))GO TO 997
      DO 144 KK=1,N
      IF(LCSUPR(KK,1).NE.0)GO TO 144
      COUT(KK,K)=COUT(KK,K)-DELTA(K,2)
  144 CONTINUE
      DO 145 KK=1,N
      IF(LCSUPR(KK,2).NE.0)GO TO 145
      COUT(L,KK)=COUT(L,KK)-DELTA(L,1)
  145 CONTINUE
      COUT(L,K)=2000000
  997 BORNE=PILE(NIV-1,2)
      NIV1=NIV-1
      DO 141 KK=1,KNIV
      IF(K.NE.CCLIN(1,KK))GO TO 141
      K1=KK
      DO 142 LL=K1,KNIV
      CCLIN(1,LL)=CCLIN(1,LL+1)
  142 CCLIN(2,LL)=CCLIN(2,LL+1)
      KNIV=KNIV-1
      GO TO 115
  141 CONTINUE
   21 PILE(KBP,4)=NIV
      BORNE=PILE(NIV,2)
      BOOL=0
      CALL MINLC(COUT,LCSUPR,N,B1,BOOL,DELTA)
      NIV1=NIV
  115 LL=0
      DO 125 KK=1,N
      IF(DELTA(KK,1).EQ.0)GO TO 126
      LL=LL+1
      DEBORD(LLANT+2*LL)=KK
      DEBORD(LLANT+2*LL+1)=DELTA(KK,1)
      GO TO 125
  126 IF(DELTA(KK,2).EQ.0)GO TO 125
      LL=LL+1
      DEBORD(LLANT+2*LL)=-KK
      DEBORD(LLANT+2*LL+1)=DELTA(KK,2)
  125 CONTINUE
      FONC(NBIT,3)=LL
      FONC(NBIT,4)=LLANT
      LLANT=LLANT+2*LL
      IF(LLANT.GE.4000)WRITE(6,180)
  180 FORMAT(10X,'ATTENTION LA BORNE SUPERIEURE DE DEBORD EST GT 4000
  165 FONC1(NBIT,1)=MK
      FONC1(NBIT,2)=ML
C RECHERCHE DU BOUT PENDANT MINIMAL
      DO 116 KK=2,NIV
      IF(PILE(KK,4).NE.0)GO TO 116
      IF(PILE(KK,2).GT.BORNE)GO TO 116
      KBP=KK
      BORNE=PILE(KBP,2)
  116 CONTINUE
```

```
C    RECHERCHE DU CHEMIN ALLANT DU SOMMET ACTUEL AU BOUT PENDANT
      IF((KBP.EQ.NIV).OR.(KBP.EQ.(NIV-1)))GO TO 23
      IF(PILE(KBP,3).LE.PILE(NIV,3))GO TO 198
      DNIV=PILE(KBP,3)-PILE(NIV,3)
      KBP1=KBP
      AVANC(1)=KBP1
      DO 199 KK=1,DNIV
      KBP1=PILE(KBP1,1)
  199 AVANC(KK+1)=KBP1
      LL=DNIV+1
  196 K1=PILE(KBP1,1)
      K2=PILE(NIV1,1)
      CALL REBROU(COUT,LCSUPR,N,FONC,FONC1,NIV1,CCLIN,KNIV,DEBORD,SYM
      IF(K1.EQ.K2)GO TO 96
      AVANC(LL+1)=K1
      LL=LL+1
      KBP1=K1
      NIV1=K2
      GO TO 196
   96 PILE(K1,4)=AVANC(LL)
      DO 200 KK=1,LL
      K1=AVANC(LL-KK+1)
  200 CALL PROGR(COUT,LCSUPR,N,FONC,FONC1,K1,CCLIN,KNIV,DEBORD,SYM)
      GO TO 23
  198 LL=0
      DNIV=PILE(NIV,3)-PILE(KBP,3)
      KBP1=NIV1
      IF(DNIV.EQ.0)GO TO 298
      DO 299 KK=1,DNIV
      K1=PILE(KBP1,1)
      CALL REBROU(COUT,LCSUPR,N,FONC,FONC1,KBP1,CCLIN,KNIV,DEBORD,SYM
  299 KBP1=K1
  298 KBP2=KBP
  297 K1=PILE(KBP1,1)
      K2=PILE(KBP2,1)
      LL=LL+1
      AVANC(LL)=KBP2
      CALL REBROU(COUT,LCSUPR,N,FONC,FONC1,KBP1,CCLIN,KNIV,DEBORD,SYM
      IF(K1.EQ.K2)GO TO 197
      KBP1=K1
      KBP2=K2
      GO TO 297
  197 PILE(K1,4)=AVANC(LL)
      DO 250 KK=1,LL
      K1=AVANC(LL-KK+1)
  250 CALL PROGR(COUT,LCSUPR,N,FONC,FONC1,K1,CCLIN,KNIV,DEBORD,SYM)
      GO TO 23
   33 WRITE(6,133)NBIT,PILE(NIV,2)
  133 FORMAT(10X,'NBRE D'' ITERATIONS',I3,'OPTIMUM',I3)
      WRITE(6,102)(CCLIN(1,KK),KK=1,KNIV)
      WRITE(6,102)(CCLIN(2,KK),KK=1,KNIV)
  102 FORMAT(4X,42I3/)
      STOP
      END
```

SION DU RESEAU 20

BORNE MINORANTE ABSOLUE 429

NOMBRE D'ITERATIONS NECESSAIRES 30
POUR ARRIVER SUR L'OPTIMUM SUIVANT 440

CIRCUIT HAMILTONIEN OPTIMAL ASSOCIE

```
4 13 19  9 11  8  2 12 18 15  5 10  3 16  1  7 20  4  6 17
3 19  9 11  8  2 12 18 15  5 10  3 16  1  7 20  4  6 17 14
```

SION DU RESEAU 10

BORNE MINORANTE ABSOLUE 75

NOMBRE D'ITERATIONS NECESSAIRES 28
POUR ARRIVER SUR L'OPTIMUM SUIVANT 85

CIRCUIT HAMILTONIEN OPTIMAL ASSOCIE

```
4  8  2  6  5  1  7 10  3  9
8  2  6  5  1  7 10  3  9  4
```

B I B L I O G R A P H I E

1. BELLMANN : Dynamic programming treatment of the traveling salesman problem.

2. R. GOMORY : The traveling salesman problem
Procedings IBM Scientific Computing Symposium
Combinatorial Problems 1964 pp. 93-121

3. C. BERGE : Théorie des graphes et ses applications
DUNOD, PARIS, 1958

4. FORD L.R. et FULMERSON D.R. : Flots dans un graphe
Gauthier Villards Paris, 1967

5. A. KAUFMANN : Introduction à la combinatorique
Dunod, Paris, 1969.

6. B. ROY et M. SIMMONARD : "Nouvelle méthode permettant d'explorer un ensemble de possibilités et à déterminer un optimum
Revue Française de Recherche Opérationnelle, 18, 1961

7. M. SIMMONARD : Programmation linéaire, technique de calcul aconomique
Tome 2, Dunod, Paris, 1973.

o o o
o o
o

NOM DE L'ETUDIANT : GOEPP Dominique

NATURE DE LA THESE : DOCTORAT DE SPECIALITE en MATHEMATIQUES

VU, APPROUVE

& PERMIS D'IMPRIMER

NANCY, le 19/11/1975

LE PRESIDENT DE L'UNIVERSITE DE NANCY I