RECHERCHE OPÉRATIONNELLE : Optimisation Combinatoire

JACQUES CARLIER

Table des matières

COURS

I

A. INTRODUCTION

La Recherche Opérationnelle est souvent réduite par les personnes extérieures à cette discipline à ses aspects mathématiques. Pourtant, et dès l'origine, elle vise essentiellement à la résolution de problèmes pratiques qui sont des défis au sens commun. C'est pourquoi il m'a paru nécessaire, dans cette introduction, de définir la Recherche Opérationnelle. Je m'appuie pour cela sur les écrits de **Robert FAURE**.

Les problèmes combinatoires sont des défis au sens commun. Ils sont donc étudiés mais ils restent méconnus y compris par de nombreux informaticiens. En effet, certains croient en la puissance absolue de l'ordinateur et ne s'inquiètent pas des problèmes de taille de données ni de complexité des algorithmes. D'autres se font une montagne des problèmes dits NP-difficiles. Or, ma déjà longue lutte contre les problèmes combinatoires m'a appris que la situation réelle est beaucoup plus complexe, en particulier si on se place d'un point de vue pratique. Certains d'entre eux sont effectivement simples et relèvent d'une heuristique, type recuit simulé, méthode tabou ou génétique. D'autres, au contraire, demandent des études fines et des programmes spécifiques. Mais, dans tous les cas, l'intervention humaine est cruciale. Il faut savoir exploiter les spécificités d'un problème et c'est une erreur de croire à l'automaticité de sa résolution.

1. La Recherche Opérationnelle

Commençons par citer Robert FAURE qui a été un des principaux initiateurs de la R.O. en France...

a) Le caractère pratique de la Recherche Opérationnelle :

Définition

"La recherche opérationnelle a été, reste et demeurera l'art d'intervenir rapidement au profit d'une entité économique déterminée (agent ou collectivité) dans une situation difficile afin de tenter d'en améliorer l'issue".

b) Heuristique et traitement interactif :

Définition

"Depuis toujours, la recherche opérationnelle a institué des méthodes heuristiques incapables de fournir l'optimum formel mais susceptibles d'aboutir à de bonnes solutions".

c) Déontologie de la R.O. :

Définition

"L'équipe de R.O., en particulier le coordonnateur, doit absolument se garder de considérer qu'il lui revient de conclure son étude par une décision". ... pour arriver à Bernard ROY, une de ses personnalités actuelles les plus marquantes, qui propose le terme "AIDE A LA DECISION".

2. Quelques problèmes de recherche opérationnelle:

a) Les problèmes combinatoires discrets :

Nous illustrons par deux exemples : le problème du voyageur de commerce et le problème de l'arbre minimal.

Exemple

Dans un problème de voyageur de commerce, un VRP doit visiter un certain nombre de villes en minimisant la distance parcourue. Sur la figure ci-dessous le voyageur doit visiter 18 villes en partant de Paris. Ce problème est modélisé par un graphe valué dont les sommets sont les villes et les arêtes les liaisons entre ces villes. Ces arêtes sont valuées par les distances kilométriques. On doit chercher dans ce graphe un cycle Hamiltonien de valeur minimale. Ici il y a 17! circuits possibles. Plus généralement, s'il y a N villes, il y a N-1! circuits. Il est en général impossible d'énumérer. C'est pourquoi ce problème est un défi au sens commun. On verra dans le cours que le problème du voyageur de commerce est probablement de complexité exponentielle (il est NP-difficile). Toutefois, il y a des méthodes qui permettent de résoudre pratiquement des problèmes de grande taille.

Figure 0.1 Carte des villes d'un problème de voyageur de commerce

Exemple

Dans le problème de l'arbre minimal on doit relier N sites pour un coût global minimal de façon à ce que tout site puisse communiquer avec tous les autres. On verra qu'il faut chercher un arbre recouvrant de coût minimal et que ce problème est facile car pouvant être résolu par un algorithme polynomial. Un exemple de graphe est rapporté sur la figure 0.2 ainsi que son arbre minimal sur la figure 0.3. Il a été obtenu en retenant successivement les arêtes de plus petits coûts, sous réserve qu'elles soient "utiles".

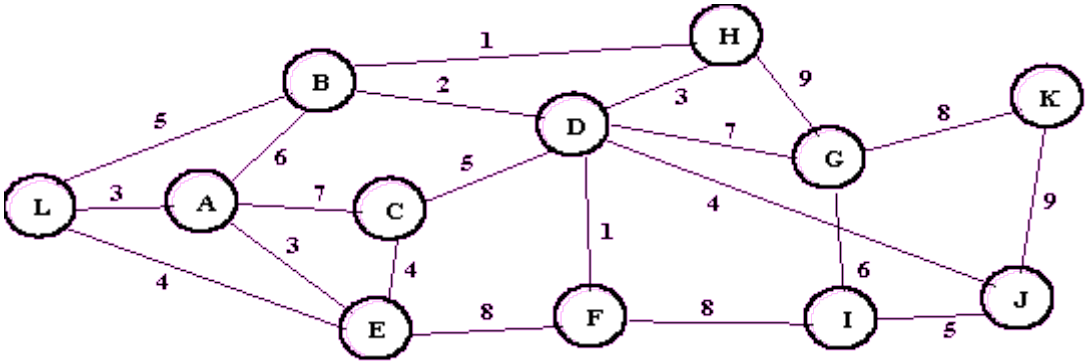


Figure 0.2 : graphe

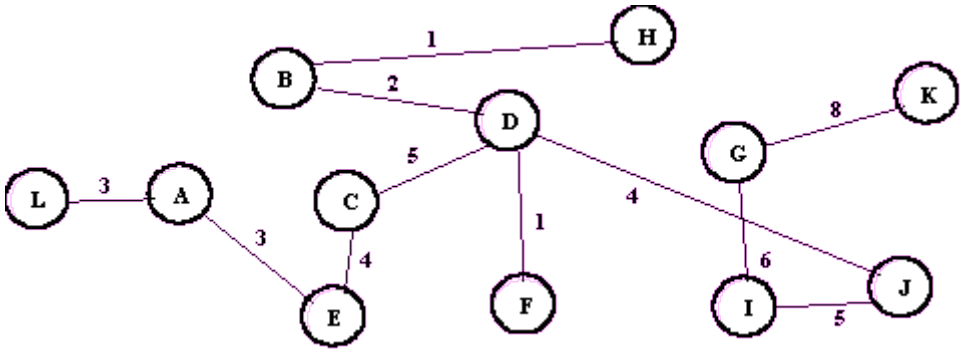


Figure 0.3 : arbre minimal

Remarque

L'UV RO03 est consacrée aux méthodes traitant les problèmes combinatoires discrets.

b) Les problèmes combinatoires continus :

Un pays veut acheter des armes et s'adresse à un marchand d'armes international qui possède des stocks volés ou achetés. Celui-ci propose 2 types de lots. Le premier type de lot contient 100 mitraillettes, 200 gilets pare-balles, 5 auto-mitrailleuses légères et 50 bazookas. Le deuxième type de lot contient 50 mitraillettes, 100 gilets pare-balles, 10 auto-mitrailleuses légères et 100 bazookas. Un lot de type 1 coûte p_1 francs et un lot de type 2, p_2 francs. Le pays désire acheter au minimum 1000 mitraillettes, 2500 gilets pare-balles, 30 auto-mitrailleuses légères et 250 bazookas. Si on note x_1 le nombre de lots 1 et x_2 le nombre de lots 2 qu'il achète alors il doit résoudre le programme linéaire suivant :

Minimiser $p_1 x_1 + p_2 x_2$

sous les contraintes :

$100 x_1 + 50 x_2 \geq 1000$

$200 x_1 + 100 x_2 \geq 2500$

$5 x_1 + 10 x_2 \geq 30$

$50 x_1 + 100 x_2 \geq 250$

$x_1 \geq 0 \; et \; x_2 \geq 0$ et x_1 et x_2 entiers

On parle de programme linéaire car les contraintes et la fonction économique sont linéaires. Ce programme est discret car les variables sont supposées entières. Cette intégrité des variables rend la résolution difficile. C'est pourquoi, on relâche la contrainte d'intégrité et on suppose les variables réelles, ce qui rend le problème

facile. Il est donc facile (polynomial) dans le cas continu et difficile (NP-difficile) dans le cas discret. Pour obtenir une solution entière, on pourra, par exemple, retenir les parties entières de la solution continue. Mais on n'aura pas la solution optimale.

Dans le cas général de fonctionnelle à optimiser sous contraintes, on parle de programmation mathématique. Un exemple est fourni par le problème qu'a eu à résoudre la reine DIDON lors de la fondation de Carthage à savoir : quelle est la figure géométrique de périmètre donné ayant la plus grande surface? La réponse est le cercle.

Remarquons toutefois que la plupart des problèmes de programmation mathématique sont difficiles.

Ces problèmes sont étudiés dans l'UV RO04 où on présente en particulier la méthode du simplexe qui permet de traiter de grands programmes linéaires en continu.

c) Les problèmes aléatoires :

Exemple

Un phénomène aléatoire se présente aux caisses d'un supermarché. Un observateur a pu mesurer la fréquence du nombre de clients qui arrive par minute pendant 30 minutes.

Nombre de clients par minute	0	1	2	3	4	5	6
Fréquence d'observation	4	8	8	6	3	1	0

Tableau 1 : t1

Il a également observé la répartition des temps de services :

Temps de service	effectif des clients
Moins de 1 mn	24
1 à 2 mn	14
2 à 3 mn	8
3 à 4 mn	5
4 à 5 mn	3
5 à 6 mn	2
6 à 7 mn	1
7 à 8 mn	1
8 à 9 mn	1

Tableau 2 : table 2

Ces statistiques permettent d'estimer les lois d'arrivées et des services. Ici par exemple on a des arrivées poissonniènes et des services exponentiels. Il faut alors trouver un compromis entre le temps d'attente des clients et le nombre de caisses ouvertes.

Remarque

D'autres problèmes aléatoires concernent les stocks, le renouvellement d'équipements et la fiabilité.

Ces problèmes sont étudiés dans l'UV RO05 où on présente en particulier les chaînes de Markov et les files d'attente.

d) Schéma du travail d'une équipe de recherche opérationnelle :

Méthode

On a trois étapes. La première étape a pour objet de modéliser le problème et de déterminer le (ou les) critère(s). La seconde étape est la résolution, c'est-à-dire la détermination d'une solution qui paraît bonne (on dira abusivement optimale). La troisième étape est une discussion avec le décideur et, si nécessaire, un retour à la première étape.

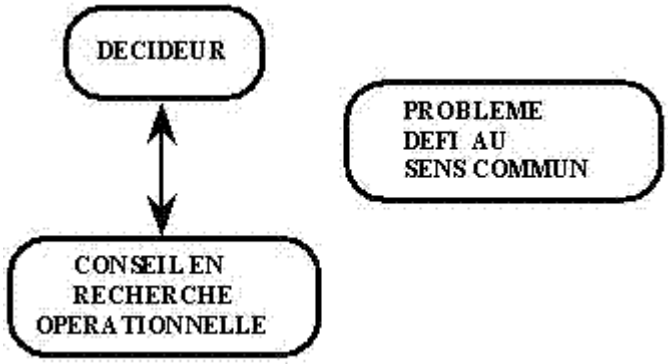


Figure 0.4 : schéma de fonctionnement d'une équipe

3. Résumons :

a) Résumons :

Rappel

La recherche opérationnelle :

- traite un problème pratique
- a un objectif limité (cette application)
- nécessite une boîte à outils (algorithmes et structures des données, optimisation combinatoire, graphes, complexité, programmations linéaire et mathématique, processus stochastiques, probabilités et statistiques, méthodes multicritères....);
- est pluridisciplinaire (Mathématique, Informatique, Economie);
- est banalisée (Programmation linéaire, PERT, ...);
- aide à la décision.

4. Les problémes combinatoires

Parmi les défis au sens commun, il y a donc les problèmes combinatoires pour lesquels il est, a priori, impossible d'énumérer.

On distingue toutefois les problèmes faciles des problèmes difficiles.

a) Problèmes faciles :

Il en est ainsi pour les problèmes de toute petite taille (il faut énumérer !), pour les problèmes fortement contraints (il suffit d'énumérer !), pour les problèmes faiblement contraints (une heuristique type recuit simulé ou méthode tabou sera parfaite surtout dans un contexte industriel) et enfin pour les problèmes résolus polynomialement, encore faut-il le savoir !

b) Problèmes difficiles :

Mon expérience pratique me permet d'affirmer qu'ils sont fréquents et que pour les résoudre il faut utiliser de multiples outils (méthodes sérielles, recuit simulé, méthodes arborescentes, programmation dynamique, algorithme A*....).

Il faut ajouter que pour certains d'entre eux la méthode actuelle la plus efficace est de reproduire l'expertise humaine.

c) Les méthodes automatiques :

(ça n'existe pas encore !) Il y a 40 ans, GOMORY découvrait ses fameuses coupes pour la Programmation Linéaire en Nombres Entiers (PLNE). Or, de très nombreux problèmes combinatoires peuvent se modéliser par un PLNE. A l'époque, on a cru pouvoir résoudre les problèmes combinatoires. L'espoir a été déçu !

40 ans après, on ne sait pas résoudre automatiquement les problèmes combinatoires.

Mais il n'est pas exclu théoriquement que cela soit possible dans l'avenir. Cela revient à dire que les problèmes NP-difficiles pourraient être résolus "pratiquement" de façon polynomiale.

Il faudrait une grande avancée algorithmique

d) Les méthodes semi-automatiques :

Modéliser un problème combinatoire ne sert à rien (1), si on ne décrit pas, en plus, son algorithme de résolution et plus particulièrement de bonnes évaluations, certains disent de bonnes contraintes. D'où l'idée de faire des langages de programmation adaptés au combinatoire et incluant des parcours arborescents avec des choix automatiques.

L'avantage est de pouvoir, au moins pour le spécialiste d'un tel langage, programmer très vite. A mon avis, un tel outil reste ambitieux car on a des surcoûts liés à la rigidité d'un tel système et à la faiblesse de leurs structures de données.

(1)En fait, tout modèle est une simplification de la réalité et, quand on modélise, il faut chercher à obtenir le modèle le plus représentatif que l'on puisse espérer résoudre de façon satisfaisante !

e) Les programmes spécifiques :

Je citerai les méthodes arborescentes, les méthodes polyédrales et la programmation dynamique, mais aussi les heuristiques. L'expérience de résolution des problèmes combinatoires montre l'importance cruciale pour construire une méthode efficace des points suivants :

- une modélisation adaptée;
- La complexité des algorithmes et des structures de données;
- La proximité de la machine (langage procédural, par exemple);
- la proximité du problème (si on n'a pas d'excellentes évaluations, celles-ci sont inutiles).

* *

*

Ce serait une galéjade de prétendre enseigner toute l'optimisation combinatoire en

une soixantaine d'heures (cours et exercices inclus). Nous ne le prétendrons pas. Nous insisterons sur les idées principales et la présentation des algorithmes les plus fondamentaux. En effet ces algorithmes sont les outils de bases pour des méthodes plus élaborées. Nos objectifs sont de faire prendre conscience de la complexité des problèmes, du danger du combinatoire et de l'utilité des graphes pour modéliser. Espérons que cela vous évitera sur le terrain de concevoir de belles maquettes parfaites pour des exemples d'écoles de petites tailles mais inutilisables sur des problèmes réels.

B. LES GRAPHES

1. Pourquoi les graphes ? :

a) Pourquoi les graphes ? :

Définition

Les graphes sont des outils irremplaçables pour modéliser et résoudre de nombreux problèmes concrets. En effet, ils permettent, d'une part de guider l'intuition lors d'un raisonnement, d'autre part de se rattacher aux résultats connus de la théorie des graphes. Nous les définissons succinctement ci-dessous et illustrons leur intérêt par quelques exemples.

Définition

Un graphe orienté est un couple **G = (X,U)**, où X est un ensemble dont les éléments sont appelés sommets et U une partie de X x X dont les éléments sont appelés arcs.

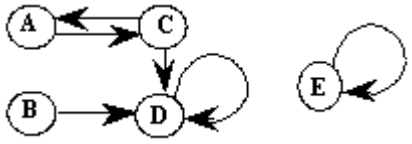


Figure 1-1

Exemple

Un premier exemple de graphe est dessiné sur la figure 1.1. Il est donné par :

X = {A, B , C , D , E} et U = {(A , C) (C ,A) (C ,D) (B ,D) (D ,D) (E , E) }

Exemple

On a 3 bocaux de contenances respectives 8, 5 et 3 litres. Initialement, le plus grand bocal est plein, les autres sont vides. On veut atteindre la situation où les 2 plus grands bocaux contiennent 4 litres sous la contrainte suivante :

quand on vide un bocal dans un autre, soit on remplit l'autre bocal entièrement, soit on vide le premier entièrement.

A ce problème, on associe un graphe dont les sommets représentent les états du système. Un sommet est un triplet (i ,j ,k) où i, j, et k sont les contenus des 3 bocaux. Les arcs correspondent aux possibilités de transition entre états (Cf. figure 1.2).

Définition

Un graphe non orienté est un couple G=(X,E), où X est un ensemble de sommets

et E un ensemble de paires de sommets appelées arêtes. On a un multigraphe quand on autorise l'existence simultanée de plusieurs paires identiques.

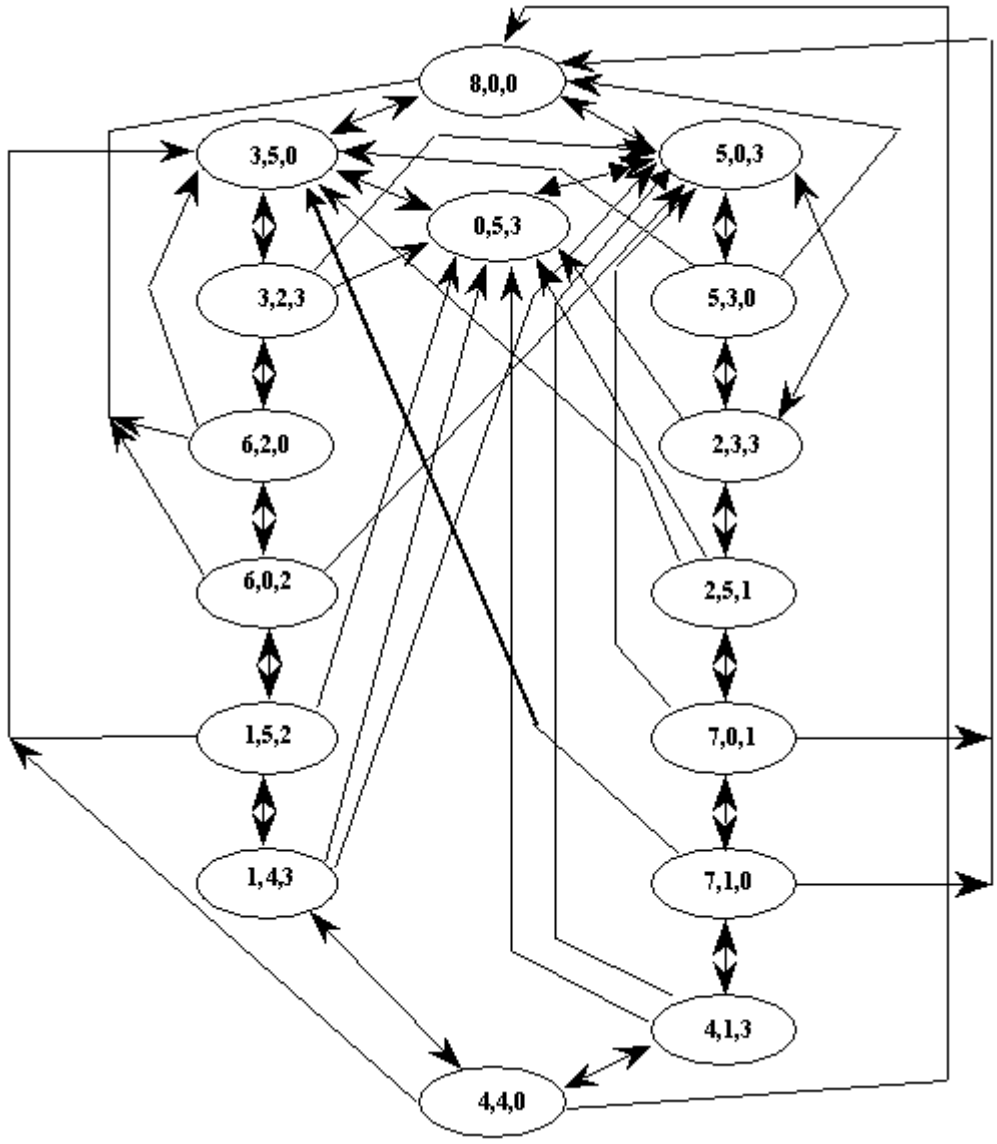


Figure 1-2

Exemple : Exemple 3 :

Le pont de Königsberg (Kaliningrad) surplombe la Pregel. Au milieu de cette rivière, on a l'île de Kneiphoff (Cf figure 1.3).

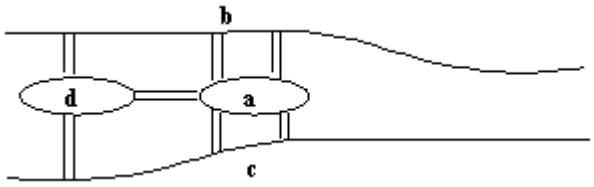


Figure 1-3

Un piéton peut-il traverser une fois et une seule chaque pont ? En 1736, Euler a démontré que cela est impossible. Pour le montrer, il a associé au problème le multigraphe de la figure 1.4.

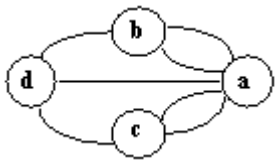


Figure 1-4

A chaque portion de terre correspond un sommet, à chaque pont, une arête. Le problème se formule alors ainsi : existe-t-il une chaîne passant une fois et une seule par chaque arête du multigraphe (on parlera de chaîne eulérienne) ? La réponse résulte du théorème suivant que nous ne démontrerons pas :

Fondamental : Théorème d'Euler

Un multigraphe simple (sans boucle) G admet une chaîne eulérienne si et seulement si il est connexe ("d'un seul tenant") et si le nombre de sommets de degré impair est 0 ou 2.

Dans le cas des ponts de Königsberg, les 4 sommets du graphe de la figure 1.4 sont de degré impair, il n'y a donc pas de solution.

2. Vocabulaire de la théorie des graphes :

Le but de ce paragraphe est de donner les définitions de base de la théorie des graphes. Ces définitions sont nombreuses mais très intuitives, ce qui facilite leur apprentissage !

a) Graphe orienté et non orienté, graphe valué :

Définition : graphe orienté

Un graphe orienté est un couple **G = (X,U)**, où X est un ensemble dont les éléments sont appelés **sommets** et U une partie de X x X dont les éléments sont appelés arcs.

Définition : graphe non orienté

Un graphe non orienté est un couple G=(X,E), où X est un ensemble dont les éléments sont appelés **sommets** et E un sous-ensemble de parties de X contenant chacune au plus 2 éléments et dont les éléments sont appelés **arêtes**.

Remarque

on notera n le nombre de sommets d'un graphe et m son nombre d'arcs (ou d'arêtes).

(cf. 'Figure 1-5' p 15)A un graphe orienté, on associe un graphe non orienté en "laissant tomber l'orientation". De même à un graphe non orienté, on associe un graphe orienté, soit en introduisant pour toute arête les deux arcs qui lui correspondent, soit en choisissant un seul des deux arcs (Cf. Figure 1.5). (cf. 'Figure 1-5' p 15)

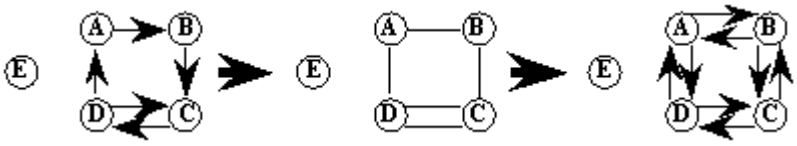


Figure 1-5

Définition : Graphe valué :

Un graphe valué est un triplet G = (X, U, v) où (X, U) est un graphe et v une application de U dans IR (ensemble des réels).

Définition : Extrémité initiale et terminale (successeur et prédécesseur) :

Soit un arc (i,j); i est dit **extrémité initiale** de l'arc (i ,j), j, **extrémité terminale**. On dit aussi que j est un **successeur** de i, et i un **prédécesseur** de j. L'arc (i ,j) est dit **incident** vers l'extérieur en i et vers l'intérieur en j.

On note $U^+(i)$ l'ensemble des successeurs de i, $U^-(i)$ l'ensemble des prédécesseurs de i, $d^+(i)$ le demi-degré extérieur de i, c'est-à-dire le cardinal de $U^+(i)$, et $d^-(i)$ le demi-degré intérieur de i, c'est-à-dire le cardinal de $U^-(i)$ (Cf. figure 1.6).

b) Degré :

Définition : Degré d'un sommet :

Le degré d'un sommet est le nombre d'arcs incidents en ce sommet. Quand il n'y a pas de boucle en un sommet, c'est-à-dire d'arc dont l'extrémité initiale se confond avec l'extrémité terminale, son degré est la somme de son demi-degré intérieur et de son demi-degré extérieur (Cf. figure 1.6).

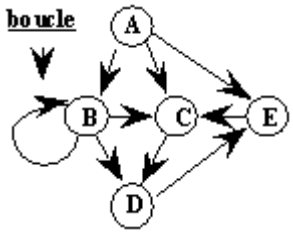


Figure 1-6

Sur la figure 1.6 on a : $U^-(E) = \{A, D\}$ $d^-(E) = 2$ $d^-(A) = 0$ $d(C) = 4$ $U^+(E) = \{C\}$ $d^+(E) = 1$ $d^+(A) = 3$ $d(B) = 4$

c) Chemins, circuits

Définition : Chemins

Un **chemin** est une suite de sommets $[x_0, x_1, \ldots, x_p]$ telle que les arcs (x_0, x_1), (x_1, x_2), ... , (x_{p-1}, x_p) appartiennent au graphe. La valeur d'un chemin est alors la somme des valuations des arcs de ce chemin. La longueur d'un chemin est son nombre d'arcs.

Le sommet x_0 est appelé **extrémité initiale** du chemin et le sommet x_p, **extrémité terminale**.

Exemple

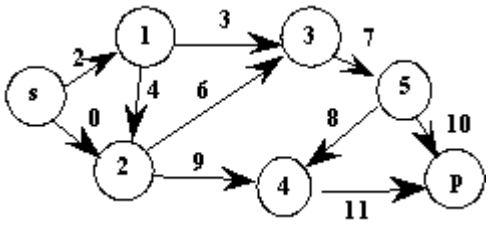


Figure 1-7

Par exemple sur le graphe de la figure 1.7 la valeur du chemin {s, 1 ,2 ,4 ,p} est 26 et sa longueur 4.

Définition : Circuit :

Un **circuit** est un chemin dont l'extrémité initiale se confond avec l'extrémité terminale.

Définition : Chemin élémentaire et simple:

Un chemin est dit **élémentaire** (resp. **simple**) s'il ne passe pas deux fois par le même sommet (resp. arc).

Exemple

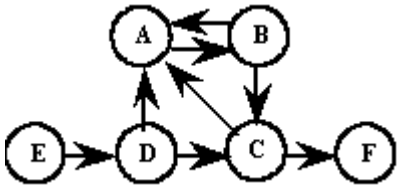


Figure 1-8

Sur la figure 1.8 : Les chemins [E ,D ,C ,F] et [E ,D ,A ,B ,C , F] sont élémentaires alors que le chemin [E ,D ,A ,B ,A ,B ,C ,F] n'est pas élémentaire.

Définition : Chemin eulérien et hamiltonien :

Un chemin est dit **eulérien** (resp **hamiltonien**) s'il passe une fois et une seule par chaque arc (resp. sommet) du graphe.

Définition : Descendant, ascendant :

Un sommet j est un **descendant** d'un sommet i s'il existe un chemin allant de i à j ou si i=j; on dit alors que i est un **ascendant** de j.

Définition : Source, puits :

Une **source** s est un sommet ascendant de tous les autres sommets, un **puits** p, un sommet descendant de tous les autres sommets.

Note : Un synonyme de source (resp. puits) est **racine** (resp. **antiracine**).

d) Chaînes, cycles :

Définition : Chaîne :

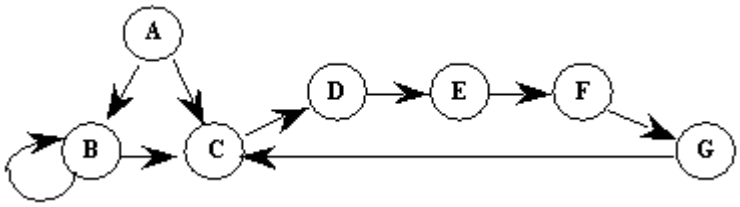


Figure 1-9

Une chaîne est une suite d'arcs u_1, u_2 , ... , u_p telle qu'une extrémité de l'arc u_i ($2 \leq i \leq p-1$) est commune avec l'arc u_{i+1}, alors que l'autre extrémité est commune avec l'arc u_{i-1}. La longueur d'une chaîne est son nombre d'arcs.

Sur la figure 1.9, (A B) (B C) (G C) (F G) (E F) (D E) est une chaîne.

Une chaîne peut être "vue" comme une suite de sommets telle que deux sommets consécutifs soient liés par un arc, certains arcs peuvent être parcourus dans le sens de la chaîne (sens +), les autres arcs dans le sens opposé (sens -).

Définition : Cycle :

Un cycle est une chaîne dont l'extrémité initiale se confond avec l'extrémité terminale.

On définira de même une chaîne simple, hamiltonienne, élémentaire, eulérienne, un cycle hamiltonien, eulérien.

e) Graphe partiel, sous-graphe :

Définition : Sous-graphe :

Soit G=(X,U) un graphe. Le sous-graphe associé au sous-ensemble A de X est par définition le graphe GA défini par : GA = ($A, U \cap A \times A$).

Exemple

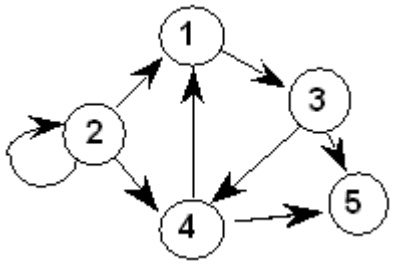


Figure 1-10

Considérons l'exemple de la figure 1.10 où n = 5 et m = 8, le sous-graphe d'ensemble A = {1, 4, 5} est rapporté sur la figure 1.11.

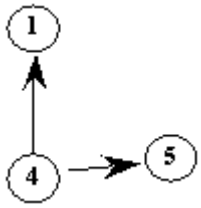


Figure 1-11

Définition : Graphe partiel :

Un **graphe partiel** G' de G est un graphe ayant même ensemble de sommets que G et dont l'ensemble des arcs est inclus dans l'ensemble des arcs de G : G' = (X ,U'

) avec U' sous-ensemble de U.

Exemple

sur la figure 1.12 est rapporté un graphe partiel du graphe de la figure 1.10.

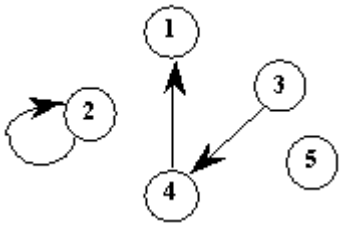


Figure 1-12

Un **sous-graphe partiel** est un sous-graphe d'un graphe partiel.

f) Connexité :

Définition : Connexité simple (non orienté) :

La relation i C j (lire i connecté à j) si i = j ou s'il existe une chaîne allant de i à j est une relation d'équivalence dont les classes sont appelées composantes connexes.

Définition : Connexité forte (orienté)

La relation i FC j si i = j ou s'il existe un circuit passant par i et j est une relation d'équivalence dite de forte connexité. Les classes s'appellent les composantes fortement connexes.

Exemple

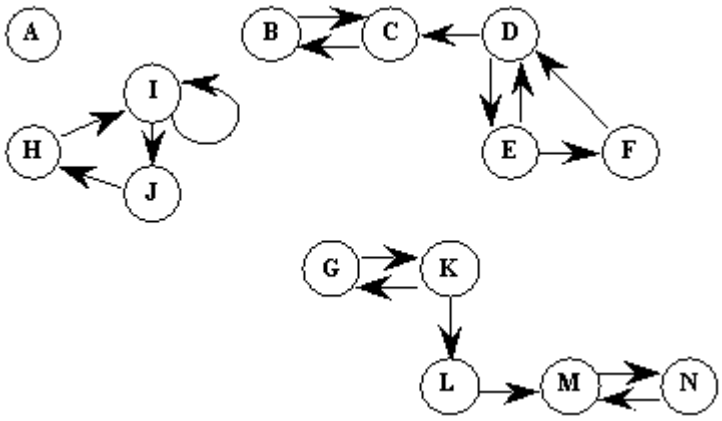


Figure 1-13

Dans l'exemple de la figure 1.13, on a les 4 composantes connexes : {A}, {B, C, D, E, F}, {H, I, J}, {G, K, L, M, N }.

Dans l'exemple précédent, on a les 7 composantes fortement connexes :{A}, {B, C}, {D, E, F}, {H, I, J}, {G, K}, {L}, {M, N}.

Définition : Graphe réduit :

Le graphe réduit GR est le graphe (X /FC ,V) dont l'ensemble des sommets est l'ensemble des classes de forte connexité et dont tout arc relie deux classes distinctes dont deux éléments sont reliés dans G. Le graphe réduit est sans circuit.

Exemple

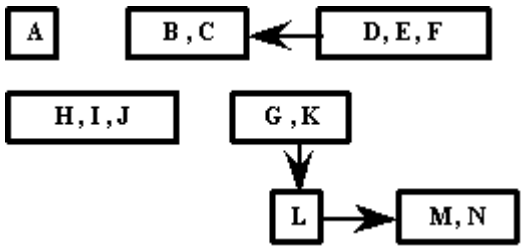


Figure 1-14

Sur la figure 1.14 on a dessiné le graphe réduit de la figure 1.13.

g) Cocycle, cocircuit :

Définition : Cocycle :

Un **cocycle** est un ensemble d'arcs reliant deux parties complémentaires du graphe. Si $Y \subseteq X$, on note $\omega(Y)$ le cocycle associé, on a donc $\omega(Y)=\omega(X-Y)$.

Définition : Cocircuit :

Un **cocircuit** est un cocycle dont tous les arcs sont dans le même sens.

Exemple

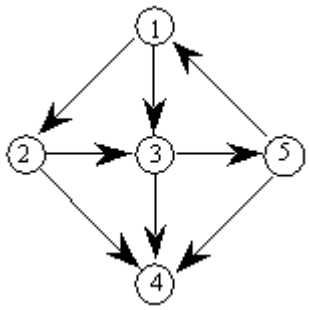


Figure 1-15

Sur la figure 1.15, $\omega(4)$ = {(2,4),(3,4),(5,4)} est un cocircuit, alors que ω {(1,2)} = {(2,4),(2,3),(1,3),(5,1)} n'est pas un cocircuit.

h) Fermeture transitive :

Définition : Fermeture transitive :

La fermeture transitive d'un graphe G =(X,U) est le graphe G"=(X ,U") de même ensemble de sommets que G et tel que (i ,j) ∈ U" s'il existe dans G un chemin allant de i à j.

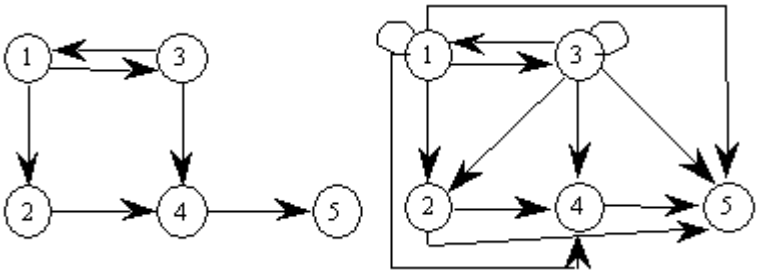


Figure 1-16 : G = (X , U) et G" = (X, U")

i) Nombre chromatique et nombre de stabilité :

Définition : Stable :

Un sous-ensemble S de X est stable s'il n'y a pas d'arc reliant deux sommets de S. Le nombre de stabilité $\alpha(G)$ est le cardinal maximal d'un ensemble stable.

Définition : Nombre chromatique :

Un graphe G est c-chromatique s'il est possible d'en colorier les sommets avec c couleurs sans que deux sommets adjacents soient de même couleur. Le nombre chromatique $\gamma(G)$ est le plus petit c tel que G soit c-chromatique.

Exemple

On considère un graphe dont l'ensemble des sommets est l'ensemble des examens du second semestre à l'U.T.C : deux sommets sont reliés par un arc si un même étudiant doit passer les deux examens correspondants. On veut déterminer un emploi du temps en un nombre minimal de périodes (une période étant égale à une demi-journée). Chaque couleur va correspondre à une période de temps durant laquelle se passera l'ensemble des examens de cette couleur.

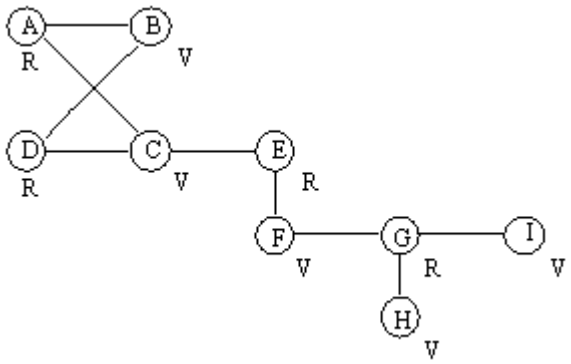


Figure 1-17

Sur la figure 1.17 : A ,D ,E ,G sont de couleur rouge et B ,C ,F ,H ,I sont de couleur verte. Le nombre chromatique $\gamma(G)=2$.

3. Graphes particuliers :

a) Forêt

Définition : Forêt :

Une forêt est un graphe sans cycle (Cf figure 1.18)

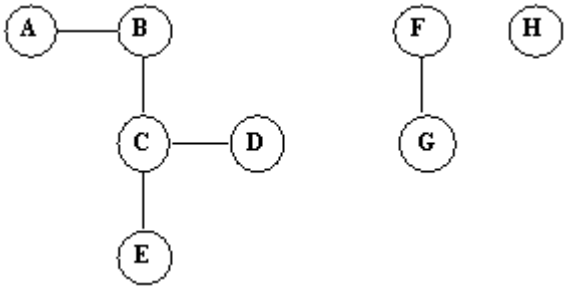


Figure 1-18

b) Arbre :

Définition

Un arbre est un graphe connexe sans cycle (avec $I X I \geq 2$).

Sur la figure 1.19 est dessiné un arbre avec 9 sommets et 8 arêtes.

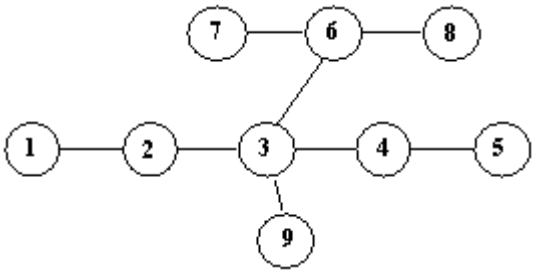


Figure 1-19

Propriétés caractéristiques (voir TD):

- Il est connexe et comporte n-1 arêtes.
- Il est sans cycle et comporte n-1 arêtes.
- Il est connexe et si on enlève une arête il n'est plus connexe
- Il est sans cycle et on crée un cycle si on ajoute une arête.
- Il existe une chaîne élémentaire et une seule entre toute paire de sommets.

c) Arborescence :

Définition : Arborescence :

On appelle **arborescence** un arbre orienté tel que chaque sommet (sauf un) a exactement un prédécesseur. Le sommet sans prédécesseur est la racine de l'arborescence.

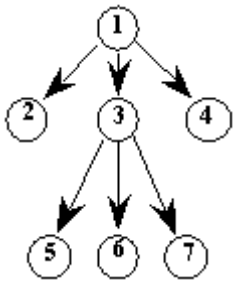


Figure 1-20

Pour tout sommet, il existe un chemin et un seul reliant la racine à ce sommet. Les sommets sans successeur sont appelés **sommets pendants** ou **sommets terminaux**.

d) Graphe biparti :

Exemple : Bataille d'Angleterre.

En 1941, les escadrilles anglaises se composaient d'avions biplaces mais certains pilotes ne pouvaient pas faire équipe avec certains mécaniciens pour des raisons de langues ou d'habitudes.

A ce problème, on associe un graphe dont l'ensemble des sommets comprend les pilotes et les mécaniciens. Un pilote étant "relié" à un mécanicien, s'ils peuvent embarquer tous deux dans le même avion.

Dans la figure 1.21, les traits gras représentent un couplage entre l'ensemble des pilotes et l'ensemble des mécaniciens.

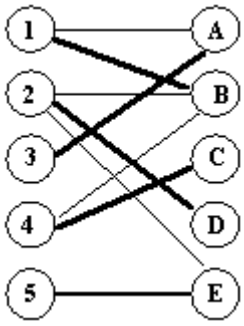


Figure 1-21

Définition : Couplage :

Un couplage est un ensemble d'arcs tel que deux arcs de cet ensemble n'ont pas d'extrémité commune.

Exemple

{ (1 ,B), (2 ,D) } est un couplage du graphe de la figure 1.21 alors que { (1 ,B), (2,D), (3 ,A), (4 ,C), (5 ,E) } est un couplage maximal.

e) Graphes planaires :

Définition : Graphes planaires :

Un graphe est dit planaire si on peut le représenter dans le plan de telle sorte que les sommets soient des points distincts et que les arêtes ne s'intersectent pas.

Exemple

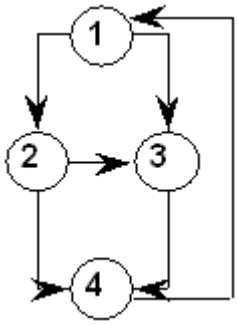


Figure 1-22 : exemple de graphe planaire

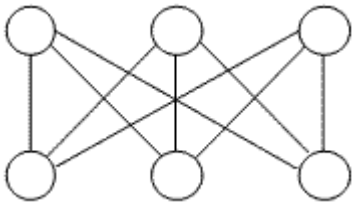


Figure 1-23 : graphe des 3 usines (non planaire)

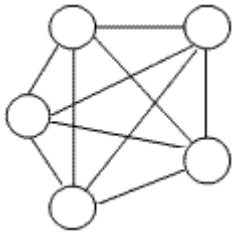


Figure 1-24 : graphe complet à 5 sommets (non planaire)

Sur les figure 1.23 et 1.24 sont dessinés les 2 graphes non planaires minimaux. En effet, si on enlève une arête à un de ces 2 graphes, le graphe devient planaire. De plus, tout graphe non-planaire contient un sous-graphe partiel qu'on peut mettre en correspondance avec un de ces deux graphes en associant une chaîne du graphe partiel à une arête du graphe (théorème de Kuratowski).

Définition : Conjecture des 4 couleurs (1875 Pertersen) :

"Tout graphe planaire est 4 coloriable" (les sommets sont coloriables en 4 couleurs). Cette conjecture a été "démontrée" à l'aide d'un ordinateur (1982). A l'origine, c'est le problème du coloriage d'une carte de géographie : est-ce qu'une carte de géographie peut être coloriée en 4 couleurs ?

f) Cliques:

Cliques:

Une clique est un graphe complet sans boucles (cas orienté et non orienté). Dans le cas orienté, une clique a n(n-1) arcs et n(n-1)/2 arêtes dans le cas non orienté. Seules les cliques de cardinalité 1,2,3 et 4 sont planaires.

C. ALGORITHMES POLYNOMIAUX DE BASE POUR LES GRAPHES

Nous allons présenter dans ce chapitre deux algorithmes polynomiaux de graphe à savoir un algorithme pour bien numéroter les sommets d'un graphe, et un algorithme pour chercher un couplage de cardinal maximal dans un graphe biparti. Auparavant nous décrirons les différentes structures de données pour représenter un graphe en machine, et nous définirons la notion d'algorithme polynomial. Cette notion est d'importance car basique pour l'efficacité des programmes. Nous verrons que la complexité d'un algorithme est fortement dépendante de la structure de données utilisée et que certains problèmes sont intrinsèquement exponentiels. D'autres algorithmes de graphes seront étudiés dans la suite du cours et en travaux dirigés.

1. Définition de la polynomialité :

a) Définition de la polynomialité :

Définition

On dit qu'une fonction f(x) est en $O(x^p)$, s'il existe un polynôme P(x) de degré p tel que $\forall\ x \in N, f(x) \leq P(x)$.

On distingue pour les algorithmes les complexités en mémoire et en temps de calcul. La première donne une borne supérieure du nombre de mots mémoires nécessaires pour représenter les données dans une structure. La seconde donne une borne supérieure du nombre d'opérations élémentaires nécessaires pour exécuter un algorithme. Dans les 2 cas x représente la taille de la donnée du problème à traiter, ou plus précisément le nombre de digits de la donnée.

Remarque

Pour les graphes on exprime la complexité en fonction du nombre n de sommets et du nombre m d'arcs.

2. Codage des graphes :

Il existe 4 structures de données principales pour coder un graphe : la matrice d'adjacence, la matrice d'incidence, la file des successeurs, et la file des prédécesseurs.

a) Matrice d'adjacence :

Définition

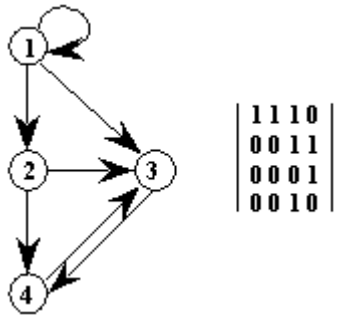


Figure 2.1 : exemple de graphe avec sa matrice d'adjacence

La matrice d'adjacence sommets-sommets est la matrice carrée d'ordre n notée $A=(a_{ij})\ telle\ que : a\ ij=1\ si\ (i,j)\in U, a_{ij}=0\ sinon.$

Remarque

Le codage par la matrice associée est en **O(n²)** (complexité du codage) car ce codage utilise n^2 mots mémoires. Cela est donc coûteux, mais c'est la façon la plus simple de coder un graphe.

b) Matrice d'incidence sommets-arcs :

Définition

La matrice d'incidence sommets arcs N est définie par N_{ij}:

N_{ij} = 1 si le sommet i est extrémité initiale de uj.

N_{ij} = -1 si i est extrémité terminale de uj.

N_{ij} = 0 sinon.

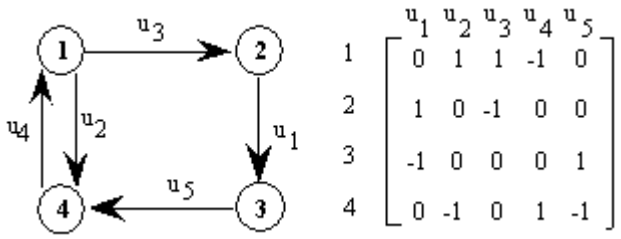


Figure 2.2 : graphe et sa matrice d'incidence

Remarque

La complexité du codage en fonction de n, le nombre de sommets et m, le nombre d'arcs est en O(n x m). Cette matrice est utile quand on a besoin de modéliser les problèmes de graphes par programmation linéaire (flot par exemple).

c) File des successeurs :

Définition

La file des successeurs est formée de 2 tableaux, ALPHA et BETA. BETA est la liste des successeurs des sommets rangés dans l'ordre de leur numérotation, et ALPHA le tableau donnant à un numéro de nœud donné l'emplacement dans le tableau BETA du premier successeur de ce nœud. Cela suppose que les sommets soient numérotés dans l'intervalle [1..n] (n étant le nombre de nœuds du graphe).

Remarque

Les informations relatives aux successeurs du sommet I se trouvent dans le tableau BETA entre les adresses ALPHA(I) et ALPHA(I+1)-1.

La complexité du codage est en O(N + M).

Cette structure est très utilisée, car elle est très économe en place mémoire pour les graphes peu denses et efficace en temps de calcul, quand on a besoin d'un accès direct aux successeurs des sommets.

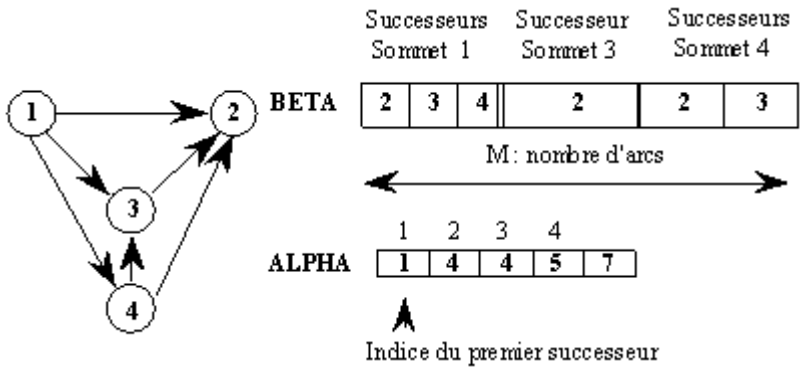


Figure 2.3 : exemple de graphe et de sa file de successeurs

d) File des prédécesseurs :

Cette file est définie de façon analogue. Elle permet un accès direct aux prédécesseurs.

3. Algorithme de bonne numérotation d'un graphe :

a) Algorithme de bonne numérotation d'un graphe :

Définition : Bonne numérotation :

On dit que numéro(x) (application de X -> [1..n]) est une bonne numérotation si, pour tout arc (x,y) appartenant à U, numéro(x) est strictement plus petit que numéro(y).

Nous allons montrer ci-dessous que pour qu'il existe une bonne numérotation il faut et il suffit que le graphe soit sans circuit. Nous énonçons également un résultat qui nous permet de construire un algorithme de complexité linéaire pour déterminer une bonne numérotation d'un graphe sans circuit.

Proposition 1

Une condition nécessaire et suffisante pour qu'un graphe soit sans circuit est que tout sous-ensemble de sommets A non vide admette au moins un élément dont tous les prédécesseurs sont dans le complémentaire de A. C'est-à-dire le sous-graphe GA a au moins un sommet sans prédécesseur.

Démonstration :

Supposons que G ait un circuit $[x_0, x_1, ..., x_r]$. Posons $A = \{x_0, x_1, ..., x_r\}$ et considérons le sous-graphe G_A : tout sommet de G_A appartenant au circuit a au moins un prédécesseur dans A. Il existe donc un ensemble A pour lequel la propriété est fausse. Réciproquement supposons G sans circuit et la propriété fausse. Il existe donc un graphe G_A dont tous les sommets ont au moins un prédécesseur dans A. Partons de xi0 dans G_A , x_{i0} a un prédécesseur $x_{i1}...x_{ih-1}$ a un prédécesseur x_{ih}. On peut donc construire un chemin $[x_{in}, x_{in-1}, ...x_{i0}]$. Le graphe ayant n sommets distincts, deux sommets de ce chemin sont identiques. Ce graphe a donc un circuit ce qui est absurde. (Q.E.D.).

Proposition 2 :

Une condition nécessaire et suffisante pour qu'un graphe soit sans circuit est qu'il existe une bonne numérotation.

Démonstration :

Si G a un circuit $[x_{i0}, x_{i1},...., x_{ip},x_{i0}]$, il ne peut pas exister de bonne numérotation car sinon on aurait : $numéro(x_{i0}) < numéro(x_{i1}) < ..... < numéro(x_{ip}) < numéro(x_{i0})$. Si G n'a pas de circuit, il existe un sommet sans prédécesseur, d'après la proposition 1, en prenant A = X. On démontre le résultat par récurrence. On numérote 1 ce sommet, et on raisonne sur le sous-graphe obtenu en l'enlevant. Ce sous-graphe a n-1 sommets. Donc, d'après l'hypothèse de récurrence, il a une bonne numérotation. On utilise cette bonne numérotation pour numéroter 2, ..., n, ses sommets, ce qui permet d'obtenir une bonne numérotation globale. (Q.E.D.).

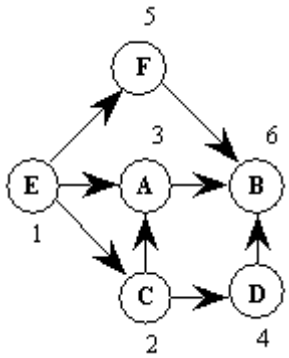


Figure 2.4 : exemple de bonne numérotation.

L'algorithme NUMEROTER ci-dessous permet de bien numéroter les sommets d'un

graphe sans circuit. Le graphe a initialement une numérotation arbitraire. Il est codé par la file des successeurs.

Syntaxe : ALGORITHME NUMEROTER :

(i) Initialisation

Lire N, M, ALPHA, BETA.{ALPHA est un tableau de dimension N+1 et BETA, de dimension M}

(ii) Calcul du nombre de prédécesseurs de chaque sommet

Pour I = 1 à N faire NOMBRE(I) = 0
Pour K = 1 à M faire
Début
I = BETA(K)
NOMBRE(I) = NOMBRE(I) + 1
Fin

(iii) Initialisation de la pile des sommets sans prédécesseurs

SOMMET = 0
Pour I = 1 à N faire
Début
Si NOMBRE(I) = 0 faire
Début
SOMMET = SOMMET + 1
PILE(SOMMET) = I
Fin
Fin

(iv) Numérotation des sommets

Pour J = 1 à N
Début
I = PILE(SOMMET)
SOMMET = SOMMET - 1
NUMERO(I) = J
Pour H = ALPHA(I) à ALPHA(I+1) - 1
Début
L = BETA(H)
NOMBRE(L) = NOMBRE(L) - 1
Si NOMBRE(L) = 0 alors
Début
SOMMET = SOMMET + 1
PILE(SOMMET) = L
Fin
Fin
Fin

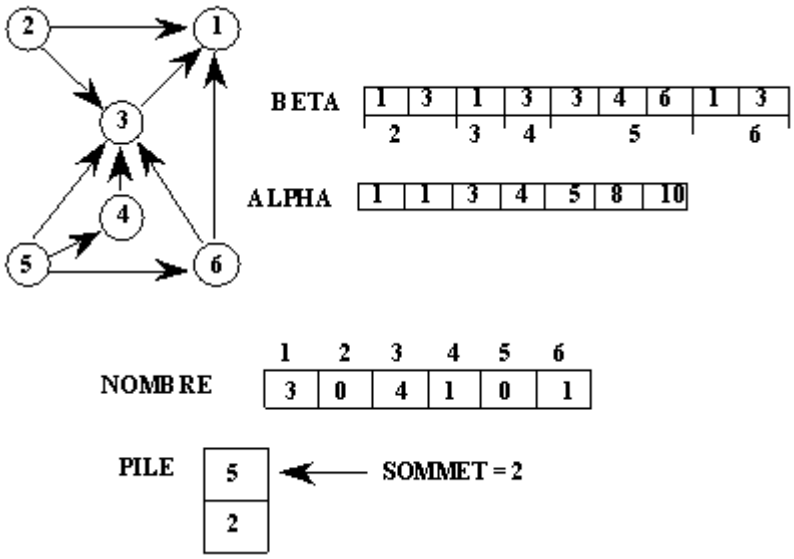


Figure 2.5 : graphe de départ avec initialisation de NOMBRE et PILE

Application Numérique :

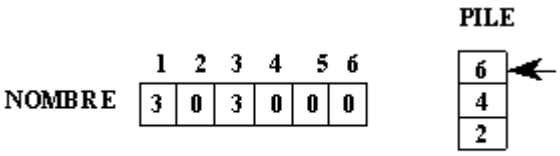


Figure 2.6 évolution de la pile et du tableau après la numérotation de 5

Lors de la numérotation de 5 on obtient :
I = 5
NUMERO(5) = 1
Pour H = 5 à 7
L = 3
NOMBRE(3) = NOMBRE(3) - 1
L = 4
NOMBRE(4) = 0 (Sommet 4 --> PILE)
L = 6
NOMBRE(6) = 0 (Sommet 6 --> PILE)
etc..

Complexité de l'Algorithme :

(i) Initialisation :
Lire N et M est en O(1).
Lire BETA et ALPHA est en O(N + M).
(ii) Calcul de Nombre :
La première boucle est en O(N) alors que la deuxième boucle est en O(M).
(iii) Pile :
La première instruction est en O(1) alors que la boucle est en O(N).
(iv) Numérotation :
La boucle est en O(N) pour les trois premières instructions.

Les autres instructions sont en O(M) car la boucle correspondant à H revient à parcourir une fois le tableau BETA.

4. Recherche d'un couplage maximal :

Nous allons étudier dans ce paragraphe les définitions et les propriétés des couplages, en particulier la notion de chaîne améliorante. On verra qu'un couplage est maximal si et seulement si le graphe ne contient pas de chaîne améliorante. Les algorithmes de recherche d'un couplage maximal sont fondés sur ce résultat. Nous présentons ci-dessous un tel algorithme pour le cas particulier des graphes biparti de complexité $O(n^3)$. Signalons l'existence d'algorithmes polynomiaux dans le cas général, mais ces algorithmes sont trop techniques pour être présentés dans ce cours.

a) Couplage :

Définition : Couplage

Un couplage C est un ensemble d'arêtes (resp. d'arcs) ne contenant pas deux arêtes (resp. arcs) adjacentes à un même sommet. Les arcs du couplage seront dits épais les autres minces.

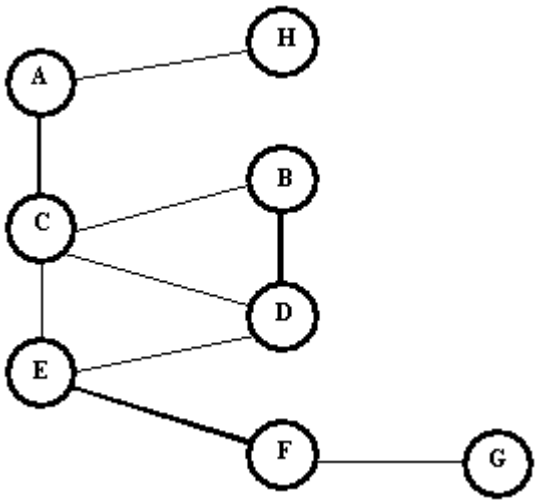


Figure 2.7 : un couplage de cardinal 3.

Définition : Transversal :

Un transversal T d'un graphe est un sous-ensemble de sommets rencontrant au moins une fois chaque arête du graphe.

Définition : Sommet saturé, insaturé :

Pour un couplage C un sommet est dit saturé si un arc du couplage est adjacent à ce sommet sinon il est dit insaturé.

Définition : Chaîne alternée, améliorante :

Une chaîne alternée d'un couplage C est une chaîne du graphe dont les arcs sont alternativement dans le couplage et hors du couplage. Une chaîne améliorante est une chaîne alternée reliant deux sommets insaturés.

Remarque

Une chaîne alternée d'un couplage C est une chaîne du graphe dont les arcs sont alternativement dans le couplage et hors du couplage. Une chaîne améliorante est une chaîne alternée reliant deux sommets insaturés.

Exemple

Sur la figure 2.7, C = {AC, BD, EF} est un couplage de cardinal 3. Les sommets A, B, C, D, E, F sont saturés et les sommets H et G sont insaturés.

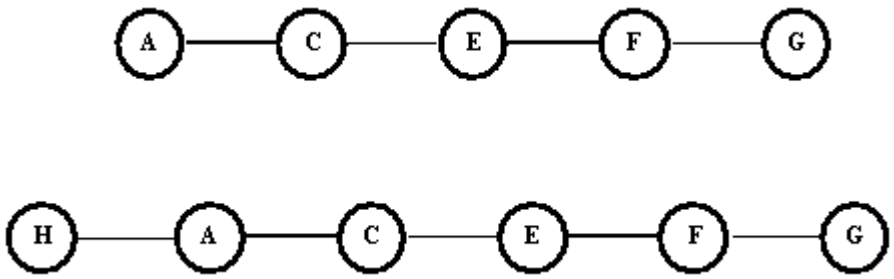


Figure 2.8 : Chaîne alternée et chaîne améliorante

Sur la figure 2.8, [A, C, E, F, G] et [H, A, C, E, F, G] sont des chaînes alternées. [H, A, C, E, F, G] est améliorante car elle relie 2 sommets insaturés. Cette chaîne permet de passer du couplage C = {AC, BD, EF} de cardinal 3 au couplage C'={HA, CE, FG, BD} de cardinal 4. C' est un couplage de cardinal maximal car tous les sommets du graphe sont saturés (Cf figure 2.9)

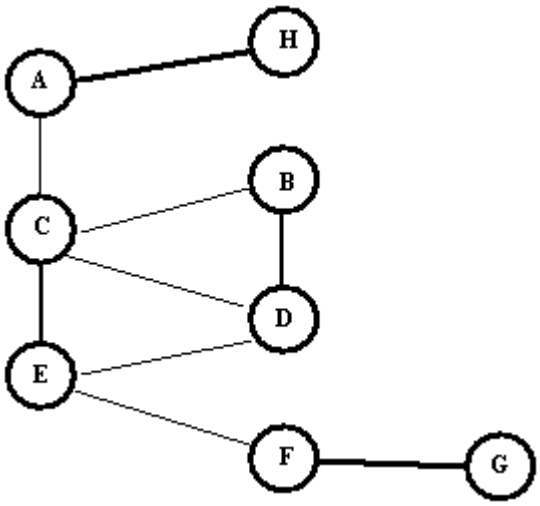


Figure 2.9 : couplage maximal.

b) Théorème de BERGE :

Définition : Théorème de BERGE :

Un couplage est de cardinal maximal si et seulement si le graphe ne contient pas de chaîne améliorante pour ce couplage.

Preuve :

Condition nécessaire : s'il existe une chaîne améliorante elle contient p arcs épais et p+1 arcs minces, on améliore donc le couplage en remplaçant les arcs épais de cette chaîne par les arcs minces de cette chaîne.

Condition suffisante : soit G=(X,U) un graphe, C_0 un couplage de G de cardinal maximal et C_1 un couplage de G sans chaîne améliorante.

Posons $D_0 = C_0 - C_0 \cap C_1 et\, D_1 = C_1 - C_0 \cap C1$.

On raisonne sur le graphe H=(X, $D0 \cup D1$). Les sommets du graphe H sont de degré 0, 1 ou 2, car en un sommet au plus 2 arcs sont incidents : un arc de D0 et un arc de D1. En conséquence les composantes connexes de H sont, soit des sommets isolés, soit des chaînes comportant alternativement des arcs de D0 (donc de C0) et des arcs de D1 (donc de C1), c'est-à-dire des chaînes alternées pour C0 et C1.Si une de ces chaînes était de longueur impaire, elle serait améliorante pour C0 ou pour C1, ce qui est impossible par hypothèse. Ces chaînes sont donc de longueurs paires et comportent autant d'éléments de D0 que d'éléments de D1. Les ensembles D0 et D1 sont donc de même cardinal. En conséquence les ensembles C0 et C1 sont de même cardinal. Il en résulte que C1 est de cardinal maximal.

c) Algorithme dans le cas d'un graphe biparti :

Le théorème de BERGE fournit une méthode pour chercher un couplage maximal. Il suffit en effet de chercher successivement des chaînes améliorantes et de les utiliser pour améliorer le couplage. Le couplage sera maximal quand il n'y aura plus de chaîne améliorante.

Nous présentons ici un algorithme dans le cas des graphes bipartis.

Définition : Algorithme COUPLAGE MAX

Etape 0 : Initialisation .

Un graphe biparti G=(X,Y,U) est donné ainsi qu'un couplage initial C (éventuellement C est vide); aucun sommet n'est marqué ;

Etape 1 : Recherche d'une chaine améliorante .

1.0 Marquer * tout sommet insaturé x dans X ;

1.1 Si tous les sommets marqués ont été examinés, aller à l'étape 3 ;

Sinon, soit i un sommet marqué non examiné

Si i = x ∈ X aller en 1.2

Sinon , i = y ∈ Y , aller en 1.3 ;

1.2 Pour chaque arête [x, y] $\notin C$ incidente au sommet x faire

Si y n'est pas marqué alors marquer y par x;

Retourner en 1.1;

1.3 Si y est insaturé, il est extrémité d'une chaîne améliorante, aller à l'étape 2

Sinon

début

Déterminer l'unique arête [x, y] ∈ C;

Marquer x par y;

Retourner en 1.1 ;

fin;

Etape 2 : Amélioration du couplage .

Améliorer le couplage C en inversant la chaîne améliorante obtenue en 1.3;

Effacer les marques; Retourner à l'étape 1;

Etape 3 : Fin .

Ecrire le couplage maximal C;

d) Application de l'algorithme :

Exemple

Considérons le graphe biparti G = (X, Y, U) avec X = {a, b, c, d, e} et Y = {1, 2, 3,

4, 5) rapporté sur la figure 2.10.

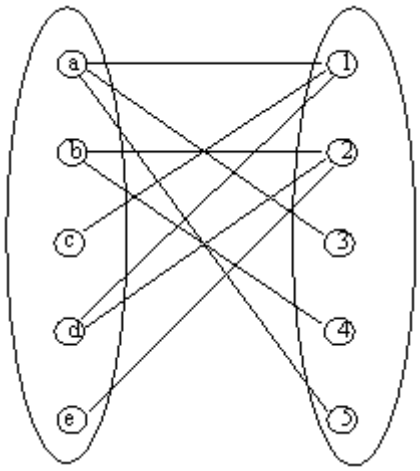


Figure 2.10 : exemple.

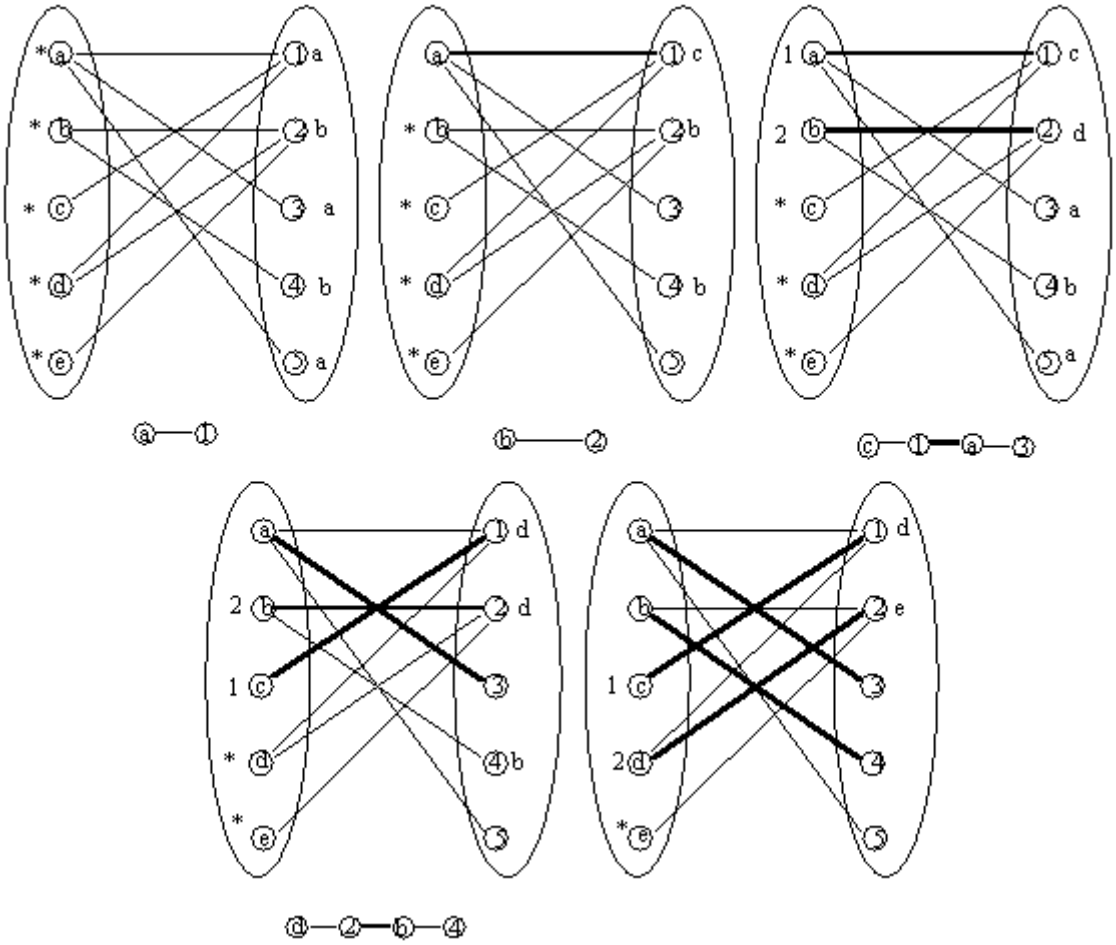


Figure 2.11 : application de l'algorithme.

On part initialement du couplage formé de l'ensemble vide et on modifie ce couplage à l'aide de chaînes améliorantes successives jusqu'à obtenir un couplage sans chaîne améliorante qui sera maximal d'après le théorème de BERGE.

Sur la figure 2.11 sont rapportées les étapes successives de l'algorithme en appliquant la règle suivante : premier sommet marqué, premier examiné.

e) Structures de données :

On suppose que le graphe biparti et le couplage sont codés par un tableau TAB tel que, pour $x. \in X et y \in. Y, TAB(x,y) = (0 si [x,y] \notin U) ou (1 si [x,y] \notin. C \cap U) ou (2 si [x,y] \in C \cap U)$

Si on suppose sans perte de généralité que Cardinal(X) = Cardinal(Y) = N/2, le tableau TAB est de dimension $N^2/4$.

Le tableau TAB permet la lecture de données et servira de tableau de travail pour le couplage.

Enumérons les autres tableaux qui seront utiles pour cet algorithme. Un tableau SATURE (de taille N) servira à noter les sommets saturés. Un tableau MARQUE de taille N sert à stocker les marques des sommets. Il sera initialisé à -1 (aucune marque), on mettra 0 quand un sommet est marqué * par l'algorithme sinon il contiendra la marque du sommet. Un tableau FILE permettra de stocker les sommets marqués non examinés (règle : premier marqué, premier examiné). FILE permet un accès direct. Un tableau CHAINE permettra de reconstituer la chaîne améliorante.

f) Complexité de l'algorithme :

Complexité de l'algorithme

La complexité de l'étape 0 est $O(N^2)$ (lecture d'une matrice).

Une étape 1.0 coûte O(N) car il y a au plus N/2 sommets insaturés. Une étape 1.1 coûte O(1) car il y a un accès direct par FILE. Une étape 1.2 coûte O(N) car il faut lire la ligne x de la matrice TAB. Une étape 1.3 coûte O(N) car il faut lire une colonne y de la matrice TAB.

Quand on exécute l'étape 1 on retourne en 1.1 chaque fois que l'on examine un nouveau sommet, donc au plus N fois. En conséquence les étapes 1.2 et 1.3 sont exécutées au plus N/2 fois. L'étape 1.1 est exécuté N fois. L'étape 1.0 est exécuté 1 fois.

Il en résulte que la complexité de chaque étape 1 est $O(N^2) = O(N)O(1) + O(1)O(N) + O(N)O(N) + O(N)O(N)$. Or l'étape 1 s'exécute au plus N/2 fois, nombre maximal d'améliorations. Donc sur l'ensemble de l'algorithme cette étape coûtera $O(N^3)$.

La complexité d'une étape 2 est O(N) : on construit la chaîne améliorante en lisant les marques. L'étape 2 s'exécute au plus N/2 fois, nombre maximal d'améliorations. Donc sur l'ensemble de l'algorithme cette étape coûtera $O(N^2)$.

La complexité de l'étape 3 est O(1).

Donc la complexité de l'algorithme est $O(N^3) = O(N^2) + O(N^3) + O(N^2) + O(1)$.

g) Preuve de l'algorithme :

Nous allons prouver que cet algorithme détermine dans l'étape 1 une chaîne améliorante si et seulement s'il en existe. Dans l'étape 2 on améliore le couplage. Donc en partant du couplage vide et après au plus N/2 opérations on obtiendra un couplage sans chaîne améliorante qui sera optimal d'après le théorème de BERGE.

Remarquons d'abord qu'une chaîne améliorante étant de longueur impaire relie un sommet de X à un sommet de Y (graphe biparti). On peut donc partir de X et chercher les chaînes alternées issues d'un sommet insaturé de X. Une chaîne améliorante sera une telle chaîne dont le dernier sommet qui est dans Y est un sommet insaturé.

Faisons l'hypothèse de récurrence suivante : $si y \in Y (resp. x \in X)$ est un des p premiers sommets marqués il est extrémité d'une chaîne alternée issue d'un sommet $x_1 \in X$ insaturé de longueur impaire (resp. paire) dont la dernière arête est mince (resp. épaisse).

La propriété est vraie à la fin de l'étape 1.0 : seuls sont marqués les sommets de X insaturés et les chaînes correspondantes sont de longueur nulle. Montrons que si la propriété est vraie à l'ordre p elle reste vraie à l'ordre p+1. Notons j le p+1ème sommet marqué et i sa marque et examinons deux cas :

- $i = x \in X \; (j = y \in Y)$ alors par hypothèse de récurrence, x est extrémité d'une chaîne alternée issue d'un sommet $x_1 \in X$ insaturé de longueur paire dont la dernière arête est épaisse. En concaténant à cette chaîne l'arête (x, y) on obtient une chaîne alternée issue d'un sommet $x_1 \in X$ insaturé de longueur impaire dont la dernière arête est mince et allant à j = y (l'arête (xy) est mince car y est marqué en 1.1 de l'algorithme).
- $i = y \in Y \; (j = x \in X)$ alors par hypothèse de récurrence, y est extrémité d'une chaîne alternée issue d'un sommet $x_1 \in X$ insaturé de longueur impaire dont la dernière arête est mince. En concaténant à cette chaîne l'arête (x, y) on obtient une chaîne alternée issue d'un sommet $x_1 \in X$ insaturé de longueur paire dont la dernière arête est épaisse et allant à j = x (l'arête (x, y) est épaisse car x est marqué en 1.2 de l'algorithme).

La propriété est donc vraie dans les deux cas ce qui la démontre par récurrence.

Définissons M comme l'ensemble des sommets marqués quand on arrive à l'étape 3 et montrons que M est alors exactement l'ensemble des sommets extrémités d'une chaîne alternée issue d'un sommet x_{1} ∈ X insaturé. D'après la propriété démontrée par récurrence tous les sommets de M ont cette propriété. Il faut montrer que réciproquement un sommet ayant cette propriété est dans M.

Avant de le prouver, remarquons que, quand on arrive à l'étape 3, les sommets de M appartenant à Y sont saturés. Il résultera de cette réciproque qu'il n'existe pas de chaîne alternée reliant un sommet insaturé de X à un sommet insaturé de Y, ce qui prouvera le résultat, à savoir qu'il n'y a pas de chaîne améliorante et que le couplage est maximal.

Soit donc k un sommet relié par une chaîne alternée à un sommet x_{1} ∈ X insaturé. On définit j comme le premier sommet de cette chaîne n'appartenant pas à M (un tel sommet existe car par hypothèse k ∉ M). De deux choses l'une, ou bien j = x ∈ X ou bien j = y ∈ Y. Montrons que dans l'un et l'autre cas on arrive à une contradiction. Nous notons i l'antécédent de j sur cette chaîne.

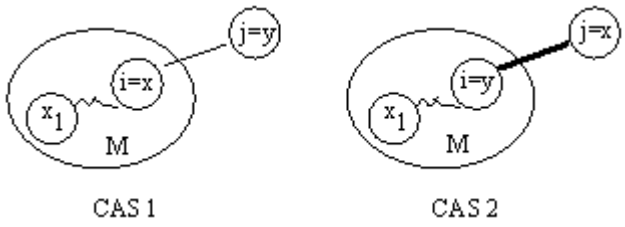


Figure 2.12 : Les deux cas possibles

Dans le premier cas (Cf. figure 2.12) la chaîne allant jusqu'à y est de longueur impaire. Donc quand on examine x, on marque y (l'arête (x, y) est mince car la chaîne est alternée). Ceci contredit le fait que j = y n'est pas marqué. Dans le second cas (Cf. figure 2.12) la chaîne allant jusqu'à x est de longueur paire. Donc quand on examine y, on marque x (l'arête (x, y) est épaisse car la chaîne est alternée). Ceci contredit le fait que j =x n'est pas marqué. En conséquence quand on arrive à l'étape 3, on a bien marqué tous les sommets reliés à un sommet insaturé x_{1} ∈ X par une chaîne alternée. On a vu par ailleurs que les sommets de $M \cap Y$. ayant cette propriété sont saturés. Il en résulte qu'il n'existe pas de chaîne améliorante, c'est-à-dire de chaîne alternée reliant un sommet insaturé x_{1} ∈ X à un sommet insaturé $y \in Y$. Le couplage est donc maximal. Q.E.D.

5. Autres algorithmes de graphe :

a) Autres algorithmes de graphe :

Autres algorithmes de graphe : Tarjan a proposé des algorithmes de complexité linéaire O(M) pour la recherche des composantes connexes (Cf. T.D) et pour la recherche des composantes fortement connexes.

Nous discuterons dans le chapitre suivant de la complexité de la recherche d'un circuit hamiltonien, du coloriage d'un graphe, et du nombre de stabilité.

D. COMPLEXITÉ DES PROBLÈMES COMBINATOIRES

1. Qu'est-ce-que l'optimisation combinatoire ? :

a) Qu'est-ce-que l'optimisation combinatoire ? :

Jusqu'au 19e siècle, on s'intéressait aux Problèmes d'existence. On répondait à la question : existe-t-il un objet ayant certaines propriétés ?

Ensuite, on a résolu des Problèmes de dénombrement (Analyse combinatoire). On répondait à la question : combien y a-t-il d'objets ayant certaines propriétés ?

Enfin avec l'informatique, on traite des Problèmes combinatoires. On se propose de déterminer un objet ayant certaines propriétés, on cherche un algorithme calculant un objet.

2. Différents types de problèmes :

a) Différents types de problèmes :

Nous énonçons ci-dessous une liste de problèmes combinatoires.

Problème du plus court chemin $O(n^3)$:

Soit G un graphe valué et x_0 une racine de G; déterminer les valeurs minimales des chemins allant de x_0 à tout sommet du graphe.

Problème du plus petit nombre O (n) :

Déterminer le plus petit nombre parmi n nombres. Problème de tri O(nlogn) :

Ordonner en ordre croissant n nombres a1, a2, ..., an.

Problème de l'arbre minimal $O(n^3)$:

Soit G = (X , U) un graphe valué (non orienté); déterminer un graphe connexe de coût global minimal.

Problème d'affectation $O(n^3)$ (Algorithme Hongrois) :

Etant donné une matrice W(n, n) représentant les coûts d'affectation de n individus à n travaux, il faut affecter un travail différent à chaque individu pour un coût global minimal.

	T 1	T 2	T 3
P 1	6	5	8
P 2	15	20	14
P 3	8	5	3

Tableau 3 : Figure 3.1 : matrice des coûts.

Pour la matrice de la figure 3.1, l'affectation (P1, T1) (P2, T2) (P3, T3) est de coût 29. L'affectation (P1, T2) (P2, T1) (P3, T3) est optimale et a pour coût 23.

Problème de partition : Etant donné n nombres a1, a2, ...,an, est-il possible de partitionner ces n nombres en deux sous-ensembles de même poids ? Exemple : Pour l'ensemble {8, 6, 15, 37, 42 ,4}, la réponse est oui car 8 + 6 + 42 = 15 + 37 + 4 = 56.

Problème du stable maximum : Soit G = (X , U) un graphe, déterminer un stable de G de cardinal maximal.

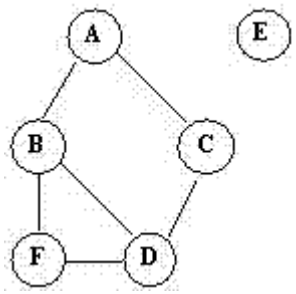


Figure 3.2 : graphe.

Sur le graphe de la figure 3.2, {E, A, F} est un stable de cardinal maximal.

Problème du voyageur de commerce : Soit G = (X , U) un graphe valué, déterminer un circuit hamiltonien de valeur minimale.

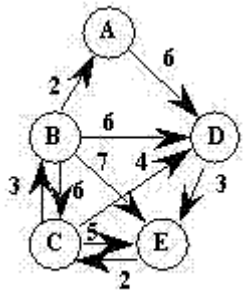


Figure 3.3 : un graphe valué.

Sur la figure 3.3, [A, D, E, C, B, A] est un circuit hamiltonien de valeur minimale.

Problème de satisfiabilité : Etant donné une fonction booléenne sous une forme conjonctive, existe-t-il une affectation des variables rendant vraie cette fonction ?

Exemple : Pour la fonction donnée par : (x1 + x2 + ¬x3) (¬x2) (x3 + x1) (x3 + x2) = 1, la réponse est oui car x2 = 0, x3 = 1, x1 = 1 est solution. De plus cette solution est unique.

3. Quand un problème est-il résolu ? :

a) Quand un problème est-il résolu ? :

Pour chaque problème, il y a un nombre fini de solutions (1). On peut donc semble-t-il énumérer toutes ces solutions pour déterminer une solution optimale (ou une solution réalisable). Par exemple, pour le problème du voyageur de commerce : il y au plus (n-1)! circuits hamiltoniens, pour le problème de satisfiabilité : il y a au plus 2n affectations, pour le problème de partition : il y a 2n-1 solutions. Il est irréaliste d'avoir recours à une méthode d'énumération dans le domaine combinatoire quand la complexité résultante est exponentielle. Le tableau suivant rapporte les durées d'énumération, si l'on dispose d'un ordinateur calculant chaque solution en une

microseconde (10-6 s).

	10	20	40	60
n	10^{-5}s	2.10^{-5}s	4.10^{-5}s	6.10^{-5}s
n^2	10^{-4}s	4.10^{-4}s	16.10^{-4}s	36.10^{-4}s
n^3	10^{-3}s	8.10^{-3}s	64.10^{-3}s	216.10^{-3}s
n^5	0,1s	3,2s	1,7 mn	13 mn
2^n	10^{-3}s	1s	12,7 jours	366 siècles
3^n	0,059s	58 mn	3855 siècles	$1,3.10^{13}$siècles
$n!$	3,63s	100 siècles	...	...

Tableau 4 : les durées d'énumération,

Les algorithmes efficaces sont principalement les algorithmes de complexité polynomiale (2).

On dira qu'un problème est bien résolu si on a trouvé un algorithme polynomial (de faible degré) le résolvant (3).

On peut aussi étudier la variation de la taille maximum que l'on peut traiter en 1000s quand on multiplie par 10 la puissance de la machine :

T(n)	Taille maximum pour 1000s avec la première machine	Taille maximum pour 1000s avec la seconde machine	Augmentation
$100n$	10	100	x 10
$10n^2$	10	32	x 3,2
n^3	10	22	x 2,2
2^n	10	13	x 1,3

Tableau 5 : t5

1. La finitude du nombre de solutions garantit l'existence d'algorithmes de résolution, c'est-à-dire de programmes s'arrêtant. Il n'en est pas toujours ainsi en informatique où certains problèmes sont indécidables, c'est-à-dire non traitables par un programme en un temps fini.
2. On évaluera dans ce cours la complexité en se plaçant dans le plus mauvais cas. Toutefois, quand un algorithme tourne fréquemment, il serait préférable de faire une analyse en moyenne. Malheureusement une telle analyse est très difficile et même hors de portée pour des programmes moyennement complexes.
3. On peut en première approximation négliger les coefficients du polynôme

dans la mesure où l'expérience montre que la plupart des algorithmes polynomiaux ont des petits coefficients. Il existe toutefois des exceptions pour lesquelles même des algorithmes linéaires sont impraticables (Cf : Théorie de ROBERTSON et SEYMOUR).

4. Les problèmes NP-difficiles :

a) Les problèmes NP-difficiles :

On n'a pas trouvé d'algorithmes polynomiaux pour les problèmes de la partition, du stable maximal, du voyageur de commerce et de la satisfiabilité. Il en est de même pour un très grand nombre d'autres problèmes. On n'a pas montré qu'il n'existait pas d'algorithme polynomial pour un de ces problèmes, mais on a le résultat suivant: "Si un seul de ces problèmes est polynomial alors tous les problèmes "raisonnables" réputés difficiles sont polynomiaux". Ces problèmes sont dits NP-difficiles.

5. Méthodes arborescentes :

Les **méthodes arborescentes** permettent de résoudre des problèmes **NP-difficiles**. Toutefois, il ne sera pas possible d'évaluer la complexité pratique de ces méthodes car cette complexité est fortement fonction de la donnée du problème. Elle est d'ailleurs en général exponentielle pour certaines données, mais peut être "praticable" pour d'autres données.

a) Les Dames de Gauss :

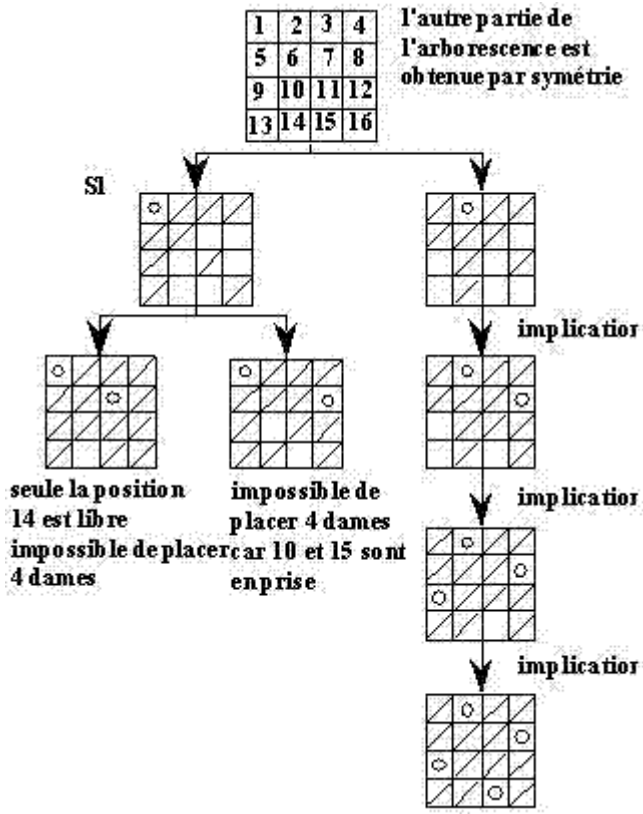


Figure 3.4 : arborescence des dames de Gauss.

A la racine correspond l'ensemble de toutes les solutions. Au sommet S1 correspond l'ensemble de toutes les solutions telles que la dame sur la première ligne est sur la case n°1. A tout sommet correspond un ensemble de solutions.

Il s'agit de placer n dames (ici n = 4) sur un échiquier n*n de façon à ce que deux dames quelconques ne soient pas en prise. Pour résoudre ce problème, on construit une arborescence dont la racine correspond à l'ensemble des solutions du problème. On examine les différents cas possibles en construisant une arborescence (Cf. Figure 3.4), sur laquelle apparaît une solution, l'autre s'obtenant par symétrie.

b) Méthode de LITTLE pour le problème du voyageur de commerce :

Un voyageur de commerce doit visiter n villes. Il faut choisir un ordre de parcours minimisant la distance totale parcourue, c'est-à-dire trouver un circuit hamiltonien de valeur minimale. Pour résoudre ce problème, on va construire une arborescence dichotomique, comme on peut le voir sur la figure 3.5.

Figure 3.5 : arborescence dichotomique.

Nous allons nous appuyer sur un exemple dont les données sont rapportées sur la figure 3.6 pour expliquer la méthode de Little. Remarquons d'abord qu'à un circuit hamiltonien correspond exactement un élément par ligne et un élément par colonne de la matrice d'adjacence. Donc, on ne modifie pas le problème si ligne par ligne on soustrait le minimum de chaque ligne, puis colonne par colonne, le minimum de chaque colonne.

	A	B	C	D	E	F
A	∞	6	7	3	1	3
B	7	∞	8	2	9	7
C	5	10	∞	10	1	7
D	8	6	5	∞	5	1
E	7	7	6	7	∞	4
F	9	8	8	5	3	

Tableau 6 : Figure 3.6 : matrice d'adjacence valuée.

En soustrayant en ligne, on obtient le tableau du milieu de la figure 3.7, et en soustrayant en colonne le tableau de droite.

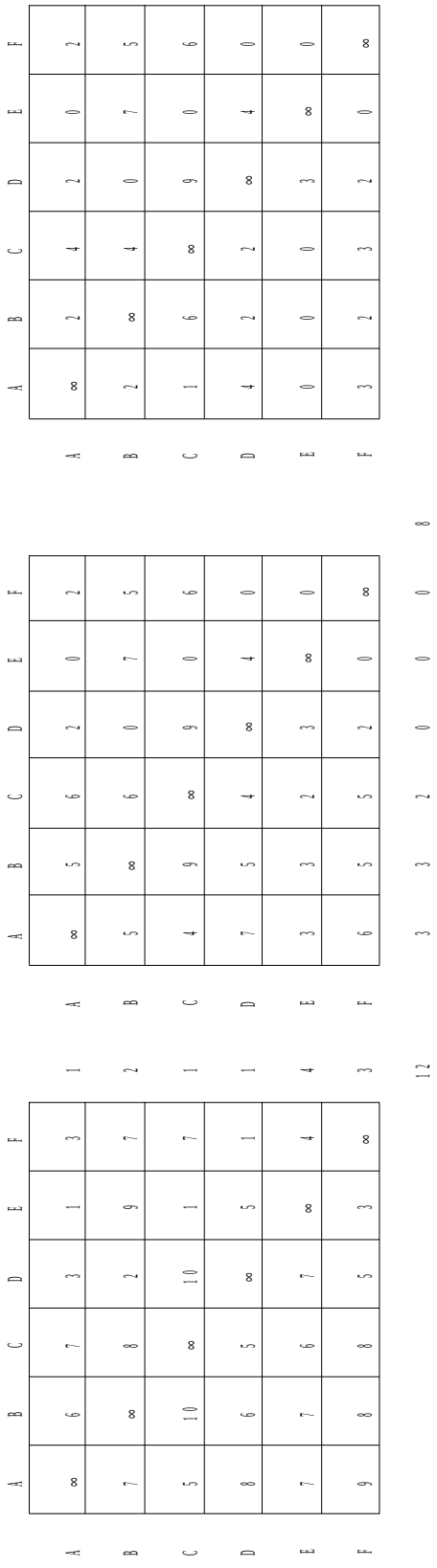


Tableau 7 : Figure 3.7 : apparition des zéros par ligne et par colonne.

Le circuit Hamiltonien aura une valeur supérieure à 20 car on a soustrait 12 en lignes et 8 en colonnes. 20 est une évaluation par défaut de la valeur optimale pour la matrice initiale. On va travailler sur la matrice réduite, pour laquelle les circuits hamiltoniens ont tous une valeur translatée de 20. On cherche à construire un

circuit hamiltonien de valeur zéro (d'où forcément optimal car les valeurs sont positives). Donc pour construire l'arborescence d'énumération, on va choisir des arcs de valuation 0.

Pour sélectionner un de ces arcs, on introduit la notion de regret. On choisira un 0 de regret maximal. Par exemple, si on ne prend pas l'arc AE, on prendra obligatoirement un autre élément de la ligne A et un autre élément de la colonne E. Le coût supplémentaire ici 2, s'appelle le regret. Ce regret se calcule en sommant les minima en ligne et en colonne (le 0 sur lequel on calcule le regret étant exclu). En effet, si on ne passe pas par cet arc de valeur 0, on ne pourra au mieux que passer par deux arcs de coût minimum des ligne et colonne correspondantes, il en coûtera le regret.

	A	B	C	D	E	F
A	∞				0 2 + 0	
B		∞		0 2 + 2		
C			∞		0 1 + 0	
D				∞		0 2
E	0 1	0 2	0 2		∞	0 0
F					0 2	∞

Tableau 8 : Figure 3.8 : matrice des regrets.

Les regrets des différents 0 sont rapportés sur la figure 3.8. Il apparaît que BD est le 0 de plus grand regret. On choisit de prendre l'arc BD, et on introduit deux nouveaux sommets dans l'arborescence (Cf. figure 3.9). Le sommet BD correspond à l'ensemble des circuits hamiltoniens empruntant l'arc BD et le sommet ¬BD à l'ensemble des circuits hamiltoniens n'empruntant pas l'arc BD. Le sommet BD a une évaluation par défaut de 20, le sommet ¬BD a une évaluation par défaut de 24, c'est-à-dire 20 plus le regret.

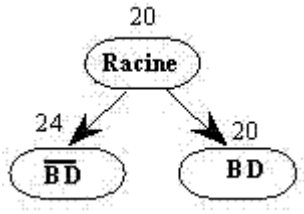


Figure 3.9 : arborescence courante après la première séparation.

Dans la matrice associée au sommet BD, on doit interdire l'arc DB en posant $M(D,B)=+\infty$, afin d'éviter le circuit parasite [D, B, D], et réitérer le processus. Sur la figure 3.10, on a sur la gauche la matrice associée au sommet BD de l'arborescence, et à droite les regrets obtenus en éliminant la ligne D et la colonne B.

	A	B	C	D	E	F
A	∞	2	4	~~2~~	0	2
B	~~2~~	~~∞~~	~~4~~	~~0~~	~~7~~	~~5~~
C	1	6	∞	~~0~~	0	6
D	4	∞	2	~~∞~~	4	0
E	0	0	0	~~3~~	∞	0
F	3	2	3	~~2~~	0	∞

	A	B	C	E	F
A				0 2	
C				0 1	
D					0 2
E	0 1	0 2	0 2		0 0
F				0 2	

Tableau 9 : Figure 3.10 : matrices associées au sommet BD.

On choisit un 0 de plus grand regret et on introduit deux nouveaux sommets dans l'arborescence courante (Cf. figure 3.11), AE et ¬AE.

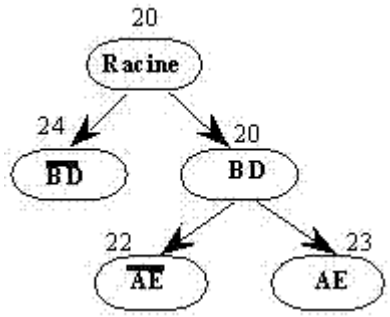


Figure 3.11 : état suivant de l'arborescence courante.

La matrice associée au sommet AE après interdiction de EA est rapportée sur la figure 3.12. On n'a pas un zéro par ligne; d'où, on va soustraire la valuation minimale sur chaque ligne et chaque colonne. On soustrait donc globalement 3. Ceci explique que la valuation du sommet AE soit égale à 23, alors que celle du sommet ¬AE est 22.

	A	B	C	F	
C	1	6	∞	6	1
D	4	∞	2	0	0
E	∞	0	0	0	0
F	3	2	3	∞	2

Tableau 10 : Figure 3.12 : nouvelle matrice avec EA = □

Présentons maintenant l'algorithme de LITTLE.

Syntaxe : Algorithme de LITTLE :

1. Initialisation :
   Faire apparaître des zéros (au moins un par ligne et un par colonne) {L'évaluation par défaut de la racine de l'arborescence est la somme des chiffres soustraits}. Poser$f_0 = \infty$ {f_0 sera la valeur de la meilleure solution connue}.
2. Boucle principale : Tant que le sommet S de l'arborescence de plus petite évaluation par défaut f(S) est tel que f(S) < f_0 faire {largeur d'abord}
   - évaluer le regret de chacun des zéros de la matrice associée à S et retenir (i, j) correspondant à un zéro de regret maximal.
   - Introduire les sommets (S + (i, j)) et (S + ¬(i, j)) en interdisant le circuit parasite dans (S + (i, j)), en faisant apparaître des zéros dans les deux matrices associées, et en valuant ces sommets par leur évaluation par défaut.
   - Si le sommet (S + (i, j)) correspond à un circuit hamiltonien alors écrire le circuit, et calculer sa valeur f_1.
   - poser f_0 = min (f_0, f_1)

   Fin tant que.

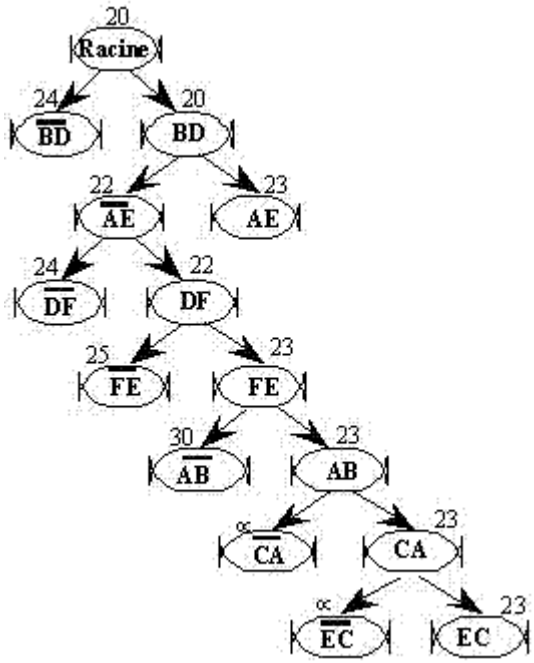


Figure 3.13 : arborescence globale.

Suite de l'exemple : Sur la figure 3.13 est rapportée l'arborescence après application de l'algorithme. Nous commentons ci-dessous sa construction. Sur la figure 3.14, on a la matrice associée au sommet ¬AE, qui est le sommet pendant de l'arborescence courante de plus petite évaluation par défaut, ainsi que ses regrets. Le plus grand regret est obtenu pour l'arc DF. On interdit l'arc FB qui créerait un circuit parasite.

	A	B	C	E	F
A	∞	0	2	∞	0
C	1	6	∞	0	6
D	4	∞	2	4	0
E	0	0	0	∞	0
F	3	2	3	0	∞

	A	B	C	E	F
A		0 0			0 0
C				0 1	
D					0 2
E	0 1	0 0	0 2		0 0
F				0 2	

Tableau 11 : Figure 3.14 : matrices associées au sommet ¬AE.

Il faut interdire le circuit parasite, décrit sur la figure 3.16, en mettant ∞ dans la case correspondant à EB.

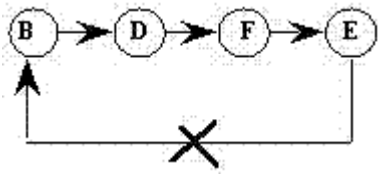


Figure 3.16 : circuit parasite.

Ayant interdit l'arc (EB), il faut faire apparaître de nouveaux zéros dans la matrice associée au sommet FE (Cf figure 3.17). Pour obtenir un zéro, on soustrait 1 et on rajoute donc 1 à l'évaluation. AB est alors le zéro de regret maximal. Pour la matrice associée au sommet AB il faut interdire le circuit parasite de la figure 3.18.

	A	B	C
A	∞	0	2
C	1	6	∞
E	0	∞	0

1

	A	B	C
A	∞	0 7	2
C	0 5	5	∞
E	0 0	∞	0 2

Tableau 12 : matrice

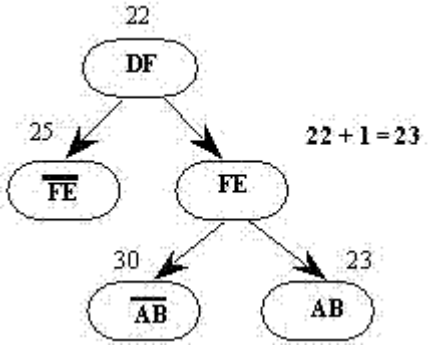


Figure 3.17 : séparation de FE.

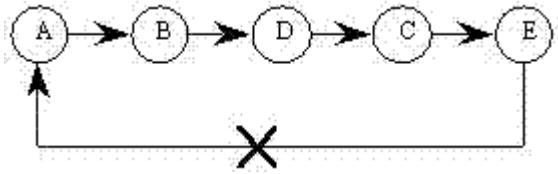


Figure 3.18 : élimination du circuit parasite du sommet AB.

Sur la figure 3.19, le choix des arcs CA et EC est obligatoire. On obtient donc le circuit hamiltonien de la figure 3.20 de valeur 23. Ce circuit hamiltonien est optimal car tous les sommets pendants de l'arborescence globale ont une évaluation supérieure ou égale à 23.

Discussion :

La méthode la plus efficace actuellement pour résoudre le problème du voyageur de commerce n'est pas l'algorithme de LITTLE qui permet de traiter optimalement des exemples à 40 villes environ. On distingue en fait les cas symétrique et asymétrique. Dans le cas asymétrique, des méthodes utilisant l'affectation linéaire couplée à de la relaxation lagrangienne (Cf. RO04) pour l'évaluation par défaut permettent de traiter des exemples à 300 villes environ. Dans le cas symétrique, la programmation linéaire généralisée consistant à introduire des coupes d'intégrité (méthodes polyédrales) permettent de traiter des exemples à 400 villes environ.

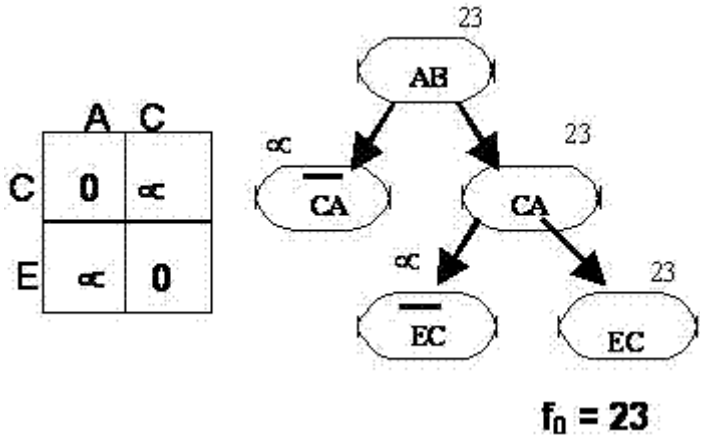


Figure 3.19 : séparation de AB.

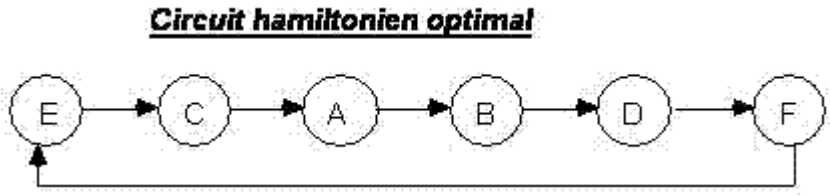


Figure 3.20 : circuit hamiltonien optimal.

c) Programmation linéaire en variables booléennes : méthode de FAURE-MALGRANGE :

Nous allons présenter sur un exemple la méthode de FAURE-MALGRANGE qui permet de traiter un programme linéaire en variables booléennes. On veut résoudre le programme linéaire suivant :

$$Max F = 3x1 + 5x2 - x3 + 2x4$$

sous les contraintes :

$$x1, x2, x3, x4 \in \{0, 1\}$$

$$3x1 + 4x2 - 2x3 + x4 \leq 5$$

$$2x1 - x2 + 3x3 - x4 \leq 4$$

$$4x1 - x2 - x3 + 2x4 \leq 5$$

Nous introduisons ¬x1, ¬x2, ¬x3, ¬x4 définies par xi + ¬xi = 1.

Ceci permet de remplacer les - par des + dans les contraintes d'inégalités qui deviennent :

$$3x1 + 4x2 + \neg 2x3 + x4 \leq 5$$

$$2x1 + \neg x2 + 3x3 + \neg x4 \leq 4$$

$$4x1 + \neg x2 + \neg x3 + 2x4 \leq 5$$

En remplaçant les + par des - dans la fonction économique, on a :

F = 10 - (3¬x1 + 5¬x2 + x3 + 2¬x4)

On remarque que F ne peut dépasser 10.

Le problème est mis sous une forme facilitant une énumération implicite. Mais cette fois il s'agit d'un problème de maximisation. On va donc construire une arborescence dont les sommets seront valués par une évaluation par excès. La séparation consistera à choisir une variable booléenne et à fixer la valeur de vérité de cette variable.

La meilleure solution construite fournira une évaluation par défaut de la solution optimale. Sur la figure 3.21 est rapporté l'arborescence globale dont nous commentons ci-dessous la construction.

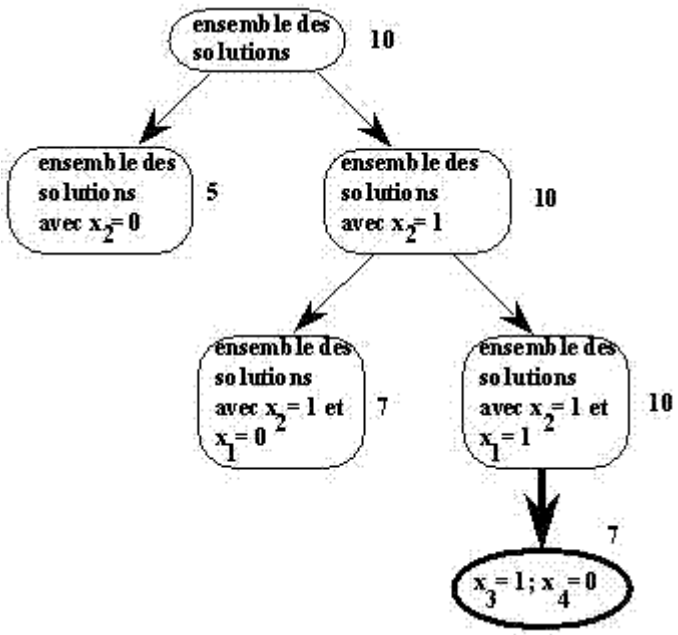


Figure 3.21 : arborescence de recherche de solution.

Premier choix :

On choisit de séparer sur x2 car son coefficient dans la fonction économique est le plus grand (x2 est de regret maximal). Au sommet x2 = 1 de l'arborescence correspond de nouvelles contraintes : 3

$$x1+2\neg x3+x4\leq 3$$

$$2x1+3x3+\neg x4\leq 6$$

$$4x1+\neg x3+2x4\leq 7$$

F = 10 - (3¬x1 + x3 + 2¬x4)

On sépare ce sommet :

Deuxième choix :

On choisit de séparer sur x1 car son coefficient dans la fonction économique est le plus grand (x1 est de regret maximal).

Au sommet x1 = 1 de l'arborescence correspond de nouvelles contraintes :

$$2\neg x3+x4\leq 0$$

$$3x3+\neg x4\leq 4$$

$$\neg x3+2x4\leq 3$$

F = 10 - (x3 + 2¬x4)

On a alors les implications x3 = 1 et x4 = 0. On a donc trouvé une solution de valeur 7. Cette solution est optimale car toutes les évaluations par excès des sommets pendants de l'arborescence sont inférieurs à 7.

Ici apparaît l'intérêt d'avoir des inégalités avec des termes positifs, en effet si $\sum a\,i\,y\,i\leq b$ avec les yi variables booléennes, les ai positifs et pour un certain i, ai > b, nécessairement yi = 0.

Ici on a parcouru l'arborescence en profondeur d'abord en choisissant de séparer le dernier sommet introduit. Une telle politique est gérable à l'aide d'une pile, ce qui renforce son intérêt.

d) Programmation linéaire en variables entières :

Dans l'U.V RO04, on présente la méthode du simplexe qui est efficace pour résoudre un programme linéaire dont les variables sont rationnelles. Quand les

variables sont astreintes à être entières le problème devient NP-difficile. On peut le résoudre, soit par des méthodes de coupes (Cf. RO04), soit par des méthodes arborescentes. Considérons le programme linéaire en nombres entiers suivant :

Max F = 3x1 + 8x2

x1 + 4x2 $\leq$ 20

x1 + 2x2 $\leq$ 11

3x1 + 2x2 $\leq$ 19

4x1 - x2 $\leq$ 22

Avec x1, x2 $\in N$

Une première méthode pour résoudre ce type de problème est de borner les variables de façon à se ramener à des variables booléennes. La première contrainte entraîne x2 $\leq$ 5. Une combinaison linéaire des troisième et quatrième contraintes entraîne x1 $\leq$ 5. On pose alors :

x1 = 4X11 + 2X12 + X13 et x2 = 4X21 + 2X22 + X23, ce qui permet de se ramener à des variables booléennes et d'utiliser la méthode de FAURE-MALGRANGE.

Une deuxième méthode consiste à utiliser directement le fait que les variables sont bornées et à séparer successivement sur x1 et x2. On a rapporté sur la figure 3.22 l'arborescence résultante. Dans cette arborescence les évaluations sont par excès, on a fait un parcours en profondeur d'abord et trouvé 2 solutions : x2 = 5 , x1 = 0 et x2 = 4 , x1 = 3. La deuxième solution de valeur 41 est optimale car les sommets pendants de l'arborescence ont une évaluation plus faibl

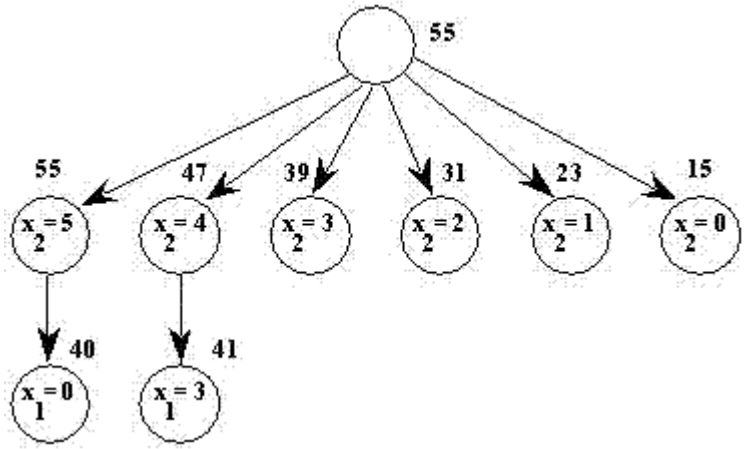


Figure 3.22 : arborescence globale.

6. Récapitulation :

a) Récapitulation :

a) Dans une méthode arborescente, on construit une arborescence dont les sommets correspondent à des sous-ensembles de solutions. L'union des successeurs d'un sommet est égal à ce sommet. Les sommets terminaux de cette arborescence sont les solutions du problème. On parle d'énumération implicite car les évaluations et les implications permettent de n'énumérer qu'un nombre limité de solutions.

b) A chaque sommet S de l'arborescence, est associé une évaluation f(S), par défaut pour un problème de minimisation, par excès pour un problème de maximisation.

c) On a également une évaluation globale f0, par excès pour un problème de minimisation et, par défaut pour un problème de maximisation. f0 est souvent la valeur de la meilleure solution construite.

d) Le choix du sommet à séparer est essentiel. On distingue principalement les stratégies en profondeur d'abord et en largeur d'abord.

e) En profondeur d'abord, on séparera le sommet le plus profond de l'arborescence. En conséquence, on parcourra l'arborescence branche par branche. L'arborescence courante sera un chemin, donc gérable à l'aide d'une pile. Cette stratégie est la plus utilisée pour cette raison.

f) En largeur d'abord, on séparera le sommet de l'arborescence de plus petite évaluation. En conséquence, on parcourra l'arborescence de façon anarchique. L'idée est de séparer un sommet prometteur.

g) La séparation permet d'introduire pour le sommet choisi des sommets successeurs. Pour être efficace, il faut que l'évaluation par défaut pour un problème de minimisation (resp. par excès pour un problème de maximisation) augmente (resp. diminue) pour ces sommets introduits.

h) Des implications, que l'on appelle aussi règles de dominance, permettent de rendre la méthode plus performante.

7. Méthodes heuristiques :

a) Méthodes heuristiques :

Les méthodes arborescentes sont des méthodes efficaces pour traiter de nombreux problèmes. Toutefois, au contraire des algorithmes polynomiaux, leur efficacité est fortement fonction des données du problème. De plus elles sont lourdes à mettre en oeuvre et ne permettent souvent de résoudre optimalement que des problèmes de tailles réduites. Pour toutes ces raisons, il est souvent nécessaire d'utiliser des méthodes heuristiques qui permettent de construire des solutions d'un problème, mais ne garantissent plus l'optimalité.

Expliquons une heuristique simple pour le problème du voyageur de commerce. On commence par choisir la ville de départ. La deuxième ville choisie est alors la ville la plus voisine. Une fois choisies un certain nombre de villes, on choisit la ville non choisie la plus proche de la dernière ville introduite et on itère jusqu'à ce que l'on revienne au point de départ.

Cette heuristique évidemment très naïve donne des résultats très mauvais pratiquement. C'est pourquoi, on introduit les méthodes de voisinage. Présentons d'abord les méthodes de descente. L'idée est de définir une relation de voisinage sur l'ensemble des circuits hamiltoniens de telle sorte qu'il soit facile de calculer le meilleur voisin d'un sommet. On part alors du circuit hamiltonien construit à l'aide de l'heuristique naïve. Puis on calcule son meilleur voisin , si celui-ci est de coût plus faible que le tour initial, on le retient. On itère , en descendant, jusqu'à obtenir un tour dont tous les voisins sont moins bons que lui-même. On a alors trouvé un optimum local par rapport à la relation de voisinage. Remarquons qu'il n'y a pas de raison pour que cet optimum local soit un optimum global.

Toutefois ces méthodes de descente sont extrêmement efficaces quand la relation de voisinage est bien choisie. Par exemple, pour un problème de voyageur de commerce symétrique, la méthode K-opt de LIN et KERNINGHAN, permet de trouver très rapidement, même pour des problèmes de grande taille (par exemple 1000 villes) des solutions proches à 2 ou 3 % de la solution optimale. Toutefois les résultats des méthodes de voisinage sont rarement aussi bons. On a constaté empiriquement que le problème du voyageur de commerce est un problème NP-difficile pour lequel il est relativement facile, le plus souvent, de construire de bonnes solutions. Ceci explique que, pour des problèmes plus difficiles

pratiquement, on cherche à échapper à l'optimum local. Il y a trois techniques principales pour y parvenir: le recuit simulé, le tabou et le génétique.

Dans le recuit simulé, quand le voisin d'un sommet est meilleur, on descend comme précédemment. Si au contraire le voisin d'un sommet est moins bon, on le retient éventuellement à la place du sommet courant et ceci avec une certaine probabilité p. On fait diminuer cette probabilité au cours du temps de façon à s'autoriser au début de nombreuses remontées et à la fin de l'algorithme très peu de remontées. Techniquement, un paramètre T décroissant, appelé température permet cela. On a : [p = 1 - exp (-T)]. Les inconvénients du recuit simulé sont les suivants : on n'a pas de garantie d'optimalité, du moins en un temps raisonnable; il est souvent très coûteux en temps de calcul ; enfin la température est délicate à régler.

Dans les méthodes tabous, si on ne peut plus descendre, on s'autorise également les remontées afin d'échapper à un minimum local. Mais il y a alors un risque de cyclage. Pour l'éviter, on maintient une liste dynamique et de longueur limitée, contenant une liste de modifications récemment effectuées. Cette liste est gérée selon la règle premier entré, premier sorti. Les méthodes tabous ont des inconvénients similaires à ceux du recuit simulé.

Citons, pour terminer, les méthodes génétiques. Cette fois, à chaque itération, on conserve un nombre de solutions plus grand que 1 qui forment la population. On choisit alors une sous population, souvent de cardinal 2, et on a une règle, dite génétique, pour construire à partir de cette sous-population, une nouvelle solution. On itèr, après avoir mis dans la population cette nouvelle solution (naissance) et retiré une autre solution (mort). Ces méthodes ne garantissent pas non plus l'optimalité.

Les performances de ces trois méthodes dépendent des problèmes et de l'habileté de leur concepteur. On ne peut donc pas dire que l'une est meilleure que les autres. Elles sont efficaces si l'on arrive à définir une bonne relation de voisinage facile à calculer.

E. PROBLÈMES DE CHEMINEMENT

Dans ce chapitre, nous allons étudier des algorithmes polynomiaux résolvant des problèmes de cheminement fondamentaux. G = (X,U,v) (v application de U dans R) sera un graphe valué, avec X = {x0,x1,....,xn-1}. De plus, par commodité, nous faisons l'hypothèse que x0 est une racine de G (il existe donc un chemin allant de x0 à tout $xi \in X$). On considère principalement le problème fondamental suivant :

Déterminer les valeurs minimales (si elles existent) des chemins issus de x0, puis calculer un chemin minimal allant de x0 à xi (pour tout i).

Nous allons voir que ce problème est résolu en $O(n^3)$ dans le cas général (algorithme de FORD), en O(m) dans le cas particulier des graphes sans circuit (algorithme de BELLMAN) et en O(n2) dans le cas particulier des graphes dont les arcs sont valués positivement (algorithme de DIJKSTRA).

1. Propriété des chemins minimaux :

a) Propriété des chemins minimaux :

On rappelle que la valeur d'un chemin est la somme des valuations de ses arcs. Un chemin allant de x0 à xi est de valeur minimale si sa valeur est plus petite que celle de tout autre chemin allant de x0 à xi.

Lemme 1

Tout sous-chemin d'un chemin de valeur minimale est un chemin de valeur minimale.

Démonstration

Soit μ un chemin allant de x_i à x_j et soit μ' un sous-chemin de μ allant de x_k à x_l, alors μ = $\mu_1\mu'\mu_2$ où μ_1 = [x_i, x_k] et μ_2 = [x_l, x_j]. On suppose μ de valeur minimale. Si μ' n'était pas de valeur minimale, il existerait un chemin μ" de valeur strictement plus faible allant de x_k à x_l et le chemin $\mu_1\mu''\mu_2$ serait de valeur strictement inférieure à celle de μ, ce qui contredit le fait que μ est de valeur minimale. Q.E.D.

Cette propriété extrêmement simple est à la base de la programmation dynamique (Cf. TD): pour ces problèmes de cheminement l'optimisation globale est le résultat d'optimisations locales.

Lemme 2

Une condition nécessaire et suffisante pour que, pour tout i, il existe un chemin de valeur minimale allant de x_0 à x_i est que le graphe G soit sans circuit de valeur strictement négative (ces circuits sont dit absorbants).

Démonstration :

Supposons qu'il existe un circuit μ1 = [x_i ,x_i] de valeur strictement négative et que μ =[x_0,x_i] soit un chemin de valeur minimale allant de x_0 à x_i. Alors la valeur du chemin

μ' = μμ1 est strictement inférieure à celle de μ, ce qui contredit sa minimalité.

Réciproquement. Supposons G sans circuit absorbant. De tout chemin allant de x_0 à x_i, on peut extraire un chemin élémentaire de valeur plus faible (si [x_0 , x_i] n'est pas élémentaire, on enlève successivement les différents circuits qu'il emprunte). On peut donc limiter la recherche des chemins minimaux à celle des chemins minimaux élémentaires. Or le nombre de chemins élémentaires allant de x_0 à x_i étant fini, il existe un chemin élémentaire de valeur minimale allant de x_0 à x_i. Q.E.D.

Les circuits absorbants pour la minimisation sont les circuits de valeur strictement négative. Dualement, si on maximise, les circuits absorbants sont les circuits de valeur strictement positive. Il faut toujours prendre garde à l'existence de tels circuits quand on recherche des chemins extrêmaux.

Théorème 1 :

Soit G un graphe sans circuit de valeur strictement négative et λ_i des valeurs de chemins entre x_0 et x_i. Une condition nécessaire et suffisante pour que ces $\{\lambda_i / 0 \le i \le n-1\}$ soient l'ensemble des valeurs des chemins minimaux issus de x_0 est que :

1. $\lambda_0 = 0$
2. $\lambda_i - \lambda_j \le v_{ij}$, pour tout arc $(x_i, x_j) \in U$.

Corollaire :

3. l'ensemble des arcs pour lesquels l'inégalité de 2°) est une égalité est l'ensemble des arcs appartenant à des chemins minimaux.

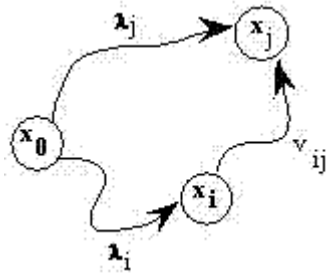


Figure 4.1 : chemins de x0 à xj .

Démonstration :

Les conditions sont nécessaires :

Soit λ_i iles valeurs minimales des chemins issus de x0.

1°) $\lambda_0=0$ (un sommet est un chemin de valeur 0).

2°) λ_j est la valeur minimale d'un chemin allant de x_0 à x_j et est donc inférieure à λ_i + v_{ij} qui est la valeur d'un chemin allant x_0 à x_j empruntant l'arc (x_i ,x_j).

Corollaire :

3°) Si $\lambda_j-\lambda_i=v_{ij}$, l'arc (x_i, x_j) appartient à un chemin de valeur minimale allant de x_0 à x_j, obtenu en concaténant un chemin de valeur minimale allant x0 à xi avec l'arc (x_i, x_j).

Réciproquement :

Si (x_i, x_j) appartient à un chemin [x_0, x_k] de valeur minimale, d'après le lemme 1, les sous-chemins allant de x_0 à x_j et de x_0 à x_i sont de valeur minimale;

il en résulte que $\lambda_j=\lambda_i+v_{ij}$, donc l'inégalité est une égalité.

Les conditions sont suffisantes :

On suppose que les λ_i satisfont 1°), 2°) et sont des valeurs de chemins ; on va montrer que ce sont les valeurs minimales des chemins issus de x_0.

Considérons un chemin [x_0, x_i] = μ de valeur minimale, noté aussi : μ= [x_{j0} = x_0 , ... , x_{jp} = x_i]. On a : $\lambda_{j0}=0$, $\lambda_{j1}-\lambda_{j0}\leq v_{j0j1}$, $\lambda_{j2}\lambda_{j1}\leq v_{j1j2}$, ..., $\lambda_{jp}-\lambda_{jp-1}\leq v_{jp-1jp}$.

En sommant, on obtient $\lambda_{jp}=\lambda_i\leq v(\mu)$. Il en résulte que λ_i est la valeur minimale d'un chemin allant de x0 à xi.

Cette valeur λ_i est en effet inférieure et supérieure à la valeur minimale d'un chemin entre x_0 et x_i. Q.E.D. Le théorème 1 est à la base des algorithmes présentés ci-dessous. On partira d'un ensemble de majorants des valeurs optimales et on les ajustera jusqu'à ce que la condition 2 soit satisfaite.

2. Algorithme de FORD :

a) Algorithme de FORD :

Syntaxe

Il est utilisable quel que soit le graphe, mais dans des cas particuliers, moins performant que d'autres algorithmes.

(i) Initialisation

Poser $\lambda_0=0$ et $\lambda_i=+\infty$ pour i > 0.

(ii) Examen des arcs

Pour chaque sommet x_i examiné, parcourir l'ensemble des arcs (x_i,x_j) issus de xi en remplaçant λ_j par $\lambda_i+v_{ij} si \lambda_i+v_{ij}<\lambda_j$.

(iii) Test d'arrêt

Itérer (ii) tant qu'un λ_j aura été modifié dans (ii).

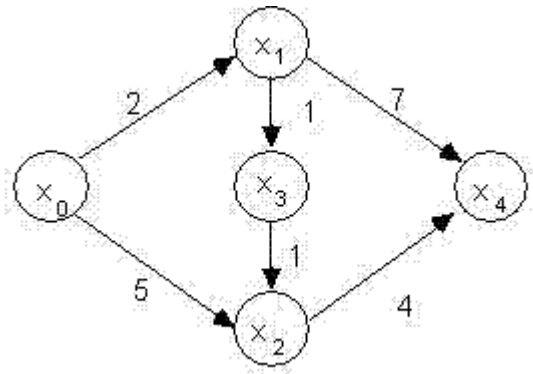


Figure 4.2 : exemple.

Le tableau ci-dessous rapporte les valeurs successives des λ_i lors de l'application de l'algorithme de FORD à l'exemple de la figure 4.2.

On doit se poser les questions suivantes:

1. Cet algorithme donne-t-il toujours les valeurs minimales ?
2. Quelle est la complexité de cet algorithme ?
3. Comment déterminer les chemins minimaux ?

	λ_0	λ_1	λ_2	λ_3	λ_4	Arcs issus de
Initialisation	0					
Première itération	0	2	5			x_0
	0	2	5	3	9	x_1
	0	2	5	3	9	x_2
	0	2	4	3	9	x_3
Deuxième itération	0	2	4	3	9	x_0
	0	2	4	3	9	x_1
	0	2	4	3	8	x_2
	0	2	4	3	8	x_3
troisième itération	0	2	4	3	8	x_0
	0	2	4	3	8	x_1
	0	2	4	3	8	x_2
	0	2	4	3	8	x_3
	Aucun	λi	modifié	STOP		

Tableau 13 : tableau

Le théorème 2 répond à la question 1.

Théorème 2 :

L'algorithme de Ford calcule les valeurs des chemins minimaux issus de x_0 quand le graphe est sans circuit absorbant.

Démonstration

Notons $\lambda_i(k)$ la valeur de λ_i à la fin de la kème itération de l'algorithme et λ_i^k la valeur d'un chemin minimal allant de x_0 à x_i parmi les chemins empruntant au plus k arcs.

On démontre le théorème par récurrence. On suppose la propriété vraie après k itérations et on démontre qu'elle reste vraie après k+1 itérations. A la fin de la kème itération de (ii), $\lambda_i = \lambda_i(k)$ est la valeur d'un chemin de x_0 à x_i et est inférieure à la valeur minimale d'un chemin allant de x_0 à x_i empruntant au plus k arcs : $\lambda_i^k \geq \lambda_i(k)$. Soit $\mu k+1(x_0, x_j)$ un chemin minimal allant de x_0 à x_j parmi les chemins empruntant au plus k+1 arcs : $\mu_{k+1}(x_0, x_j)=\mu_k(x_0, x_i)$ concaténé à (x_i, x_j), pour x_j successeur de x_i; λ_j^{k+1} est la valeur de ce chemin. Par hypothèse de récurrence, $\lambda_i^k \geq \lambda_i(k)$. En utilisant $\lambda_j^{k+1} = \lambda_i(k) + v_{ij}$. On a donc : $\lambda_j^{k+1} \geq \lambda_i(k) + v_{ij}$. et du fait de l'algorithme $\lambda_i + v_{ij} \geq \lambda_j(k+1)$. D'où le résultat. La propriété est donc vraie à l'ordre k+1. Il en résulte qu'à la n-1 ème itération, les λ_i sont les valeurs minimales des chemins issus de x_0. Q.E.D.

Le théorème 3 répond à la question 2.

Théorème 3 :

La complexité de l'algorithme de Ford est : O(n x m) où n = | X | et m = | U |.

Démonstration

Démonstration On exécute (n - 1) (ii), (ii) coûte O(m); d'où la complexité globale. Q.E.D.

Enfin pour répondre à la question 3, on utilise le corollaire : les arcs appartenant aux chemins minimaux sont les (x_i, x_j) tels que $\lambda_j = \lambda_i + v_{ij}$. Les chemins du graphe partiel formé de ces arcs sont exactement les chemins minimaux.

3. Algorithme de DIJKSTRA :

a) Algorithme de DIJKSTRA :

Cet algorithme ne sera applicable que si les valuations des arcs sont positives ou nulles. Dans cet algorithme, on va de nouveau ajuster les λ_i, mais cette fois le nombre d'ajustements sera minimal, donc égal au nombre d'arcs du graphe.

Syntaxe : Algorithme

(i) Poser S ={x_0}, $\lambda_o = 0, \lambda = v_{0i}, si\ (x_0, x_i) \in U, \lambda_i = \infty$,sinon.

(ii) Tant que $S \neq X$ faire

(a) Choisir $x_i \in X - S$ - S de λ_i minimum.

(b) Poser S = S +{ x_i }.

(c) Pour tout $x_j \in (X - S)$, successeur de x_i , poser: λ_j = min($\lambda_i + v_{ij}, \lambda_j$).

Exemple

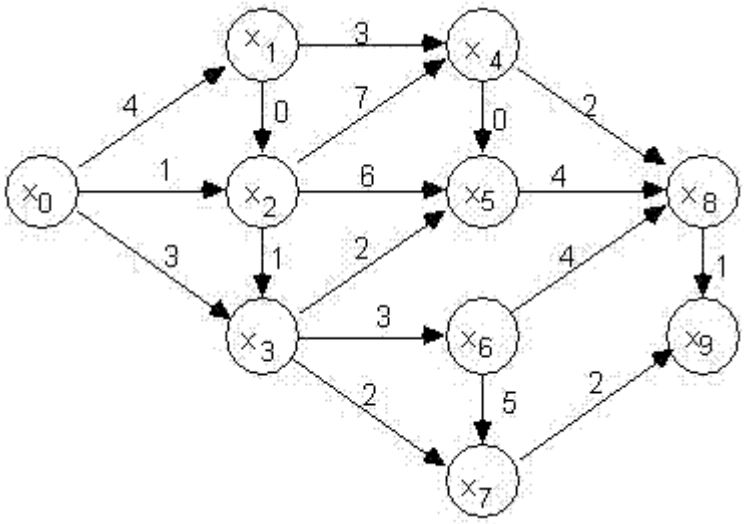


Figure 4.3: exemple

	x i	S	λ0	λ1	λ2	λ3	λ4	λ5	λ6	λ7	λ8	λ9
S	{ x0 }	{ x0 }	0	**4**	**1**	3						
S	{ x2 }	{ x0 , x2 }	0	4	1	2	8	7				
S	{ x3 }	{ x0 , x2 , x3 }	0	4	1	2	8	**4**	**5**	**4**		
S	{ x1 }	{ x0 , x1 , x2 , x3 }	0	4	1	2	**7**	4	5	4		
S	{ x5 }		0	4	1	2	7	4	5	4	**8**	
S	{ x7 }		0	4	1	2	7	4	5	4	8	**6**
S	{ x6 }		0	4	1	2	7	4	5	4	8	6
S	{ x9 }		0	4	1	2	7	4	5	4	8	6
S	{ x4 }		0	4	1	2	7	4	5	4	8	6
S	{ x8 }		0	4	1	2	7	4	5	4	8	6

Tableau 14 : tableau14

Arborescence de chemins minimaux :

Si on définit pour chaque sommet x_j son père comme le sommet x_i ayant permis de fixer λ_j (donc $\lambda_j - \lambda_j = v_{ij}$), on a une arborescence de chemins minimaux (Cf. figure 4.4).

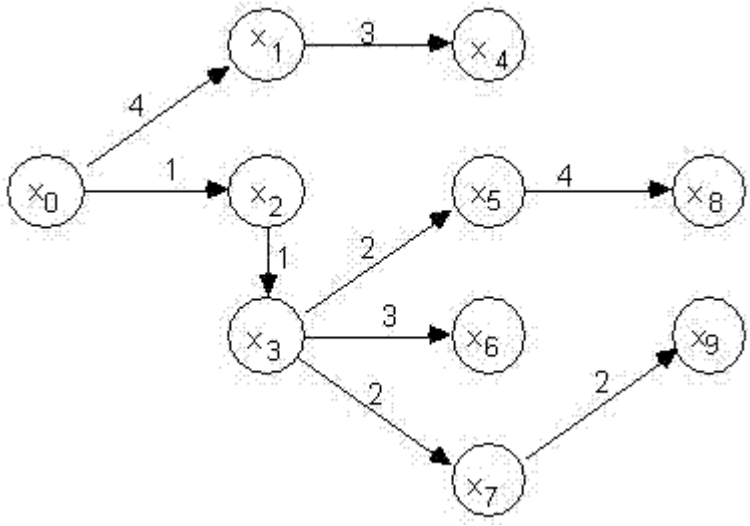


Figure 4.4: arborescence de chemins minimaux

Complexité de l'algorithme de Dijkstra :

Lemme 3

L'algorithme de Dijkstra se termine en un nombre fini d'étapes et a pour complexité $O(n^2)$. **Démonstration :**

(i), l'initialisation, coûte O(n).

(ii) (a) coûte O(n) et est faite n fois. (b) coûte O(1) et est faite n fois. (c) coûte globalement O(m) car chaque arc est examiné une fois. La complexité globale est donc: $O(n^2)$. Q.E.D.

Théorème 4

Les λ_i à la fin de l'algorithme sont les valeurs minimales des chemins issus de x0.

Démonstration

Les valuations étant positives ou nulles, il n'y a pas de circuit absorbant. Il y a donc des chemins minimaux. On va noter λ_i * la valeur minimale d'un chemin allant de x_0 à x_i et on va montrer par récurrence qu'à la fin de chaque itération (ii), on a :

1°) si $x_j \in S, \lambda_i = \lambda_j$ *

2°) si $x_j \in X - S, \lambda_j = min(\lambda_k + v_{kj}) sur, x_k \in S$, , et prédécesseur de x_j. La propriété est vraie à la fin de l'initialisation. Supposons la propriété vraie à la fin de la kème itération et montrons qu'elle reste vraie à la fin de la k+1ième itération. Soit x_i le sommet de X-S de λ_i minimal au début de la k+1ième itération. D'après l'hypothèse de récurrence, λ_i est la valeur minimale d'un chemin élémentaire allant de de x_0 à x_i et ne passant pas par un sommet de X-S. On va montrer que λ_i = λ_i * en comparant la valeur d'un chemin élémentaire allant de x_0 à x_i et passant par un sommet de X - S à λ_i et en utilisant le fait que λ_i est la valeur minimale d'un chemin allant de x_0 à x_i et ne passant que par les sommets de S.

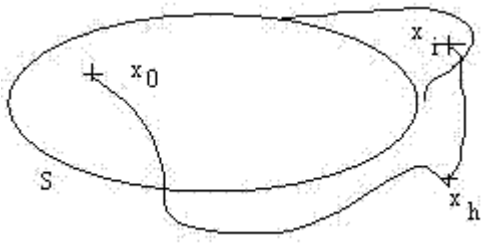


Figure 4-5: un chemin de x0 à xi

Soit $\mu(x_0, x_i)$ un tel chemin. On note x_h le premier sommet de X-S appartenant à $\mu(x_0,x_i)$.

Le sous-chemin allant de x0 à xh est de valeur supérieure à λ_h. Les valuations étant positives, la valeur du chemin $\mu(x_0,x_i)$ est supérieure à λ_h donc à λ_i, d'après l'algorithme (Cf. (ii) a)). Il en résulte que $\lambda_i = \lambda_i^*$ (Cf. figure 4.5).

D'après l'algorithme (Cf (ii) a)) :

1°) est vérifié pour les autres sommets de S par hypothèse de récurrence;

2°) résulte de l'hypothèse de récurrence et de l'ajustement dans l'étape (c) de l'algorithme. En effet, posons S' = S + x_i et considérons un sommet x_j n'appartenant pas à S'. Montrons qu'après l'ajustement $\lambda_j = min(\lambda_k + v_{kj}) pour, x_k \in S'$ et prédécesseur de xj.

Avant l'ajustement, on a par hypothèse de récurrence:

$\lambda_j = min(\lambda_k + v_{kj})\ pour, x_k \in S$ On pose ensuite $\lambda_j = min(\lambda_j, \lambda_i + v_{ij})$. Q.E.D

4. Algorithme de BELLMAN :

a) Algorithme de BELLMAN :

Il est applicable aux graphes sans circuit. On a vu qu'on pouvait numéroter les sommets d'un graphe sans circuit en O(m) opérations. Le nombre d'ajustements est également minimal car égal au nombre d'arcs du graphe.

Syntaxe : Algorithme de Bellman :

(i) Numéroter les sommets du graphe , poser $\lambda_0 = 0$.

(ii) Pour j = 1 à n - 1 poser : $\lambda_j = min(\lambda_i + v_{ij})$ sur l'ensemble des prédécesseurs x_i de x_j.

Exemple

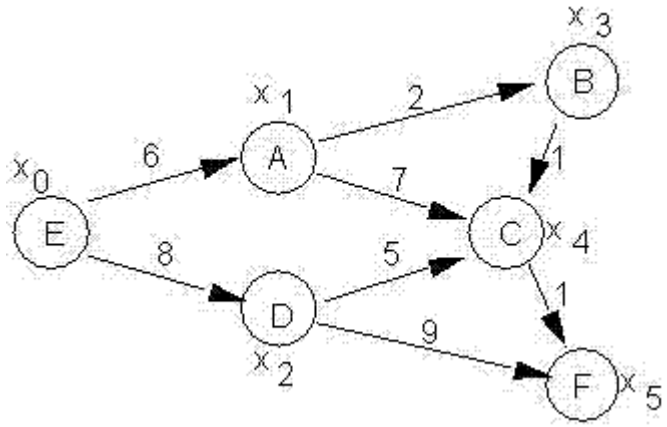


Figure 4.6 : un exemple pour Bellman

$\lambda_0 = 0 . \lambda_1 = 6 (0+6) . \lambda_2 = 8 (0+8) . \lambda_3 = 8 (6+2) . \lambda_4 = 9 = min(8+5, 6+7) . \lambda_5 = 10 = min(9+1, 8+9).$

Théorème 5 :

L'algorithme de Bellman calcule les valeurs λ_i des chemins minimaux issus de x_0 en O(m) itérations.

Démonstration :

(i) coûte O(m). (ii) coûte O(m) car on examine un arc exactement une fois. On suppose par récurrence que, à l'ordre j, on a : $\lambda_1 = \lambda_1^*, \lambda_2 = \lambda_2^*, \ldots, \lambda_j = \lambda_j^*$ où les λ_j^* sont les valeurs minimales des chemins allant de x_0 à x_i, qui existent puisque le graphe est sans circuit, donc sans circuit absorbant. La propriété est vraie à l'ordre j+1, car un chemin minimal allant de x_0 à x_{j+1} est obtenu en ajoutant l'arc (x_k ,x_{j+1}) à un chemin minimal allant de x_0 à x_k pour un prédécesseur x_k de x_{j+1}. Ceci démontre la propriété par récurrence. Q.E.D.

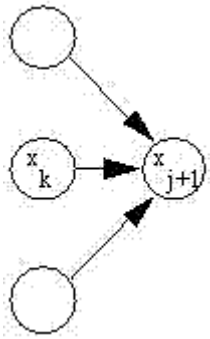


Figure 4.7 : preuve du théorème 5

5. Méthode matricielle :

a) Méthode matricielle :

La méthode matricielle permet de calculer la valeur des chemins minimaux entre toute paire de sommets dans un graphe sans circuit absorbant pour une complexité $O(n^3)$.

Syntaxe : Algorithme

(i) Soit $V_0=(v_{0ij})$ la matrice des valuations des arcs de G .

($v_{ij}^0=\infty$ si (x_i, x_j) n'appartient pas à U)

(ii) Pour k = 1 à n

Pour i = 1 à n

Pour j = 1 à n Faire

v_{kij} = min (v_{k-1ij} , v_{k-1ik} + v_{k-1kj})

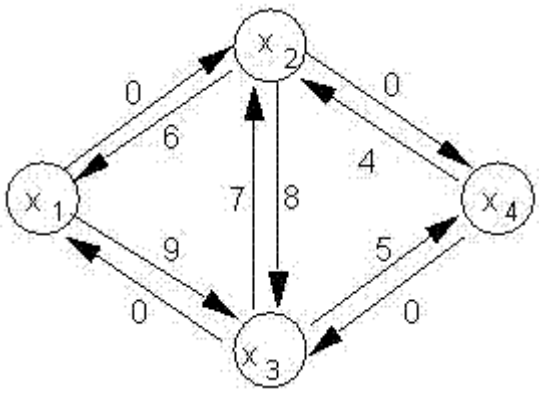


Figure 4-8: un exemple pour la méthode matricielle

V^0=

∞	0	9	∞
6	∞	8	0
0	7	∞	5
∞	4	0	∞

V^1=

∞	0	9	∞
6	6	8	0
0	0	9	5
∞	4	0	∞

V^2=

6	0	8	0
6	6	8	0
0	0	8	0
10	4	0	4

V^3=

6	0	8	0
6	6	8	0
0	0	8	0
0	0	0	0

V^4=

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Tableau 15 : Figure 4.9 : application de la méthode matricielle à l'exemple

Théorème 5 :

En l'absence de circuit absorbant, à la fin de la kième itération, v^k_{ij} est la valeur minimale d'un chemin élémentaire allant de i à j et ne passant que par les sommets 1 , 2 , ... , k.

Démonstration :

Ce résultat se démontre immédiatement par récurrence. L'absence de circuit absorbant permet de limiter la recherche des chemins minimaux aux chemins élémentaires. Un chemin élémentaire entre i et j, s'il existe, ne passant que par 1, 2, ...,k est soit un chemin passant une fois par k, soit un chemin ne passant pas par k. Ceci explique la formule de l'algorithme. Q.E.D.

Remarque

S'il y a des circuits absorbants des termes négatifs apparaissent sur la diagonale principale d'une des matrices.

6. Autres problèmes de cheminement :

a) Autres problèmes de cheminement :

D'autres problèmes de cheminement se résolvent de manière similaire. Nous citons les problèmes suivants ainsi que leurs méthodes de résolution :

- Chemins de valeur maximale : Ford et Bellman restent valables en remplaçant MIN par MAX et en initialisant les $\lambda_i à -\infty, et, \lambda_i à 0$.

- Existence d'un chemin de i à j : méthode matricielle ou matrice booléenne associée élevée à une puissance (Cf. : TD).

- Dénombrement du nombre de chemins entre i et j : M matrice réelle associée (nombre de chemins de longueur 1), M^k (nombre de chemins de longueur k).

- Chemins de fiabilité maximale entre i et j . La fiabilité d'un chemin est le produit des valuations des arcs de ce chemin (fiabilité : probabilité de ne pas tomber en

panne). On utilise Ford ou Bellman en remplaçant + par x et MIN par MAX. Les λ_i sont initialisés à 0, sauf λ_0 qui est initialisé à 1.

- Chemins de capacité maximale. La capacité maximale d'un chemin est la plus petite des valuations (≥ 0) de ce chemin. On utilise également Ford et Bellman en remplaçant + par MAX et MIN par MAX. Les λ_i sont initialisés à -1, sauf λ_0 qui est initialisé à ∞ .

F. PROBLÈMES D'ORDONNANCEMENT

Jusqu'en 1958, on utilisait les diagrammes de Gantt (appelés aussi planning à barres).

Cette année là, se sont développées deux méthodes fondées sur les graphes :

1. La méthode potentiel-tâches, à l'occasion de la construction du paquebot FRANCE;
2. La méthode PERT, lors de celle des fusées POLARIS.

Ces méthodes ne prennent en compte que les contraintes potentielles (succession et localisation temporelle) et ne gèrent pas les contraintes de ressources. Elles sont fondées toutes les deux sur la recherche de chemins maximaux, que l'on appelle chemins critiques. Cela n'était pas trop pénalisant dans la mesure où, si les projets correspondants comportaient de très nombreuses tâches, ils étaient aussi de prestige et la fonction coût était donc secondaire. L'important était de terminer le projet le plus tôt possible, ce que permet les méthodes de chemin critique.

1. Graphes conjonctifs, ensemble de potentiels

a) Graphes conjonctifs, ensemble de potentiels :

Graphes conjonctifs, ensemble de potentiels : Dans l'une ou l'autre méthode, on associe au problème un graphe conjonctif. Un ordonnancement sera un ensemble de potentiels sur le graphe conjonctif.

Définition : Graphe conjonctif :

Un graphe conjonctif est un graphe G = (X, U) valué ayant une racine 0 et une antiracine n+1 tel qu'il existe un chemin de valeur positive entre la racine et tout autre sommet et entre tout sommet différent de l'antiracine et l'antiracine.

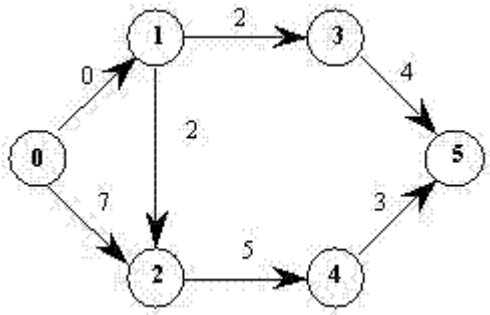


Figure 5.1 : exemple de graphe conjonctif

Définition : Ensemble de potentiels :

Un ensemble de potentiels sur un graphe conjonctif G=(X, U) est une application t de X dans R telle que :

1. $t_o = 0$,
2. $v_{ij} \leq t_j - t_i$, pour tout arc (i,j) de U.

Par abus d'écriture, on écrira T= $\{t_i / i \in X\}$.

Théorème d'existence :

Une condition nécessaire et suffisante pour qu'il existe un ensemble de potentiels sur un graphe conjonctif G = (X, U) est que ce graphe n'ait pas de circuit de valeur strictement positive.

Démonstration :

La condition est nécessaire :

S'il existe un circuit [i1, i2, ... , ir , i1] de valeur strictement positive, c'est-à-dire tel que : $v_{i1i2} + v_{i2i3} + \ldots + v_{iri1} > 0$, on a, pour tout ensemble de potentiels T :

$t_{i2} - t_{i1} \geq v_{i1i2}, t_{i3} - t_{i2} \geq v_{i2i3}, \ldots, t_{i1} - t_{ir} \geq v_{iri1}$.

En sommant, on trouve : 0 > 0, ce qui est absurde.

La condition est suffisante :

S'il n'existe pas de circuit de valeur strictement positive, il existe un chemin maximal entre 0 et i.

On va noter r_i la valeur d'un tel chemin et montrer que R = $\{r_i / i \in X\}$ est un ensemble de potentiels. En effet, la valeur maximale r_j d'un chemin de 0 à j est supérieure à $r_i + v_{ij}$ qui est la valeur d'un chemin de 0 à j obtenu en concaténant un chemin maximal de 0 à i à l'arc (i,j). Il en résulte :

1. $r_o = 0$,

2. $r_j - r_i \geq v_{ij}$. Q.E.D.

Ensemble de potentiels calé à gauche et calé à droite:

On suppose dans la suite que le graphe est sans circuit de valeur strictement positive et nous notons l(i,j) la valeur maximale d'un chemin allant de i à j

On a vu que R = $\{r_i = l(0,i) / i \in X\}$ était un ensemble de potentiels (il sera dit calé à gauche ou au plus tôt).

On va montrer que F = $\{ f_i = l(0,n+1) - l(i,n+1) / i \in X \}$ est également un ensemble de potentiels (il sera dit calé à droite ou au plus tard). Posons t* = l(0,n+1).

Proposition 1 :

Pour tout ensemble de potentiels T= $\{t_i / i \in X\}$, $t_{n+1} \geq t^* = l(0,n+1)$. En particulier t* est la durée optimale de l'ordonnancement. De plus, pour tout i, on a: $r_i \leq t_i$ (r_i : date au plus tôt). Enfin, si T est un ordonnancement optimal, donc de durée t*, alors $t_i \leq f_i$, (f_i : date au plus tard).

Démonstration :

Soit [i_o=0 , ... , i_r = n+1] un chemin de valeur maximale entre 0 et n+1, donc de valeur l(0,n+1) (un tel chemin est dit critique). $t_{i1} - t_{io} \geq v_{ioi1}$, $t_{i2} - t_{i1} \geq v_{i1i2}$, ..., $t_{ir} - t_{ir-1} \geq v_{ir-1ir}$. En sommant, on obtient: $t_{ir} - t_{io} = l(0,n+1)$ et puisque $i_r = n+1$ et $i_o = 0$, on a: $t_{n+1} \geq t^* = l(0,n+1)$.

On vérifie de façon analogue que R = $\{r_i = l(0,i) / i \in X\}$ est un ensemble de potentiels d'écart minimal entre t_o et t_i, d'où $t_i \geq r_i$. L'écart minimal entre t_i et t_{n+1}

est égal à l(i,n+1), en conséquence si F = {f_i = l(0,n+1)-l(i,n+1) / i ∈ X}, on a : $t_i \leq f_i$ si $t_{n+1} = t^*$. Q.E.D.

2. La méthode potentiel-tâches

a) La méthode potentiel-tâches :

Méthode

Tâches	Durée	contraintes potentielles
1	3	
2	7	
3	4	La tâche 1 précède la tâche 3
4	6	Les tâches 1 et 2 précèdent la tâche 4
5	5	La tâche 3 précède la tâche 5
6	3	Les tâches 3 et 4 précèdent la tâche 6
7	2	La tâche 6 précède la tâche 7

Tableau 16 : Figure 5-2 : exemple

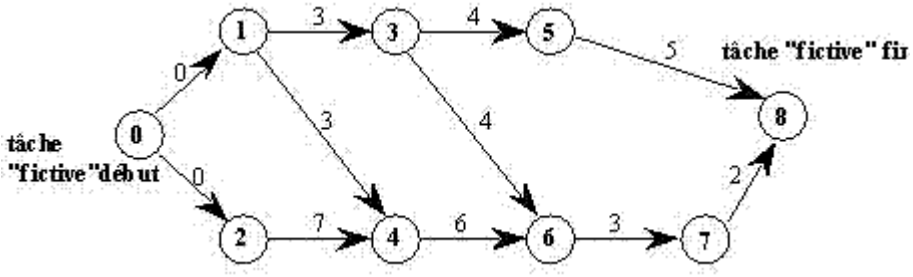


Figure 5.3 : le graphe potentiel-tâches de l'exemple.

On associe de façon canonique à ce problème un graphe G = (X , U) où X = {0, 1, 2, ... , n+1}

(0 : tâche fictive début; n+1 : tâche fictive fin), et U est associée aux contraintes potentielles, qui comprennent d'une part les contraintes initiales, d'autre part les contraintes dues aux tâches 0 et n+1 : on relie tout sommet sans prédécesseur à 0 par un arc de valuation 0 et on relie tout sommet i sans successeur à n+1 par un arc de valuation égale à la durée de la tâche i.

Définition : Le chemin critique:

Les chemins maximaux entre 0 et n+1, dits critiques, jouent un rôle central dans cette méthode. Les tâches appartenant à ce chemin critique sont dites critiques. Si on retarde d'un certain délai une tâche critique, l'ordonnancement sera retardé du même délai. Cela résulte du fait que les tâches critiques ont des dates au plus tard égales aux dates au plus tôt : elles sont de marge nulle.

Définition : Ordonnancement :

Un ordonnancement est un ensemble de potentiels sur le graphe conjonctif associé.

Tout ordonnancement est de durée supérieure à l(0,n+1) = t* valeur du chemin critique. On calcule le plus souvent, les ordonnancements calés à gauche et à droite (au plus tôt et au plus tard), en résolvant deux problèmes de cheminements: r_i = l(0, i) , f_i = l(0,n+1) - l(i,n+1).

On utilisera BELLMAN en l'absence de circuit ou FORD, s'il y a un circuit, en utilisant les formules suivantes.

pour les dates au plus tôt :

$r_0 = 0$, $r_j = \max (r_i + v_{ij})$ (avec $i \in U^-(j)$).

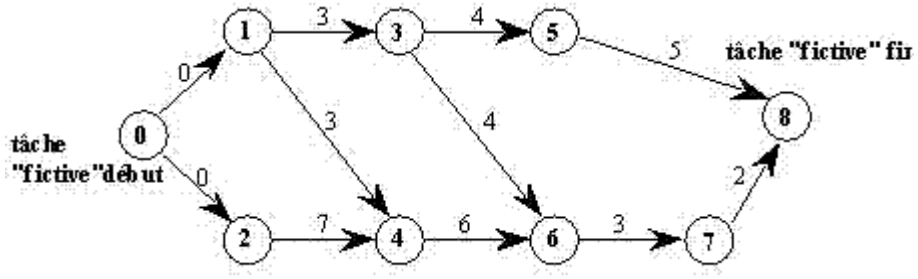


Figure 5.3 : le graphe potentiel-tâches de l'exemple.

pour les dates au plus tard :

$f_{n+1} = t^*$, $f_i = \min (f_j - v_{ij})$ (avec $j \in U^+(i)$).

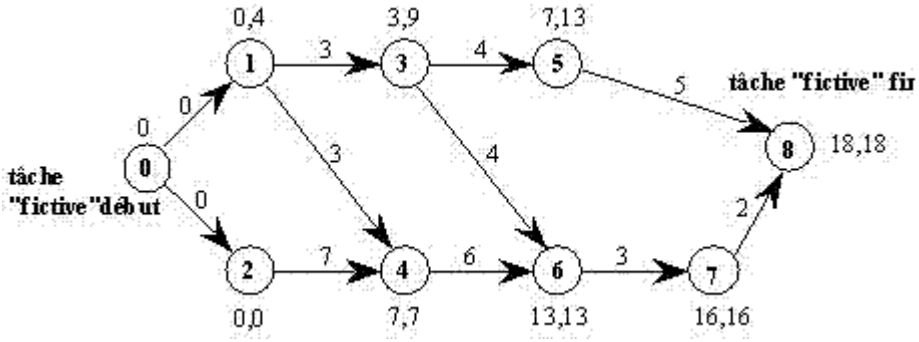


Figure 5.5 : dates au plus tard.

Les différents types de contraintes potentielles :

Nous rapportons ci-dessous les différents types de contraintes potentielles qui comprennent les contraintes de succession et de localisation temporelle.

Contraintes de succession :

i précède j, en appelant t_i (resp. t_j) la date de début de i (resp. j) et p_i la durée de i, la contrainte s'écrit $t_j \geq t_i + p_i$, c'est-à-dire $t_j - t_i \geq p_i$: on associe un arc (i,j) valué par p_i.

Contraintes de localisation temporelle :

La tâche i est disponible à la date a_i, $t_i \geq a_i$ ou $t_i - t_0 \geq a_i$ ($t_0 = 0$). On associe un arc (0,i) valué par a_i. La tâche i doit être achevée à la date d_i : $t_i + p_i \leq d_i$, qui s'écrit: $t_o - t_i \geq p_i - d_i$. On associe un arc (i,0) valué par $p_i - d_i$ (cette valuation est négative ou nulle).

Contraintes de succession au sens large :

La tâche j pourra débuter après la tâche i plus un temps de réglage r_{ij}: $t_j \geq t_i + p_i + r_{ij}$, d'où $t_j - t_i \geq p_i + r_{ij}$. La tâche j pourra commencer après que i soit commencée d'un tiers : $t_j \geq t_i + p_i/3$.

3. La méthode PERT

a) La méthode PERT :

Méthode

On associe à chaque tâche i, un événement D_i, début de la tâche i et un événement

F_i, fin de la tâche i. On introduit également un événement D début de l'ordonnancement et F fin de l'ordonnancement.

Le graphe conjonctif associé a pour sommets l'ensemble des événements et pour arcs l'ensemble des tâches, plus des arcs dits fictifs permettant de représenter les contraintes.

Exemple

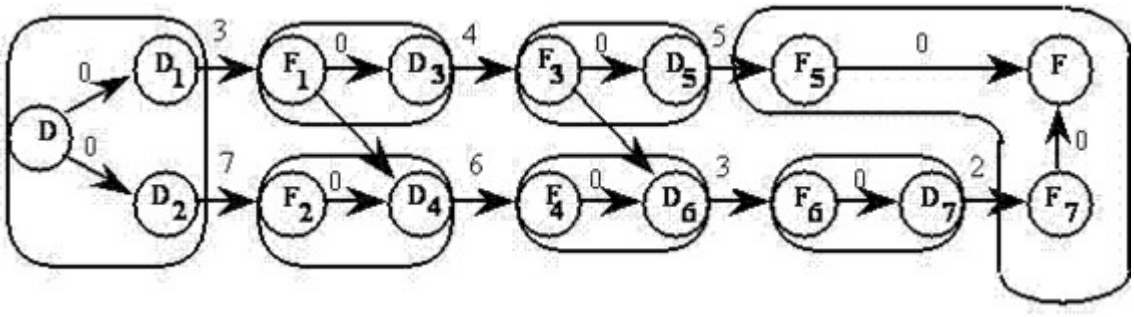


Figure 5.6 graphe PERT développé

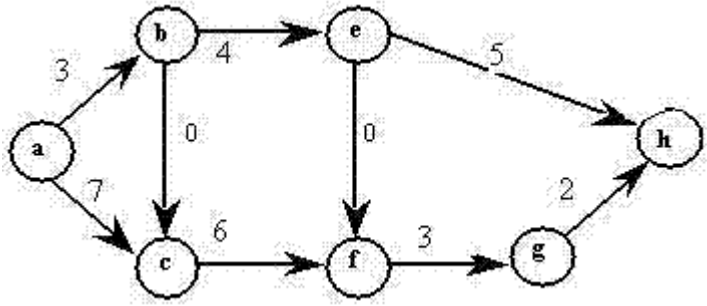


Figure 5.7 : graphe PERT simplifié

On peut simplifier le graphe PERT en fusionnant des événements début ou fin de tâches. Mais les inconvénients du graphe simplifié sont :

- la non-automaticité de sa construction;
- le fait qu'il n'est pas unique;
- le fait que si on modifie une contrainte (ajout ou retrait), il faut reconstruire tout le graphe simplifié

L'avantage du graphe PERT est une meilleure lisibilité pour des non-spécialistes. En effet une tâche qui a une certaine durée est représentée par une "flèche".

G. LE PROBLÈME DU FLOT MAXIMAL

Nous allons étudier dans ce chapitre le problème du flot maximal. Ce problème apparaît quand il s'agit de maximiser la quantité transportée d'une ou plusieurs sources vers une ou plusieurs destinations. Il est modélisé par un graphe valué, appelé réseau de transport, que nous définissons. Les arcs représentent les possibilités de transport entre deux sites et sont valués par les capacités correspondantes. Le problème du flot maximal est résolu par un algorithme efficace fondamental du à FORD et FULKERSON, que nous présentons et justifions ci-dessous. Pour terminer nous discutons de la polynomialité de ce problème en

évoquant des travaux de recherche plus récents qui ont permis la mise au point d'algorithmes plus performants.

1. Réseau de transport

a) Réseau de transport :

Définition

On appelle réseau de transport un graphe valué positivement sans boucle, ayant une racine s, un puits p et contenant l'arc (p,s) de valuation infinie. Les valuations des arcs sont appelées capacités.

Un exemple de réseau de transport est rapporté sur la figure 6.1 ci-dessous.

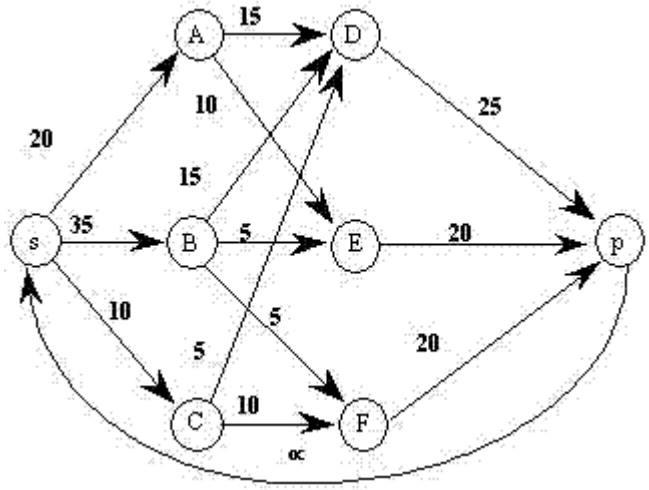


Figure 6.1 : réseau de transport

Définition

On appelle flot sur un réseau de transport G = (X, U) une application f : U dans R qui vérifie :

1. Les contraintes de capacité :
   pour tout arc (i,j) de $U : 0 \leq f_{ij} \leq c_{ij}$;
2. Les contraintes de conservation (Kirchoff) (Cf figure 6.2) pour tout sommet i de X, on a : $\sum_{j \in U^+(i)} f_{ij} = \sum_{k \in U^-(i)} f_{ki}$

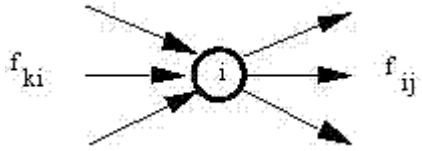


Figure 6.2 : contrainte de conservation

Un premier exemple de flot est le flot nul. Un deuxième exemple de flot est rapporté sur la figure 6.3.

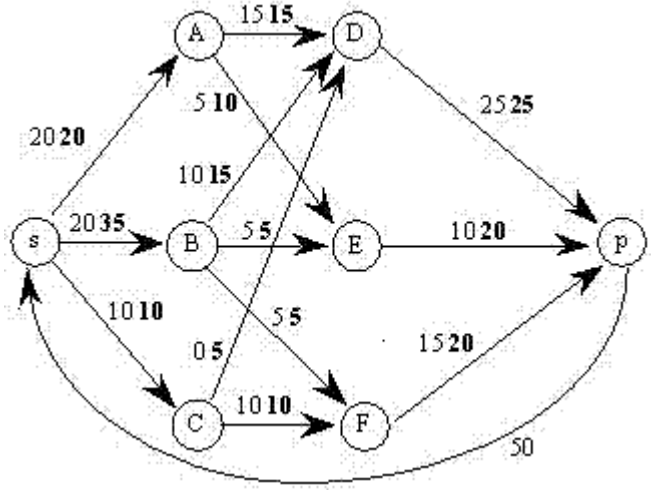


Figure 6.3 : un flot complet.

Exemple

En trois dépôts A, B, C, on dispose respectivement de 20, 35 et 10 tonnes de marchandises. On a des demandes de 25, 20 et 20 tonnes aux destinations D, E et F. Il existe des possibilités de transport à l'aide de camions. Ces possibilités sont rapportées dans le tableau suivant :

	D	E	F
A	15	10	0
B	15	5	5
C	5	0	10

Tableau 17 : tableau17

Problème :

Déterminer un plan de transport permettant de transporter des origines aux destinations une quantité maximale.

A ce problème, on associe le graphe de la figure 6.1. Plus généralement, il faut résoudre le problème suivant:

Déterminer un flot de valeur maximale sur l'arc (p, s), donc maximiser fps.

2. Lemme

a) Lemme :

Avant d'introduire les algorithmes pour maximiser le flot, nous montrons que la propriété de conservation du flot se généralise à un sous-ensemble de sommets.

Lemme :

Si Y est un sous-ensemble de X, le flot sortant de Y est égal au flot entrant dans Y.

Démonstration :

On somme l'égalité de Kirchoff sur l'ensemble des sommets de Y. Les flux sur les arcs qui ont une extrémité dans Y et l'autre en dehors de Y apparaissent de chaque côté de l'égalité. On peut donc les soustraire. Il reste alors d'un côté de l'égalité la somme des flux des arcs sortants de Y et de l'autre côté de l'égalité, la somme des flux des arcs entrants dans Y. On a donc le résultat. Q.E.D.

Figure 6.4: preuve du lemme.

3. Flot complet

a) Introduction :

Introduction : Une première idée pour optimiser le flot est de saturer successivement les chemins de s à p. On obtiendra alors un flot dit complet qui, comme nous le verrons ci-dessous, n'est pas maximal, mais fournit une excellente solution de départ pour appliquer l'algorithme de Ford-Fulkerson que nous présenterons au paragaphe suivant.

Définition

On dit qu'un flot est complet si tout chemin du réseau de transport allant de s à p contient au moins un arc saturé , c'est-à-dire un arc (i,j) tel que $f_{ij} = c_{ij}$.

b) Algorithme de recherche d'un flot complet :

Syntaxe

On part d'un flot f (par exemple, f = 0) et on l'améliore pas à pas par une procédure de marquage:

(I) Marquer s.

(II) Soit i un sommet marqué non encore examiné; marquer j si j est un successeur non marqué de i avec $f_{ij} < c_{ij}$. La marque de j est +i.

(III) Si p est marqué, aller en (IV).

Si tous les sommets marqués sont examinés, le flot est complet FIN.

Sinon aller en (II).

(IV) Améliorer le flot. Effacer les marques (sauf celle de s) et aller en (I).

c) Fonctionnement sur l'exemple précédent :

On trouve successivement les chemins améliorants suivants : sADp (15), sAEp (5), sBDp (10), sBEp (5), sBFp (5), et sCFp (10). Le flot résultant a pour valeur 50 et est complet (Cf figure 6.3). En effet tous les chemins passant par l'arc sA sont saturés car cet arc est saturé; tous les chemins passant par l'arc sC sont saturés car cet arc est saturé; tous les chemins passant par l'arc sB sont saturés car les arcs Dp,BE et BF sont saturés.

4. Algorithme de FORD-FULKERSON :

Pour construire un flot complet, on a supprimé les chemins améliorants. Il n'est pas optimal car il reste une chaîne améliorante. Nous expliquons ci-dessous la notion de chaîne améliorante et nous rapportons l'algorithme de Ford-Fulkerson qui permet

de chercher les chaînes améliorantes et qui s'arrête quand il n'y a plus de chaîne améliorante. Le flot est alors optimal.

a) Chaîne améliorante :

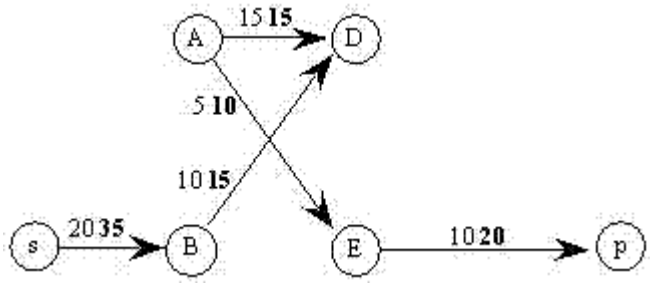


Figure 6.5: une chaîne améliorante

Dans un chemin de s à p, le sens des arcs est respecté. Dans une chaîne de s à p, au contraire, le sens des arcs n'est pas forcément respecté. On peut alors distinguer deux types d'arcs: les arcs qui respectent l'orientation (arcs dits dans le sens +) et les arcs qui ne respectent pas l'orientation (arcs dits dans le sens-). Une chaîne améliorante est une chaîne allant de s à p dont les arcs allant dans le sens + sont insaturés et les arcs dans le sens - transportent un flux strictement positif. Elle permet d'améliorer le flot, en ajoutant une quantité δ sur les arcs dans le sens +, et en retirant une quantité δ sur les arcs dans le sens -.

L'algorithme de Ford - Fulkerson part d'un flot quelconque (par exemple, le flot nul ou un flot complet). Il cherche une chaîne améliorante. Il améliore le flot tant qu'il existe une chaîne améliorante et sinon il s'arrête. Le flot est alors optimal.

b) Algorithme de Ford - Fulkerson :

Syntaxe

(I) Marquer l'entrée s par *.

(II) Soit i un sommet marqué non examiné.

Etudier tous les successeurs j de i :

Marquer j s'il est non marqué et si $f_{ij} < c_{ij}$ par +i.

Etudier tous les prédécesseurs k de i:

Marquer k s'il est non marqué et si $f_{ki} > 0$ par -i.

(III) Si p est marqué, aller en (IV)

S'il reste des sommets marqués non examinés, aller en (II).

Sinon le flot est optimal, FIN.

(IV) Améliorer le flot à l'aide de la chaîne améliorante ayant permis de marquer p.

Effacer les marques, sauf celle de s, et aller en (I).

c) Exemple :

Exemple

(I) On marque s par * et nous partons du flot complet construit précédemment (Cf figure 6.3). (II) On examine s

On marque B par +s, car seul l'arc sB est insaturé.

(II) On examine B

BD est le seul arc insaturé sortant de B; on marque donc D par +B (successeur de B).

B n'a pas de prédécesseur non marqué. (

II) On examine D

Il n'y a pas d'arc insaturé sortant de D.

Par contre, l'arc AD transporte un flux non nul. Cela entraîne que A est marqué -D (prédécesseur de D).

(II) On examine A

On marque E par +A, car il est successeur insaturé de A.

Le seul prédécesseur de A est déjà marqué.

(II) On examine E

On marque p par +E, c'est-à-dire comme successeur de E.

(IV) p étant marqué, on a trouvé une chaîne améliorante. On utilise les marques "en remontant" pour reconstituer cette chaîne améliorante: c'est la chaîne sBDAEp, dont les quatre arcs dans le sens + sont insaturés, alors que l'arc dans le sens - transporte un flux non nul. Cette chaîne permet d'améliorer le flot de 5 unités.

On efface les marques sauf celles de s et on recommence.

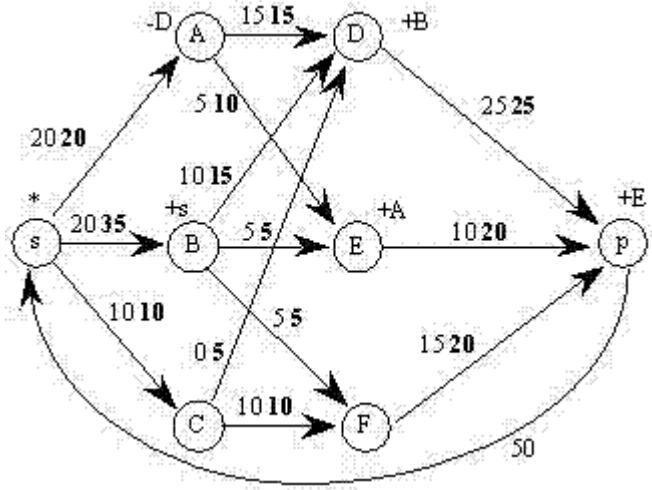


Figure 6.6 : marquage de Ford-Fulkerson

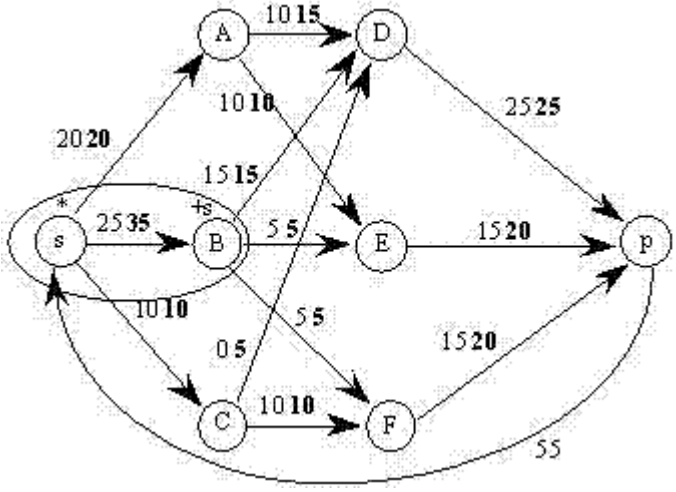


Figure 6.7 : flot optimal.

(ii) Cette fois, on marque le sommet B et c'est fini car tous les arcs sortants de B sont maintenant saturés. Le flot est maximal comme nous allons le montrer ci-dessous dans le cas général.

d) Théorème de Ford - Fulkerson :

Définition

Une coupe est un ensemble de sommets contenant s et ne contenant pas p. La capacité d'une coupe est la somme des capacités des arcs sortant de cette coupe.

Exemple

à la fin de l'algorithme précédent l'ensemble des sommets marqués M = {s, B} est une coupe qui a la propriété suivante :

- tous les arcs sortants de M sont saturés;
- les arcs entrants dans M, à l'exception de (p,s), ont un flux nul.

Théorème de Ford-Fulkerson :

Théorème de Ford-Fulkerson : La capacité minimale d'une coupe est égale au flot maximal.

Démonstration :

Il faut montrer que : Max f_{ps} = Min C(Y) (avec f, flot, $Y \subseteq X, s \in Y, p \notin Y$)

1. **Montrons que Max $f_{ps} \leq$ Min C(Y) :**
   Soit f un flot quelconque et Y une coupe quelconque, il suffit de montrer que : $f_{ps} \leq C(Y)$.
   Le flot sortant de Y est inférieur à C(Y). Le flot entrant dans Y est supérieur à f_{ps}. Comme le flot entrant dans Y est égal au flot sortant de Y, on a : $f_{ps} \leq C(Y)$.
2. Montrons que Max $f_{ps} \geq$ Min C(Y) :
   On va montrer qu'il existe un flot f ' et une coupe M telle que : $f'_{ps} = C(M)$. L'inégalité en résultera car Max $f_{ps} \geq f'ps$ et Min C(Y) $\leq$ C(M).

Quand on applique Ford-Fulkerson, à la fin, si on désigne par M l'ensemble des sommets marqués et par f ' le flot, on a C(M) = f'_{ps}.

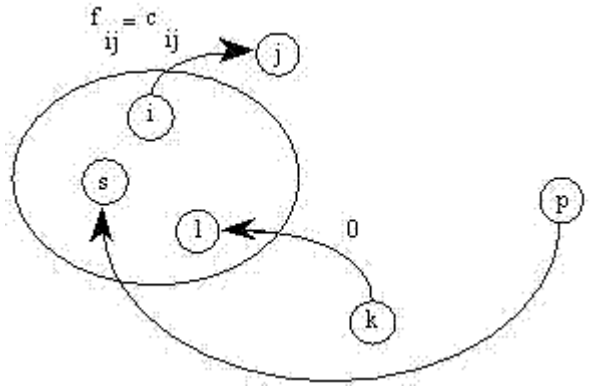


Figure 6.8 : coupe minimale.

Après un nombre fini d'itérations, Ford-Fulkerson s'arrête. En effet , chaque itération permet d'améliorer le flot d'au moins une unité; ce flot est borné, par exemple, par C({s}). A la fin , les arcs sortants de M sont saturés et les arcs entrants dans M ont un flux nul. Le flot entrant dans M est f_{ps}. Le flot sortant de M est C(M). D'où C(M) = f_{ps}. Q.E.D.

On démontre immédiatement les corollaires suivants :

Corollaire 1 : Une C.N.S pour qu'un flot soit maximal est qu'il n'existe pas de chaîne améliorante.

Corollaire 2 : L'algorithme de Ford-Fulkerson construit un flot maximal.

e) Complexité de l'algorithme de Ford-Fulkerson :

La figure 6.9 rapporte un exemple où le nombre d'itérations de Ford-Fulkerson est égal à 2M si on choisit successivement les chaînes améliorantes {s x y p} et {s y x p} de capacités 1. Le nombre d'améliorations peut donc être exponentiel. L'algorithme de Ford-Fulkerson n'est pas polynomial.

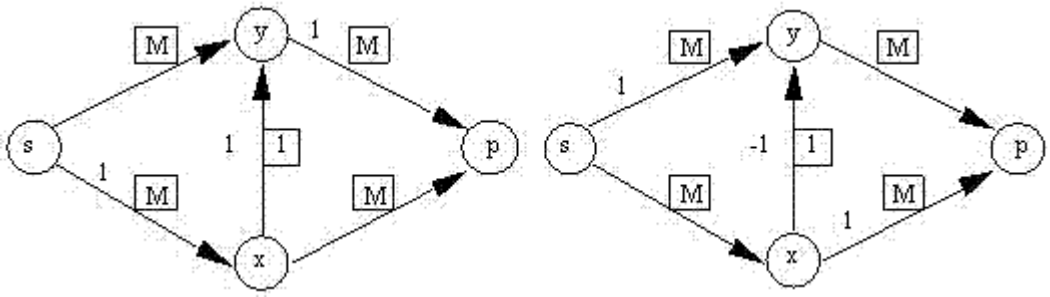


Figure 6.9 : exemple.

Par contre, si on applique la règle "premier marqué, premier examiné" l'algorithme de Ford-Fulkerson devient polynomial de complexité $O(n^5)$: le nombre de chaînes améliorantes est majoré par 1/4 (n^3 - n) d'après le théorème d'EDMONS et KARP. Cela revient à chercher les chaînes améliorantes les plus courtes. Remarquons pour terminer qu'il existe de meilleurs algorithmes, en particulier, l'algorithme de KARZANOV de complexité globale $O(n^3)$.

5. Flot maximal à coût minimal :

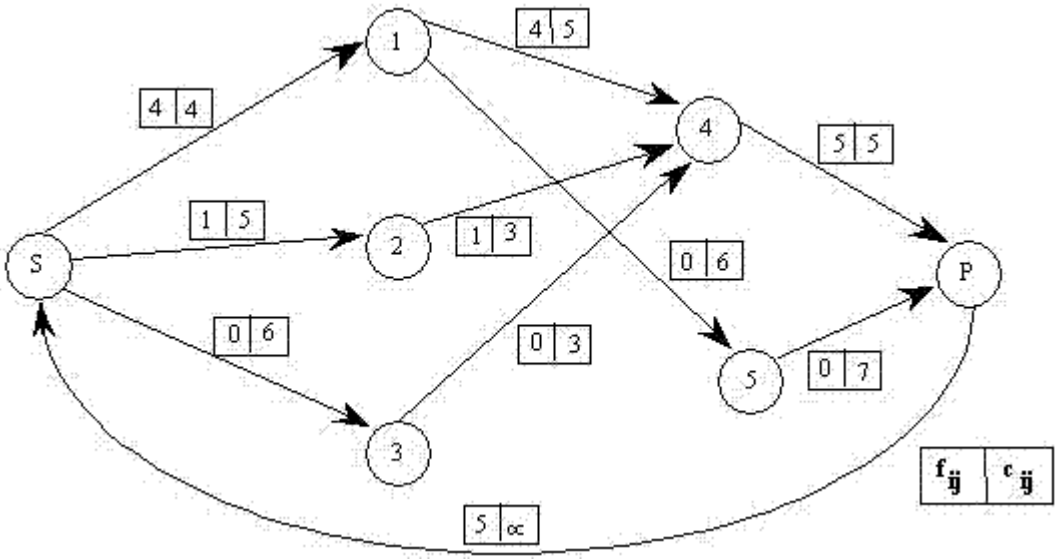


Figure 6.10 : flot sur un réseau de transport

Nous revenons sur l'algorithme de Ford-Fulkerson en montrant que la notion de chaîne améliorante sur le graphe initial est équivalente à la notion de chemin améliorant sur le graphe d'écart. Sur la figure 6.10, on a un flot sur le réseau de transport. Le graphe d'écart associé est rapporté sur la figure 6.11.

a) Problème du flot maximal :

Définition : Définition du graphe d'écart Ge(f) :

Au couple formé d'un flot f et d'un réseau de transport G, on associe le graphe d'écart Ge(f)=(X,Ue(f)) défini par :

à tout arc (i, j) de G, correspond deux arcs : un arc (i, j) de Ge(f) de capacité c_{ij} - f_{ij} et un arc (j, i) de Ge(f) de capacité f_{ij} (si la capacité d'un arc est nulle dans Ge(f), on ne représente pas l'arc, car il est inutile).

Remarquons qu'ajouter (resp. retirer) un flux sur un arc du graphe initial revient à ajouter ce flux sur l'arc correspondant (resp. inverse) du graphe d'écart. Les chaînes améliorantes du graphe initial correspondent à des chemins du graphe d'écart. L'algorithme de Ford-Fulkerson devient donc l'algorithme suivant :

Syntaxe : Algorithme du graphe d'écart :

Cet algorithme permet de construire un flot de valeur maximale sur l'arc de retour (p, s) dans G.

(I) Initialiser f sur tous les arcs (par exemple, f = 0, si b(u)= 0, pour toutu).

(II) Construire Ge(f) et chercher un chemin de s à p dans Ge(f). Aller en (III)

si un chemin a été trouvé

sinon FIN.

(III) Améliorer le flot f le long du chemin obtenu et retourner en (II).

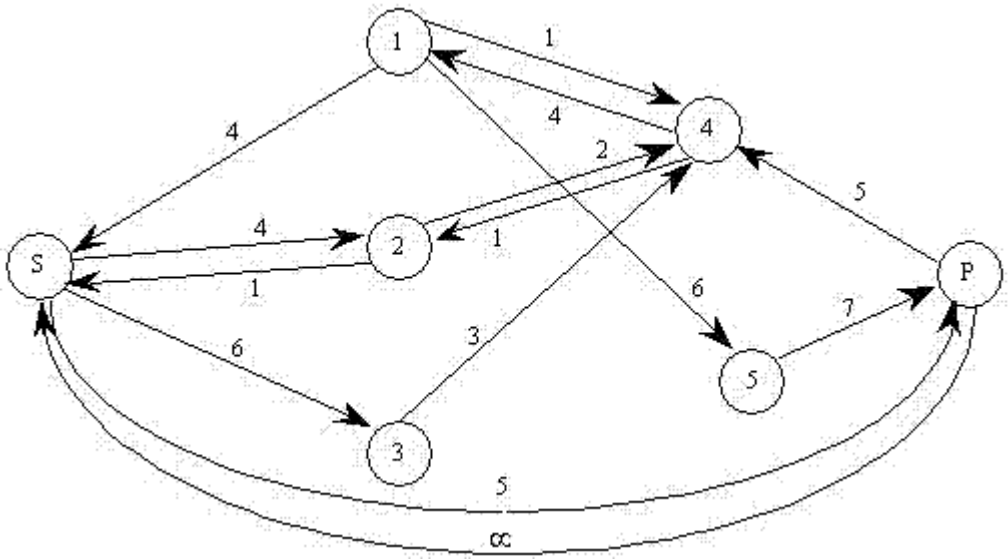


Figure 6.11 : premier graphe d'écart.

Application de l'algorithme à l'exemple précédent

Sur le premier graphe d'écart (figure 6.11), on a le chemin [s, 3, 4, 1, 5, p] qui permet d'améliorer le flot de 3. Sur le deuxième graphe d'écart (figure 6.12), on a le chemin [s, 2, 4, 1, 5, p] qui permet d'améliorer le flot de 1. Le troisième graphe d'écart (figure 6.13), ne contient plus de chemin de s à p; le flot est donc maximal. Les flux sont égaux à la capacité des arcs inverses de ce troisième graphe d'écart (Cf. fig 6.14).

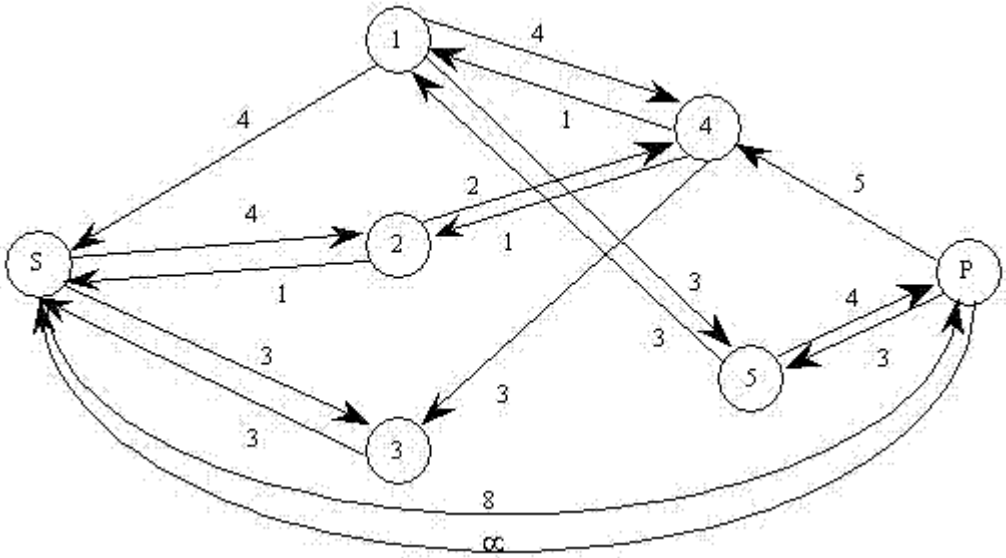


Figure 6.12 deuxième graphe d'écart

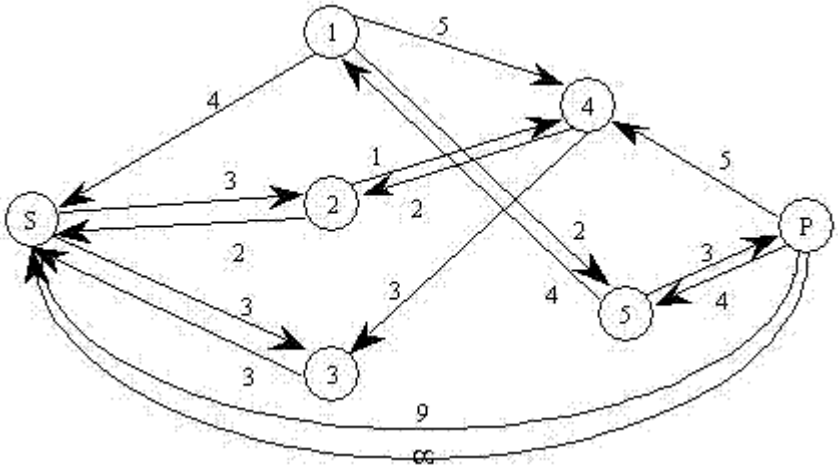


Figure 6.13: troisième graphe d'écart

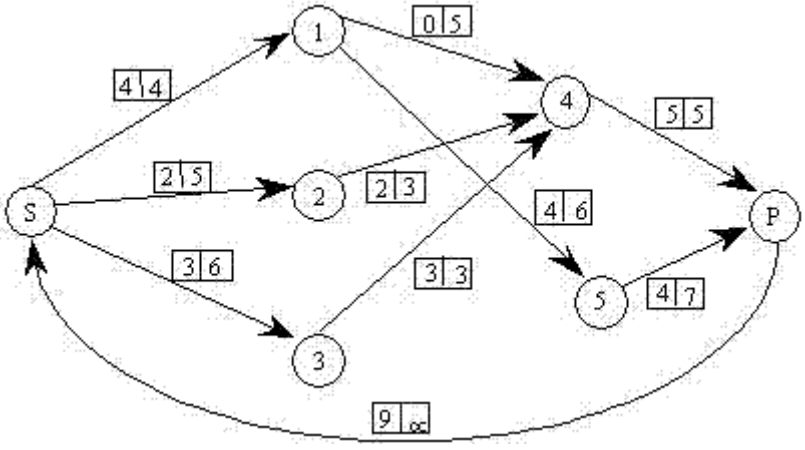


Figure 6.14 : flot maximal.

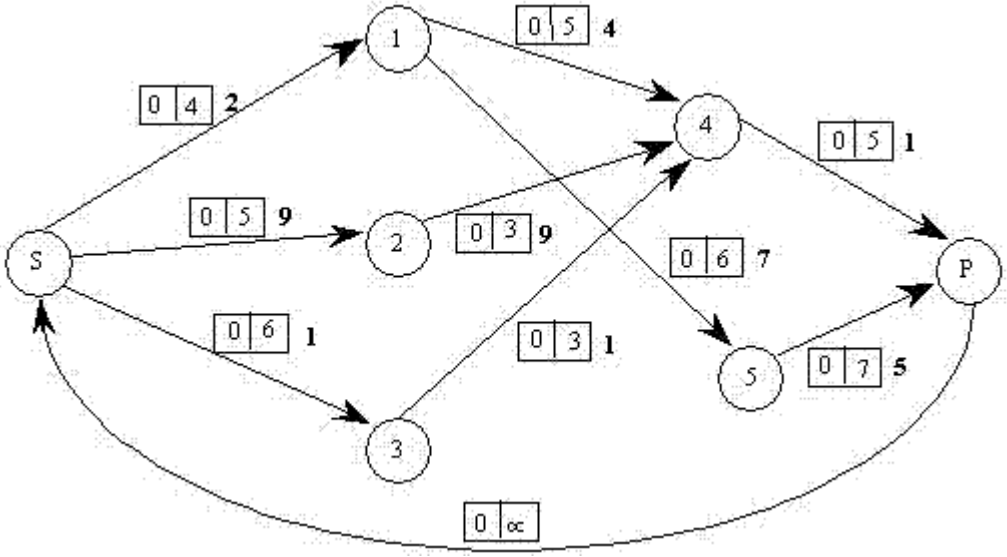


Figure 6.15: un problème avec des coûts

b) Le problème du flot maximal à coût minimal :

Le problème du flot maximal à coût minimal : Nous allons maintenant modifier cet algorithme du graphe d'écart pour pouvoir tenir compte de coûts unitaires de transport sur les arcs. Pour chaque arc (i ,j) , on a en outre , un coût d_{ij} ($\geq$ 0). Il faut déterminer un flot maximal de coût global minimal, c'est-à-dire qui minimise : $\sum_{(i,j)\in U} d_{ij} f_{ij}$

Syntaxe : Algorithme de ROY :

(I) Poser f = 0.

(II) Construire Ge(f) en introduisant un coût d_{ij} sur l'arc (i , j) et un coût - d_{ij} sur l'arc (j , i).

(III) Chercher un chemin de coût minimal dans Ge(f) allant de s à p.S'il n'y a pas de chemin de s à p, le flot est maximal de coût minimal alors FIN.

(IV) Sinon améliorer le flot f et retourner en (II).

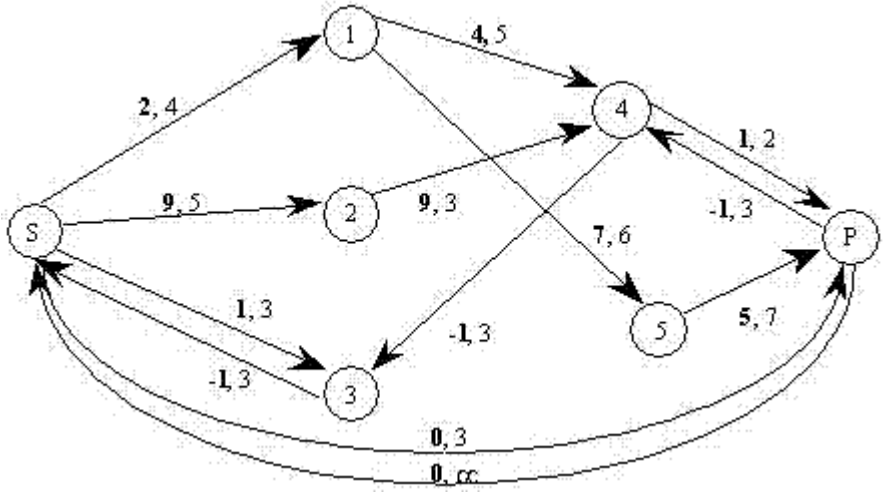


Figure 6.16 : deuxième graphe d'écart.

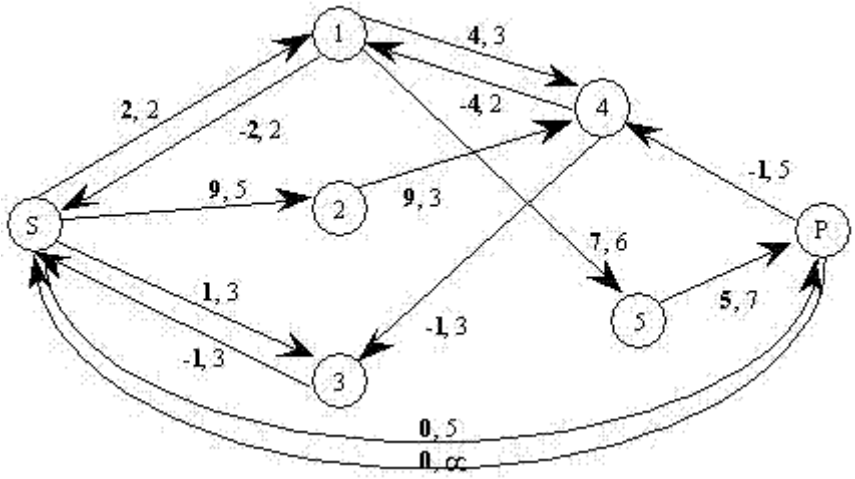


Figure 6.17 : troisième graphe d'écart.

Exemple

Le premier graphe d'écart se confond avec le graphe initial (Cf. figure 6.15). Sur ce graphe, le chemin de coût minimal allant de s à p est [s, 3, 4, p] de coût 3. On fait alors passer un flux de valeur 3 à travers ce chemin et on obtient un deuxième graphe d'écart (Cf. figure 6.16). Sur ce graphe, le chemin de coût minimal allant de s à p est [s, 1, 4, p] de coût 7. On fait alors passer un flux de valeur 2 à travers ce chemin et on obtient un troisième graphe d'écart (Cf. figure 6.17). Sur ce graphe, le chemin de coût minimal allant de s à p est [s, 1, 5, p] de coût 14. On fait alors passer un flux de valeur 2 à travers ce chemin et on obtient un quatrième graphe d'écart (Cf. figure 6.18). Sur ce graphe, le chemin de coût minimal allant de s à p est [s, 2, 4, 1, 5, p] de coût 26. On fait alors passer un flux de valeur 2 à travers ce chemin et on obtient un dernier graphe d'écart (Cf. figure 6.19).

Il n'y a plus de chemin de s à p. Le flot est maximal de coût minimal (Cf. figure 6.20). Le coût de ce flot est : 3*3 + 2*7 + 2*14 + 2*26 = 103.

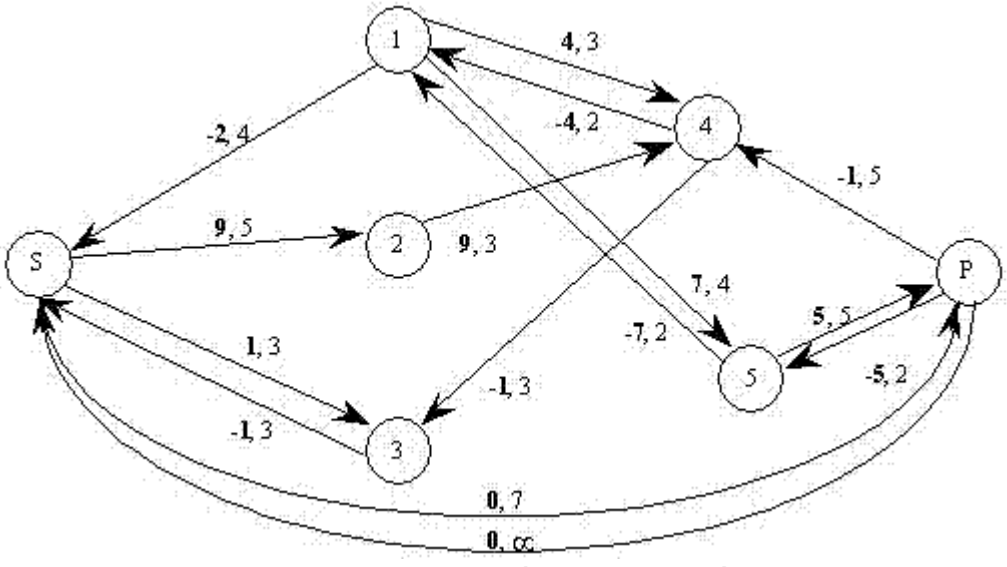


Figure 6.18 : quatrième graphe d'écart.

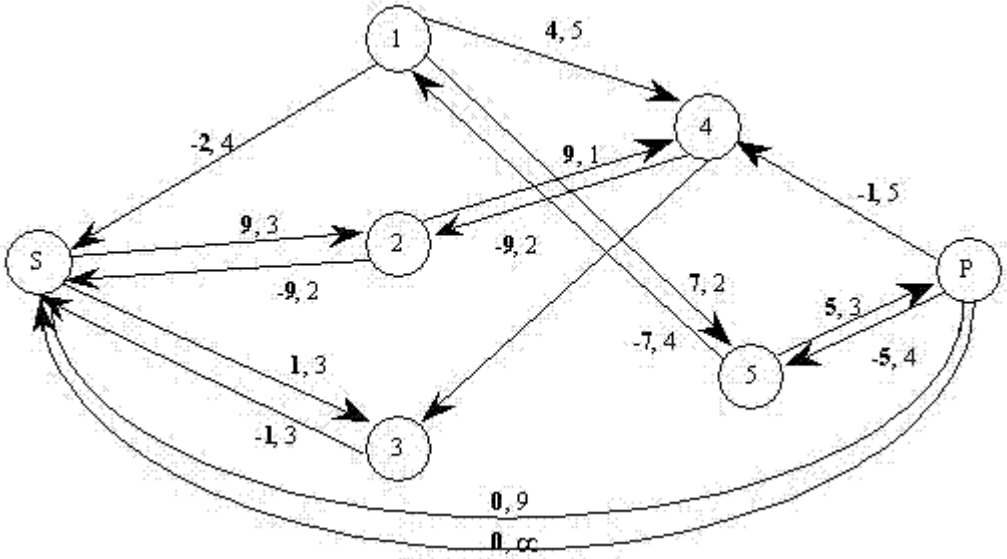


Figure 6.19 : dernier graphe d'écart.

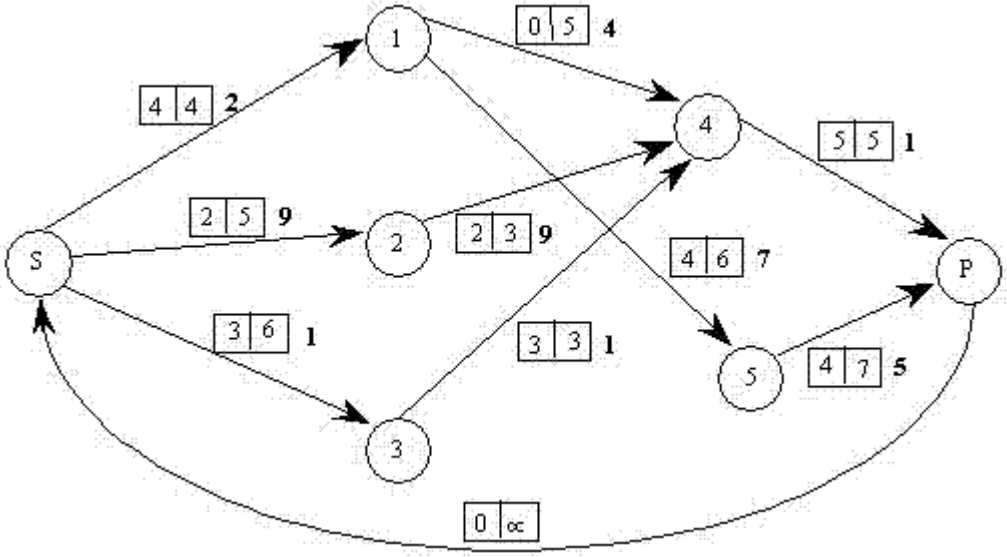


Figure 6.20 : flot maximal de coût minimal..

H. LES PROBLÈMES DE FLOTS CANALISES A COÛT MINIMAL

Dans ce chapitre, nous considérons des généralisations du problème du flot maximal en traitant des problèmes de flots à coût minimal et des problèmes de flots canalisés. Un flot canalisé est un flot qui doit satisfaire outre les contraintes de KIRCHOFF et de capacités, des contraintes supplémentaires dites de bornes : à chaque arc (i, j) est associé une borne b_{ij} et le flux sur l'arc (i, j) doit être supérieur ou égal à b_{ij}. Un flot sera de coût minimal si une fonction linéaire des coûts de transport est minimisée.

L'outil de base reste la chaîne améliorante. Nous le simplifions en introduisant la notion de graphe d'écart qui permet de se ramener à la recherche de chemins améliorants. Nous expliquons comment chercher un flot canalisé et un flot de coût

minimal.

1. Flot canalisé et graphe d'écart :

a) Flot canalisé et graphe d'écart :

Flot canalisé et graphe d'écart : Il arrive qu'une borne inférieure b(u) entière du flux sur l'arc u ne soit pas nulle, auquel cas, le problème de la recherche d'un flot compatible (on dira canalisé) se pose. On rappelle que c(u) la capacité de l'arc u est une borne supérieure entière du flux sur l'arc u.

Définition

Un flot canalisé est une application f de U dans N (ensemble des entiers) satisfaisant les contraintes de Kirchoff et les contraintes de bornes et de capacités.

On cherche un flot f tel que pour tout u, $b(u) \leq f(u) \leq c(u)$. Le théorème d'Hoffman donne une condition d'existence d'un flot canalisé.

Théorème d'HOFFMAN :

Une C.N.S pour qu'il existe un flot canalisé dans le réseau G = (X, U, b, c) est que : pour tout $Y \subseteq X$, la somme des bornes des arcs entrants dans Y est inférieure ou égale à la somme des capacités des arcs sortants de Y, c'est-à-dire :

$$\sum_{u \in U^-(Y)} b(u) \leq \sum_{u \in U^+(Y)} c(u)$$

Démonstration :

La condition est évidemment nécessaire. En effet, le flux entrant dans Y est égal au flux sortant de Y. Le flux entrant est supérieur à la somme des bornes et le flux sortant inférieur à la somme des capacités on a donc l'inégalité dès qu'il existe un flot. La suffisance de la condition est assurée par l'algorithme de recherche d'un flot compatible que nous introduisons et justifions ci-dessous. Q.E.D.

Syntaxe : Algorithme de recherche d'un flot compatible :

{Par la suite fu est le flux sur l'arc u}

(I) Partir du flot f nul.

(II) Chercher un arc u tel que f_u < b(u)

- si un tel arc n'existe pas alors FIN 1;

- sinon poser s = extrémité terminale de u et p = extrémité initiale de u.

(III) Chercher une chaîne de s à p dont les arcs dans le sens + sont insaturés et les arcs dans le sens - ont un flux strictement supérieur à la borne; si ce n'est pas possible alors FIN 2.

(IV) Utiliser cette chaîne pour améliorer le flot en cherchant à satisfaire la contrainte de borne sur l'arc u et retourner en (ii).

Cet algorithme s'arrête, soit après avoir trouvé un flot compatible (FIN1), soit parce que la condition nécessaire du théorème n'est pas vérifiée (FIN2).

Nous allons prouver ce résultat après avoir fait fonctionner l'algorithme sur l'exemple de la figure 7.2. On trouve successivement les quatre chaînes améliorantes suivantes:

- pour s = x3 et p = x1, on a la chaîne (x3, x4, x1) de capacité 2.
- pour s = x3 et p = x1, on a la chaîne (x3, x4, x5, x1) de capacité 2.
- pour s = x2 et p = x1, on a la chaîne (x2, x5, x1) de capacité 3.

- pour s = x3 et p = x5, on a la chaîne (x3, x4, x5) de capacité 1.

On a alors un flot compatible.

Validité de l'algorithme :

L'algorithme s'arrête. On ne peut pas boucler indéfiniment car l'ensemble des valeurs est fini et on est sûr de ne jamais retrouver le même flot entre deux étapes. En effet, $\sum_{u \in U} max\{b_u - f_u, 0\}$ est strictement décroissant au cours de l'algorithme et minoré par 0.

Puisque l'on part du flot nul et que l'on respecte toujours les contraintes de capacités, on est sûr que pour tout u : $f_u \leq c(u)$.

Si FIN1, alors on a obtenu un flot compatible.

Si FIN2, alors l'algorithme de marquage d'une chaîne de s à p est bloqué. Notons A l'ensemble des sommets marqués à la fin de cette phase.

On vérifie que le flot entrant dans A est strictement inférieur à la somme des bornes des arcs entrants dans A et est égal à la somme des capacités des arcs sortants de A (Cf. 7.1).

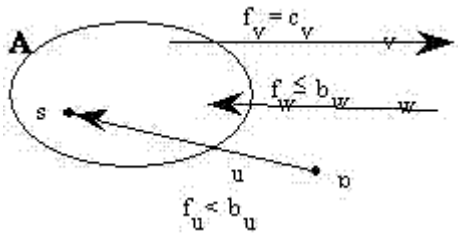


Figure 7.1 : cas de non-existence d'un flot compatible (FIN2)

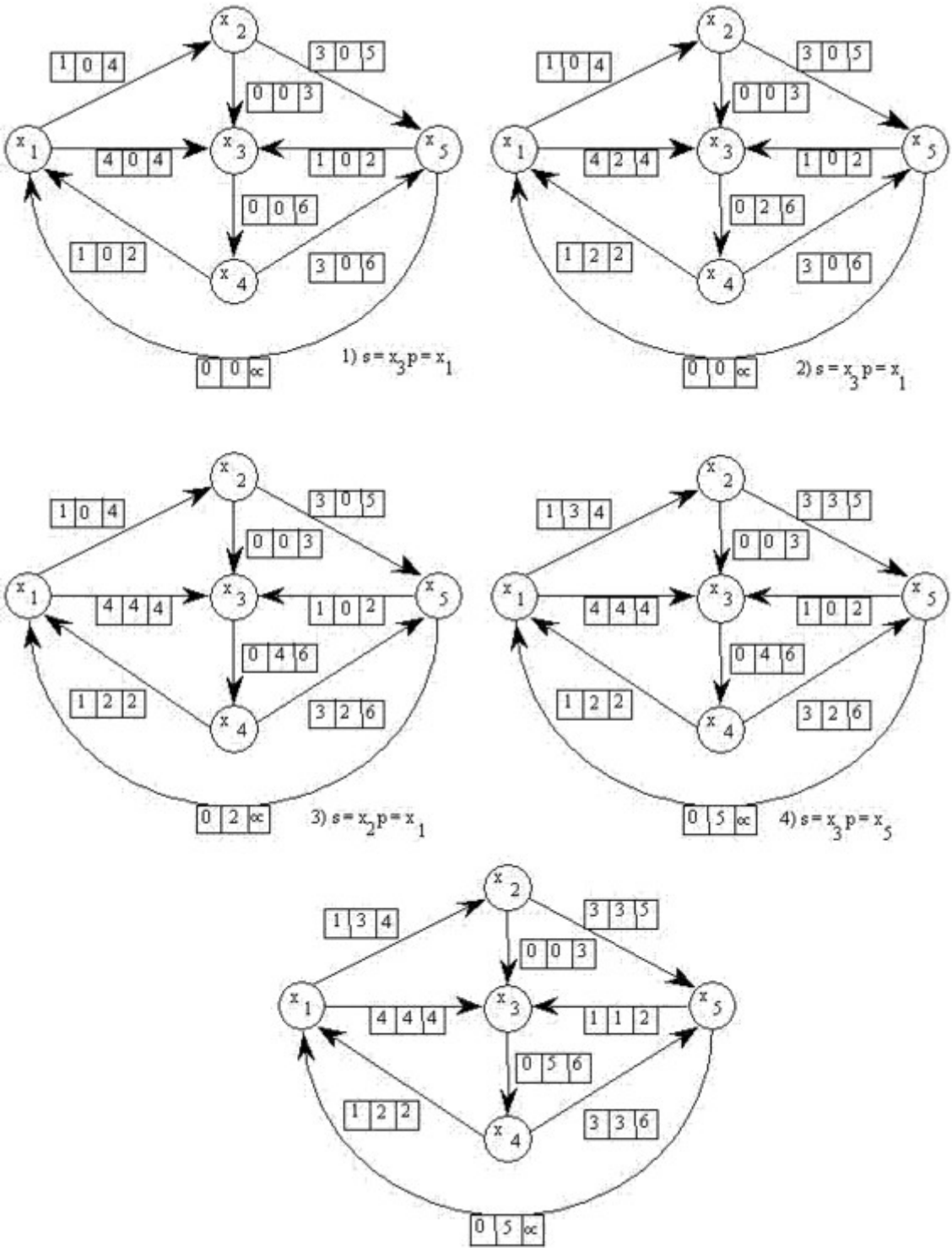


Figure 7.2 : application de l'algorithme.s

2. Flot canalisé à coût minimal :

a) Décomposition d'un flot sur une base de circuits :

Proposition :

Une condition nécessaire et suffisante pour qu'un flot f soit positif est qu'il existe une famille de circuits $\mu_1, \mu_2, \ldots, \mu_p$ et des coefficients $\lambda_1, \lambda_2, \ldots, \lambda_p$ entiers

positifs tels que : $f = \mu_1 \lambda_1 + \mu_2 \lambda_2 + \ldots + \mu_p \lambda_p$.

Démonstration :

La condition est suffisante car chaque circuit peut être considéré comme un flot positif. Montrons que la condition est également nécessaire. En partant d'un arc de flux non nul, on peut trouver également un arc suivant, également de flux non nul, à cause de la condition de Kirchoff. En itérant on va repasser par un même sommet. On aura donc trouvé un circuit μ_1 . Soit λ_1 la valeur minimale des flux sur ce circuit; $f - \lambda_1 \mu_1$ est encore un flot positif et on peut donc lui appliquer le même raisonnement ce qui nous permet d'obtenir successivement toutes les composantes de f, car à chaque fois un nouvel arc de flux nul apparaît. Q.E.D.

b) Bijection d'écart :

Définition : Définition du graphe d'écart Ge(f) (incluant les bij) :

Au couple formé d'un flot f et d'un réseau de transport G, on associe le graphe d'écart Ge(f)=(X,Ue(f)) défini par :

à tout arc (i, j) de G, correspond deux arcs : un arc (i, j) de Ge(f) de capacité c_{ij} - f_{ij} et un arc (j, i) de Ge(f) de capacité f_{ij} - b_{ij} (si la capacité d'un arc est nulle dans Ge(f), on ne représente pas l'arc, car il est inutile).

Soit f_0 un flot compatible et $G^e(f_0)$ le graphe d'écart associé. Soit φ un flot sur $G^e(f_0)$ on définit un nouveau flot f compatible sur G par la formule notée $f = f_0 \oplus \varphi$ telle que :

$(f)_u = (f_0)_u + (\varphi)_{u+} - (\varphi)_{u-}$

Le coût du flot f sur G est égal au coût du flot f_0 sur G plus le coût du flot φ sur le graphe d'écart.

On cherche à minimiser le coût de f où f est un flot compatible. La méthode que nous allons voir consiste à supprimer les circuits de coût strictement négatif dans le graphe d'écart. Elle est fondée sur le théorème d'optimalité que nous démontrons.

c) Théorème d'optimalité :

Théorème d'optimalité :

Un flot f_0 compatible est de coût minimal si et seulement si Ge(f_0) ne contient pas de circuit de coût strictement négatif.

Démonstration

La condition est évidemment nécessaire. En effet, s'il existe un circuit de coût strictement négatif, ce circuit permet de construire un flot strictement meilleur, comme nous l'expliquons avec l'exemple ci-dessous. La condition est également suffisante. Soit un flot f_0 tel que $G^e(f_0)$ ne contient pas de circuit de coût strictement négatif et f un autre flot compatible; on a $f = f_0 \oplus \varphi$ se décompose sur une base de circuits de $G^e(f_0)$: $\varphi = \mu_1 \lambda_1 + \mu_2 \lambda_2 + \ldots + \mu_p \lambda_p$. Alors : Coût(φ) = $\sum_{i=[1,\,p]} \lambda_i$ Coût(μ_i) est positif car les coûts de tous les circuits du graphe d'écart sont positifs et les λ_i également. En utilisant Coût(f) = Coût(f_0) + Coût(φ)on a Coût(f) $\geq$ Coût(f_0) et donc f_0 est un flot compatible de coût minimal. Q.E.D.

Exemple

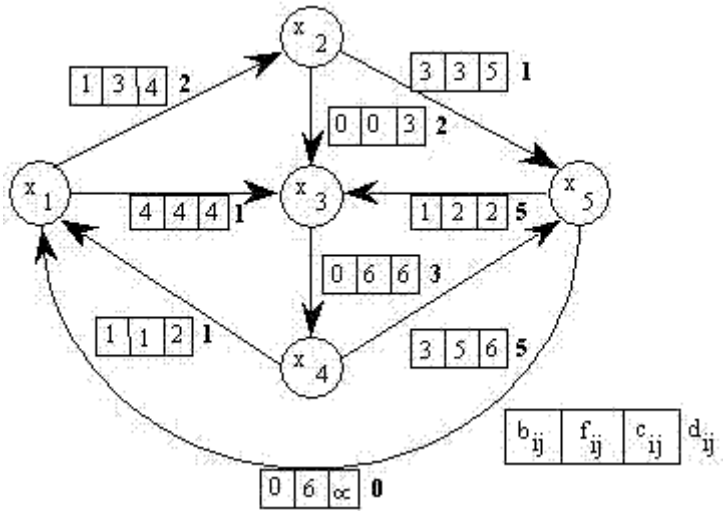
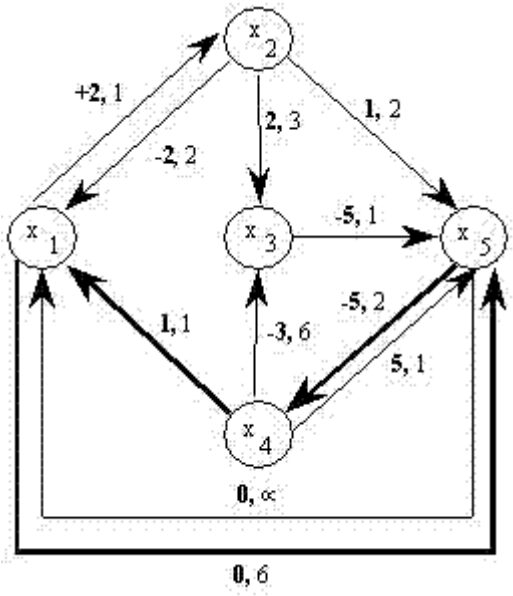
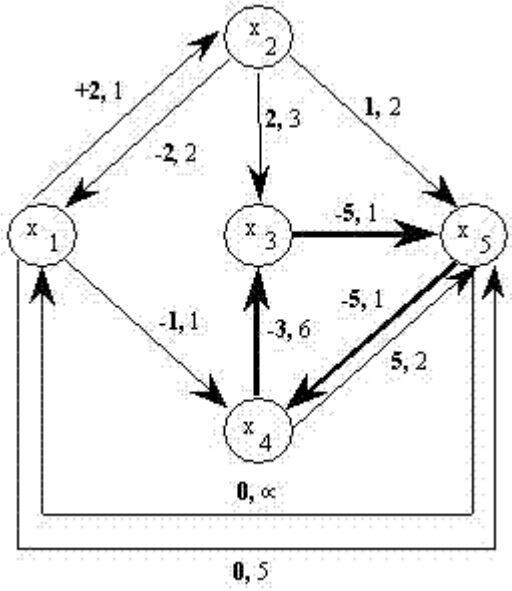


Figure 7.3 : problème de flot canalisé à coût minimal.

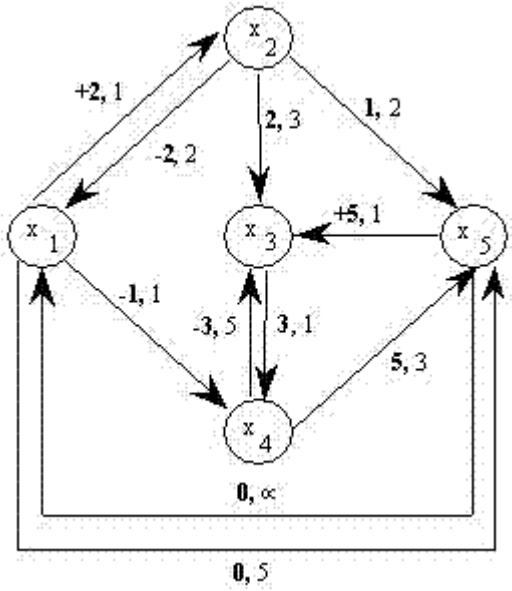
Sur la figure 7.3, on a un exemple de flot compatible. Nous allons utiliser le résultat précédent pour trouver un flot compatible de coût minimal. Sur la figure 7.4, on a le graphe d'écart associé. Ce graphe d'écart contient un circuit de coût strictement négatif : le circuit [x5, x4, x1]. On fait alors passer un flux de 1 à travers ce circuit dans le graphe d'écart. Sur la figure 7.5, on a le graphe d'écart associé. Ce graphe d'écart contient un circuit de coût strictement négatif : le circuit [x3, x5, x4]. On fait alors passer un flux de 1 à travers ce circuit dans le graphe d'écart. On obtient alors un dernier graphe d'écart dont tous les circuits sont de coûts positifs ou nuls (Cf. figure 7.6). En conséquence, le flot compatible résultant, qui est rapporté sur la figure 7.7 est un flot compatible de coût minimal.



7.4 : premier graphe d'écart.s



7.5 : deuxième graphe d'écart.



7.6 : dernier graphe d'écart.

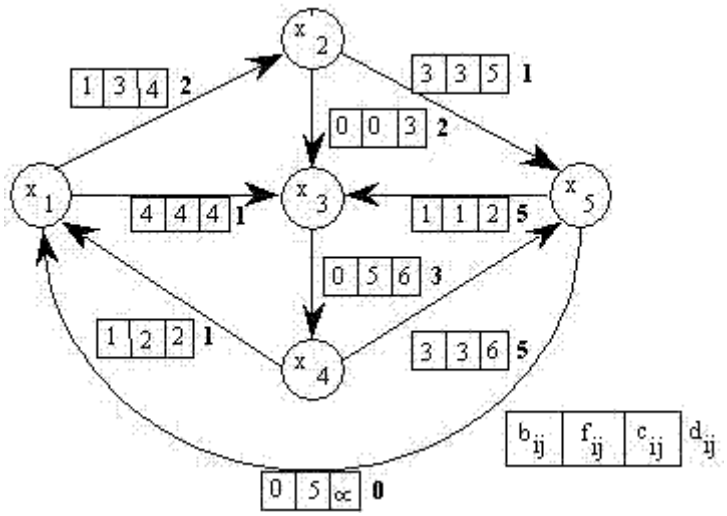


Figure 7.7 : flot canalisé de coût minimal.

d) Preuve de l'algorithme de ROY :

Nous terminons ce chapitre en utilisant le théorème d'optimalité pour démontrer la validité de l'algorithme de ROY.

Proposition

l'algorithme de ROY détermine un flot maximal de coût minimal.

Démonstration :

On montre par récurrence que tous les flots f_k construits au cours de l'algorithme de ROY sont de coût minimal parmi les flots de valeur v_k. Il en résultera que le dernier flot sera maximal à coût minimal.

La propriété est vraie pour $f_0 = 0$ qui est le flot de valeur nulle de coût minimal, supposons que f_k soit de coût minimal parmi les flots de valeur v_k on a $f_{k+1} = f_k \oplus \varepsilon\, \mu$ où μ est un circuit de coût minimal passant par l'arc de retour dans le graphe d'écart $G^e(f_k)$ et ε la valeur maximale admissible du flot. Soit f un flot quelconque de même valeur sur l'arc de retour que f_{k+1}. On sait que $f = f_k \oplus \varphi$ où φ est un flot sur le graphe d'écart $G^e(f_k)$. Appliquons le théorème de décomposition : $\varphi = \mu_1\lambda_1 + \mu_2\lambda_2 + \ldots + \mu_p\lambda_p$. En utilisant le fait que f_{k+1} et f ont mêmes composantes sur l'arc de retour on obtient :

$\sum_{i\in I} \lambda_i = \varepsilon$ où I est l'ensemble des circuits passant par l'arc de retour.

On en déduit : $Coût(f) = \sum_{i\in[1,p]} \lambda_i Coût(\mu_i) + Coût(f_k) \geq \sum_{i\in I} \lambda_i Coût(\mu_i) + Coût(f_k)$ car tous les circuits ne passant pas par l'arc de retour sont de coût positif ou nul du fait que, par hypothèse de récurrence, f_k est de coût minimal parmi les flots de valeur v_k. En utilisant $Coût(\mu_i). \geq Coût(\mu)$ pour $i \in I$ et $= \sum_{i\in I} \lambda_i = \varepsilon$ ..on a : $Coût(f) \geq \varepsilon.\ Coût(\mu) + Coût(f_k) = Coût(f_{k+1}).$ Ce qui démontre la minimalité du coût de f_{k+1} et le résultat par récurrence. D'où la validité de l'algorithme de ROY. Q.E.D.

Bibliographie

[2]

R. PENROSE, *L'esprit, l'ordinateur et les lois de la physique*, INTEREDITIONS, 1992.

[3]

B. ROY, *"Recherche Opérationnelle et Aide à la Décision"*, Discours de remerciements à l'occasion de la remise du diplôme de Docteur Honoris Causa de l'Université de POZNAN, 1992.

[4]

M. GONDRAN; M. MINOUX *"Graphes et Algorithmes"* Eyrolles 1985.

[5]

ROSEAUX. *"Exercices corrigés de Recherche Opérationnelle"* 3 Tomes. MASSON.

[6]

M.R. GAREY, D.S JOHNSON, *"Computers and Intractability"* W. H. FREEMAN, San Francisco 1978

[1]

A. ALJ, R. FAURE, *Guide de la Recherche Opérationnelle*, 2 tomes, MASSON, 1990.