GRAPHES ET ALGORITHMES DE TRAITEMENT

Bruno ROUZEYRE
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

Département ERII Polytech'Montpellier
Master EEA 2eme année

Université de Montpellier II : Sciences et Techniques du Languedoc
Place Eugène Bataillon 34095 Montpellier CEDEX 2

BIBLIOGRAPHIE

”Graphes et Algorithmes” M. Gondran et M. Minoux. Coll. de la direction des études et recherche d’électricité de France. Editions Eyrolles.

”Graph theory with applications to engineering and computer science” N. Deo. Series in Automatic Computation. Editions Prentice Hall.

”Graph theory : an algorithmic approach” N. Christofides. Series in computer science and applied mathematics. Editions Academic Press.

”Graphes et hypergraphes” C. Berge. Editions Dunod.

”Algèbre moderne et théorie des graphes” B. Roy. Editions Dunod.

INTRODUCTION ET TERMINOLOGIE

I DEFINITIONS

1- Notions orientées

Un **graphe** G[X,U] est un ensemble déterminé par :
- X : ensemble des **sommets** (ou noeuds).
N = Card(X) est appelé l'**ordre** du graphe, G est dit graphe d'ordre N.
- U : ensembles des **arcs**, ou un arc est un couple ordonné u=(i,j), i∈X, j∈X.
i est appelé l'origine de l'arc u, j l'extrémité. On note M=Card(U)

Un graphe G[X,U] est appelé **p-graphe** si p est le nombre maximum d'arcs joignant deux sommets quelconques.

Un arc de la forme u=(x,x) est appelé une **boucle**.

Un sommet j est dit **successeur** de i si il existe u=(i,j). i est alors appelé **prédécesseur** de j.
L'ensemble des sommets successeurs de i est noté Γi, Γ application multivoque de X dans X.
L'ensemble des sommets prédécesseurs de i est noté $\Gamma^{-1} i$

2- Notions non orientées

Si l'on ne considère plus l'orientation les arcs sont appelés **arêtes**.

Un **multigraphe** est un graphe pour lequel il peut exister plusieurs arêtes entre paires de sommets.

Un graphe est dit **simple** si :
- il est sans boucle
- il existe une arête au plus entre 2 sommets

3- Principales définitions

- 2 arcs (arêtes) sont **adjacents** s'ils ont un sommet commun.

- le **demi-degré extérieur** d'un sommet i est le nombre d'arcs d'origine i (noté $d^+ i$). Dans un 1-graphe $d^+ i = Card(\Gamma i)$.

- le **demi-degré intérieur** d'un sommet i est le nombre d'arcs d'extrémité i (noté $d^- i$). Dans un 1-graphe : $d^- i = Card(\Gamma^{-1} i)$.

- le **degré** de i noté $di = d^+ i + d^- i$

- G[X,U] est **symétrique** <=> pour tout u=(i,j), u'=(j,i) ∈ U

- G[X,U] est **antisymétrique** <=> pour tout u=(i,j), u'=(j,i) ∉ U.

- **graphe partiel** : soit G=[X,U], soit V ⊂ U, le graphe partiel engendré par V est égal à [X,V].

- **sous-graphe** : soit G=[X,U], soit A ⊂ X, le sous-graphe engendré par A est le graphe noté G_A dont les sommets sont ceux de A et dont les arcs sont ceux de G ayant leur origine et leur extrémité dans A.

- **sous-graphe partiel** : soient G=[X,U], V⊂U, A ⊂X. Le sous-graphe partiel engendré par A et V est le graphe partiel de G_A engendré par V.

- **graphe complémentaire** : G=[X,U'] est défini par :
  si (i,j) ∈ U => (i,j) ∉ U'
  si (i,j) ∉ U => (i,j) ∈ U'

- **graphe complet - clique** :
G [X,U] est dit complet si pour toute paire de sommets (i,j), il existe au moins un arc (i,j) ou (j,i).
Un 1-graphe complet non orienté d'ordre n est noté Kn.
Un sous-graphe complet est appelé une clique (c-à-d un sous-graphe dont tous les sommets sont connectés 2 à 2).

- **graphe biparti** : G [X,U] est biparti si X = X1 ∪ X2 avec :
  X1 ∩ X2 = Ø
  pour tout u=(i,j), i∈X1 => j∈X2 et i∈X2 => j∈X1

- **graphe contracté** :
Soit Y ⊂ X , le graphe contracté de G par rapport à Y est le graphe noté G/Y = [X-Y+{y},V] où : V est défini par :
  (i,j) ∈ V si i ∈ X-Y, j ∈ X-Y
  (i,y) ∈ V si i ∈ X-Y, et s'il existe j ∈ Y/ (i,j) ∈ U.

4- Chemins et connexité

a - <u>non orienté</u> :

Une **chaîne** de longueur q est une séquence de q arcs: L = {u_1, u_2, ..., u_q} telle que :
∀ r/ 1<r<q, u_r a une extrémité commune avec u_{r-1} et l'autre avec u_{r+1}.
Les extrémités de u_1 et u_q non adjacentes à un autre arc sont appelées les extrémités de la chaîne.
Si G est un graphe simple, une chaîne est définie par la donnée des sommets qu'elle rencontre.

- une **chaîne élémentaire** est une chaîne telle qu'on ne rencontre pas deux fois le même sommet en la parcourant.

- un **cycle** est une chaîne dont les extrémités coïncident.

- un **cycle élémentaire** est un cycle minimal pour l'inclusion (i.e. une chaîne élémentaire dont les extrémités coïncident).

- **Connexité** : G [X,U] est dit **connexe**, si pour tout couple i et j il existe une chaîne de i à j.
Soit R la relation binaire :
i R j <=> soit i=j
  soit, il existe une chaîne de i à j.

R est une relation d'équivalence.
Les classes d'équivalence sont appelées les **composantes connexes** de G.

Le **nombre de connexité** de G, noté ν(G), est le nombre de composantes connexes de G.
Si ν(G) = p, G est dit **p-connexe**.

- **Cocycle** :
Soit $A \subset X$, on note :

$\omega^+(A) = \{ u=(i,j)/\ i \in A, j \in X-A \}$ = ensemble des **arcs incidents extérieurement** à A

$\omega^-(A) = \{ u=(i,j)/\ i \in X-A, j \in A \}$ = ensemble des **arcs incidents intérieurement** à A

$\omega(A) = \omega^+(A) \cup \omega^-(A)$ = ensemble des **arcs incidents** à A

Tout ensemble d'arcs de la forme $\omega(A)$ est appelé un **cocycle**.

Un cocycle est élementaire s'il est formé d'arcs reliant 2 sous-graphes connexes GA1 et GA2 de G avec A1 non vide, A2 non vide, $A1 \cap A2 = \emptyset$, et $A1 \cup A2$ est une composante connnexe de G.
Un cocycle élémentaire peut être défini de façon équivalente comme un cocycle minimal pour l'inclusion.

b- <u>orienté</u> :

- Un **chemin** de longueur q est une chaîne de longueur q dont les arcs sont orientés dans le sens de parcours.

- **Chemin élémentaire** $\leftrightarrow$ chaîne élémentaire.

- **Circuit** $\leftrightarrow$ cycle.

- **Circuit élémentaire** $\leftrightarrow$ cycle élémentaire.

- **Fermeture transitive** : on appelle fermeture transitive de l'application Γ, l'application Γ' définie par :

$\Gamma'_i = \Gamma_i \cup \Gamma^2_i \cup \Gamma^3_i \cup .....\cup \Gamma^{N-1}_i$ où Γ^k_i représente l'ensemble des sommets que l'on peut atteindre à partir de i part un chemin de longueur k (N est l'ordre du graphe).

Γ'_i représente l'ensemble des sommets que l'on peut atteindre à partir de i.

Γ'_i = l'ensemble des **descendants** de i.

Γ'^{-1}_i = l'ensemble des **ascendants** de i.

- **Forte connexité** : Un graphe est dit **fortement connexe** si pour tout couple de sommets i,j il existe un chemin de i à j et un chemin de j à i.
Soit R la relation binaire :
i R j <=> soit i=j
soit, il existe un chemin de i à j et un chemin de j à i.
R est une relation d'équivalence.

Les classes d'équivalence sont appelées les **composantes fortement connexes** de G.

<u>Remarque</u> : forte connexité => connexité.

- **Cocircuit** : c'est un cocycle dont tous les arcs sont orientés dans le même sens c-à-d une ensemble d'arcs de la forme $\omega^+(A)$ ou bien $\omega^-(A)$.

II Lemme de Minty - Nombre cyclomatique - Formule d'Euler :

1 Lemme de Minty :

Soit un graphe dont les arcs sont colorés soit en rouge, soit en noir, soit en vert, soit laissés incolores. Si l'on suppose que l'arc $u_0 = (i_0,j_0)$ est coloré en noir, alors l'<u>une et une seule</u> des deux propositions suivantes est vraie :

1- il passe par l'arc u_0 un cycle élémentaire dont tous les arcs sont noirs,rouges ou verts (pas d'incolore) avec tous les noirs orientés dans le même sens (celui de u_0), tous les verts en sens inverses.

2- il existe une partie A de X telle que $j_0 \in A$, $i_0 \notin A$ et telle que le cocycle $\omega(A)$ soit sans arc rouge avec tous les arcs noirs dans le même sens (celui de u_0) et tous les verts dans l'autre sens.

<u>remarque</u> :

- $\omega(A)$ peut contenir des arcs incolores.
- $u_0 \in \omega^-(A)$ => tous les noirs intervenant $\in \omega^-(A)$ et tous les verts $\in \omega^+(A)$.

<u>Démonstration</u> : elle consiste à chercher un ensemble de chemins de j_0 à i_0 vérifiant les conditions du cas 1 par une procédure de marquage.
On part de j_0 marqué (extrémité de l'arc de départ).
Soit i un sommet marqué, on marque j non encore marqué uniquement dans l'un des cas suivants :
- il existe un arc (i,j) de couleur noire,
- il existe un arc (j,i) de couleur verte,
- il existe un arc (i,j) ou (j,i) de couleur rouge.

et on réitère.
A la fin, deux cas possibles :
- on a réussi à marquer le sommet i_0, => proposition 1.
- on ne rencontre jamais i_0. Soit A l'ensemble des sommets marqués ($i_0 \notin A$, $j_0 \in A$).

A ne contient pas d'arc d'arc rouge (sinon les deux extrémités d'un tel arc appartiendrait à A., tous les arcs noirs sont dans $\omega^-(A)$, et tous les verts dans $\omega^+(A)$. Donc A vérifie bien les conditions de la proposition2. c.q.f.d.

Conséquence :

<u>tout arc appartient soit à un circuit soit à un cocircuit mais pas aux deux</u>.

dém : il suffit de colorier tous les arcs en noir et d'appliquer le lemme de Minty.

2. Bases de cycles, de cocycles :

Soit μ un cycle quelconque, soit μ^+ l'ensemble des arcs orientés dans le sens de parcours, μ^- l'ensemble des arcs orientés dans le sens inverse.
A μ on peut faire correspondre un vecteur à M composantes $\mu=(\mu_1,\mu_2,.....,\mu_M)$ avec M = Card(U)
avec $\mu_u = 0$ si $u \notin \mu^+ \cup \mu^-$

1 si $u \in \mu^+$

-1 si $u \in \mu^-$

Propriété : Tout cycle est une somme (au sens vectoriel) de cycles élémentaires sans arcs communs.
Dém : Lorsque l'on parcourt un cycle quelconque, on définit un nouveau cycle chaque fois que l'on arrive à un sommet déjà rencontré.

Définitions :
- dépendance linéaire de cycles
- indépendance linéaire
- **base de cycles** = ensemble minimal de cycles indépendants tel que tout cycle puisse se mettre sous forme de combinaison linéaire des cycles de la base.
- **nombre cyclomatique** : $\nu(G)$ = dimension de la base de cycle.

De façon identique, on définit des b**ases de cocycles**, => $\lambda(G)$ = **nombre cocyclomatique**.
rappel : On dit qu'un cocycle est élémentaire si et seulement si il est formé d'un ensemble d'arcs joignant deux sous-graphes connexes non vides A1 et A2 / $A1 \cap A2 = \emptyset$, $A1 \cup A2$ est une composante connexe de G.

Propriétés :
-Tout cocycle est une somme (au sens vectoriel) de cocycles élémentaires disjoints (sans arcs communs).
-Un cocycle est élémentaire si et seulement si c'est un cocycle minimal (i.e. si et seulement si on ne peut pas en déduire un autre cocycle par suppression d'arcs)

Théorème :
Soit G un graphe avec n sommets, m arcs et p composantes connexes.
Alors : $\nu(G) = m - n + p$
$\lambda(G) = n - p$

Application : Formule d'Euler :

Dans un graphe planaire, on a :

$n - m + f = 2$ où f est le nombre de faces.

Exercice : chercher tous les types de polyèdres convexes réguliers (il y en a 5).

III Matrices associées à un graphe :

1 - Matrice d'incidence sommets-arcs :

La **matrice d'incidence sommets-arcs** A d'un graphe G[X,U] est une matrice à N lignes et M colonnes telle que si u(i,j) est un arc de U la colonne u vaut 0 partout sauf en
$a_{iu} = +1$
$a_{ju} = -1$.

On a : $\forall i \in X$, $\omega^+(i) = \{ u / a_{iu} = +1 \}$

$\omega^-(i) = \{ u / a_{iu} = -1 \}$

Propriété : la matrice d'incidence sommets-arcs est totalement unimodulaire (i.e. toute sous-matrice carrée extraite de A est à déterminant +1 ,-1 ou 0.

2- Matrice d'incidence sommet-arêtes :

La **matrice d'incidence sommets-arêtes** A d'un graphe G[X,U] est une matrice à N lignes et M colonnes telle que si u(i,j) est un arc de U la colonne u vaut 0 partout sauf en

$a_{iu} = +1$

$a_{ju} = +1$.

Propriété : la matrice d'incidence sommets-arêtes est totalement unimodulaire ssi G[X,U] est un graphe biparti.

3- Matrice d'adjacence :

La **matrice d'adjacence** d'un graphe orienté G [X,U] est une matrice carrée d'ordre N telle que :

$a_{ij} = +1 <=> (i,j) \in U$

La matrice d'adjacence d'un graphe non orienté est une matrice symétrique.

IV Représentations des graphes en mémoire d'un calculateur :

On peut toujours représenter un graphe soit à l'aide de sa matrice d'adjacence soit de sa matrice d'incidence, mais perte de place inutile puisque beaucoup de termes nuls.

1 - A partir de la matrice d'adjacence :

La matrice d'adjacence nécessite N^2 mots mémoire en orienté, 1/2 N (N+1) en non orienté.

Si le graphe est peu dense $M << N^2$ ou $M << 1/2\ N\ (N+1)$, on utilise deux "tableaux" :

LP [1..N+1] et LS [1..M+1] (orienté) ou LS[1..2M+1] (non orienté)

où LP[i] donne l'adresse dans LS de la liste des successeurs de i.

Cette liste est comprise dans LS entre les adresses LP [i] et LP [i+1] exclus.

Sur cette structure, on a :

$d^+(i)$= LP[i+1] - LP [i] (orienté)

d(i) = LP[i+1] - LP [i] (non orienté).

$LP[i] = \Sigma\ d^+(j) + 1$, pour j=1,....,i-1.

2- A partir de la matrice d'incidence sommets-arcs :

a - Par la liste des arcs :

Avec un tableau des origines OR [1..M] et un tableau des extrémités EX [1..M]

OR[u] donne l'origine de l'arc u, EX [u] donne l'extrémité de l'arc u.

Cette structure permet de décrire des p-graphes.

b - Par le liste des cocycles $\omega^+(i)$ ou $\omega^-(i)$:

On décrit la liste des arcs d'origine un sommet i (et/ou d'extrémité i) à l'aide de deux tableaux :
LP [1..N+1] et LA [1..M+1] (et LS[1..M+1])
Où LP [i] donne l'adresse de la liste des arcs d'origine i dans le tableau LA .

$\omega^+(i)$ = l'ensemble des éléments de LA compris entre LP[i] et LP[i+1] exclus.

<u>Remarque</u> : le choix de la structure de données dépend du problème à traiter.

RECHERCHE EN PROFONDEUR D'ABORD DANS UN GRAPHE (DEPTH FIRST SEARCH - DFS)

I INTRODUCTION

Il s'agit d'une méthode de parcours des sommets du graphe. Elle permet de résoudre les problèmes de connexité sur un graphe par des algorithmes efficaces (i.e. linéaires). Par exemple :

- un sommet j est-il accessible à partir d'un sommet i donné ?

- le graphe est-il connexe ?

- le graphe est-il fortement connexe ?

On suppose que G est un 1-graphe (sans perte de généralité).

II PRINCIPE

Il s'agit d'explorer tous les sommets d'un graphe, une fois et une seule, atteignables à partir d'un sommet i0 donné et en cheminant le long des arcs en accord avec leur orientation. On avance tant qu'on peut avancer. A chaque "embranchement" possible, on prend une direction au hasard. Lorsqu'on ne peut plus avanceer ou que l'on retombe sur un sommet déjà rencontré, on repart du dernier embranchement en prenant une nouvelle direction.

Au cours de l'exploration, tout sommet i se trouve dans l'un des 2 états suivants :
S1 : i n'a pas encore été rencontré
S2 : i a déjà été rencontré. On note **P(i)** le sommet à partir duquel on a rencontré i.
Soient les 2 sous-états :

E1 : tous les successeurs de i ont déjà été examinés.

E2 : certains successeurs j de i (j appartient à Γi) n'ont pas été examinés.

- Conditions initiales :
P(i0) = i0;
P(i) = 0, pour tout i $\neq$ i0 , i.e. i non encore atteint.
i dans l'état S1 <=> P(i) = 0

Soit **n(i),** un compteur associé à chaque sommet, donnant le nombre de successeurs de i non encore examinés.

initialement n(i) = d^+i = Card (Γi)
i dans l'état E1 <=> n(i)=0

- A l'étape courante :
On considère le sommet i atteint(au départ i=i0)

Cas 1 : n(i) > 0

il existe j $\in \Gamma$i non encore examiné.
- sélectionner j dans Γi non encore examiné
- et faire n(i) <- n(i)-1

Cas 1a : c'est la première fois que l'on rencontre le sommet j i.e. P(j)=0. On marque alors j comme atteint à partir de i i.e. P(j) <- i et on recommence une nouvelle étape en prenant j comme sommet courant i.e. j <- i

Cas 1b : j a déjà été rencontré (P(j) ≠ 0) à partir d'un autre sommet au cours d'une étape précédente. On ne met pas en évidence de nouvelles possibilités de cheminement. On recommence une nouvelle étape avec le même sommet i.

Cas 2 : n(i) =0 i.e. tous les successeurs de i ont été examinés.
On ne peut trouver d'autres possibilités de cheminement qu'en repartant du dernier ancêtre de i pour lequel tous les successeurs n'ont pas été examinés. On fait alors un retour arrière ("backtrack") en faisant un déplacement vers le sommet k tel que k=P(i). On recommence une nouvelle étape en prenant k comme sommet courant.

L'algorithme se termine lorsque on revient à i0 et que tous les successeurs de i0 ont été examinés (i.e. lorsque on a parcouru tous les chemins issus de i0). La condition d'arrêt est : i=i0 et n(i0)=0.

Remarque : lorsque l'algorithme se termine le tableau P décrit une arborescence maximale d'origine i0.

Dans l'algorithme formel, on note **num(i)** le numéro d'ordre avec lequel on rencontre le sommet i.

Enoncé formel :

```
Initialisations :
a- Pour i = 1,2,....,N :
        P(i) <- 0
        n(i) <- d+(i)
        num(i) <- 0
soit i0 le sommet de départ :
P(i0) <- i0
k <- 1
num(i0) <- k
i <- i0

b- Tant que (i ≠ i0 ou n(i) ≠ 0) faire :
        Si n(i) ≠ 0 :                       {CAS 1}
                - sélectionner le n(i)ème successeur j de i dans la liste des successeurs de i.
                - n(i) <- n(i)-1
                - Si P(j) = 0 alors faire :             {CAS 1.a}
                        P(j) <- i
                        k    <- k+1
                        num(j) <- k
                        i <- j
                - Sinon : rien                  {CAS 1.b}
        Sinon (n(i)=0). "backtrack"             {CAS 2}
                i <- P(i)
```

Complexité :

Chaque arc est parcouru au plus une fois et on n'effectue jamais plus d'un retour arrière pour chaque sommet => le nombre d'opérations élémentaires est en O(N)+O(M).
Dans un graphe connexe on a toujours M ≥ N-1. Donc O(M).

III APPLICATIONS :

III.1 : TEST DE CONNEXITE - DETERMINATION DES COMPOSANTES CONNEXES

Connexité notion non orientée => on remplace chaque arête (i,j) par deux arcs (i,j) et (j,i). Soit G'[X,U'] le graphe obtenu.

Test de connexité :

Si G non connexe, il n'existe pas de chaîne entre 2 sommets i0 et j0 <=> il n'existe pas de chemin entre i0 et j0 dans G'.
Donc, si en appliquant l'algorithme D.F.S. il existe un sommet j / P(j)=0, G n'est pas connexe.

Détermination des composantes connexes :

Dans l'algorithme D.F.S., un sommet j est atteint dan G' <=> il appartient à la même composante connexe que i0 dans G.

=> on applique la procédure D.F.S. en marquant i0 et tous les sommets atteints à 1.

On réitère à partir d'un autre sommet non atteint en prenant une marque égale à 2.

etc....

En pratique il suffit d'utiliser un tableau **ncomp()**, ncomp(i) donne le numéro de composante connexe du sommet i.
Si nc est le numéro de composante connexe en cours de construction, le cas 1a devient :

```
Si (P(j)=0) alors  P(j)<-i
                   ncomp(j)<-nc
                   i<-j
```

Enoncé formel de détermination des composantes connexes :

a- Initialisations :

pour i=1,2,...,N faire
ncomp(i) <-0
nc <-0

b- Etape courante :
Sélectionner i0 tel que ncomp(i0)=0
Faire : nc <- nc+1
ncomp(i0) <- nc

Détermination de la composante connexe de numéro nc :
Appliquer l'algorithme D.F.S à partir de i0

c- test d'arrêt :
Si pour tout i:1,....,N , ncomp(i)>0 FIN
Sinon retour en b.

En sortie nc donne le nombre de composantes connexes.

Complexité :

Test de connexité : O(M).
Détermination de toutes les composantes connexes : O(M)

III.2 : COMPOSANTES FORTEMENT CONNEXES :

Rappel : i et j appartiennent à la même composante fortement connexe <=> il existe un chemin de i à j et un chemin de j à i.

a- Amélioration de D.F.S. :

- On garde dans num(i) l'ordre de traitement du sommet i.

- Soit A(i) = ensemble des ancêtres de i dans l'arborescence D.F.S.. L'ensemble A(i) est déterminé par les P(i), P(P(i)), P(P(P(i))), etc...

Soit D(i) = l'ensemble des descendants de i dans l'arborescence D.F.S., i.e. l'ensemble des sommets non atteints avant i et qui peuvent être atteints par un chemin dans G[X,U] à partir de i.

Soit inf(i) = numéro d'ordre du sommet ayant le plus petit numéro d'ordre dans l'ensemble A(i) que l'on peut atteindre avec un seul arc de U à partir d'un des sommets de D(i).

$\inf(i) = \text{MIN}[\ num(i)\ ,\ m\ ]$ (1)

où $m = \underset{j \in D(i)}{\text{MIN}} \quad \underset{k \in \Gamma j \cap A(i)}{\text{MIN}} [num(k)]$ (2)

remarque : pour tout i , $num(i) \geq \inf(i)$

Soit **ninf()** un tableau / ninf(i) donne le j de (2), initialement ninf(i)=i.

Calcul de inf(i) :

inf(i) peut être calculé directement dans DFS :
Si chaque fois que l'on atteind un sommet i, on initialise inf(i)<- num(i) et ninf(i)<-i, on ne doit modifier inf(i) pour un sommet i que lorsque on met en évidence un sommet k successeur direct d'un descendant de i ($k \in \Gamma j$, $j \in D(i)$) tel que $k \in A(i)$ et num(k)<inf(i).

Test d'appartenance à A(i) : l'ensemble des sommets atteints peut être divisé en deux :
- ceux pour lesquels il y a eu retour arrière.
- ceux pour lesquels il n'y a pas eu retour arrière donc étant sur le chemin menant à i (sur l'arborescence) = A(i).

Dans l'algorithme, on marque ceux pour lesquels il y a eu retour arrière en faisant : P(j)<- -P(j). Donc $j \in A(i)$ ssi P(j)>0 et num(j)<num(i).

Algorithme de calcul de inf(i) :

Cas 1a : j est un nouveau successeur de i dans l'arborescence, l'arc (i,j) est rajouté à l'arborescence
=> . inf(i) ne doit pas être modifié
. inf(j)<-num(j) et ninf(j)<-j

Cas 1b : l'arc (i,j) ne fait pas partie de l'arborescence DFS : il joint i qui fait partie de D(i) au sommet j qui a déjà été atteint.

D'après (2), si k=ninf(j) appartient à A(i), donc si P(ninf(j))>0, il faut remplacer inf(i) par inf(j) si inf(j)<inf(i), i.e. :

inf(i) <- MIN (inf(i) , inf(j)) (3)

ninf(i) <- ninf(j) si (3) est atteint pour inf(j).

Cas 2 : tous les successeurs de i ont été examinés :

Si i est un sommet terminal de l'arborescence :
Comme (3) a été effectué pour chacun de ses successeurs directs, inf(i) a exactement la valeur des formules (1) et (2). Cette valeur doit être répercutée au prédécesseur de i dans l'arborescence c-à-d pour j=P(i).

Donc pour j=P(i), faire :

inf (j) = MIN (inf (i), inf(j)) (4)

ninf(j)<-ninf(i) si (4) est atteint pour inf(i).

Sinon, inf(i) et ninf(i) ont été mis à jour par le cas précédent, il reste à faire de même pour son prédécesseur.

D'où l'algorithme DFS2 :

Enoncé formel de l'algorithme DFS2 :

a- initialisation
Pour i=1,2,....,N, faire :
    P(i) <- 0
    n(i) <- d+(i)
    num(i) <- 0
    inf(i) <- 0
    ninf(i) <- 0

Soit i0 le sommet de départ, faire :
P(i0) <- i0
k <- 1
num(i0) <- k
inf(i0) <- k
ninf(i0) <- i0
i <- i0

b- Tant que (i ≠ i0 ou n(i) ≠ 0) faire :
- Si n(i) ≠ 0 :
    - sélectionner le n(i)ème successeur j de i dans la liste des successeurs de i.
    - n(i) <- n(i)-1
    - Si P(j) = 0 alors faire : (cas 1a)
        P(j) <- i
        k <- k+1
        num(j) <- k
        inf(j) <- num(j)
        ninj(j) <- j
        i <- j
    - Sinon (P(j) ≠ 0) et si (P(ninf(j)) > 0), faire : (cas 1b)
        inf(i) = MIN (inf(i),inf(j))
        si le minimum est atteint pour inf(j), faire:
            ninf(i) <- ninf(j)
- Sinon (n(i)=0) , faire : (cas 2)
    j <- P(i)
    P(i) <- -P(i)
    inf (j) <- MIN (inf(i),inf(j))
    si le minimum est atteint pour inf(i), faire:
        ninf(j) <- ninf(i)
        i <- j

b- <u>Utilisation de DFS2 pour le calcul des composantes fortement connexes</u> :

<u>Lemme</u> : Si (i,j) ∈ U et si j ∉ D(i) alors num(j)<num(i).

<u>Théorème</u> :
Un sommet i0 est le premier sommet de sa composante fortement connexe que l'on rencontre dans DFS si et seulement si inf(i0) = num(i0).

Démonstration :

Soit un graphe comportant p composantes fortement connexes C1,C2,..Cp.
Soit Cq dont l'un des sommets est atteint dans DFS2 (=> tous les sommets sont atteints).

condition nécessaire :

Soit i0 le premier sommet de Cq atteint, on a : $\forall$ j $\in$ Cq, num(j) > num(i0) (cf. lemme).
Soit un circuit passant par i0 (il existe puisque Cq composante fortement connexe).
De façon générale il est constitué de :
- un chemin entre i0 et i1 $\in$ D(i0) dont tous les sommets intermédiaires appartiennent à D(i0) (ce n'est pas forcément un chemin de l'arborescence),
- d'un arc entre i1 $\in$ D(i0) et i2 $\notin$ D(i0),
- d'un chemin de i2 à i0.

On a :
num(i2) > num(i0) puisque i0 est le premier rencontré.
num(i2) < num(i1) d'après le lemme
=> num(i0) < num(i2) < num(i1)

Mais comme i1 $\in$ D(i0), tous les sommets j / num(i0) < num(j) < num(i1) sont dans l'arborescence issue de i0 => contradiction sur i2.

Donc i2 $\in$ D(i0)

=> tout les sommets d'un circuit passant par i0 appartiennent à D(i0).

Ceci étant vrai pour tout circuit, il n'existe pas dans D(i0) de sommet i1 relié à j $\in$ A(i0) avec num(j) < num(i0)

Donc i0 est tel que inf(i0) = num(i0).

condition suffisante :

Soit i0 / inf(i0) = num(i0).

Par l'absurde :

Soit j $\in$ Cq / num(j) < num(i0).

Comme il existe un chemin entre j et i0, j $\in$ A(i0).
D'autre part, par i0 et j il passe au moins un circuit => il existe au moins un arc allant d'un sommet de D(i0) à j.

On a alors inf(i0) $\leq$ num(j) < num(i0). D'où contradiction. c.q.f.d.

Application : détermination des composantes fortement connexes

Pour trouver toutes les c.f.c., on utilise une pile :
Lorsque on explore un nouveau sommet, on empile son numéro d'ordre.
Lors d'un retour arrière et lorsque num(i)=inf(i) les éléments de la composantes fortement connexe sont dans la pile entre i et le haut de la pile. Il suffit donc de dépiler les (haut-de-pile)-i éléments.

Enoncé formel :

a- initialisation
Pour i=1,2,....,N, faire :
  P(i) <- 0
  n(i) <- d+(i)
  num(i) <- 0
  inf(i) <- 0
  ninf(i) <- 0
Soit i0 le sommet de départ, faire :
  P(i0) <- i0
  k <- 1
  num(i0) <- k
  inf(i0) <- k
  ninf(i0) <- i0
  i <- i0
  np <- 1, pile(np) <- i0, nc <- 1 (nc : numéro de c.f.c.)
b- Tant que (i p i0 ou n(i) p 0) faire :
  - Si n(i) p 0 :
    - sélectionner le n(i)ème successeur j de i dans la liste des successeurs de i.
    - n(i) <- n(i)-1
    - Si P(j) = 0 (cas 1a) alors faire :
      P(j) <- i
      k <- k+1
      num(j) <- k
      np <- np+1, pile(np) <- j
      inf(j) <- num(j)
      ninj(j) <- j
      i <- j
    - Sinon (P(j)p0) (cas 1b) et si (P(ninf(j)>0), faire :
      inf(i) = MIN (inf(i),inf(j))
      si le minimum est atteint pour inf(j), faire:
        ninf(i) <- ninf(j)
  - Sinon (n(i)=0) (cas 2), faire :
    si (inf(i)=num(i)) alors faire : {on marque le num de cfc}
      tant que (pile(np)pi), faire : {on dépile}
        j <- pile(np)
        ncomp(j) <- nc
        np <- np-1
      ncomp(i) <- nc
      np <- np-1
      nc <- nc+1
    faire :
    j <- P(i)
    P(i) <- -P(i)
    inf (j) <- MIN (inf(i),inf(j))
    si le minimum est atteint pour inf(i), faire:
      ninf(j) <- ninf(i)
    i <- j

PROBLEMES DE CHEMIN DANS LES GRAPHES

I EXISTENCE DE CHEMINS, FERMETURE TRANSITIVE :

Définition : On appelle fermeture transitive d'un graphe G(X,U) le graphe G'(X,U') où U'= {(i,j) si il existe un chemin de i à j }.

Calcul de G' :

I.1 Méthode matricielle :

Elever la matrice d'adjacence à la puissance N-1 dans l'algèbre ((0,1),max,min) <=> algèbre ((0,1),ou,et) :

$M^2 = (x_{ij})$ où $x_{ij} = \Sigma\, a_{ik} \cdot a_{kj}$.

$x_{ij} = 1$ <=> il existe k0 / $a_{ik0} \cdot a_{k0j} = 1$
<=> il existe un arc (i,k0) et un arc (k0,j)
<=> il existe un chemin de i à j de longueur 2

$M_l = M_{l-1} \cdot M$ où $x_{ij} = y_{ik} \cdot a_{kj}$.

$x_{ij} = 1$ <=> il existe k0 / $y_{ik0} \cdot a_{k0j} = 1$
<=> il existe un chemin (i,k0) de longueur $\leq$ l-1 et un arc (k0,j)
<=> il existe un chemin de i à j de longueur $\leq$ l.

Complexité : $O(N^4)$

I.2- Deuxième méthode :

1- Déterminer les composantes fortement connexes.
2- Déterminer la fermeture transitive du graphe réduit G/R.
3- En déduire la fermeture transitive du graphe.

point 1 : cf. chap. précédent . complexité en O(M).

point 3 : algorithme de reconstruction

a- pour chaque c.f.c. Xi de cardinal Ni, établir la liste des Ni(Ni-1) arcs du sous-graphe complet sur Xi.

b- si Xk et Xl sont deux sommets tels que arc (k,l) $\in$ G'/R, alors établir la liste des Nk*Nl arcs de la forme (i,j) avec i $\in$ Xk, j $\in$ Xl.

Complexité : O(M').

le problème se réduit au point 2 => fermeture transitive d'un graphe sans circuit.

I.3- Fermeture transitive d'un graphe sans circuit :

Soit G(X,U) dont les sommets sont numérotés suivant un ordre topologique inverse i.e. (i,j) $\in$ U => j < i (cf. § III).

Algorithme :

a- Pour tout i faire : Li <- Γi.
b- Les sommets étant supposés numérotés suivant l'ordre topologique inverse :
Pour tout sommet k:1,...,N faire :

Lk <- Lk U Lj, pour tout j ∈ Γk

à la fin Li représente l'ensemble des sommets pouvant être atteints à partir de i.
Complexité : si les listes sont représentés par des vecteurs (i.e. Vi(j)=1 <=> le sommet l appartient à Li), l'examen du sommet k nécessite O (N Card Γk) opérations et la complexité de l'algorithme est O(MN) (puisque $\sum_{k \in X}$ Card Γk = M).

II PROBLEMES DE PLUS COURTS CHEMINS :

A chaque arc, on associe une longueur lu ∈ R, si u=(i,j) notée aussi $l_{i,j}$. Il s'agit de trouver un chemin C entre 2 sommets i et j tel que l(C) = $\sum_{u \in C}$ lu soit minimale.

Conditions d'existence :
Si C n'est pas élémentaire => C(i,j) = C'(i,j) U Cir où C'(i,j) est un chemin élémentaire de i à j et Cir est un circuit.
. si l(Cir) < 0, il n'existe pas de plus court chemin => pas de solution
. si l(Cir) ≥ 0 alors l(C') ≤ l(C), on se restreint au chemin élémentaire.

Plusieurs algorithmes suivant que :
- lu ≥ 0 , pour tout u
- lu = 1, pour tout u
- G et lu quelconques
- G sans circuit

et que l'on s'intéresse au plus court chemin entre :
- 1 sommet donné et un autre sommet donné
- 1 sommet donné et tous les autres sommets
- tous les couples de sommets

II.1 : Plus courts chemins d'origine fixée i0 :

II.1.1 Longueurs quelconques : algorithme de Ford-Bellman

On utilise le principe d'optimalité de Bellman :
"toute sous-politique d'une politique optimale est une politique optimale".

En termes de longueurs de chemins :
si L*(j) représente la longueur du plus court chemin entre i0 et j, on a :

$$\mathbf{L^*(j) = \min_{i \in Pred(j)} [\, L^*(i) + l(i,j) \,]}$$

Principe de l'algorithme : procédure de marquage.

A chaque sommet i on associe une marque M(i). Les M(i) sont modifiés itérativement de telle sorte qu'à la fin M(i) représente la longueur du plus court chemin de i0 à i.

initialement : M(i0)=0 et M(i)=+∞ pour i≠i0.

D'après le principe d'optimalité :
Les M(i) représentent les longueurs des plus courts chemins de i0 à i <=> pour tout (i,j) de U : $M(j) \leq M(i) + lij$

Le principe de l'algorithme consiste à balayer tous les arcs et à vérifier la condition pour chaque arc :

- lorsque la condition est satisfaite pour chaque arc, le pb est résolu.
- lorsque on rencontre un arc (i,j) / M(j) > M(i)+lij, on met à jour la valeur de M(j) par : M(j) = M(i)+lij.

Au passage, on note dans un tableau P que le sommet j a pour prédécesseur i sur le + court chemin en faisant P(j) <- i.

Théorème : Si le graphe n'admet pas de circuit de longueur négative, les valeurs finales des M(i) sont obtenues en au plus N-1 itérations, où une itération est un balayage complet de la liste des arcs.

Dem :
M1(i) représente la longueur du plus court chemin de i0 à i ne contenant au plus que 1 arc.
M2(i) représente la longueur du plus court chemin de i0 à i ne contenant au plus que 2 arcs.
Mk(i) représente la longueur du plus court chemin de i0 à i ne contenant au plus que k arcs.
S'il n'existe pas de circuit de longueur négative, il existe un plus court chemin entre i0 et tous les autres sommets de moins de N-1 arcs.

Corollaire : si au bout de N itérations, les valeurs M(i) continuent d'être modifiées c'est qu'il existe un circuit de longueur négative et donc pas de solution.

Complexité : N itérations, à chaque itération on balaye les M arcs => O(MN).

Enoncé formel :

a- Initialisations :
M(i0) <- 0
M(i) <- +∞ pour tout i≠i0
P(i) <- i (* P tableau des prédécesseurs sur le plus court chemin *)
K <- 1 (* compteur d'itérations *)
Modif <- Vrai

b- Tant que (Modif = vrai) et ($k \leq N$) faire
Modif <- faux
Pour tout i ∈ X
Pour tout j ∈ Γi, calculer z=M(i)+lij
si (z < M(j)) alors
M(j) <- z
P(j) <- i
Modif <- vrai
k <- k+1

c - Si k=N+1 et Modif=vrai (i.e. il y a eu une modification à la Nième itération) alors il existe un circuit de longueur négative et donc il n'existe pas de solution au problème,FIN
Sinon on a trouvé la solution FIN

II.1.2 : Longueurs ≥ 0 , algorithme de MOORE-DIJKSTRA :

Principe identique de marquage des sommets.

A chaque itération l'ensemble des sommets est partagé en deux : X = S U X/S = S U S'
où S et S' sont définis par :

pour tout k $\in$ S , M(k)=L*(k).

pour tout k $\in$ S', $M(k)= \min_{i \in S \cap \Gamma^{-1}k} (M(i)+lik)$ (1)

= la longueur minimale des chemins de i0 à k à condition que tous les sommets

S est l'ensemble des sommets dont la marque est définitive, et S' est l'ensemble des sommets dont la marque est provisoire.

Lemme : Soit $j \in S'$ / $M(j)= \min_{i \in S'} \{ M(i) \}$ alors M(j)=L*(j)

Dém :
- il existe un chemin de i0 à j de longueur M(j) par définition de M
- soit Ch un chemin quelconque de i0 à j .

Ch peut être décomposé en 2 sous-chemins :
Ch1: i0 -> k où k est le premier sommet de S' rencontré
et Ch2 : k -> j.

on a : L(Ch1) = M(k) $\geq$ M(j) par définition de j et L(Ch2) $\geq$ 0
donc L(Ch) = L(Ch1) + L(Ch2) $\geq$ M(j) , et ceci pour tout chemin Ch de i0 à j.
Donc M(j)=L*(j).

Enoncé formel de l'algorithme de MOORE-DIJKSTRA :

a- Initialisations :
S' = {1,2,....N}
M(i0) <- 0
M(i) <- + ∞, pour tout i≠i0
P(i0) <- - i0 (* par convention les sommets j de S' sont identifiés par le fait que P(j)<0 *)
P(i) <- - i, pour tout i≠i0
j <- i0, (* j designe le j du lemme *)

b- Tant que M(j) < +∞, Faire : (* mise a jour de M(i) pour tout i $\in$ S' $\cap$ Γj *)
Pour tout i $\in$ S' $\cap$ Γj, Faire : (* et adjonction de j a S *)
calculer z <- M(j) + lji
Si z<M(i), faire :
M(i) <- z
P(i) <- -j
Sélectionner $j \in S'$ / $M(j)= \min_{i \in S'} \{M(i)\}$
S' <- S'- {j}
P(j) <- -P(j)

Complexité :
Au maximum, on a N itérations. A chaque itération, lorsque le nouveau sommet inclus dans S est j, il faut examiner tous les arcs (j,i) pour la mise à jour des M(i), cette phase est en

$O(d^+(j))$ opérations.
Pour la sélection du sommet j de S' de marque minimale, il faut N opérations => O(N).
Or $\sum d^+(j) = M$ donc au total la complexité est en $O(M)+O(N^2)$.
Or, on a toujours $M \leq N^2$, et donc on a une complexité en $O(N^2)$.

II.1.3 longueurs constantes <-> longueurs en nombre d'arcs :

Les informations stockées sont identiques (M(i) et P(i)), mais en plus on utilise une file (FIFO) contenant les sommets marqués avec le règles suivantes :
- chaque fois qu'un sommet i (précédemment non marqué reçoit une marque provisoire, il entre dans la file à l'adresse top+1.
- à chaque itération, le sommet en tête de file est le sommet sélectionné et il est extrait de la file.

Justification :

Il faut donc que le sommet de tête de file soit le sommet de marque provisoire minimum :

- au début de la première itération, lorsque i0 est sélectionné la file est vide, puis contient tous les successeurs j de i0 munis de la marque provisoire M(j)=1 qui sont stockés de manière consécutive dans la file.

-le sommet sélectionné a donc une marque M(j)=1. Les seuls nouveaux sommets mis dans la file sont les successeurs de j, non marqués, et ils reçoivent la marque M(j)+1=2.
Les sommets successeurs de j, qui étaient déjà marqués à 1, sont déjà dans la file avec la marque 1

- etc....

Donc tout sommet de marque provisoire k est examiné après tout sommet de marque provisoire $\leq$ k.

D'où le choix de l'élément de marque provisoire minimale se fait en O(1), ce qui économise le terme en $O(N^2)$ de MOORE-DIJKSTRA.
D'où complexité en O(M).

Enoncé formel de l'algorithme :

a- Initialisations :
M(i0) <- 0
M(i) <- $+\infty$, pour tout i $\neq$ i0
P(i0) <- -i0 (* par convention les sommets j de S' sont identifiés par le fait que P(j)<0 *)
P(i) <- $-\infty$
pile(k) <- 0 (pour tout k)
top <- 0
bottom <- 0
j <- i0

b- Tant que bottom $\leq$ top, Faire :

Pour tout i $\in \Gamma$j tel que P(i) < 0, Faire :
M(i) <- M(j)+1
P(i) <- j
top <- top+1
pile(top) <- i
bottom <- bottom + 1

si bottom $\leq$ top faire : j <- file (bottom)

Remarque : les trois algorithmes précédents construisent une arborescence de racine i0 qui détermine un ordre sur les sommets (M(i) croissant). Cet ordre peut être utilisé ensuite pour explorer les sommets du graphe.
C'est une manière d'explorer une arborescence dans les procédures d'exploration par séparation et évaluation (branch and bound).
Dans le cas des longueurs constantes, cela revient à faire une exploration en largeur d'abord.

II.2 Plus courts chemins entre toutes les paires de sommets, algorithme de FLOYD:

On peut évidemment utiliser les algorithmes précédents en partant de tous les sommets, mais :

$l \geq 0$ Moore-Dijkstra => $O(N^3)$
$l <> 0$ Ford-Bellman => $O(MN^2)$
$l = 1$ => $O(MN)$

L'algorithme de Floyd a une complexité en $O(N^3)$ même si les longueurs sont négatives ou nulles. Il s'agit d'une méthode matricielle.

Principe :

Soit $L=(l_{ij})$ la matrice des longueurs d'arcs avec $l_{ij} = +\infty$ si $(i,j) \notin U$ et $l_{ii}=0$.

On note $L(k) = (l_{ij}(k))$ la matrice dont le terme $l_{ij}(k)$ représente la longueur minimale d'un chemin de i à j à condition que tous les sommets intermédiaires appartiennent à {1,2,....k} et L(0) = L. La matrice L(N) est donc la matrice des longueurs des plus courts chemins

Propriété : Pour tout i et tout j , $l_{ij}(k) = \text{Min} \{ l_{ij}(k-1) , l_{ik}(k-1)+l_{kj}(k-1) \}$ (1)

Dém : le plus court chemin de i à j à sommets intermédiaires dans { 1,2,...k } passe par k (cas 1) ou ne passe pas par k (cas 2).
- cas 1 : ce chemin est formé d'un sous-chemin Ch1 : i -> k et d'un sous-chemin Ch2 : k -> j
Chacun d'eux a ses sommets intermédiaires dans {1,2,...k-1} et doit être minimal (principe d'optimalité de Bellman) et sont donc de longueur $l_{ik}(k-1)$ et $l_{kj}(k-1)$. Donc $l_{ij}(k) = l_{ik}(k-1) + l_{kj}(k-1)$.
- cas 2 : on a évidemment $l_{ij}(k) = l_{ij}(k-1)$

Algorithme :
La matrice des plus courts chemins L(N) va être calculée en N itérations grâce à la formule (1).
C-à-d à l'itération 1, le sommet 1 est inséré dans le chemin de i à j si $l_{ij} > l_{i1}+l_{1j}$, etc...
=> complexité en $O(N^3)$.

Remarques :
- cet algorithme permet de détecter l'existence d'un circuit de longueur négative et donc pas de solution. Le circuit apparait dès qu'un $l_{ii}(k)$ devient négatif.
- appliqué tel quel, l'algorithme de Floyd ne détermine que les longueurs des plus courts

chemins. Si l'on veut expliciter le chemin, on utilise une matrice P=(Pij) qui contient le prédécesseur sur le plus courts chemins avec :

- initialement : pij=i si lij < +∞
  pij=0 sinon
- a la kième itération, si le sommet k est inséré entre i et j, l'élément Pij est remplacé par le Pkj courant, c-à-d Pij <- Pkj si lij(k) = lik(k-1) + lkj (k-1).

- informatiquement, on n'a besoin que d'une seule matrice L.

Enoncé formel de l'algorithme de Floyd :

```
a- initialisation
Pour i=1,2,...,N
        Pour j=1,2,...,N
                Si (i,j) ∈ U, alors faire
                        L(i,j) <- lij
                        P(i,j) <- i
                Sinon, alors faire
                        L(i,j) <- +∞
                        P(i,j) <- 0
                Si i=j, alors faire
                        L(i,j) <- 0
b- Pour k=1,2,....N
        (* itération *)
        Pour i=1,2,...,N
                Pour j = 1,2,...,N
                        t <- L(i,k)+L(k,j)
                        si t < L(i,j) alors faire :
                                L(i,j) <- t
                                P(i,j) <- P(k,j)
                        si i=j et L(i,j)<0  alors
                                il existe un chemin de longueur négative -> FIN
```

III : Graphes sans circuit :

Problèmes :
- un graphe est-il sans circuit ?
- définition d'un ordre topologique et de la fonction rang.
- plus courts (plus longs) chemins dans un graphe sans circuit.

III.1 Ordre topologique - fonction rang :

Propriété : G(X,U) sans circuit => il existe un sommet de degré intérieur nul.

Définitions :

- une application f : X -> [1,2,...,k] est un *ordre topologique* ssi pour tout u=(i,j) ∈ U, f(i) < f(j).
Si l'on parcourt le graphe suivant l'ordre topologique, on ne rencontre un sommet que si l'on a déjà rencontré tous ses prédécesseurs.
On ne peut définir un ordre topologique que si G est sans circuit.

- soit R = { i ∈ X / d^-(i)=0 }, G[X,U] sans circuit, la fonction *rang* X -> N est définie par :
  rang (i) = 0 si i ∈ R
  rang (i) = le nombre d'arcs dans un chemin de cardinal maximal joignant R à i.

La fonction rang partitionne X en k classes appelées *niveaux*.

Propriété : Pour tout k>0, l'ensemble Xk des sommets de niveaux k est égal à l'ensemble des sommets de degré intérieur nul dans le graphe obtenu en éliminant tous les sommets de niveau 0,1,...,k-1.

Algorithme de détermination d'un ordre topologique si G est sans circuit, et de détection d'un circuit dans le cas contraire :

a- Soit G'(X',U') = G(X,U)
k <- -1

b- Tant que X' ≠ Ø, faire :

si tous les sommets i de G' sont tels que $d^-(i) \neq 0$ alors
il existe un circuit et FIN
sinon soit j / $d^-(j) = 0$, Faire :
k <- k+1
F(j) <- k
U' <- U' - {arc(j,i)}
X' <- X' - {j}

Justification : le sommet j n'est sélectionné que si son degré intérieur est devenu nul i.e. si tous ses ascendants ont été numérotés.

Remarque : - appliqué tel quel, l'algorithme détermine un ordre topologique. Pour déterminer la fonction rang, il suffit de sélectionner simultanément tous les sommets j / $d^-(j)=0$. Pour l'implantation de l'algorithme, on utilise une pile dans laquelle seront stockés successivement tous les éléments de rang 0, puis ceux de rang 1, etc... Les sommets compris entre oldtop et newtop sont de même rang.

Enoncé formel de l'algorithme de détermination de la fonction rang :

a- Initialisations :
Pour i=1,2,...,N Faire :
pile(i) <- 0
rang (i) <- - Card Γ^{-1}_i (degré intérieur du sommet i) (* rang(i)<0 ssi i ∈ X' *)
(* initialisation de la pile *)
k <- 0
Pour i=1,2,...,N Faire :
Si rang(i)=0 alors :
newtop <- newtop+1
pile(newtop) <- i
oldtop <- newtop
bottom <- 0

b- Etape courante : on examine successivement tous les sommets de rang k i.e. ceux qui se trouvent dans la pile entre bottom et oldtop
Tant que (bottom ≠ oldtop) faire :
bottom <- bottom+1
i <- pile(bottom)
rang(i) <- k
pour tout j ∈ Γi, faire :

rang(j) <- rang(j)+1
si rang(j)=0 alors faire : {i.e. tous les prédécesseurs de j ont été traités}
newtop <- newtop+1
pile(newtop) <- j

c- Passage à l'itération suivante et test d'arrêt :
(*si tous les sommets ont une valeur de rang alors l'algorithme se termine *)
si bottom=N alors FIN
sinon
(* s'il existe des sommets n'ayant pas reçu de valeur de rang alors que la pile est vide, c'est que tous les sommets non encore examiné ont un degré intérieur non nul ; il existe donc un circuit dans le graphe restant *)
si oldtop=newtop, alors il existe un circuit. FIN
sinon Faire :
k <- k+1
oldtop <- newtop
retourner en (b)

Complexité : L'algorithme consiste à parcourir la liste des sommets dans un certain ordre qui est l'ordre topologique. A chaque étape l'indice i du sommet à considérer est obtenu directement comme le contenu de la liste pile à l'adresse bottom (complexité O(1)). Puis les degrés intérieurs des sommets de Γi doivent être mis à jour, ce qui nécessite O(Card Γi) opérations élémentaires. Comme il y a au plus N étapes, le nombre total d'opérations élémentaires est :

$O(N)+O(\Sigma \text{ Card } \Gamma i)=O(N)+O(M)$ => complexité globale O(M).

III.2 : Plus court (plus long) chemin dans un graphe sans circuit :

Les longueurs des arcs sont supposées quelconques.

Le problème du plus long chemin ~ problème du plus court en inversant les signes des longueurs.

On se restreint ici à trouver le plus court chemin entre un sommet i0 donné et tous le autres sommets.

rappel : Si L*(j) est la longueur du plus court chemin entre i0 et j, on a :

$$L^*(j) = \min_{i \in \Gamma^{-1}j} \{ L^*(i) + l_{ij} \}, \text{ pour tout } j \neq i0 \quad (1)$$

$$L^*(i0) = 0$$

Principe :

Puisque tous les sommets peuvent être ordonnés suivant un ordre topologique F, il est possible de résoudre l'équation ci-dessus en une seule itération. En effet, soit j quelconque et supposons que l'on ait déterminé L*(i) pour tous les i / F(i) < F(j). Or pour tout $i \in \Gamma^{-1}j$, on a F(i) < F(j). Donc (1) peut être calculé directement en suivant un ordre topologique.

Algorithme : quasiment identique à celui de la fonction rang en utilisant une pile.

Remarque : dans le cas d'une maximisation, il suffit de remplacer Min par Max dans (1).

Application : le problème central de l'**ordonnancement**.

Un projet donné est composé de N tâches ayant entre elles des contraintes d'antériorité. Soit chaque tâche i a un durée déterminée di ou le passage d'une tâche i à une tâche j a un délai dij. Le problème est de trouver un ordonnancement des tâches pour lequel le projet sera fini le plus tôt possible et de déterminer pour chaque tâche la date au plus tôt à laquelle elle peut commencer et la date au plus tard à laquelle on peut la faire commencer sans retarder la date de fin des travaux.

Pour cela on crée un graphe dont chaque sommet est une tâche et dont les arcs représentent les contraintes d'antériorité.
Chaque arc (i,j) est valué soit par di soit par dij. Ce graphe est évidemment sans circuit.
On rajoute un sommet D de début des travaux reliés par des arcs valués à 0 aux sommets sans prédécesseurs et un sommet F de fin des travaux reliés aux sommets sans successeurs par des arcs valués soit à di soit à 0.

- La *date au plus tôt* de chaque tâche est donc égale à :

$$t_i = \max_{j \in \Gamma^{-1} i} (t_j + d_{ji}) \quad \text{et} \quad t_D = 0$$

c-à-d la longueur du plus long chemin entre D et i.

- la durée totale du projet est tF.

- Si on fixe la durée du projet à tF, la *date au plus tard* Ti de la tâche i est :

$$T_i = \min_{j \in \Gamma i} (T_j - d_{ij}) \quad \text{et} \quad T_F = t_F$$

- la *marge* est mi = Ti - ti.
Les tâches de marge nulles sont appelées *tâches critiques*. Ce sont celles qui ne peuvent prendre aucun retard.

Donc pour déterminer les ti, on utilisera l'algorithme précédent.
Pour déterminer les Ti, on utilisera l'algorithme précédent en prenant les sommets par ordre décroissant et en initialisant Tf à tf.

ARBRES ET ARBORESCENCES

I DEFINITIONS

- un **arbre** est un graphe connexe sans cycles (non orienté)
- une **forêt** est un graphe sans cycles => k arbres
- un **forêt maximale** est une forêt telle que si l'on ajoute un arc on introduit un cycle.

Propriétés :

a) G(X,U) est un arbre <=> entre 2 sommets quelconques il existe une chaîne unique.
b) Si G a N sommets et P composantes connexes, une forêt maximale de G comporte exactement N-P arêtes. Donc tout arbre extrait de G comporte N-1 arcs.
c) Soit F(X,T) une forêt maximale de G(X,U) Alors F et G ont même nombre de connexité.
d) G admet un graphe partiel qui est un arbre <=> G est connexe.
e) Soit F(X,T) une forêt maximale de G(X,U). Pour tout u de U-T, il passe un cycle et un seul C dont toutes les arêtes, sauf u, appartiennent à F (conséquence de a- et définition de forêt maximale).

II- ARBRE DE POIDS MINIMUM (MINIMUM SPANNING TREE)

Problème : Soit G(X,U) un graphe connexe. A chaque arête u de U est associé un nombre réel **p(u)** appelé le poids de l'arête.

Le problème est de trouver F*(X,T) tel que :

$$p(F^*) = \sum_{u \in F^*} p(u) \text{ soit minimal.}$$

Rappel : un cocycle CC est un ensemble d'arcs tel qu'il existe $A \subset X$ et tel que chacun des arcs de CC ait une extrémité dans A et l'autre dans X-A.

Propriété : Soit F(X,T) un arbre de G(X,U) connexe. Soit u(x,y) une arête de F. Le cocycle CCu tel que $CCu \cap T = \{u\}$ est composé d'arêtes v(a,b) telles que il existe une chaîne dans F de a à x et il existe une chaîne de y à b dans F.

Théorème : F(X,T) est un arbre de poids minimum ssi pour toute arête u de T, le cocycle CCu (tel que $CCu \cap T = \{u\}$) vérifie :

$p(u) \leq p(v)$ pour tout v de CCu ($v \neq u$).

Autrement dit, une C.N.S. pour qu'un arbre soit de poids minimum est que pour toute arête u de l'arbre, toutes les arêtes de CCu soient de poids supérieur à u.

Dém :

Condition nécessaire :

Soit $u \in T$ et $v \in CCu$ / $p(u) > p(v)$. Le graphe partiel (X,T+{v}-{u}) est un arbre (cf.propriété) et de poids strictement inférieur à F(X,T). Contradiction.

Condition suffisante :
Soit F(X,T) un arbre vérifiant la condition et soit F*(X,T*) un arbre de poids minimum avec F ≠ F*. Montrons que p(F)=p(F*).

Soit u ∈ T et u ∉ T*.
Soit CCu le cocycle associ à u.
Soit Cu le cycle introduit par u dans F*.

Soit v ∈ CCu ∩ Cu (v ≠ u), v ∉ T et v ∈ T*.
Par définition de T, on a : p(v) ≥ p(u).
On ne peut pas avoir p(v) > p(u), car en remplaçant v par u dans F*, on obtiendrait un arbre de poids inférieur à F* (qui est de poids minimum). Donc p(v)=p(u).
En résumé, pour tout u de F, il existe v de F* / p(u)=p(v) et donc p(F)=p(F*). c.q.f.d.

Algorithme de PRIM : (cas général) issu directement du théorème.

Principe : On construit progressivement à partir d'un sommet i0 arbitraire un sous-ensemble de sommets A de X contenant i0 et un sous-ensemble T de U tel que (A,T) soit un arbre de poids minimal du sous-graphe engendré par A.

Pour cela on sélectionne dans le cocycle CC(A)= { (k,l) / k ∈ A et l ∈ X-A } l'arête de poids minimum. Soit u=(i,j) cette arête. On ajoute alors j et u à A et T respectivement.

Mise en oeuvre :

- à chaque sommet i, on associe une marque m(i) telle qu'à une étape quelconque, pour i ∈ X-A, m(i) représente le poids de l'arête de poids minimal joignant i à un sommet de A.
- Il suffit alors de prendre le sommet i ayant la marque m(i) minimale.
- dans Lien(i) on conserve l'arête ayant permis d'attribuer à i la marque m(i).

Lorsque le sous-ensemble A est augmenté d'un sommet i, on met à jour les arêtes u(i,j) avec j ∈ X-A par : m(j) <- Min { m(j) , p(u) }
Pour identifier A et X-A, on prend par convention :
Lien(i)>0 ssi i ∈ A
Lien(i)<0 ssi i ∈ X-A.
A la fin, tous les éléments positifs de Lien(.) représentent l'arbre de poids minimum.

Enoncé formel de l'algorithme de PRIM :

a- initialisations
m(i0) <- - ∞
m(i) <- + ∞ pour tout i ≠ i0
Lien(i) <- - ∞ pour tout i

b- étape courante
sélectionner i ∈ X-A tel que m(i) = Min {m(j)}
si m(i)= +∞ ou X=A FIN
sinon : faire
Lien(i) <- -Lien(i) (équivalent à faire A <- A + { i })

c- mise à jour des marques:
pour toutes les arêtes u=(i,j) ayant i comme extrémité, faire :
si (p(u)<m(j) et Lien(j)<0) alors :

m(j) <- p(u)
Lien(j) <- - u
retourner en b.

Complexité :
à chaque étape il faut déterminer le sommet de marque minimale -> N opérations. D'où complexité en $O(N^2)$.
Remarque :
Si A=X, on a obtenu un arbre de poids minimum. Sinon c'est que X n'est pas connexe => cet algorithme permet de tester la connexité.

Théorème : F(X,T) est un arbre de poids minimum ssi pour tout arc u de U-T, le cycle Cu (tel que $Cu \subset T+\{u\}$) vérifie :

$p(u) \geq p(v)$, pour tout v de Cu ($v \neq u$).

Dém : conséquence du théorème précédent.
Conséquence : si les poids des arcs sont tous différents, l'arbre de poids minimum est unique. D'où l'algorithme (dans le cas où tous les poids sont différents).

Algorithme de Kruskal:

a- ranger les arêtes par ordre de poids croissant.
b- en suivant cet ordre, on reconstruit l'arbre de la façon suivante :
- soit u l'arête de poids minimum restant.
- on rajoute u à l'arbre. Si u introduit un cycle, on le retire, sinon on le garde.

Enoncé formel de l'algorithme de Kruskal

a)- ranger les arêtes par ordre de poids croissant dans U'.

b)- T <- Ø

cc (i) <- i, pour tout $i \in X$ (le tableau cc représente l'ensemble des composantes connexes de l'arbre en cours de construction).

c)- soit u=(i,j) l'arête de poids minimum restant dans U'.
U'=U-{u}

Si cc (i) $\neq$ cc(j), (l'arête u est valide) faire
- pour tout k / cc (k) = cc (j), faire : cc(k) <- cc(i);
- T <- T+{u}

Si U'= Ø alors FIN
Sinon retourner en c)

III ARBORESCENCES :

Définitions :
Soit G(X,U) orienté. On appelle **arborescence** F(X,T) de racine i0, un graphe partiel de G tel que :

- F est une arbre de G
- Pour tout j ≠ i0, il existe un chemin de i0 à j.

Un graphe est dit **quasi fortement connexe** si pour tout couple de sommet (i,j) il existe un sommet k relié à i et à j par deux chemins, l'un entre k et i, l'autre entre k et j.

forte connexité => quasi forte connexité => connexité

Propriétés : les propriétés suivantes sont équivalentes.
Soit F(X,T) un graphe d'ordre N>1.
a) F est un arbre admettant une racine i0.
b) Il existe un sommet i0 qui est relié à chacun des autres sommets par un chemin unique d'origine i0.
c) F est quasi fortement connexe et est minimal pour cette propriété.
d) f est connexe et il existe i0 / $d^-(i0)=0$ et pour tout $j \neq i0$, $d^-(j)=1$.
e) F est sans cycle et il existe i0 / $d^-(i0)=0$ et pour tout $j \neq i0$, $d^-(j)=1$.
f) F est quasi fortement connexe et sans cycle.
g) F est quasi fortement connexe et comporte N-1 arcs.

IV - ARBORESCENCE DE POIDS MINIMUM :

Problème : On associe à chaque arc u un poids p(u), on veut chercher l'arborescence F* de racine i0 telle que :

$$p(F^*) = \sum_{u \in F^*} p(u) \text{ soit minimal.}$$

Le principe de l'algorithme est basé sur la propriété e).
Pour chaque sommet j ≠ i0, on choisit l'arc incident à j de poids minimum. Si le graphe obtenu est sans circuit (ou de façon équivalente s'il est connexe) alors il s'agit de l'arborescence de poids minimum. Sinon il contient un circuit C.

Théorème : Si F*(X,T*) est une arborescence de poids minimal de racine i0 alors F* contient tous les arcs du circuit C sauf un. (C est le circuit obtenu par la méthode précédente).

Dém :

F* ne peut contenir tous les arcs de C (évident).

Supposons que F* contienne C sauf 2 arcs (a,b) et (c,d).
Comme F est une arborescence, il existe un chemin de i0 à b et un chemin de i0 à c.
Soit a' le prédécesseur de b sur ce chemin, a' ≠ a par hyp.
Soit c' le prédécesseur de d sur ce chemin, c' ≠ c par hyp.
Par construction de C, on a : p((a',b)) > p((a,b))
En remplaçant (a',b) par (a,b) on obtiendrait une arborescence de poids inférieur. c.q.f.d.

Le principe de l'algorithme est, lorsque un circuit C est créé, de contracter G(X,U) suivant C, c-à-d à remplacer C par un pseudo-sommet => G(X1,U1) = G/C.
Soit F1*=(X1,T1*) la trace de F sur G1.
F* et F1* sont liés par la relation :

p(F*)=p(F1*)+p(C)-p(u') où u' est l'arc de C qui n'appartient pas à F.

Lors de la contraction on modifie les poids de la façon suivante :

p1(u) = p(u)-p(u') si u=(i,j) avec i $\notin$ C, j $\in$ C et u'=(k,j), u'$\in$ C.
p1(u) = p(u) sinon.

On a alors p(F*)=p1(F1*)+p(C).

Donc F1* est une arborescence de poids minimal par rapport aux poids p1 dans G1. On peut alors reconstruire F en prenant F1 plus C moins l'arc u' intérieur à C correspondant à l'arc incident à C.

On a ramené le problème à un problème de taille plus petite, etc...

Enoncé formel de l'algorithme de recherche d'une arborescence de poids minimal

a- initialisations
t <- 0 (t numéro de l'itération)
G_t <- G
$p_t(u)$ <- p(u) pour tout u

b- A l'itération courante t déterminer le graphe partiel H_t de G_t en sélectionnant, pour chaque sommet de G_t (différent de la racine) l'arc incident intérieurement et de poids minimal (relativement aux poids p_t).

c- Si H_t ne contient pas de circuit (en particulier si G_t est réduit à un seul sommet, la racine), alors H_t est une arborescence de poids minimal dans G_t : on reconstitue alors l'arborescence de poids minimal induite par H_t sur G_0 et l'algorithme se termine.

sinon:

d- Si C est une circuit, définir : G_{t+1} <- G_t/C (graphe contracté de G_t par rapport aux sommets de C) et le poids des arcs G_{t+1} par :

$p_{t+1}(u) = p_t(u)$ pour u(i,j), j $\notin$ C

$p_{t+1}(u) = p_t(u) - p_t(u')$ pour u=(i,j), i $\notin$ C, j $\in$ C et u'=(k,j) $\in$ C.

(si H_t contient plusieurs circuits C1,C2,...,Cp contracter Gt par rapport à c1, puis le graphe obtenu par rapport à C2, etc..)
Faire t <- t+1, retourner en b.

Complexité :
A chaque niveau il faut :
- balayer la liste des arcs pour déterminer l'arc de poids minimum incident à chaque sommet => au plus M opérations.
- détecter les circuits de Ht => N opérations.

N niveaux maximum => O(N max(M,N)) = O(MN).

ARBRES ET ARBORESCENCES

I DEFINITIONS

- un **arbre** est un graphe connexe sans cycles (non orienté)
- une **forêt** est un graphe sans cycles => k arbres
- un **forêt maximale** est une forêt telle que si l'on ajoute un arc on introduit un cycle.

Propriétés :

a) G(X,U) est un arbre <=> entre 2 sommets quelconques il existe une chaîne unique.
b) Si G a N sommets et P composantes connexes, une forêt maximale de G comporte exactement N-P arêtes. Donc tout arbre extrait de G comporte N-1 arcs.
c) Soit F(X,T) une forêt maximale de G(X,U) Alors F et G ont même nombre de connexité.
d) G admet un graphe partiel qui est un arbre <=> G est connexe.
e) Soit F(X,T) une forêt maximale de G(X,U). Pour tout u de U-T, il passe un cycle et un seul C dont toutes les arêtes, sauf u, appartiennent à F (conséquence de a- et définition de forêt maximale).

II- ARBRE DE POIDS MINIMUM (MINIMUM SPANNING TREE)

Problème : Soit G(X,U) un graphe connexe. A chaque arête u de U est associé un nombre réel **p(u)** appelé le poids de l'arête.

Le problème est de trouver F*(X,T) tel que :

$$p(F^*) = \sum_{u \in F^*} p(u) \text{ soit minimal.}$$

Rappel : un cocycle CC est un ensemble d'arcs tel qu'il existe $A \subset X$ et tel que chacun des arcs de CC ait une extrémité dans A et l'autre dans X-A.

Propriété : Soit F(X,T) un arbre de G(X,U) connexe. Soit u(x,y) une arête de F. Le cocycle CCu tel que $CCu \cap T = \{u\}$ est composé d'arêtes v(a,b) telles que il existe une chaîne dans F de a à x et il existe une chaîne de y à b dans F.

Théorème : F(X,T) est un arbre de poids minimum ssi pour toute arête u de T, le cocycle CCu (tel que $CCu \cap T = \{u\}$) vérifie :

$p(u) \leq p(v)$ pour tout v de CCu ($v \neq u$).

Autrement dit, une C.N.S. pour qu'un arbre soit de poids minimum est que pour toute arête u de l'arbre, toutes les arêtes de CCu soient de poids supérieur à u.

Dém :

Condition nécessaire :

Soit $u \in T$ et $v \in CCu$ / $p(u) > p(v)$. Le graphe partiel (X,T+{v}-{u}) est un arbre (cf.propriété) et de poids strictement inférieur à F(X,T). Contradiction.

Condition suffisante :
Soit F(X,T) un arbre vérifiant la condition et soit F*(X,T*) un arbre de poids minimum avec F ≠ F*. Montrons que p(F)=p(F*).

Soit u ∈ T et u ∉ T*.
Soit CCu le cocycle associ à u.
Soit Cu le cycle introduit par u dans F*.

Soit v ∈ CCu ∩ Cu (v ≠ u), v ∉ T et v ∈ T*.
Par définition de T, on a : p(v) ≥ p(u).
On ne peut pas avoir p(v) > p(u), car en remplaçant v par u dans F*, on obtiendrait un arbre de poids inférieur à F* (qui est de poids minimum). Donc p(v)=p(u).
En résumé, pour tout u de F, il existe v de F* / p(u)=p(v) et donc p(F)=p(F*). c.q.f.d.

Algorithme de PRIM : (cas général) issu directement du théorème.

Principe : On construit progressivement à partir d'un sommet i0 arbitraire un sous-ensemble de sommets A de X contenant i0 et un sous-ensemble T de U tel que (A,T) soit un arbre de poids minimal du sous-graphe engendré par A.

Pour cela on sélectionne dans le cocycle CC(A)= { (k,l) / k ∈ A et l ∈ X-A } l'arête de poids minimum. Soit u=(i,j) cette arête. On ajoute alors j et u à A et T respectivement.

Mise en oeuvre :

- à chaque sommet i, on associe une marque m(i) telle qu'à une étape quelconque, pour i ∈ X-A, m(i) représente le poids de l'arête de poids minimal joignant i à un sommet de A.
- Il suffit alors de prendre le sommet i ayant la marque m(i) minimale.
- dans Lien(i) on conserve l'arête ayant permis d'attribuer à i la marque m(i).

Lorsque le sous-ensemble A est augmenté d'un sommet i, on met à jour les arêtes u(i,j) avec j ∈ X-A par : m(j) <- Min { m(j) , p(u) }
Pour identifier A et X-A, on prend par convention :
Lien(i)>0 ssi i ∈ A
Lien(i)<0 ssi i ∈ X-A.
A la fin, tous les éléments positifs de Lien(.) représentent l'arbre de poids minimum.

Enoncé formel de l'algorithme de PRIM :

a- initialisations
m(i0) <- - ∞
m(i) <- + ∞ pour tout i ≠ i0
Lien(i) <- - ∞ pour tout i

b- étape courante
sélectionner i ∈ X-A tel que m(i) = Min {m(j)}
si m(i)= +∞ ou X=A FIN
sinon : faire
Lien(i) <- -Lien(i) (équivalent à faire A <- A + { i })

c- mise à jour des marques:
pour toutes les arêtes u=(i,j) ayant i comme extrémité, faire :
si (p(u)<m(j) et Lien(j)<0) alors :

m(j) <- p(u)
Lien(j) <- - u
retourner en b.

Complexité :
à chaque étape il faut déterminer le sommet de marque minimale -> N opérations. D'où complexité en $O(N^2)$.
Remarque :
Si A=X, on a obtenu un arbre de poids minimum. Sinon c'est que X n'est pas connexe => cet algorithme permet de tester la connexité.

Théorème : F(X,T) est un arbre de poids minimum ssi pour tout arc u de U-T, le cycle Cu (tel que $Cu \subset T+\{u\}$) vérifie :

$p(u) \geq p(v)$, pour tout v de Cu ($v \neq u$).

Dém : conséquence du théorème précédent.
Conséquence : si les poids des arcs sont tous différents, l'arbre de poids minimum est unique. D'où l'algorithme (dans le cas où tous les poids sont différents).

Algorithme de Kruskal:

a- ranger les arêtes par ordre de poids croissant.
b- en suivant cet ordre, on reconstruit l'arbre de la façon suivante :
- soit u l'arête de poids minimum restant.
- on rajoute u à l'arbre. Si u introduit un cycle, on le retire, sinon on le garde.

Enoncé formel de l'algorithme de Kruskal

a)- ranger les arêtes par ordre de poids croissant dans U'.

b)- T <- Ø

cc (i) <- i, pour tout $i \in X$ (le tableau cc représente l'ensemble des composantes connexes de l'arbre en cours de construction).

c)- soit u=(i,j) l'arête de poids minimum restant dans U'.
U'=U-{u}

Si cc (i) $\neq$ cc(j), (l'arête u est valide) faire
- pour tout k / cc (k) = cc (j), faire : cc(k) <- cc(i);
- T <- T+{u}

Si U'= Ø alors FIN
Sinon retourner en c)

III ARBORESCENCES :

Définitions :
Soit G(X,U) orienté. On appelle **arborescence** F(X,T) de racine i0, un graphe partiel de G tel que :

- F est une arbre de G
- Pour tout j ≠ i0, il existe un chemin de i0 à j.

Un graphe est dit **quasi fortement connexe** si pour tout couple de sommet (i,j) il existe un sommet k relié à i et à j par deux chemins, l'un entre k et i, l'autre entre k et j.

forte connexité => quasi forte connexité => connexité

Propriétés : les propriétés suivantes sont équivalentes.
Soit F(X,T) un graphe d'ordre N>1.
a) F est un arbre admettant une racine i0.
b) Il existe un sommet i0 qui est relié à chacun des autres sommets par un chemin unique d'origine i0.
c) F est quasi fortement connexe et est minimal pour cette propriété.
d) f est connexe et il existe i0 / $d^-(i0)=0$ et pour tout $j \neq i0$, $d^-(j)=1$.
e) F est sans cycle et il existe i0 / $d^-(i0)=0$ et pour tout $j \neq i0$, $d^-(j)=1$.
f) F est quasi fortement connexe et sans cycle.
g) F est quasi fortement connexe et comporte N-1 arcs.

IV - ARBORESCENCE DE POIDS MINIMUM :

Problème : On associe à chaque arc u un poids p(u), on veut chercher l'arborescence F* de racine i0 telle que :

$$p(F^*) = \sum_{u \in F^*} p(u) \text{ soit minimal.}$$

Le principe de l'algorithme est basé sur la propriété e).
Pour chaque sommet j ≠ i0, on choisit l'arc incident à j de poids minimum. Si le graphe obtenu est sans circuit (ou de façon équivalente s'il est connexe) alors il s'agit de l'arborescence de poids minimum. Sinon il contient un circuit C.

Théorème : Si F*(X,T*) est une arborescence de poids minimal de racine i0 alors F* contient tous les arcs du circuit C sauf un. (C est le circuit obtenu par la méthode précédente).

Dém :

F* ne peut contenir tous les arcs de C (évident).

Supposons que F* contienne C sauf 2 arcs (a,b) et (c,d).
Comme F est une arborescence, il existe un chemin de i0 à b et un chemin de i0 à c.
Soit a' le prédécesseur de b sur ce chemin, a' ≠ a par hyp.
Soit c' le prédécesseur de d sur ce chemin, c' ≠ c par hyp.
Par construction de C, on a : p((a',b)) > p((a,b))
En remplaçant (a',b) par (a,b) on obtiendrait une arborescence de poids inférieur. c.q.f.d.

Le principe de l'algorithme est, lorsque un circuit C est créé, de contracter G(X,U) suivant C, c-à-d à remplacer C par un pseudo-sommet => G(X1,U1) = G/C.
Soit F1*=(X1,T1*) la trace de F sur G1.
F* et F1* sont liés par la relation :

p(F*)=p(F1*)+p(C)-p(u') où u' est l'arc de C qui n'appartient pas à F.

Lors de la contraction on modifie les poids de la façon suivante :

p1(u) = p(u)-p(u') si u=(i,j) avec i $\notin$ C, j $\in$ C et u'=(k,j), u'$\in$ C.
p1(u) = p(u) sinon.

On a alors p(F*)=p1(F1*)+p(C).

Donc F1* est une arborescence de poids minimal par rapport aux poids p1 dans G1. On peut alors reconstruire F en prenant F1 plus C moins l'arc u' intérieur à C correspondant à l'arc incident à C.

On a ramené le problème à un problème de taille plus petite, etc...

Enoncé formel de l'algorithme de recherche d'une arborescence de poids minimal

a- initialisations
t <- 0 (t numéro de l'itération)
G_t <- G
$p_t(u)$ <- p(u) pour tout u

b- A l'itération courante t déterminer le graphe partiel H_t de G_t en sélectionnant, pour chaque sommet de G_t (différent de la racine) l'arc incident intérieurement et de poids minimal (relativement aux poids p_t).

c- Si H_t ne contient pas de circuit (en particulier si G_t est réduit à un seul sommet, la racine), alors H_t est une arborescence de poids minimal dans G_t : on reconstitue alors l'arborescence de poids minimal induite par H_t sur G_0 et l'algorithme se termine.

sinon:

d- Si C est une circuit, définir : G_{t+1} <- G_t/C (graphe contracté de G_t par rapport aux sommets de C) et le poids des arcs G_{t+1} par :

$p_{t+1}(u) = p_t(u)$ pour u(i,j), j $\notin$ C

$p_{t+1}(u) = p_t(u) - p_t(u')$ pour u=(i,j), i $\notin$ C, j $\in$ C et u'=(k,j) $\in$ C.

(si H_t contient plusieurs circuits C1,C2,...,Cp contracter Gt par rapport à c1, puis le graphe obtenu par rapport à C2, etc..)
Faire t <- t+1, retourner en b.

Complexité :
A chaque niveau il faut :
- balayer la liste des arcs pour déterminer l'arc de poids minimum incident à chaque sommet => au plus M opérations.
- détecter les circuits de Ht => N opérations.

N niveaux maximum => O(N max(M,N)) = O(MN).

PROBLEMES DE FLOTS

I DEFINITIONS ET PROBLEME DE FLOT MAXIMUM :

Définition : Soit G(X,U) avec Card(X)=N et Card(U)=M. Un flot dans G est un vecteur à M composantes f=(f1,...,fM) tel que :

$$\sum_{u \in \omega^+(i)} fu = \sum_{u \in \omega^-(i)} fu \text{ , pour tout i de X (première loi de Kirchoff).}$$

A chaque arc u, on associe :
$cu > 0$ borne supérieure de flux ou capacité de u
$bu \geq 0$ borne inférieure de flux.
Un flot f est dit compatible avec les bornes bu et cu si $bu \leq fu \leq cu$ pour tout u

Soit G(X,U) dans lequel à chaque arc u est associée une capacité $cu > 0$ et deux sommets particuliers s (source) et t (puits).
Soit G0(X,U0) obtenu en rajoutant l'arc (t,s) (arc de retour) de numéro 0 et de capacité infinie.
On dit que le vecteur f=(f1,...,fM) est un flot de s à t et de valeur f0 si et seulement si f'=(f0,f1,...,fM) est un flot de G0.
Pour f=(f1,f2,..,fm) les lois de conservation aux noeuds sont vérifiées sauf en s et t où l'on a :

$$\sum_{u \in \omega^+(i)} fu = \sum_{u \in \omega^-(i)} fu = f0 \text{ , pour tout i de X.}$$

En général, on se limite à G 1-graphe.

Le problème de flot maximum de s à t dans G muni des capacités cu et bu revient à déterminer f'=(f0,f1,...,fM) tel que $bu \leq fu \leq cu$ et que f0 soit maximum.

Définition : On appelle graphe d'écart associé à un flot f vérifiant les contraintes aux bornes le graphe G'(f) = (X,U'(f)) où U'(f) est construit de la façon suivante :
à chaque arc u=(i,j) on fait correspondre au plus deux arcs:
u+ = (i,j) si $fu < cu$ de capacité (résiduelle) $cu-fu > 0$
u- = (j,i) si $fu > bu$ de capacité (résiduelle) $fu-bu > 0$.

Rappel : LEMME DE MINTY .

II CAS DES BORNES INFERIEURES NULLES :

Remarque : dans le cas des bornes inférieures nulles, le flot f=(0,0,...,0) est un flot compatible.

Théorème : Soit f un flot de s à t, compatible avec les cu. Soit G'(f) le graphe d'écart associé à f. Alors f est un flot maximum si et seulement si il n'existe pas de chemin de s à t dans G'(f).

Dém : conséquence du lemme de Minty (cf. Gondran et Minoux).

Le graphe d'écart permet d'une part de vérifier une solution mais aussi d'améliorer une solution :

Soit f de s à t ne vérifiant pas la condition du théorème. Il existe donc un chemin de s à t dans G'(f). Ce chemin correspond sur G à une chaîne L joignant s à t.
Par définition de G'(f), si la chaîne L, parcourue de s à t, c-à-d dans le sens direct, emprunte un arc u(i,j) de G dans le sens direct alors on a nécessairement $fu < cu$. Inversement, si elle emprunte u(i,j) dans le sens inverse alors $fu > 0$.

Soit L^+ = ensemble des arcs parcourus dans le sens direct.

L^- = ensemble des arcs parcourus dans le sens inverse.

Pour $\varepsilon > 0$ petit, soit la transformation :

$f'u = fu + \varepsilon$ pour $u \in L^+$.

$f'u = fu - \varepsilon$ pour $u \in L^-$.

Cette transformation préserve les conditions de conservation du flot.
De plus pour les sommets s et t, on a :

$$\sum_{u \in \omega^+(s)} f'u = \left(\sum_{u \in \omega^-(s)} fu \right) + \varepsilon = f0 + \varepsilon = \sum_{u \in \omega^-(t)} f'u \ .$$

Donc la valeur de f0 a été augmenté de ε.

Remarque : la valeur maximale possible pour ε est la capacité du chemin L du graphe d'écart c-à-d :

$\varepsilon = MIN(\varepsilon+ , \varepsilon-)$ où $\varepsilon+ = MIN_{u \in L^+}(cu- fu)$, $\varepsilon- = MIN_{u \in L^-}(fu)$

ALGORITHME DU FLOT MAXIMUM (FORD ET FULKERSON) .

a) Initialisations :
k <- 0
Partir d'un flot initial f0 compatible avec les contraintes de capacité, par exemple
f0=(0,0,0,....,0)

b) Test d'optimalité : recherche d'un chemin dans le graphe d'écart .
à l'itération courante k, soit f_k le flot courant. Rechercher un chemin de s à t dans le graphe d'écart $G'(f_k)$.
S'il n'en existe pas, FIN. Le flot f_k est maximum.
S'il en existe, soit L_k un tel chemin.

c) Changement de flot :

Soit ε_k la capacité du chemin L_k sur le graphe d'écart (minimum des capacités des arcs du chemin).
Définir le flot f_{k+1} par :

$fu_{k+1} = fu_k + \varepsilon_k$ si $u+ \in L_k$

$fu_{k+1} = fu_k - \varepsilon_k$ si $u^- \in L_k$

$f0_{k+1} = f0_k + \varepsilon_k$.

Faire k <- k+1 et retourner en b).

Complexité : elle dépend du choix de Lk.

- si Pk est le chemin de capacité maximale => algorithme non polynomial.
- si Pk est le plus court chemin en nombre d'arcs => $O(M.N^2)$.

III - PROBLEME DU FLOT MAXIMUM AVEC BORNES INFERIEURES NON NULLES

On peut généraliser la méthode précédente avec des bu quelconques et utiliser l'algorithme précédent en prenant : ε = valeur maximale telle que $\varepsilon \leq cu - fu$ et $\varepsilon \leq fu$.

Toutefois il reste le problème de la détermination du flot compatible initial. f=(0,0,..,0) n'est pas forcément compatible.

Théorème d'Hoffman et recherche d'un flot compatible :

Il existe un flot compatible ($bu \leq fu \leq cu$ pour tout u) si et seulement si pour tout cocycle $\omega(A) = \omega^+(A) \cup \omega^-(A)$:

$$\sum_{u \in \omega^+(A)} cu \; - \sum_{u \in \omega^-(A)} bu \geq 0.$$

Dém :

condition nécessaire :

si f compatible existe, on a , pour tout A

$$0 = \sum_{u \in \omega^+(A)} fu \; - \sum_{u \in \omega^-(A)} fu \leq \sum_{u \in \omega^+(A)} cu \; - \sum_{u \in \omega^-(A)} bu .$$

condition suffisante et construction d'un flot compatible :

Si f est un flot quelconque, on définit :

$du(fu) = 0$ si $fu \in [bu,cu]$
$bu - fu$ si $fu < bu$
$fu - cu$ si $fu > cu$

Donc en général $d(f) = \sum_{u \in U} du(fu) \geq 0$

L'idée est de minimiser de proche en proche la quantité d(f) jusqu'à trouver d(f)=0. Alors f sera un flot compatible.

Si à une étape quelconque, f vérifie $d(f) > 0$, il existe u0 tel que $d(fu_0)>0$ c-à-d par exemple $fu_0 < bu_0$. soit $u_0 = (y,x)$.

Soit la coloration du graphe suivante :
- l'arc u_0 est colorié en noir.
- tous les arcs / $fu \leq bu$ sont noirs
- tous les arcs / $bu < fu < cu$ sont rouges
- tous les arcs / $fu \geq cu$ sont verts.

D'après le lemme de Minty, on a les cas suivants :

a)- Il existe un cycle noir, rouge et vert, avec tous les arcs noirs orientés dans le même sens, tous les verts orientés dans le sens contraire.

Soit μ le vecteur associé à ce cycle avec $(\mu)_{u0} = +1$.

Soit $\varepsilon = \text{Min}(\varepsilon 1, \varepsilon 2)$ avec :

$\varepsilon 1$ = Min {cu-fu}, μ^+ ={u / μ_u =+1} c-à-d les arcs noirs du cycle et certains rouges
$u \in \mu^+$
$\varepsilon 2$ = Min {fu-bu}, μ^- = {u / μ_u =-1} c-à-d les arcs verts du cycle et les autres arcs rouges
$u \in \mu^-$

Soit alors f' = f + $\varepsilon.\mu$. On a d(f')<d(f).
L'algorithme se poursuit avec f'.
remarque : si cu et bu sont entiers, la convergence est assurée en un nombre fini d'itérations puisque d(f') décroît d'une unité à chaque itération.

b)- Si ce cycle n'existe pas, il existe un cocycle ω(A) tel que les arcs de ω^+(A) sont verts (c-à-d fu $\geq$ cu), les arcs de ω^-(A) sont noirs (c-à-d fu $\leq$ bu) avec fu0 < bu0.
Or $\sum_{u \in \omega^+(A)} fu - \sum_{u \in \omega^-(A)} fu = 0.$

$$\sum_{u \in \omega^+(A)} cu - \sum_{u \in \omega^-(A)} bu < 0.$$

Ce qui montre que la condition nécessaire n'est pas vérifiée; et donc il n'existe pas de flot compatible et l'algorithme s'arrête.

remarque : si fu0 > cu0 (au lieu de fu0 < bu0), on procède de façon analogue en inversant les rôles de vert et noir.

Pour obtenir un cycle μ faisant intervenir le cas a) du lemme de Minty, on utilise une procédure de marquage, ou de façon équivalente, on recherche un chemin de x à y dans le graphe d'écart.

La condition du théorème est suffisante :
si on suppose qu'elle est vérifiée , à chaque étape de l'algorithme précédent, on a toujours la condition a) du lemme de Minty, et on obtient toujours un flot f / d(f)=0

PARCOURS EULERIENS

Définitions : G[X,U] non orienté connexe. Card(U)=M

- Une chaîne eulérienne est une chaîne empruntant une fois et une seule chaque arête.
- Un cycle eulérien est une chaîne dont les extrémités coïncident.

Théorème : G connexe. G admet une chaîne eulérienne si et seulement si le nombre de sommets de degré impairs est 0 ou 2 (0 pour un cycle,2 dans l'autre cas).

Dém :

condition nécessaire : Seules les deux extrémités sont de degré impair. Pour les autres sommets i, il y a le même nombre d'arêtes avec lesquelles on arrive sur i que d'arêtes avec lesquelles on repart de i (donc i de degré pair).

condition suffisante : (par récurrence sur le nombre d'arêtes M).

- évident pour M =0
- on suppose la condition vraie pour tout graphe ayant moins de M arêtes.

- Soit G[X,U] / Card(U)=M et vérifiant la condition.

Soit a et b les 2 sommets de degré impairs (dans l'autre cas il suffit de diviser un sommet a en 2 sommets a et b ayant chacun la moitié des arêtes).

Soit L une chaîne parcourue en partant de a, en prenant n'importe qu'elle direction sans emprunter 2 fois la même arête.

A un instant donné soit x le sommet atteint. On aura donc utilisé un nombre impair d'arêtes incidentes a x => on peut donc en repartir.

On ne sera donc bloqué que pour le sommet b. On a défini ainsi une chaîne de a à b.

Soit G' le graphe partiel engendré par les arêtes non parcourues. Tous les sommets de G' sont de degré pair.

Soit $G'_1, G'_2, ..., G'_p$ les composantes connexes de G'.

Dans G'i il existe un cycle eulérien c_i (par hypothèse de récurrence).

L rencontre G'_i aux sommets $i_1, i_2, ..., i_p$.

Soit la chaîne L' composée de (à l'ordre de rencontre près des sommets i_k) :

L entre a et i_1,

du cycle c_1 entre i_1 et i_1,

L entre entre i_1 et i_2,

.... ,

L entre i_p et b.

La chaîne L' est donc une chaîne eulérienne entre a et b. c.q.f.d

L'algorithme de détermination d'une chaîne eulérienne découle directement de la démonstration.

On part de a et on suit un chemin quelconque sans emprunter la même arête. On arrive forcement en b. Si toutes les arêtes ont été rencontrées fin. Sinon. Soit x un sommet par lequel on est passé et de degré restant non nul. On recherche un circuit eulérien dans le graphe restant de x à x que l'on recolle à la première chaîne. On réitère.

Enoncé formel de l'algorithme :

a - Initialisations :

S(u) <- 0 pour tout u de U {S est un tableau / s(u) donne le numéro de l'arête suivante dans la chaîne}

D(i)<- d(i) pour tout i de X { A est un tableau donnant le degré restant de chaque sommet}

n<-0 { nombre d'arêtes dans la chaîne}

u'<-M+1 {dernière arête empruntée}

F(i) <- 0, pour tout i de X {F est un tableau donnant pour chaque sommet le numéro de la dernière arête utilisée pour arriver au sommet i}

u'' <- -1

i <- a

t <- b (on prend b=a quelconque si aucun sommet de degré impair}

F(a) <- M+1

b - Construction de la chaîne eulérienne par adjonction successive d'une arête adjacente à la dernière arête rencontrée :

Si D(i)=0 aller en d)

Sinon : soit u l'arête numéro D(i) dans la liste des arête adjacentes à i dans G.
Faire D(i) <- D(i)-1
Si u=u' ou si S(u)≠0 alors : retourner en b) (arête déjà empruntée)
Sinon : soit j l'autre extrémité de u. Faire :
F(j) <- i
S(u') <- u
n <- n+1
i <- j
Si (i=t) aller en c)
sinon Faire
u' <- u
retourner en b)

c- Arrêt de l'extension de la chaîne :

soit on atteint l'autre sommet de degré impair soit on est revenu au sommet de départ d'un cycle eulérien.

Faire S(u) <- u''.

Si n=M alors Fin sinon aller en d)

d) Initialisation de la recherche d'un cycle eulérien dans une composante connexe du graphe partiel des arêtes non encore utilisées.

Sélectionner un sommet k quelconque tel que D(k) ≠ 0 et F(k) ≠ 0.

Poser :

i <- k

t <- k

u' <- F(i)

u'' <- S(u')

retourner en b)

Complexité : chaque arête est sélectionnée au plus 2 fois pour construire le cheminement => O(M).

PARCOURS HAMILTONIENS

Définitions : G[X,U] non orienté connexe. Card(U)=M

- Une chaîne hamiltonienne est une chaîne passant une fois et une seule par chaque sommet.
- Un cycle hamiltonien est une chaîne dont les extrémités coïncident.

Problèmes : - trouver un chemin hamiltonien dans un graphe quelconque
- on attribue à chaque arête un longueur, trouver un chemin hamiltonien de longueur minimale.

A l'heure actuelle, on ne connait pas d'algorithme polynomial permettant de trouver une solution aux problèmes ci-dessus. (et selon toute vraisemblance il n'en existe pas)

=> pour résoudre ces 2 problèmes on donc explorer tout l'arbre des possibilités => algorithme de croissance exponentielle (dit NP-difficile).

PROBLEMES DE COUPLAGE

Définitions : Soit G(X,U) un graphe non orienté, un couplage est un ensemble d'arêtes K ⊂ U, tels que deux arêtes quelconques de K ne sont pas adjacentes.
On appelle couplage maximal un couplage de cardinalité maximale.

Un couplage correspond donc à un appariement possible d'objets,étant donné des contraintes d'appariement (du type tel objet ne peut être apparié qu'avec tels autres, etc...).

Problème : Trouver un couplage maximal.

Remarque : Dans le cas des graphes biparti, ce problème peut être résolu par un algorithme de flots.

Définitions : Si K ⊂ U est un couplage de G, un sommet i de X est dit saturé par le couplage K s'il existe une arête de K incidente à i. Dans le cas contraire i est dit insaturé.
L'ensemble des sommets saturés pour K est noté S(K).
On appelle chaîne alternée (relativement à K) une chaîne élémentaire de G dont les arêtes sont alternativement dans K et U-K.

Lemme 1 : Soient K et K' deux couplages distincts de G. Alors les composantes connexes du graphe partiel H(X,V) où V=K Δ K' sont de trois types :
type 1 : point isolé
type 2 : cycle élémentaire pair dont les arêtes sont alternativement dans K et K'
type 3 : chaîne élémentaire dont les arêtes sont alternativement dans K et K'.

Dém : Le degré maximum d'un sommet de H est 2 puisqu'il ne peut y avoir plus d'une arête de K et plus d'une arête de K' incidente à ce sommet.

a- Soit $x \notin S(K-K')$ et $x \notin S(K'-K)$ => x est isolé.
b- Soit $x \in S(K-K')$ et $x \notin S(K'-K)$. x est extrémité d'une unique arête de K et d'aucune arête de K'.
c- Soit $x \in S(K-K')$ et $x \in S(K'-K)$. x est extrémité d'une unique arête de K et d'une arête unique de K'.
D'où les composantes connexes.

Conséquence : Soit K ⊂U est un couplage et soit L une chaîne alternée telle que chaque extrémité est soit un sommet insaturé, soit telle que l'unique arête de K qui lui est incidente soit dans L. Alors le sous-ensemble K'=K Δ L est encore un couplage de G (cf. lemme 1).

K' est obtenu en échangeant les rôles des arêtes de L de "appartient à K" en "n'appartient pas à K" et inversement. Une telle opération est appelée transfert le long de L.

Définitions : Une chaîne alternée augmentante est une chaîne alternée joignant 2 sommets insaturés. Le transfert le long d'une telle chaîne augmente le couplage d'une unité.
Une chaîne alternée réductrice est une chaîne alternée élémentaire impaire dont les extrémités sont dans K (réduction d'une unité de couplage).
Une chaîne alternée conservatrice est une chaîne alternée élémentaire dont une des extrémités est un sommet isolé (un transfert ne change rien à la cardinalité du couplage).

Théorème : un couplage K est maximum si et seulement si il n'existe pas de chaîne alternée augmentante relativement à K.

=> évident
<= soit K vérifiant la condition du théorème et soit K' un couplage maximum.
K' vérifie donc la condition.

D'après le lemme 1, le graphe le graphe (X, K Δ K') n'admet pas de composante connexe de type 3 augmentante => Card(K-K') = Card (K'-K) => Card (K) = Card(K').

Corollaire 1 : Un couplage qui ne laisse pas plus d'un sommet insaturé est maximum.

Corollaire 2 : Soit K un couplage maximum de G. Tout couplage maximum K' de G peut s'obtenir à partir de K par une suite de transferts le long des chaînes alternées deux à deux disjointes et qui sont :
soit des cycles élémentaires pairs,
soit des chaînes alternées impaires.

Algorithme de recherche d'un couplage maximum :

D'après le théorème 1, il suffit de trouver une chaîne alternée augmentante (et donc qui joint 2 sommets insaturés) relativement à un couplage K ⊂ U quelconque. Il suffit ensuite d'effectuer un transfert le long de ce chemin.