Algorithmique de graphes

Lucas Létocart
LIPN - UMR CNRS 7030
Institut Galilée, Université Paris 13
99 av. Jean-Baptiste Clément
93430 Villetaneuse - FRANCE
lucas.letocart@lipn.univ-paris13.fr

Chapitre Contenu du document.

Chapitre 1. Introduction

La théorie des graphes et l'algorithmique qui lui est liée est un des outils privilégiés de modélisation et de résolution de problèmes dans un grand nombre de domaines allant de la science fondamentale aux applications technologiques concrètes. Par exemple, les graphes déterministes et/ou aléatoires sont utilisés en physique, en sciences sociales (pour représenter des relations entre groupes d'individus), en mathématique combinatoire, en informatique (structures de données et algorithmique). Concernant les très nombreuses applications, on peut citer : les réseaux électriques et transport d'énergie, le routage du trafic dans les réseaux de télécommunications et les réseaux d'ordinateurs, le routage de véhicules et l'organisation des tournées ou rotations, les problèmes de localisation (d'entrepôts, d'antennes, ...) et de placement, les problèmes d'ordonnancement de tâches et d'affectation de ressources, ...

Chapitre 2. Notions élémentaires

2.1. Quelques problèmes modélisables par des graphes

2.1.1. Choix d'un trajet en train

Comment faire pour aller en train le plus rapidement possible de Bordeaux à Grenoble sachant que la SNCF donne les temps de parcours suivants et que l'on suppose que toutes les correspondances sont possibles sans attente :

Bordeaux → Nantes	4h
Bordeaux → Marseille	6h
Bordeaux → Lyon	6h
Nantes → Paris-Montparnasse	2h
Nantes → Lyon	4h30
Paris-Montparnasse → Paris-Lyon	0h30
Paris-Lyon → Grenoble	3h
Marseille → Lyon	1h30
Marseille → Grenoble	3h
Lyon → Grenoble	1h15

Ce problème peut facilement être représenté par un graphe dont les arêtes sont valuées par les durées des trajets. Il s'agit alors de déterminer un plus court chemin entre Bordeaux et Grenoble.

2.1.2. Planification d'une session d'examens

8 groupes d'étudiants (numérotés de G1 à G8) doivent passer des examens dans différents enseignements, chaque examen occupant une demi-journée :

Chimie	G1 et G2
Électronique	G3 et G4
Informatique	G3, G5, G6 et G7
Mathématiques	G1, G5, G6 et G8
Physique	G2, G6, G7 et G8

On cherche à organiser la session d'examens la plus courte possible.

On peut représenter chaque enseignement par un sommet et relier par des arêtes les sommets qui correspondent aux examens ne pouvant se dérouler simultanément. Le problème est alors de colorier tous les sommets du graphe en utilisant le moins de couleurs possible sachant que deux sommets reliés par une arête doivent être de couleurs différentes.

2.1.3. Ordonnancement de tâches

La mise en exploitation d'un nouveau gisement minier demande l'exécution d'un certain nombre de tâches :

Codes	Tâches	Durées	Tâches antérieures
1	Obtention d'un permis d'exploitation	120	-
2	Établissement d'une piste de 6km	180	1
3	Transport et installation de 2 sondeuses	3	2
4	Création de bâtiments provisoires pour le bureau des plans et le logement des ouvriers	30	2
5	Goudronnage de la piste	60	2
6	Adduction d'eau	90	4
7	Campagne de sondage	240	3 et 4
8	Forage et équipement de 3 puits	180	5, 6 et 7
9	Construction de bureaux et de logements définitifs	240	5, 6 et 7
10	Transport et installation du matériel d'exploitation	30	8 et 9
11	Traçage et aménagement du fond	360	8 et 9

Cela revient à résoudre un problème d'ordonnancement qui consiste à déterminer les dates de début au plus tôt et au plus tard de chaque tâche ainsi que le temps minimum de réalisation de l'ensemble. Pour cela, on doit rechercher un plus long chemin dans un graphe, appelé graphe potentiels-tâches, dont les sommets représentent les tâches à réaliser et les arcs les contraintes d'antériorité entre les tâches : un arc (i, j) représente la nécessité de réaliser la tâche i avant la tâche j et l'arc (i, j) est valué par la durée d'exécution de i.

2.1.4. Routage dans les réseaux de télécommunications

Le problème de savoir s'il est possible d'acheminer un ensemble d'informations entre deux centres reliés par un réseau de télécommunications revient à chercher un flot de valeur maximale dans un graphe représentant ce réseau, les sommets représentant les centres et les arêtes les liaisons entre ces centres.

2.2. Définitions

Un **graphe orienté** $G = (X, U)$ est défini par :

- Un ensemble $X = \{x_1, x_2, \ldots, x_n\}$ dont les éléments sont appelés des **sommets** ou des **nœuds**. L'ordre du graphe G est le nombre de sommets n.

– Un ensemble $U = \{u_1, u_2, \ldots, u_m\}$ dont les éléments, appelés des **arcs**, sont des couples ordonnés de $X \times X = \{(x, y)|x \in X, y \in X\}$. Pour un arc $u = (x, y)$ avec $(x, y) \in X^2$, on dit que x est l'**extrémité initiale** et y l'**extrémité terminale** de u, y est **successeur** de x et x est **prédécesseur** de y. On note $|U| = m$.

Une **boucle** est un arc ayant le même sommet comme extrémité initiale et terminale : $u = (x, x)$ est appelé une boucle, $\forall x \in X$.

Un **p-graphe** est un graphe dans lequel il n'existe jamais plus de p arcs de la forme (x, y) entre x et y : $p = \max_{(x,y)\in X^2} |\{u \in U|u = (x, y)\}|$.

$\Gamma^+(x)$, **l'ensemble des successeurs** de x, $\forall x \in X$, est une application multivoque de X dans une partie de X telle que : $\Gamma^+(x) = \{y \in X|(x, y) \in U\}$. De même, $\Gamma^-(x)$, **l'ensemble des prédécesseurs** de x, $\forall x \in X$, est une application multivoque (réciproque) de X dans une partie de X telle que : $\Gamma^-(x) = \{y \in X|(y, x) \in U\}$. $\forall x \in X$, $\Gamma(x) = \Gamma^+(x) \bigcup \Gamma^-(x)$ est **l'ensemble des voisins** de x. Un sommet x est dit **isolé** si $\Gamma(x) = \{x\}$. Un sommet x est **adjacent** à Y si $x \notin Y$ et $x \in \Gamma^-(Y) \bigcup \Gamma^+(Y)$ où $\Gamma^-(Y) = \bigcup \Gamma^-(y) \forall y \in Y$ et $\Gamma^+(Y) = \bigcup \Gamma^+(y) \forall y \in Y$.

Si G est un 1-graphe, il est parfaitement défini par l'ensemble X et l'application multivoque Γ^+ ou Γ^-. On peut alors le noter $G = (X, \Gamma^+)$ ou $G = (X, \Gamma^-)$.

Le **demi-degré extérieur (resp. intérieur)** du sommet x, noté $d^+(x)$ (resp. $d^-(x)$) désigne le nombre d'arcs ayant x comme extrémité initiale (resp. terminale), $d^+(x) = |\Gamma^+(x)|$ et $d^-(x) = |\Gamma^-(x)|$.

Soit $Y \subset X$ un sous-ensemble de sommets :

– $\omega^+(Y)$ est l'ensemble des arcs ayant leur extrémité initiale dans Y et leur extrémité terminale dans $X \setminus Y$.
– $\omega^-(Y)$ est l'ensemble des arcs ayant leur extrémité terminale dans Y et leur extrémité initiale dans $X \setminus Y$.
– $\omega(Y) = \omega^+(Y) \bigcup \omega^-(Y)$ est appelé **co-cycle** du graphe. $\omega(Y)$ est dit **co-circuit** si $\omega^+(Y) = \emptyset$ ou $\omega^-(Y) = \emptyset$.

Un graphe G est dit **complet** si pour toute paire (x, y) de sommets (avec $x \neq y$) il existe au moins un arc (x, y) ou (y, x). Un 1-graphe est complet si et seulement si : $(x, y) \notin U \Rightarrow (y, x) \in U$.

Un graphe G est dit **symétrique** si $(x, y) \in U \Rightarrow (y, x) \in U$.

Un graphe G est dit **anti-symétrique** si $(x, y) \in U \Rightarrow (y, x) \notin U$.

Un graphe G est dit **biparti** si $X_1 \bigcup X_2 = X$, $X_1 \bigcap X_2 = \emptyset$ et $X_1 \bigcap \Gamma^+(X_1) = \emptyset$ (i.e. $\Gamma^+(X_1) \subset X_2$), $X_2 \bigcap \Gamma^+(X_2) = \emptyset$ (i.e. $\Gamma^+(X_2) \subset X_1$).

Un **graphe non orienté** $G = (X, E)$ est défini par :
– Un ensemble $X = \{x_1, x_2, \ldots, x_n\}$ dont les éléments sont appelés des **sommets** ou des **nœuds**. L'ordre du graphe G est le nombre de sommets n.
– Un ensemble $E = \{e_1, e_2, \ldots, e_m\}$ dont les éléments, appelés des **arêtes**, sont des couples non ordonnés de $X * X = \{(x, y)|x \in X, y \in X\}$. On note $|E| = m$.

Un **multigraphe** est un graphe non orienté pour lequel il peut exister plusieurs arêtes entre deux sommets x et y donnés.

Un graphe non orienté est dit **simple** s'il est sans boucle et s'il n'existe pas plus d'une arête entre toute paire de sommets.

Deux arcs (arêtes) sont dit(e)s **adjacent(e)s** s'ils ont au moins une extrémité commune.

Le **degré** du sommet x, noté $d(x)$, est le nombre d'arcs ou d'arêtes ayant x comme extrémité. On a $d(x) = d^+(x) + d^-(x)$ dans le cas d'un graphe orienté (attention : x est comptabilisé deux fois en cas de boucle).

Une **clique** est un graphe simple pour lequel il existe exactement une arête entre toute paire de sommets. On la note K_n si n est son nombre de sommets.

Le **sous-graphe** $SG(Y)$ de $G = (X, U)$ engendré par le sous-ensemble de sommets $Y \subset X$ est le graphe dont les sommets sont les éléments de Y et les arcs les éléments de U ayant leurs deux extrémités dans Y : $SG(Y) = (Y, U_Y)$ avec $U_Y = \{u \in U | u = (x, y), x, y \in Y^2\}$.

Le **graphe partiel** $GP(V)$ de $G = (X, U)$ engendré par le sous-ensemble d'arcs $V \subset U$ est le graphe dont les sommets sont ceux de X et les arcs ceux de V : $GP(V) = (X, V)$.

Le **sous-graphe partiel** SGP de $G = (X, U)$ engendré par le sous-ensemble de sommets $Y \subset X$ et le sous-ensemble d'arcs $V \subset U$ est le graphe partiel $GP(V)$ du sous-graphe $SG(Y)$ ou le sous-graphe $SG(Y)$ du graphe partiel $GP(V)$: $SGP = (Y, V)$ avec $Y \subset X$ et $V \subset U$.

Deux graphes $G = (X, U)$ et $G' = (X', U')$ sont dits **isomorphes** s'il existe deux bijections : $g : X \to X'$ et $h : U \to U'$ telles que $\forall u = (x, y) \in U$, $h(u) = (g(x), g(y)) \in U'$.

2.3. Connexité

2.3.1. Chaînes et chemins

Chaînes et cycles Une **chaîne** de longueur l allant du sommet x au sommet y est une suite d'arêtes $\mu(x, y) = (u_1, u_2, \ldots, u_l)$ tels que $\forall i \in \{1, \ldots, l-1\}$, u_i et u_{i+1} sont adjacents. Le sommet x (resp. y) est appelé l'extrémité initiale (resp. terminale) de la chaîne μ.

Une chaîne est dite **élémentaire** si elle ne passe jamais deux fois par le même sommet.

Une chaîne est dite **simple** si elle ne passe jamais deux fois par la même arête.

Remarques :

- Dans un graphe simple, si une chaîne est élémentaire, alors elle est simple.
- Dans le cas d'un graphe simple, une chaîne est parfaitement caractérisée par la suite des sommets qu'elle rencontre ou par l'extrémité initiale de la chaîne et la suite d'arêtes.

Une chaîne simple de longueur m (i.e. maximale) est une **chaîne eulérienne**.

Un **cycle** est une chaîne dont les extrémités initiale et terminale coïncident. Un cycle élémentaire est un cycle minimal, au sens de l'inclusion, i.e. ne contenant strictement aucun cycle.

Chemins et circuits Un **chemin** de longueur l allant du sommet x au sommet y est une suite d'arcs $\mu[x, y] = (u_1, u_2, \ldots, u_l)$ tels que $\forall i \in \{1, \ldots, l-1\}$, u_i et u_{i+1} sont adjacents et l'extrémité terminale de u_i est l'extrémité initiale de u_{i+1}. Le sommet x (resp. y) est appelé l'extrémité initiale (resp. terminale) du chemin μ.

Un chemin est dit **élémentaire** si il ne passe jamais deux fois par le même sommet.

Un chemin est dit **simple** si il ne passe jamais deux fois par le même arc.

Remarques :

- Dans un graphe simple, si un chemin est élémentaire, alors il est simple.
- Dans le cas d'un graphe simple, un chemin est parfaitement caractérisé par la suite des sommets qu'il rencontre ou par l'extrémité initiale du chemin et la suite d'arcs.

Un chemin élémentaire de longueur $n-1$ (i.e. maximale) est un **chemin hamiltonien**.

Un **circuit** est un chemin dont les extrémités initiale et terminale coïncident. Un circuit élémentaire est un circuit minimal, au sens de l'inclusion, i.e. ne contenant strictement aucun circuit.

Un **circuit hamiltonien** est un chemin hamiltonien $\mu[x_{i1}, x_{in}] = (x_{i1}, x_{i2}, \ldots, x_{in})$ complété par (x_{in}, x_{i1}).

2.3.2. Graphes et composantes connexes et fortement connexes

Connexité simple Un graphe G est dit **connexe** si $\forall (x,y) \in X^2\ x \neq y,\ \exists \mu(x,y)$.

Proposition : La relation R définie par $xRy \Leftrightarrow x = y$ ou $\exists \mu(x,y)$ est une relation d'équivalence.

Preuve :

- Elle est réflexive : xRx.
- Elle est symétrique : $xRy \Rightarrow yRx$ car il suffit d'inverser la chaîne.
- Elle est transitive : xRy et $yRz \Rightarrow xRz$ car il suffit de concaténer les deux chaînes.

Les classes d'équivalence induites sur X par R constituent une partition de X en $X_1, X_2, \ldots, X_p$. Le nombre p de classes d'équivalence est appelé le **nombre de connexité** du graphe. Un graphe est dit connexe si et seulement si son nombre de connexité est égal à 1. Les sous-graphes $G_1, G_2, \ldots, G_p$ engendrés par les sous-ensembles $X_1, X_2, \ldots, X_p$ sont appelés les **composantes connexes** de G. Chaque composante connexe est un graphe connexe.

Connexité forte Un graphe G est dit **fortement connexe** si $\forall (x,y) \in X^2\ x \neq y,\ \exists \mu[x,y]$.

Proposition : La relation R' définie par $xR'y \Leftrightarrow x = y$ ou $\exists \mu[x,y]$ et $\exists \mu[y,x]$ est une relation d'équivalence.

Preuve :

- Elle est réflexive : $xR'x$.
- Elle est symétrique : $xR'y \Rightarrow yR'x$.
- Elle est transitive : $xR'y$ et $yR'z \Rightarrow xR'z$.

Les classes d'équivalence induites sur X par R' constituent une partition de X en $X_1, X_2, \ldots, X_q$. Le nombre q de classes d'équivalence est appelé le **nombre de connexité forte** du graphe. Un graphe est dit fortement connexe si et seulement si son nombre de connexité forte est égal à 1. Les sous-graphes $G_1, G_2, \ldots, G_q$ engendrés par les sous-ensembles $X_1, X_2, \ldots, X_q$ sont appelés les **composantes fortement connexes** de G.

On appelle le **graphe réduit**, noté G_r, le graphe défini comme suit : les sommets de G_r représentent les composantes fortement connexes et il existe un arc entre deux sommets x et y si et seulement s'il existe au moins un arc entre un sommet de X_x et un sommet de X_y dans le graphe G.

Remarque : Un graphe réduit est nécessairement sans circuit.

Chapitre 3. Représentation des graphes

3.1. Matrices d'adjacence et d'incidence

3.1.1. Matrice d'adjacence

Soit $G = (X, U)$ un 1-graphe d'ordre n, la **matrice d'adjacence** $A = (a_{ij_{(n\times n)}})$ à coefficients 0 ou 1 est définie comme suit :

- $a_{xy} = 1 \iff (x, y) \in U$
- $a_{xy} = 0$ sinon.

3.1.2. Matrice d'incidence sommets-arcs

Soit $G = (X, U)$ un graphe orienté sans boucle d'ordre n, la **matrice d'incidence sommets-arcs** est une matrice $A = (a_{ij_{(n\times m)}})$ à coefficients entiers 0, $+1$ ou -1 telle que chaque colonne correspond à un arc et chaque ligne à un sommet. Si $u = (x, y)$ est un arc du graphe G alors la colonne u a tous ses termes nuls sauf :

- $a_{xu} = +1$
- $a_{yu} = -1$

3.1.3. Matrice d'incidence sommets-arêtes

Soit $G = (X, U)$ un graphe non orienté sans boucle d'ordre n, la **matrice d'incidence sommets-arêtes** est une matrice $A = (a_{ij_{(n\times m)}})$ à coefficients entiers 0 ou 1 telle que chaque colonne correspond à un arc et chaque ligne à un sommet. Si $u = (x, y)$ est une arête du graphe G alors la colonne u a tous ses termes nuls sauf :

- $a_{xu} = 1$
- $a_{yu} = 1$

3.2. Représentation des graphes en machine

3.2.1. À partir de la matrice d'adjacence

L'occupation mémoire d'une matrice d'adjacence est $O(n^2)$. Dans le cas de graphes peu denses, $m \ll n^2$ pour les graphes orientés et $m \ll n(n+1)/2$ pour les graphes non orientés, il est plus avantageux d'écrire uniquement que les termes non nuls de la matrice d'adjacence.

Représentation par liste d'adjacence On mémorise pour chaque sommet la liste de ses prédécesseurs ou de ses successeurs en utilisant des listes chaînées. Dans le cas orienté, cette représentation revient à décrire le graphe par son application multivoque Γ^- ou Γ^+.

Remarques :

- Les deux représentations seront différentes si le graphe n'est pas symétrique.
- Le désavantage d'une telle représentation est que le test d'appartenance d'un arc entre x et y peut atteindre $O(n)$ puisque l'on peut trouver $O(n)$ sommets sur la liste des successeurs de i.

3.2.2. À partir de la matrice d'incidence sommets-arcs ou sommets-arêtes

La matrice d'incidence permet de représenter un p-graphe ou un multigraphe (sans boucle). Comme il n'existe que deux termes non nuls par colonne dans cette matrice, on a toujours intérêt à représenter l'information sous une forme plus condensée.

Par liste des arcs On décrit la matrice d'incidence ligne par ligne : pour chaque sommet $x \in X$, on décrit la liste $\omega^+(x)$ des arcs ayant x comme extrémité initiale (ou comme extrémité dans le cas non orienté).

Par liste des cocycles On décrit la matrice d'incidence colonne par colonne.

Chapitre 4. Parcours des graphes

De nombreux problèmes fondamentaux en théorie des graphes concernent la connexité. On peut citer par exemple :

- ➔ Un sommet y est-il accessible par un chemin à partir d'un autre sommet ?
- ➔ Quel est l'ensemble de tous les sommets accessibles par un chemin à partir d'un sommet x ?
- ➔ Le graphe est-il connexe, c'est-à-dire tous les sommets sont-ils accessibles par une chaîne à partir d'un sommet donné x ?
- ➔ Le graphe est-il fortement connexe, c'est-à-dire existe-t-il un chemin joignant toute paire ordonnée (x, y) de sommets dans le graphe ?

Pour répondre à toutes ces questions, Tarjan a proposé en 1972 un ensemble d'algorithmes efficaces (de complexité linéaire en fonction du nombre d'arcs ou d'arêtes du graphe) qui sont fondés sur une méthode d'exploration systhématique des sommets d'un graphe connue sous le nom de parcours en profondeur d'abord.

4.1. Parcours en profondeur d'abord : DFS (Depth First Search)

Il s'agit d'une généralisation du parcours préfixé des arbres.

On explore G à partir d'un sommet x_0 quelconque. Au cours de l'exploration, chaque sommet peut se trouver dans l'un des deux états suivants :

- ➢ E_1 : non exploré ;
- ➢ E_2 : exploré.

Remarque : Si un sommet est dans l'état E_2 alors au moins un prédécesseur de ce sommet a déjà été exploré.

Un sommet x exploré peut être dans l'un deux états suivants :

➢ S_1 : fermé. C'est le cas quand tous les successeurs y de x ont été explorés ;

➢ S_2 : ouvert. C'est le cas quand certains successeurs y de x n'ont pas encore été explorés ;

Pour chaque sommet x, on dispose d'un compteur $c(x)$ indiquant à chaque instant le nombre de successeurs de x non encore explorés. Initialement $c(x) = d^+(x)\ \forall x$. Un sommet x est donc ouvert tant que $c(x) > 0$. À l'itération courante, on part du sommet exploré x (exploré à partir d'un sommet s). Si x est ouvert, on sélectionne le $c(x)$ième successeur non exploré y de x (il faut alors décrémenter $c(x)$ de 1), pour passer y à l'état exploré et commencer une nouvelle itération à partir de y. Si x est fermé, il faut remonter au sommet s pour commencer une nouvelle itération à partir de s. L'exploration à partir de x_0 se termine lorsque l'on est remonté en ce sommet et qu'il est fermé. On a donc pu atteindre tous les sommets pouvant être atteints par un chemin à partir de x_0 dans le graphe. Rappelons qu'un sommet x relié par un chemin à tous les autres sommets d'un graphe est appelé **racine**. Voici l'algorithme en pseudo-langage de cette exploration par un parcours en profondeur d'abord :

```
Procédure DFSx0(graphe G, sommet x0)
    k ← 1 ;
    Pour i de 1 à n faire                                          ;
        c(i) ← d+(i)
    Fin Pour
    Pour i de 1 à n faire                                          ;
        num(i) ← 0
    Fin Pour
    num(x0) ← k ;
    explore(G, x0) ;
Fin
```

Algorithme 1: Parcours en profondeur d'abord à partir du sommet x_0

```
Procédure explore(graphe G, sommet x)
    Tant que (c(x) > 0) faire
        Sélectionner y le c(x)ième successeur de x ;
        c(x) ← c(x) − 1 ;
        Si (num(y) = 0) Alors
            k ← k + 1
            num(y) ← k
            explore(G,y)
        Fin Si
    Fait
Fin
```

Algorithme 2: Exploration en profondeur d'abord

DFS$x_0(G, x_0)$ permet d'explorer tous les sommets de G s'il est possible de les atteindre à partir de x_0. Si ce n'est pas le cas, il reste des sommets non explorés. Pour continuer l'exploration et couvrir tous les sommets de G, on choisit un sommet x_1 parmi les sommets non explorés (tels que $num(x_1) = 0$) et on applique DFS à partir de x_1 ; et ainsi de suite tant qu'il existe des sommets non explorés. L'algorithme prend fin lorsque tous les sommets ont été explorés.

```
Procédure DFS(graphe G)
    k ← 0 ;
    Pour i de 1 à n faire                                              ;
        c(i) ← d⁺(i)
    Fin Pour
    Pour i de 1 à n faire                                              ;
        num(i) ← 0
    Fin Pour
    Pour i de 1 à n faire
        Si (num(i) = 0) Alors
            k ← k + 1 ;
            num(i) ← k ;
            explore(G, i) ;
        Fin Si
    Fin Pour
Fin
```

Algorithme 3: Parcours en profondeur d'abord

La complexité de cet algorithme est en $O(m) + O(n)$ car chaque arc est examiné au plus une fois et chaque sommet est exploré au plus une fois. Or si G est connexe, $m \geq n - 1$ et le terme $O(n)$ est absorbé.

On obtient alors une forêt décrivant l'exporation de G par DFS, mais également une caractérisation des arcs du graphe. On distingue 4 types d'arc :

- ➔ **Arc d'arbre** : arc ayant permis de visiter de nouveaux sommets lors de l'exploration par DFS.
- ➔ **Arc avant** : (x, y) est un arc avant si et seulement si (x, y) n'est pas un arc d'arbre et y est un descendant de x dans l'un des arbres de la forêt décrivant l'exploration par DFS (i.e. qu'il existe un chemin de x à y dans l'un des arbres).
- ➔ **Arc arrière** : (x, y) est un arc arrière si et seulement si (x, y) n'est pas un arc d'arbre et x est un descendant de y dans l'un des arbres de la forêt décrivant l'exploration par DFS.
- ➔ **Arc croisé** : (x, y) est un arc croisé si et seulement si y n'est un descendant de x et x n'est un descendant de y dans la forêt décrivant l'exploration par DFS.

4.2. Détermination des composantes connexes

4.2.1. Cas non orienté

Pour déterminer les composantes connexes d'un graphe $G = (X, U)$ non orienté, il suffit d'appliquer l'algorithme de parcours du graphe en profondeur d'abord en modifiant la procédure DFS en initialisant les compteurs $c(x)$ par les degrés $d(x)$ et en explorant, non pas les successeurs des sommets, mais les voisins. Chaque arbre de la forêt obtenue couvre des sommets appartenant à une même composante connexe et le nombre d'arbres de la forêt est donc le nombre de connexité de G. Ce nouvel algorithme, appelé DFSno, est présenté ci-dessous :

```
Procédure DFSno(graphe G)
    k ← 0 ; l ← 0 ;
    Pour i de 1 à n faire                                          ;
        c(i) ← d(i)
    Fin Pour
    Pour i de 1 à n faire                                          ;
        num(i) ← 0
    Fin Pour
    Pour i de 1 à n faire                                          ;
        ncomp(i) ← 0
    Fin Pour
    Pour i de 1 à n faire
        Si (num(i) = 0) Alors
            l ← l + 1 ;
            ncomp(i) ← l ;
            explore(G, i) ;
        Fin Si
    Fin Pour
Fin
```

Algorithme 4: Parcours en profondeur d'abord pour la détermination des composantes connexes

```
Procédure explore(graphe G, sommet x)
    Tant que (c(x) > 0) faire
        Sélectionner y le c(x)ième voisin de x ;
        c(x) ← c(x) − 1 ;
        Si (num(y) = 0) Alors
            k ← k + 1 ;
            num(y) ← k ;
            ncomp(y) ← l ;
            explore(G,y)
        Fin Si
    Fait
Fin
```

Algorithme 5: Exploration en profondeur d'abord pour la détermination des composantes connexes

4.2.2. Cas orienté

Pour déterminer les composantes connexes d'un graphe $G = (X, U)$ orienté, il suffit de transformer G en un graphe G' non orienté et appliquer DFSno(G'). Chaque arbre de la forêt obtenue couvre des sommets appartenant à une même composante connexe et le nombre d'arbres de la forêt est donc le nombre de connexité de G.

4.3. Détermination des composantes fortement connexes

L'exploration d'un graphe par DFS permet d'obtenir une numérotation préfixe et une numérotation suffixe des sommets comme suit :

```
Procédure DFS(graphe G)
    k ← 0 ;
    f ← 1 ;
    Pour i de 1 à n faire                                                    ;
        c(i) ← d⁺(i)
    Fin Pour
    Pour i de 1 à n faire                                                    ;
        num(i) ← 0
    Fin Pour
    Pour i de 1 à n faire
        Si (num(i) = 0) Alors
            k ← k + 1 ;
            num(i) ← k ;
            prefixe(i) ← k ;
            explore(G, i) ;
        Fin Si
    Fin Pour
Fin
```

Algorithme 6: Parcours en profondeur d'abord

```
Procédure explore(graphe G, sommet x)
    Tant que (c(x) > 0) faire
        Sélectionner y le c(x)ième successeur de x ;
        c(x) ← c(x) − 1 ;
        Si (num(y) = 0) Alors
            k ← k + 1;
            num(y) ← k;
            prefixe(y) ← k ;
            explore(G,y)
        Fin Si
    Fait
    suffixe(x) ← f ;
    f ← f + 1;
Fin
```

Algorithme 7: Exploration en profondeur d'abord

L'algorithme de Kosaraju-Sharir utilise la numérotation suffixe pour déterminer les composantes fortement connexes d'un graphe $G = (X, U)$:

1. Appliquer DFS(G) en numérotant les sommets dans l'ordre suffixe.
2. Construire le graphe $G' = (X, U')$ obtenu en inversant le sens des arcs de G.
3. Appliquer DFS(G') en démarrant par le sommet de plus grand numéro suffixe et itérer le processus à partir du sommet non marqué de plus grand numéro suffixe.

Théorème 1 *Chaque arbre de la forêt construite avec DFS(G') couvre les sommets constituant une composante fortement connexe de G.*

Preuve:

1. Il faut commencer par montrer que si x et y, deux sommets de G, appartiennent à une même composante fortement connexe, alors ils appartiennent au même arbre de la forêt d'exploration en profondeur de G'. Si x et y appartiennent à la même composante fortement connexe, il existe un chemin allant de x à y et un autre allant de y à x. Ainsi, si on effectue une recherche en profondeur d'abord dans G' à partir d'un sommet z et que l'on atteint x alors on atteindra nécessairement y car x et y sont mutuellement accessibles et inversement.
2. Il faut également montrer que si x et y appartiennent au même arbre de la forêt d'exploration en profondeur d'abord de G', alors x et y appartiennent à la même composante fortement connexe de G. Soit z la racine de l'arbre contenant x et y. Il existe nécessairement un chemin allant de z à x dans G' et donc de x à z dans G. Par ailleurs, on a par construction : suffixe(z) $>$ suffixe(x) ce qui signifie que dans DFS(G), l'appel récursif en x s'est terminé avant celui en z. Deux cas doivent alors être considérés : soit prefixe(z) $>$ prefixe(x), soit prefixe(x) $>$ prefixe(z). Si prefixe(z) $>$ prefixe(x), cela signifie que l'on explore x avant z dans G mais, comme il existe un chemin de x à z, on doit atteindre z par x ce qui contredit suffixe(z) $>$ suffixe(x). Ainsi prefixe(x) $>$ prefixe(z) et suffixe(z) $>$ suffixe(x) impliquent que x est un descendant de z dans un arbre de la forêt d'exploration en profondeur d'abord de G. Il existe donc un chemin allant de z à x dans G ce qui implique que z et x appartiennent à la même composante fortement connexe. Un raisonnement annalogue permet de montrer que z et y appartiennent aussi à la même composante fortement connexe. Donc x et y appartiennent à la même composante fortement connexe.

■

On peut ainsi construire le graphe réduit associé à G.

4.4. Tri topologique dans un graphe orienté sans circuit

On rencontre fréquemment les graphes sans circuits dans les applications. Par exemple, dans un problème d'ordonnancement, le graphe potentiels-tâches $G = (X, U)$, défini de la façon suivante : les sommets représentent les tâches à exécuter et (x, y) est un arc si et seulement si l'exécution de la tâche x doit précéder celle de la tâche y et la longueur de l'arc (x, y) représente la durée de la tâche x ; doit par construction être un graphe sans circuit.

Dans un graphe sans circuit, on peut définir les notions d'**ordre topologique** et de **fonction rang**.

Définition 1 *Étant donné un graphe $G = (X, U)$ d'ordre n, on appelle ordre topologique une numérotation v des sommets de 1 à n telle que les numéros d'ordre $v(y)$ associés aux sommets y vérifient :*

$$\forall (x, y) \in U \ : v(x) < v(y).$$

En d'autres termes, si l'on parcourt la liste des sommets dans l'ordre défini par une telle numérotation, un sommet y ne peut être rencontré que si l'on a, au préalable, rencontré tous ses prédécesseurs.

Remarque 1 *Il n'y a pas unicité : plusieurs ordres topologiques peuvent être définis sur un même graphe.*

Propriété 1 *Un graphe G est sans circuit si et seulement si il n'existe pas d'arc arrière dans le parcours en profondeur d'abord de G.*

Preuve: Montrons que s'il existe un circuit, alors il existe un arc arrière dans le parcours en profondeur d'abord de G. Soit le circuit $\mu = (x, x_0, \ldots, x_k, x)$. Si x est le premier sommet du circuit exploré par DFS, x sera la racine d'un arbre comprenant au moins tous les autres sommets du circuit. Si un arc du circuit n'est pas un arc d'arbre, alors il s'agit d'un arc arrière et il existe au moins un arc du circuit qui ne sera pas un arc d'arbre. La réciproque est triviale en utilisant un raisonnement similaire, en effet, s'il existe un arc arrière alors le graphe contient au moins un circuit. ■

Propriété 2 *L'ordre suffixe inversé est un ordre topologique.*

Preuve: Le tableau ci-dessous présente, pour un type d'arc (x, y), les relations entre les numérotations suffixes et préfixes des sommets x et y.

avant	prefixe(x) < prefixe(y)	suffixe(x) > suffixe(y)
arbre	prefixe(x) < prefixe(y)	suffixe(x) > suffixe(y)
croisé	prefixe(x) > prefixe(y)	suffixe(x) > suffixe(y)
arrière	prefixe(x) > prefixe(y)	suffixe(x) < suffixe(y)

Comme il n'existe par d'arc arrière lorsque le graphe est sans circuit, l'ordre suffixe inversé est bien un ordre topologique. ■

Définition 2 *Si l'on note R l'ensemble des sommets d'un graphe sans circuit G de demi-degré intérieur nul, on appelle fonction rang associée à G l'application rang qui à tout sommet y associe un nombre entier défini de la façon suivante :*

- ➔ $rang(y) = 0 \ \forall y \in R$
- ➔ $rang(y)$ = *nombre d'arcs dans un chemin de cardinal maximal joignant un sommet quelconque de l'ensemble R et le sommet y,* $\forall y \notin R$.

La fonction rang est unique et elle définit une partition de l'ensemble des sommets en niveaux, chaque niveau k étant formé par le sous-ensemble de sommets :

$$X_k = \{x \in X \ : rang(x) = k\}$$

4.5. Fermeture transitive d'un graphe

Considérons un 1-graphe orienté connexe $G = (X, U)$. La **fermeture transitive** d'un graphe G est un graphe $F = (X, U_F)$ défini sur le même ensemble de sommets X et dont l'ensemble des arcs est défini par :

$$U_F = \{(x, y) \in X^2 \ : \exists \mu[x, y]\}$$

Le problème de la fermeture transitive d'un graphe a été beaucoup étudié. Il existe principalement deux types d'approches, la première se basant sur l'algorithme de Kosaraju-Sharir notamment et la seconde utilise les multiplications matricielles de la matrice d'adjacence du graphe.

4.5.1. Algorithme de parcours de graphes

Pour trouver la fermeture transitive d'un graphe G, on peut décomposer le problème en trois étapes :

1. Déterminer les composantes fortement connexes de G.
2. Déterminer la fermeture transitive du graphe réduit G_r qui est sans circuit.
3. Déduire la fermeture transitive de G de la fermeture transitive de G_r.

Remarque 2 *Si $G = (X, U)$ est fortement connexe, sa fermeture transitive est un 1-graphe plein sans boucle, c'est-à-dire $\forall (x, y) \in X^2$, $(x, y) \in U_F$ et $(y, x) \in U_F$.*

✓ **Étape 1** : Recherche des composantes fortement connexes de G par l'algorithme de Kosaraju-Sharir.

✓ **Étape 2** : Recherche de la fermeture transitive du graphe $G_r = (X_r, U_r)$ (sans circuit) d'ordre k. Il faut en premier lieu définir un ordre topologique sur les sommets de G_r. Les sommets sont alors numérotés de 1 à k selon cet ordre topologique. Pour tout sommet x de X_r, L^x désigne la liste des sommets pouvant être atteints par un chemin partant de x (liste des descendants). Pour chaque sommet x, la liste L^x est initialisée par l'ensemble de ses successeurs. Puis, pour chaque sommet y allant de k à 1 (ordre topologique inverse), on établit sa liste des descendants de la façon suivante :

$$L^y \leftarrow L^y \bigcup_{z \in \Gamma^+(y)} L^z$$

Comme les sommets sont examinés dans l'ordre topologique inverse, lorsque le sommet y est examiné, les listes définitives L^z de tous ses successeurs directs z ont déjà été constituées. La fermeture transitive $F_r = (X_r, U_{F_r})$ de G_r est constituée des arcs suivants : $\forall x \in X_r$, $\forall y \in L^x$, $(x, y) \in U_{F_r}$.

✓ **Étape 3** : Déduire la fermeture transitive de G de la fermeture transitive de G_r.

1. Pour chaque composante fortement connexe X_i de cardinal n_i, établir la liste des $n_i(n_i - 1)$ arcs du sous-graphe sans boucle définis sur X_i.
2. Si k et l sont deux sommets de G_r (correspondant aux composantes fortement connexes X_k et X_l de G) tels que l'arc (k, l) appartienne à la fermeture transitive de G_r, alors établir la liste des $n_k \times n_l$ arcs de la forme (i, j) avec $i \in X_k$ et $j \in X_l$.

4.5.2. Puissances de la matrice d'adjacence

Calcul de chemins Une **matrice d'adjacence** A est une matrice carrée d'ordre n telle que :

$$a_{ij} = \begin{array}{l} 1 \text{ si } (x_i, x_j) \in U \\ 0 \text{ sinon.} \end{array}$$

On définit également $\mathbf{A^2} = (a^2_{ij})$ par : $a^2_{ij} = \sum_{l=1}^{n} a_{il}.a_{lj}$
Ainsi, $a_{il}.a_{lj} = 1 \Leftrightarrow$ il existe un chemin de x_i à x_j de longueur 2 transitant par x_l. De ce fait, le terme a^2_{ij} représente le nombre de chemins de longueur 2 avec x_i pour extrémité initiale et x_j pour extrémité terminale.

De la même manière, on définit $\mathbf{A^k} = (a_{ij}^k)$ par : $a_{ij}^k = \sum_{l=1}^{n} a_{il}^{k-1}.a_{lj}$
$a_{il}^{k-1}.a_{lj}$ représente le nombre de chemins de x_i à x_j de longueur k transitant par x_l si $a_{lj} = 1$.
Ainsi, a_{ij}^k représente le nombre de chemins de longueur k avec x_i pour extrémité initiale et x_j pour extrémité terminale.

On notera que les boucles sont prises en compte. On peut donc repérer sur la diagonale les nombres de circuits de longueur k.

Existence de chemins $\mathbf{A^{[k]}} = (a_{ij}^{[k]})$ est une **matrice booléenne** dont chaque terme est défini par :

$$a_{ij}^{[k]} = \vee_{l=1}^{n} \left(a_{il}^{[k-1]} \wedge a_{lj} \right)$$

Ainsi, $a_{ij}^{[k]}$ représente l'existence de chemins de x_i à x_j de longueur k.

Soit :

$$\widehat{\mathbf{A}} = A \vee A^{[2]} \vee A^{[3]} \vee \ldots \vee A^{[n-1]}$$

$\widehat{A}$ représente la **fermeture transitive de** Γ

$$\forall x \in X, \quad \mathbf{\Gamma(x)} = \Gamma(x) \cup \Gamma^2(x) \cup \ldots \cup \Gamma^{n-1}(x)$$

Dans ce paragraphe, nous définissons la fermeture transitive d'une autre manière qu'au paragraphe 4.5.

$\forall x_i \in X$, à $\Gamma(x_i)$, on associe la ligne i de la matrice $\widehat{A}$ telle que :

$$\widehat{a}_{ij} = \begin{array}{l} 1 \text{ si } \exists\ \mu[x_i, x_j] \\ 0 \text{ sinon.} \end{array}$$

On notera que la construction de la composante fortement connexe associée à un sommet x donné peut être réalisée par le calcul de $\Gamma(x) \cap \Gamma^{-1}(x)$ (cet ensemble correspond aux sommets qui se situent sur un circuit contenant x).

```
Procédure Fermeture transitive(Γ, Γ)
    Γ ← Γ(x);
    D ← Γ; ensemble des nouveaux sommets atteints
    Si (x ∈ D) Alors
        D ← D − {x};
    Fin Si
    Tant que (D ≠ ∅) faire
        S ← ∪_{y∈D} Γ(y) ;
        D ← S − (S ∩ Γ) ;
        Γ ← Γ ∪ D ;
    Fait
Fin
```

Procédure Fermeture transitive(Γ^{-1}, Γ^{-1})
| { idem que pour Γ }
Fin

Procédure Composante fortement connexe()
| $FC(x) \leftarrow \{x\}$;
| **Si** ($\Gamma(x) \neq \emptyset$ et $\Gamma^{-1} \neq \emptyset$) **Alors**
| | *Fermeture transitive* (Γ, Γ) ;
| | *Fermeture transitive* $(\Gamma^{-1}, \Gamma^{-1})$;
| | $FC(x) \leftarrow \{x\} \cup (\Gamma \cap \Gamma^{-1})$;
| **Fin Si**
Fin

On associe à $\widehat{A}$ un graphe $\tau(G) = (X, \tau(U))$ de la fermeture transitive.

Deux graphes $G = (X, U)$ et $G' = (X, U')$ sont dit **τ-équivalent** s'ils possèdent la même fermeture transitive, c'est-à-dire $\tau(G) = \tau(G')$ ($\Leftrightarrow \tau(U) = \tau(U')$).

Un graphe $G = (X, U)$ est dit **τ-minimal**
si $\forall u \in U, G' = (X, U - \{u\})$ n'est pas τ-équivalent à G.
i.e. si $\forall u \in U, \tau(U - u) \subset \tau(U)$ (l'inclusion est stricte)
i.e. $\tau(G')$ est un graphe partiel de $\tau(G)$.

4.5.3. Algorithmes par multiplications matricielles

Pour trouver la fermeture transitive d'un graphe G d'ordre n, il suffit d'élever la matrice d'adjacence de G à la puissance $n - 1$.

Algorithme trivial en $O(n^4)$ L'algorithme trivial correspond au
produit de matrices en $O(n^3)$ avec $n - 2$ produits
addition de matrices en $O(n^2)$ avec $n - 2$ additions

Algorithme de Warshall en $O(n^3)$

$$A \to A^{(1)} \to A^{(2)} \to \ldots \to A^{(k)} \to \ldots \to A^{(n)}$$

Soit $k \in \{1, 2, \ldots, n\}$ donné, on pose $a_{ij}^{(k)} = 1$ s'il existe au moins un chemin de x_i à x_j transitant uniquement par des sommets de numéros appartenant à $\{1, 2, \ldots, k\}$:

ou bien
il existe un chemin de x_i à x_j transitant seulement par des sommets de numéros appartenant à

$\{1, 2, \ldots, k-1\}$ i.e. $a_{ij}^{(k-1)}$.

ou bien
il existe un chemin de x_i à x_k transitant seulement par des sommets de numéros appartenant à $\{1, 2, \ldots, k-1\}$ i.e. $a_{ik}^{(k-1)}$.
et, il existe un chemin de x_k à x_j transitant seulement par des sommets de numéros appartenant à $\{1, 2, \ldots, k-1\}$ i.e. $a_{kj}^{(k-1)}$.

$\Rightarrow a_{ij}^{(k)} = a_{ij}^{(k-1)} \vee \left(a_{ik}^{(k-1)} \wedge a_{kj}^{(k-1)}\right)$.
avec initialement, $a_{ij}^{(1)} = a_{ij} \vee (a_{i1} \wedge a_{1j})$

Pour k **de** 1 **à** n **faire**
  Pour i **de** 1 **à** n **faire**
    Pour j **de** 1 **à** n **faire**
      $a_{ij} \leftarrow a_{ij} \vee (a_{ik} \wedge a_{kj})$;
    Fin Pour
  Fin Pour
Fin Pour

La compléxité temporelle est en $O(n^3)$.

On notera qu'à chaque itération k, on fait un produit de la colonne k de A "transformée" par la ligne k de A "transformée" ; c'est une **matrice anti-scalaire** qui se calcule en $O(n^2)$.
i.e.
$A^{(1)} = A \vee \left(A^1 \wedge A_1\right)$
$A^{(2)} = A^{(1)} \vee \left(A^{(1)^2} \wedge A_2^{(1)}\right)$
...
$A^{(k)} = A^{(k-1)} \vee \left(A^{(k-1)^k} \wedge A_k^{(k-1)}\right)$

Chapitre 5. Problèmes de meilleurs chemins

Les problèmes de cheminement dans les graphes (en particulier la recherche d'un plus court chemin) comptent parmi les plus classiques de la théorie des graphes et les plus importants dans leurs applications. Le problème du **plus court chemin** (pcch) peut être posé de la façon suivante : étant donné un graphe $G = (X, U)$, on associe à chaque arc $u = (i, j)$ un nombre réel, noté $l(u)$ ou l_{ij}, appelé longueur de l'arc. Le problème du pcch entre deux sommets i_0 et j_0 du graphe consiste à déterminer, parmi tous les chemins allant de i_0 à j_0 celui, noté μ^* dont la longueur totale :

$$l(\mu^*) = \sum_{u \in \mu^*} l(u)$$

soit minimale.

Condition nécessaire Le problème du pcch a une solution si et seulement si il n'existe pas dans le graphe de circuit de longueur strictement négative pouvant être atteint à partir de i_0.

Si cette condition nécessaire est vérifiée, il existe toujours un chemin de longueur minimale qui soit élémentaire. En effet, lorsque tous les circuits du graphe pouvant être atteint à partir de i_0 ont une longueur strictement positive, tout pcch est nécessairement élementaire.

Lorsque l'on cherche un pcch entre deux sommets, on doit déterminer d'autres ppch entre i_0 et d'autres sommets du graphe. Aussi, les algorithmes existant se divisent en deux catégories : ceux dans lesquels on recherche un pcch d'un sommet spécifié i_0 à tous les autres sommets du graphe ; ceux qui procèdent directement de la recherche de tous les pcch entre i et j pour tous les couples (i, j) du graphe.

Il existe un grand nombre d'algorithmes permettant de déterminer un pcch d'un sommet particulier i_0 à tous les autres sommets du graphe. Les plus efficaces d'entre eux réalisent un marquage des sommets c'est-à-dire qu'à chaque sommet i est associé une marque $\lambda(i)$ représentant à la fin de l'algorithme la longueur du pcch allant de i_0 à i.

5.1. Plus courts chemins d'origine fixée dans un graphe avec longueurs non négatives : algorithme de Dijkstra

Soit $G = (X, U)$ un graphe dont les arcs sont munis de longueurs réelles positives ou nulles. On cherche les pcch de i_0 à tous les autres sommets du graphe.

L'algorithme de Moore-Dijkstra procède en $n - 1$ itérations. A l'initialisation, $\lambda(i_0) \leftarrow 0$ et $\lambda(i) \leftarrow \infty$ pour tout $i \neq i_0$. A une itération quelconque de l'algorithme, l'ensemble des sommets est partagé en deux sous-ensembles S et $X \backslash S$. Le sous-ensemble S contient l'ensemble des sommets définitivement marqués c'est-à-dire les sommets i pour lesquels la marque $\lambda(i)$ représente effectivement la longueur d'un pcch allant de i_0 à i (à l'initialisation $S \leftarrow \{i_0\}$). $X \backslash S$ contient les sommets i ayant une marque provisoire vérifiant :

$$\lambda(i) = min_{k \in S \cap \Gamma^-(i)} \{\lambda(k) + l_{ki}\}$$

L'algorithme est basé sur le lemme suivant.

Lemme : Si i est un sommet de $X \backslash S$ de marque provisoire $\lambda(i)$ minimale :

$$\lambda(i) = min_{j \in X \backslash S} \lambda(j)$$

alors $\lambda(i)$ est la longueur d'un pcch chemin allant de i_0 à i.

Ainsi à chaque itération, on sélectionne le sommet i de plus petite marque provisoire, on l'inclut dans l'ensemble S des sommets définitivement marqués et on remet à jour les marques de ces successeurs non définitivement marqués de la façon suivante :

$$\forall j \in \Gamma^+(i) \backslash S, \lambda(j) = min \{\lambda(j); \lambda(i) + l_{ij}\}$$

Lorsque tous les sommets du graphe sont dans l'ensemble S, les marques représentent les longueurs des pcch allant de i_0 à tous les autres sommets du graphe.

Comme à chaque itération, on attribue une marque définitive à un nouveau sommet, on détermine $n-1$ pcch entre i_0 et les autres sommets du graphe en au plus $n-1$ itérations.

$\lambda(i_0) \leftarrow 0$;
$\lambda(i) \leftarrow +\infty \ \forall i \in X \backslash \{i_0\}$;
$p(i) \leftarrow i \ \forall i \in X$;
$S \leftarrow \{i_0\}$;
$i \leftarrow i_0$;
Tant que $(S \neq X)$ **faire**
  Pour tout $j \in \Gamma^+(i) \backslash S$ **faire**
    Si $(\lambda(j) > \lambda(i) + l_{ij})$ **Alors**
      $\lambda(j) \leftarrow \lambda(i) + l_{ij}$;
      $p(j) \leftarrow i$;
    Fin Si
  Fin Pour
  Sélectionner $i \in X \backslash S$ tel que $\lambda(i) = min_{j \in X \backslash S} \lambda(j)$
  $S \leftarrow S \cup \{i\}$;
Fait

Preuve par récurrence de l'exactitude de l'algorithme :
Au rang 1 : $\lambda(i_0) = 0 = \lambda^*(i_0)$
Au rang 2 : Soit $i \in \Gamma^+(i_0)$ tel que $\lambda(i) = min_{j \in X \backslash S}\{\lambda(j)\}$
$\lambda(i) = l_{i_0 i} = \lambda^*(i)$ car tout autre chemin allant de i_0 à i comportera au moins deux arcs et sera donc de valeur minimale : $v = min_{j \in \Gamma^+(i_0) \backslash \{i\}}\{\lambda(j) + q\}$. Et, comme $q \geq 0, v \geq \lambda(i)$.
On suppose que la propriété est vraie au rang $k-1$.
Au rang k : Soit i tel que $\lambda(i) = min_{j \in X \backslash S}\{\lambda(j)\}$
Raisonnons par l'absurde et supposons qu'il existe un chemin de $\mu(i_0, i) = \{i_0, \ldots, s, r, \ldots, t, i\}$ de longueur $v(\mu(i_0, i))$ tel que $v(\mu(i_0, i)) < \lambda(i)$. Soit s le premier sommet appartenant à S rencontré en remontant le chemin μ de i vers i_0. Comme la propriété est supposée vraie au rang $k-1$, on a $\lambda(s) = \lambda^*(s)$ et :

$$\lambda^*(s) + l_{sr} \leq v(\mu(i_0, s)) + l_{sr} \leq v(\mu(i_0, s)) + l_{sr} + \ldots + l_{ti} = v(\mu(i_0, i))$$

car $\forall (i, j) \in U, l_{ij} \geq 0$. Par hypothèse, on a : $v(\mu(i_0, i)) < \lambda(i)$. Or, $\lambda(r) \leq \lambda^*(s) + l_{sr} < \lambda(i)$, c'est donc le sommet non marqué r qui aurait dû être sélectionné à l'itération k et non le sommet i. On aboutit donc à une contradiction.

Complexité : A chaque itération, on sélectionne le sommet j de plus petite marque en $O(n)$ opérations dans le pire cas, et on remet à jour les marques des successeurs de j en $O(d^+(j))$ opérations. En tout, il y a n itérations pour marquer tous les sommets du graphe. La complexité totale est donc en $O(n^2) + O(m) \approx O(n^2)$.

Remarquons que lorsque le graphe est peu dense, le terme $O(n^2)$ l'emporte sur $O(m)$ et l'opération la plus coûteuse consiste à recherche le sommet de plus petite marque. Pour diminuer la complexité de cette recherche, on peut utiliser une structure de données appelée tas.

5.2. Mise en oeuvre de l'algorithme de Dijkstra pour les graphes peu denses : algorithme de Johnson

5.2.1. Présentation de la structure de données de tas (heap)

Un **maximier** est un **arbre valué partiellement ordonné**, c'est-à-dire que la valuation d'un sommet est inférieure ou égale à celle de ses fils, **aussi équilibré que possible** (les seules feuilles manquantes doivent se trouver à droite des feuilles situées au plus bas niveau de l'arbre).

Le sommet de plus petite valuation est le sommet racine. On peut donc le trouver en $O(1)$ opération. Si on le supprime, on détruit la structure d'arbre. Pour recomposer un maximier, il suffit de prendre la feuille la plus à droite du niveau le plus bas et on la place temporairement à la racine. Puis, il faut pousser cet élément aussi bas que possible en l'échangeant avec celui de ses fils ayant la plus petite valuation inférieure. On s'arrête lorsque l'élément est, soit, devenu une feuille, soit, a ses fils de plus grande valuation. La suppression de la racine a une complexité en $0(log_2 n)$ car aucun chemin de l'arbre ne contient plus de $log_2 n$ arêtes.

Pour insérer un nouvel élément tout en préservant la structure du maximier, on réalise la procédure "inverse". On commence par placer l'élément en question aussi loin que possible à gauche au plus bas niveau de l'arbre. Puis, on le pousse aussi haut que possible dans l'arbre de la façon suivante : si son père à une valuation supérieure à la sienne, il faut les échanger, et on réitère les comparaisons jusqu'à ce que l'élément inséré se trouve à la racine, ou bien, ait une priorité inférieure à celle de son père. L'insertion d'un nouvel élément a également une compléxité en $O(log_2 n)$.

Le **tas** est une structure de données pouvant être utilisée pour représenter en machine un maximier. Soit un maximier contenant n sommets. Dans le tas, on utilise les n premières cellules d'un tableau unidimensionnel T de la façon suivante : les sommets du maximier remplissent les cellules $T[1], T[2], \ldots, T[n]$ niveau par niveau à partir du haut, et à l'intérieur d'un même niveau de la gauche vers la droite. Ainsi, le fils gauche, s'il existe, du sommet $T[i]$ se trouve en $T[2^i]$ et le fils droit, s'il existe, en $T[2^i + 1]$; le père de $T[i]$ se trouve en $T[\frac{i}{2}]$.

5.2.2. Algorithme de Johnson

Les sommets i de G et leur marque $\lambda(i)$ sont stockées dans un maximier ce qui permet de trouver le sommet de marque initiale en $O(1)$ opération.

Dans l'algorithme de Johnson, on fait appel aux fonctions suivantes :

- insere(i, T) : insère le sommet i dans le tas T
- min(T) : retourne et supprime le sommet racine (de plus petite valuation) de T et recompose la structure
- diminue$(valeur, i, T)$: modifie la valuation du sommet i dans T (elle passe à $valeur$) et recompose la structure

Le tas est constitué des sommets ayant des marques temporaires finies.

```
T ← Ø ;
λ(i0) ← 0 ; λ(i) ← +∞ ∀i ∈ X\{i0} ;
p(i) ← i ∀i ∈ X ;
insere(i, T) ;
Tant que (S ≠ X) faire
    i ← min(T) ;
    S ← S ∪ {i} ;
    Pour tout j ∈ Γ+(i)\S faire
        v = λ(i) + lij ;
        Si (λ(j) > v) Alors
            Si (λ(j) = +∞) Alors
                λ(j) ← v ;
                p(j) ← i ;
                insere(j, T) ;
            Sinon
                λ(j) ← v ;
                p(j) ← i ;
                diminue(v, j, T) ;
            Fin Si
        Fin Si
    Fin Pour
Fait
```

A chaque itération, on sélectionne le sommet i de plus petite marque et on recompose le tas : la complexité est en $O(1) + O(log_2 n)$. Puis, on remet à jour les marques de tous les successeurs du sommet i sélectionné en $O(d^+(i) \times log_2 n)$. Comme il y a n itérations, on a une complexité totale de $O(nlog_2 n) + O(mlog_2 n) \approx O(mlog_2 n)$.

5.3. Plus court chemin d'origine fixée dans un graphe sans circuit avec longueurs quelconques : algorithme de Bellman

Soit $G = (X, U)$ un graphe sans circuit dont les arcs sont munis de longueurs réelles quelconques. On cherche les pcch allant de i_0 à tous les autres sommets du graphe.

Condition d'optimalité Un ensemble de valeurs $\lambda^*(i)$ pour $i = 1, \ldots, n$, avec $\lambda^*(i_0) = 0$, représente les longueurs des pcch allant de i_0 aux autres sommets du graphe si et seulement si :

$$\forall (i,j) \in U \ : \ \lambda^*(j) \leq \lambda^*(i) + l_{ij}$$

Lorsque le graphe ne présente pas de circuit, il faut déterminer une numérotation des sommets allant de 1 à n qui constitue un ordre topologique (le sommet de départ i_0 ayant le numéro 1) et marquer les sommets dans cet ordre.

$\lambda(1) \leftarrow 0$;
$p(1) \leftarrow 1$;
Pour j **de** 2 **à** n **faire**
  $\lambda(j) = min_{i \in \Gamma^-(j)} \lambda(i) + l_{ij}$
  $p(j) = argmin_{i \in \Gamma^-(j)} \lambda(i) + l_{ij}$
Fin Pour

5.4. Détermination de plus courts chemins d'origine fixée dans un graphe avec longueurs quelconques

Soit $G = (X, U)$ un graphe dont les arcs sont munis de longueurs réelles quelconques. On cherche les pcch de i_0 à tous les autres sommets du graphe.

Remarque préliminaire Soit un graphe G présentant sur certains arcs une valuation négative et, envisageons de rendre les valuations toutes positives ou nulles en ajoutant à chaque valeur la valeur absolue maximale des valeurs négatives.

Ceci change la nature du problème puisque, comme on le voit dans l'exemple ci-dessus, le plus court chemin allant de 1 à 4 dans le graphe initial (de valeur 3) n'est plus optimal dans le graphe transformé.

5.4.1. Algorithme de Ford (1956)

Dans l'algorithme de Ford, les marques $\lambda(i)$ des sommets sont modifiées itérativement de façon à converger vers la condition d'optimalité présentée en section 5.3.

L'algorithme consiste alors, à partir d'un ensemble de valeurs provisoires, à parcourir séquentiellement la liste des arcs du graphe de façon à vérifier que la condition d'optimalité est satisfaite. Si cette condition est satisfaite pour tous les arcs (i, j), le problème est résolu ; sinon, il existe un arc (i, j) tel que : $\lambda(j) > \lambda(i) + l_{ij}$ et , dans ce cas, on remet à jour la marque de j de la façon suivante : $\lambda(j) \leftarrow \lambda(i) + l_{ij}$ (ce qui signifie que l'on améliore la marque provisoire de j en empruntant le chemin de longueur $\lambda(i)$ entre i_0 et i suivi de l'arc (i, j)).

$\lambda(i_0) \leftarrow 0$;
$\lambda(j) \leftarrow +\infty \ \forall i \in X/\{i_0\}$;
Tant que (il existe $(i,j) \in U$ tel que $\lambda(j) > \lambda(i) + l_{ij}$) **faire**
  $\lambda(j) \leftarrow \lambda(i) + l_{ij}$;
  $p_j \leftarrow i$
Fait

5.4.2. Algorithme de Ford-Bellman

Cet algorithme est une variante "optimisée" de l'algorithme de Ford. Pour parcourir la liste des arcs du graphe, on regarde pour chaque sommet i l'ensemble de ses prédécesseurs. Et, à une itération k donnée, on ne va pas s'intéresser à tous les sommets du graphe mais seulement à ceux dont la marque a été modifiée au cours de l'itération précédente. L'algorithme calcule donc à chaque itération k un ensemble de marques, notées λ_j^k pour tout $j \in X$. On note $M = \left\{ j \in X \mid \lambda_j^k < \lambda_j^{k-1} \right\}$, l'ensemble des sommets dont les marques ont été modifiées à l'itération k, et seuls les sommets appartenant à $\Gamma^+(M)$ peuvent voir leurs marques modifiées au cours de l'itération $k+1$. En fait les marques λ_j^k ont une interprétation très précise : c'est la valeur du meilleur chemin de i_0 à j ne contenant pas plus de k arcs. Ainsi, en l'absence de circuit absorbant dans le graphe, l'algorithme termine nécessairement à l'issue de l'itération n car tout chemin élémentaire a une longueur maximale égale à $n-1$. Si une ou plusieurs marques sont modifiées à l'itération n, cela signifie que le graphe présente un circuit de valeur négative (car il existe un pcch de longueur $> n-1$ qui emprunte donc nécessairement un circuit de valeur < 0 !). L'algorithme de Ford-Bellman peut donc s'écrire comme suit :

$k \leftarrow 0$;
$\lambda^0(i_0) \leftarrow 0$
$\lambda^0(i) \leftarrow +\infty \ \forall i \in X/\{i_0\}$;
$p(i) \leftarrow i_0 \ \forall i \in X/\{i_0\}$;
$M \leftarrow \{i_0\}$;
Tant que ($k \leq n-1$ et $M \neq \emptyset$) **faire**
  $k \leftarrow k+1$;
  $M' \leftarrow \emptyset$;
  Pour tout $j \in \Gamma^+(M)$ **faire**
    $\lambda^k(j) \leftarrow min\left\{\lambda^{k-1}(j); \lambda^{k-1}(i) + l_{ij}, i \in \Gamma^-(j) \cap M\right\}$;
    Si ($\lambda^k(j) < \lambda^{k-1}(j)$) **Alors**
      $M' \leftarrow M' \cup \{j\}$;
      $p(j) \leftarrow i^*$ avec $i^* \in \Gamma^-(j) \cap M$ tel que $\lambda^k(j) = \lambda^{k-1}(i^*) + l_{i^*j}$;
    Fin Si
  Fin Pour
  $M \leftarrow M'$;
Fait
Si ($\Gamma(M) \neq \emptyset$) **Alors**
  alors $\exists$ un circuit de valeur négative
Fin Si

Complexité A chaque itération, on remet à jour, dans le pire cas, les marques de tous les sommets en regardant l'ensemble des arêtes : $O(m)$ opérations. Le nombre maximal d'itérations étant n, la complexité totale est en $O(nm)$.

5.5. Plus courts chemins entre toutes les paires de sommets : algorithme de Floyd

Soit la matrice $A = \{a_{ij}\}$ de taille $n \times n$ avec :

$$a_{ij} = \begin{cases} 0 & \text{si } i = j \\ l_{ij} & \text{si } (i,j) \in U \\ +\infty & \text{sinon} \end{cases}$$

L'algorithme de Floyd permet de calculer les pcch entre tous les couples de sommets de la façon suivante. A la première itération, on cherche le pcch entre chaque couple (i, j) passant éventuellement par le sommet 1 ; à l'itération l (avec $l > 1$), on cherche le pcch entre chaque couple (i, j) passant par des sommets d'indice inférieur ou égal à l. Une description formelle de l'algorithme est donnée ci-dessous.

Pour l **de** 1 **à** n **faire**
  Pour i **de** 1 **à** n **faire**
    Pour j **de** 1 **à** n **faire**
      $a_{ij} = min\{a_{ij}, a_{il} + a_{lj}\}$;
    Fin Pour
  Fin Pour
Fin Pour

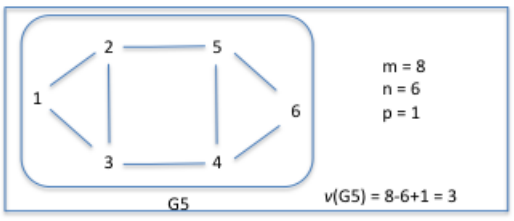
La complexité totale est en $O(n^3)$.

Chapitre 6. Arbres couvrants

6.1. Arbre et arborescence

Nombre cyclomatique Soit G un graphe de n sommets, m arêtes et p composantes connexes. Le nombre cyclomatique est noté $\nu(G)$:

$$\nu(G) = m - n + p$$

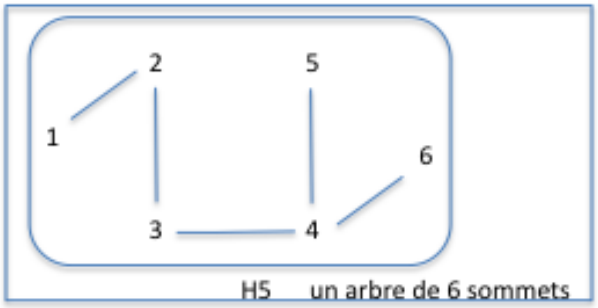


$\nu(G)$ représente le nombre d'éléments d'une base de cycles.

arbre Les 6 définitions suivantes sont équivalentes :
Soit $H = (X, U)$ un graphe de n sommets ($n \geq 2$).

(1) H connexe et sans cycle
(2) H sans cycle et $n-1$ arêtes
(3) H connexe et $n-1$ arêtes
(4) H sans cycle et en ajoutant une arête, on crée un et un seul cycle
(5) H connexe et si on supprime une arête, il n'est plus connexe
(6) $\exists$une chaîne et une seule entre toute paire de sommets

Preuve:
1 $\Rightarrow$ **2** H sans cycle $\Rightarrow \nu(H) = 0$ et H connexe $\Rightarrow p = 1 \Rightarrow m - n + 1 = 0 \Rightarrow m = n - 1$.
2 $\Rightarrow$ **3** H sans cycle $\Rightarrow \nu(H) = 0$, et $m = n - 1 \Rightarrow (n-1) - n + p = 0 \Rightarrow p = 1 \Rightarrow H$ connexe.
3 $\Rightarrow$ **4** H connexe $\Rightarrow p = 1$, et $m = n - 1 \Rightarrow \nu(H) = (n-1) - n + 1 = 0 \Rightarrow H$ est sans cycle. Si on ajoute une arête, on obtient H' t.q. $m' = m + 1 = n, n' = n$ et $p' = 1 \Rightarrow \nu(H') = n - n + 1 = 1 \Rightarrow 1$ cycle.
4 $\Rightarrow$ **5** Si H n'est pas connexe, $\exists\{x, y\}$ non reliés par une chaîne $\Rightarrow$ en ajoutant l'arête $[x, y]$ on ne crée pas de cycle et (4) n'est pas vérifié $\Rightarrow H$ est connexe. Si on ôte une arête, on obtient H" t.q. m" $= n-2$ et n" $= n \Rightarrow \nu(H") = m" - n" + p" = 0$ (car sans cycle) $\Rightarrow n - 2 - n + p" = 0 \Rightarrow p" = 2 \Rightarrow H$" est non connexe.
5 $\Rightarrow$ **6** H connexe $\Rightarrow \exists$ au moins une chaîne entre 2 sommets ; la suppression d'une arête rend H non connexe $\Rightarrow$ cette chaîne est unique.
6 $\Rightarrow$ **1** $\exists$ une chaîne entre 2 sommets $\Rightarrow H$ est connexe ; elle est unique $\Rightarrow$ pas de cycle. ■



H5 un arbre de 6 sommets

graphes orientés

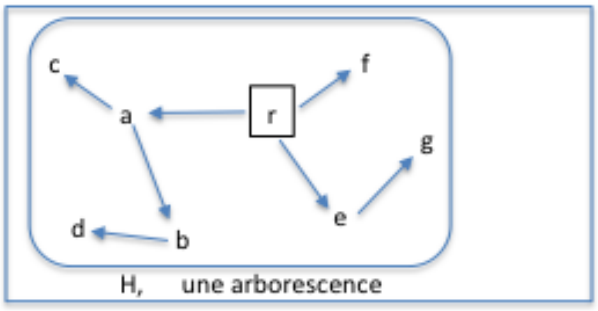
racine : une racine est un sommet r tel qu'il existe un chemin de r à tout autre sommet du graphe

degré intérieur (resp. **extérieur**) : le degré intérieur (resp. extérieur) d'un sommet x représente le nombre d'arcs d'extrémité terminale (resp. initiale) x, notés $d^-(x)$ et $d^+(x)$.

arborescence Les 4 définitions suivantes sont équivalentes :
Soit $H = (X, U)$ un graphe de n sommets ($n \geq 2$).

(1) H arbre avec une racine r
(2) $\exists r \in X$, relié à tout $x \in X$ par un chemin unique
(3) H connexe et $\exists r \in X$ t.q. $d^-(x) = 1$ pour tout $x \neq r$
(4) H sans cycle et $\exists$ un sommet $r \in X$ t.q. $d^-(r) = 0$ et $d^-(x) = 1$ pour tout $x \neq r$.

On notera qu'une arborescence = "arbre enraciné" (rooted tree) = "arbre" en informatique.
Ex : arbre généalogique, tournois, arbre des espèces animales, ...



6.2. Arbre couvrant de poids minimal d'un graphe simple

6.2.1. Formulation du problème

Soit $G = (X, U)$ un graphe connexe simple de $n =\mid X \mid$ sommets et $m =\mid E \mid$ arêtes.

G est **valué** si à tout $(x_i, x_j) \in U$ est associée une valeur. On note $v_{ij} \in \mathbb{R}^+$ le poids de l'arête.

arbre couvrant Un arbre couvrant $T = (X, U_T)$ est un graphe partiel de G qui a la propriété d'un arbre (i.e. sans cycles). On a donc $\mid U_T \mid = n - 1$.

poids de T

$$v(T) = \sum_{(x_i,x_j)\in U_T} v_{ij}$$

But : Etant donné

$$\mathbf{T} = \{T \mid T \text{ arbre couvrant de } G\}$$

trouver

$$T^* \in \mathbf{T} \text{ tel que } v(T^*) = min_{t\in\mathbf{T}} v(T)$$

T^* est appelé **arbre couvrant de poids minimal (a.c.p.m.)** de G.
On note que $\mid U_T \mid = n_1 \Rightarrow$ on peut toujours retrouver l'hypothèse $v_{ij} \in R^+ \forall (x_i, x_j) \in U$.

Théorème 2 *Un arbre couvrant de poids minimal (a.c.p.m.) de G contient au moins une arête de poids minimal de tout cocycle de G*

Preuve:
Soit T^* un a.c.p.m. ;
Raisonnons par l'absurde : on suppose qu'il existe $A \subset X$ (strictement) tel que T^* ne contienne aucune des arêtes de poids minimal de $\omega(A)$.
Soit (x_k, x_l) une arête de poids minimal de $\omega(A)$ avec $x_k \in A$ et $x_l \in X - A$.
Par hypothèse, $(x_k, x_l) \notin U_{T^*}$.
Connexité $\Rightarrow U_{T^*} \cap \omega(A) \neq \emptyset$
et par hypothèse, $\forall (x_i, x_j) \in U_{T^*} \cap \omega(A), \quad v_{ij} > v_{kl}$.
L'ajout de (x_k, x_l) crée un cycle contenant une arête $(x_i, x_j) \in U_{T^*} \cap \omega(A)$
L'échange des arêtes (x_i, x_j) et (x_k, x_l)

- maintient la connexion de A et de $X - A$, et
- permet de construire un nouvel arbre couvrant de poids inférieur à $v(T^*)$ d'une quantité égale à $v_{ij} - v_{kl}$ (strictement positive),

ce qui est impossible car T^* est un a.c.p.m. ■

Théorème 3 *Un a.c.p.m. de G ne contient pas toutes les arêtes de poids maximal de tout cycle de G.*

Preuve: Soit T^* un a.c.p.m.
Raisonnons par l'absurde : on suppose qu'il existe un cycle $C = (X_C, U_C)$ de G tel que T^* contient toutes les arêtes de poids maximal de C.
Soit $(x_i, x_j) \in U_C \cap U_{T^*}$ une des arêtes de poids maximal.
C est un cycle $\Rightarrow (U - U_{T^*}) \cap U_C \neq \emptyset$
et toutes les arêtes de C ne peuvent être de même poids, car T^* ne serait pas un arbre
$\Rightarrow \forall (x_k, x_l) \in (U - U_{T^*}) \cap U_C$ on a $v_{kl} < v_{ij}$.
La suppression de (x_i, x_j) induit une déconnexion de x_i et x_j ; la connexion est rétablie par l'ajout d'une arête $(x_k, x_l) \in (U - U_{T^*}) \cap U_C$
Cet échange aboutit à un nouvel arbre couvrant de poids inférieur à $v(T^*)$ de la quantité $v_{ij} - v_{kl} (> 0)$.
Ceci est impossible car T^* est un a.c.p.m. ■

6.2.2. Algorithme de Prim-Dijsktra

Paramètres (significations en cours d'algorithme)
$M \subset X$: ensemble des sommets marqués (i.e. inclus dans T^*).
λ_j : distance de x_j au sous-arbre de T^* déjà construit $\forall x_j \in X - M$ i.e.

$$\lambda_j = min_{x_i \in M : (x_i, x_j) \in U} v_{ij}$$

Algorithme

sélectionner $x_k \in X$; $M \leftarrow \{x_k\}$
$U_{T^*} \leftarrow \emptyset$
Pour j **de** 1 **à** n et $(j \neq k)$ **faire**
  $\lambda_j \leftarrow +\infty$;
Fin Pour
Tant que $(M \neq X)$ **faire**
  { mise à jour des distances }
  Pour tout $x_j \in X - M$ tel que $(x_k, x_j) \in U$ **faire**
    Si $(\lambda_j > v_{kj})$ **Alors**
      $\lambda_j \leftarrow v_{kj}$;
      $p_j \leftarrow k$
    Fin Si
  Fin Pour
  { marquage d'un nouveau sommet }
  déterminer k et $x_k \in X - M$ tel que $\lambda_k = min_{x_j \in X - M} \lambda_j$;
  $M \leftarrow M \cup \{x_k\}$;
  $U_{T^*} \leftarrow U_{T^*} \cup \{(x_{pk}, x_k\}$;
Fait

Preuve: A chaque itération, l'algorithme sélectionne une arête de poids minimal du cocyle engendré par l'ensemble M des sommets marqués (cf. théorème 2). ■

complexité
identique à celle de l'algorithme de Dijkstra.

6.2.3. Algorithme de Kruskal

S contient toutes les arêtes considérées en cours d'algorithme i.e. celles de U_{T^*} et celles éliminées par le théorème 3.

déterminer (k, l) tel que $v_{kl} = min_{(x_i,x_j)\in U} v_{ij}$;
$S \leftarrow \{(x_k, x_l)\}$;
$U_{T^*} \leftarrow \{(x_k, x_l)\}$ { itération 1 } ;
Pour *iter* **de** 2 **à** $n-1$ **faire**
  fin $\leftarrow$ faux ; { permet de gérer l'élimination des arêtes par le théorème 3} ;
  Tant que (non fin) **faire**
    déterminer (k, l) tel que $v_{kl} = min_{(x_i,x_k)\in U-S} v_{ij}$
    $S \leftarrow S \cup \{(x_k, x_l)\}$;
    Si ($U_{T^*} \cup \{(x_k, x_l)\}$ ne forme pas un cycle de G) **Alors**
      $U_{T^*} \leftarrow U_{T^*} \cup \{(x_k, x_l)\}$;
      fin $\leftarrow$ vrai ;
    Fin Si
  Fait
Fin Pour

Preuve: Le but est de retenir $n-1$ arêtes (arbre) d'où le nombre d'itérations.
Les itérations sont envisagées dans l'ordre croissant de leurs poids ; toute arête candidate à être incluse dans U_{T^*}, qui engendre un cycle avec tout ou partie des arêtes déjà retenues, est une arête de poids maximal de ce cycle.
Elle doit donc être rejetée (cf. théorème 3). ■

Chapitre 7. Flots dans les graphes

Le problème des flots dans les graphes concerne la circulation de matière sur les arcs. Parmi les nombreuses applications qui relèvent de ce problème, on trouve le transport de marchandises entre différents points, les télécommunications dans les réseaux, ...

7.1. Définitions

Un **réseau de transport** $R = (N, A)$ est un 1-graphe connexe sans boucle tel que :
- il existe une **source** (il s'agit d'un sommet racine) notée s,
- il existe un **puits** (il s'agit d'un sommet antiracine) notée t,
- chaque arc $u = (i, j) \in A$ est muni d'une **capacité** c_u entière positive ou nulle (et éventuellement un coût d_u entier positif ou nul).

Un **flot** $\phi = (\phi_u)$ dans $R = (N, A)$ est un vecteur de R^m (dont l'élément générique ϕ_u est appelé **flux** sur l'arc $u = (i, j)$) tel que :
(i) $\forall u = 1, \ldots, m, \phi_u \geq 0$
(ii) En tout sommet $i \in N \backslash \{s, t\}$, la première loi de Kirchhoff est vérifiée (cette loi est également connue sous le nom de loi de conservation aux noeuds) :

$$\sum_{u \in \omega^+(i)} \phi_u = \sum_{u \in \omega^-(i)} \phi_u$$

Un flot compatible ϕ dans un réseau de transport $R = (N, A)$ est un flot tel que :

$$\forall u \in A, \phi(u) \leq c_u$$

La **valeur** f_ϕ d'un flot **compatible** ϕ dans un réseau de transport s'énonce comme suit : déterminer quelle est la valeur maximale du flot qu'il est possible de faire parvenir de la source s au puits t. On supposera par la suite qu'il n'existe pas de chemin de capacité infinie entre s et t dans R car, si un tel chemin existe, la solution est triviale.

Le problème de flot maximal de s à t peut se formuler de la façon suivante :

$$\begin{cases} max\ f \\ \sum_{u \in \omega^+(i)} \phi(u) - \sum_{u \in \omega^-(i)} \phi(u) = 0 \quad \forall i \in N \backslash \{s, t\} \\ \sum_{u \in \omega^+(s)} \phi(u) - \sum_{u \in \omega^-(s)} \phi(u) = f \\ \sum_{u \in \omega^+(t)} \phi(u) - \sum_{u \in \omega^-(t)} \phi(u) = -f \\ 0 \leq \phi_u \leq c_u \quad \forall u \in A \end{cases}$$

7.2. Propriétés fondamentales

7.2.1. Flot maximal et coupe minimale

Dans un réseau de transport $R = (N, A)$, soit S un sous-ensemble de N et $\bar{S}$ son complément par rapport à N. On dit que S et $\bar{S}$ forment une bipartition de N et on note $(S, \bar{S})$ l'ensemble des arcs de A ayant leur extrémité initiale dans S et leur extrémité terminale dans $\bar{S}$.

Soit S et $\bar{S}$ une bipartition de N telle que s appartient à S et t appartient à $\bar{S}$. Une **coupe** $s - t$, notée $C(S, \bar{S})$ est l'ensemble des arcs appartenant à $(S, \bar{S})$ et à $(\bar{S}, S)$.
La **capacité d'une coupe**, notée $v(S, \bar{S})$, est donnée par la somme des capacités des arcs de l'ensemble $(S, \bar{S})$:

$$v(S, \bar{S}) = \sum_{u \in C(S, \bar{S})} c_u$$

Théorème 4 *La valeur d'un flot ϕ dans R est inférieure ou égale à la capacité de toute coupe $s - t$.*

Preuve: Soit S et $\bar{S}$ une bipartition de N. On somme sur S les équations de conservation de flot :

$$\begin{array}{ll} \sum_{u \in \omega^+(i)} \phi_u - \sum_{u \in \omega^-(i)} \phi_u & = 0 \quad \forall i \in S \backslash \{s\} \\ \sum_{u \in \omega^+(s)} \phi_u - \sum_{u \in \omega^-(s)} \phi_u & = f \end{array}$$

On obtient :
$\sum_{i \in S} \left(\sum_{u \in \omega^+(i)} \phi_u - \sum_{u \in \omega^-(i)} \phi_u \right) = f$

$\sum_{i\in S}\left(\sum_{u=(i,j):j\in S}\phi_u+\sum_{u=(i,j):j\in\bar{S}}\phi_u-\sum_{u=(j,i):j\in S}\phi_u-\sum_{u=(j,i):j\in\bar{S}}\phi_u\right)=f$
$\sum_{u=(i,j):i\in S,j\in S}\phi_u+\sum_{u=(i,j):i\in S,j\in\bar{S}}\phi_u-\sum_{u=(j,i):i\in S,j\in S}\phi_u-\sum_{u=(j,i):i\in S,j\in\bar{S}}\phi_u=f$

En substituant $\phi_u \leq c_u$ et $\phi_u \geq 0$ on obtient :

$$\sum_{u=(i,j):i\in S,j\in\bar{S}} c_u \geq f$$

$$v(S,\bar{S}) \geq f$$

On a donc :

$$\min_{C(S,\bar{S}} v(S,\bar{S}) \geq f$$

■

En conséquence, si on trouve un flot f égal à la capacité d'une coupe $C(S,\bar{S})$, ce flot est maximal et cette coupe est minimale. Le théorème Flot Max - Coupe Min ci-dessous stipule qu'il existe un flot dont la valeur est égale à la capacité d'une certaine coupe.

Théorème 5 (Ford-Fulkerson (max-flow min-cut)) *La valeur d'un flot maximum ϕ dans R est égale à la plus petite des capacités des coupes $s-t$.*

7.2.2. Graphe d'écart et chemin augmentant

Soit ϕ un flot admissible sur R. Le **graphe d'écart** associé à ϕ sur R est le graphe $R^e(\phi) = [N, A^e(\phi)]$ défini comme suit : $\forall u = (i,j) \in A$
- Si $\phi(u) < c_u, (i,j) \in A^e(\phi)$ avec la capacité (résiduelle) $c'_{(i,j)} = c_u - \phi_u$
- Si $\phi(u) > 0, (j,i) \in A^e(\phi)$ avec la capacité $c'_{(j,i)} = \phi_u$

Dans le premier cas, un arc est valué par sa capacité résiduelle, c'est-à-dire l'augmentation de flux possible. Dans le second cas, l'idée est de diminuer le flux sur l'arc (i,j) en faisant passer une quantité de flux sur l'arc (j,i).

Soit ϕ un flot admissible sur R et $R^e(\phi) = [N, A^e(\phi)]$ le graphe d'écart associé. Soit μ un chemin allant de s à t dans $R^e(\phi)$ et $\delta = min_{u\in\mu}c'u$. Ce chemin est appelé **chemin augmentant** car il est possible d'augmenter la valeur du flot sur R de δ de la façon suivante $\forall(i,j) \in \mu$:
- si $u = (i,j) \in A$, alors $\phi_u \leftarrow \phi_u + \delta$
- si $u = (j,i) \in A$, alors $\phi_u \leftarrow \phi_u - \delta$

Théorème 6 *Un flot ϕ admissible sur R est maximum si et seulement si il n'existe pas de chemin augmentant allant de s à t dans $R^e(\phi) = [N, A^e(\phi)]$*

7.3. Recherche d'un flot maximal : algorithme de Ford-Fulkerson

7.3.1. Algorithme générique

Soit un réseau R, un flot compatible ϕ dans R et le graphe d'écart $R^e(\phi)$. Pour déterminer un flot maximum, l'algorithme générique consiste, à chaque itération, à chercher un chemin μ allant de s à t dans $R^e(\phi)$. Si un tel chemin existe, on augmente le flot ϕ de la quantité $\delta = min_{u\in\mu}c'_u$. Sinon l'algorithme termine et ϕ est le flot de valeur maximale.

$\phi \leftarrow 0$
Tant que ($R^e(\phi)$ contient un chemin de s à t) **faire**
  Identifier un chemin μ de s à t
  $\delta = min_{u \in \mu} c'_u$
  Augmenter de δ unités le flot ϕ sur R
  Mettre à jour $R^e(\phi)$
Fait

Cet algorithme ne précise pas de quelle façon déterminer un chemin μ de s à t. Dans la section suivante, nous présentons un algorithme de marquage qui, sans passer par le graphe d'écart, permet d'exhiber un chemin augmentant dans $R^e(\phi)$ (ou de façon équivalente une chaîne augmentante dans R) en travaillant directement sur R.

7.3.2. Algorithme de marquage de Ford-Fulkerson

$marque(t) \leftarrow 1$
Tant que ($marque(t) \neq 0$) **faire**
  $marque(i) \leftarrow 0 \quad \forall i \in N$
  $pred(i) \leftarrow i \quad \forall i \in N$
  $marque(s) \leftarrow +\infty$ et $LISTE \leftarrow \{s\}$
  Tant que ($LISTE \neq \emptyset$ et $marque(t) = 0$) **faire**
    Sélectionner $i \in LISTE$ et faire $LISTE \leftarrow LISTE \backslash \{i\}$
    Pour chaque $u = (i, j) \in \omega^+(i)$ **faire**
      Si ($marque(j) = 0$ et $c_u > \phi_u$) **Alors**
        $marque(j) \leftarrow c_u - \phi_u$
        $pred(j) \leftarrow i$
        $LISTE \leftarrow LISTE \cup \{j\}$
      Fin Si
    Fin Pour
    Pour chaque $u = (j, i) \in \omega^-(i)$ **faire**
      Si ($marque(j) = 0$ et $\phi_u > 0$) **Alors**
        $marque(j) \leftarrow -\phi_u$ (marquage de type -)
        $pred(j) \leftarrow i$
        $LISTE \leftarrow LISTE \cup \{j\}$
      Fin Si
    Fin Pour
  Fait
  Si $marque(t) \neq 0$ alors **augmenter**
Fait

augmenter
Identifier la chaîne μ à l'aide de $pred(i)$: $\mu = (s = i_0, i_1, \ldots, i_k = t)$
$\delta \leftarrow min_{i \in \mu} |marque(i)|$
$\phi_{s,i_0} \leftarrow \phi_{s,i_0} + \delta$
Pour j **de** 1 **à** k **faire**
  Si $(marque(i_j) < 0)$ **Alors**
    $\phi_{i_{j-1},i_j} \leftarrow \phi_{i_{j-1},i_j} - \delta$
  Sinon
    $\phi_{i_{j-1},i_j} \leftarrow \phi_{i_{j-1},i_j} + \delta$
  Fin Si
Fin Pour
$\phi_{i_k,t} \leftarrow \phi_{i_k,t} + \delta$

Exactitude de l'algorithme de marquage :
A chaque itération, soit on trouve une chaîne augmentante, soit on ne parvient pas à marquer t et l'algorithme se termine. Dans ce dernier cas, il nous faut montrer que le flot ϕ, de valeur f_ϕ, obtenu à la dernière itération est bien le flot de valeur maximale.
Soit S l'ensemble des sommets marqués à la dernière itération (avec $s \in S$) et $\bar{S}$ l'ensemble des sommets non marqués (avec $t \in \bar{S}$ et $\bar{S} = N \backslash S$). Comme il s'agit de la dernière itération, il n'est plus possible de marquer un sommet de $\bar{S}$ à partir d'un sommet de S. En conséquence :

$$\forall i \in S, \forall j \in \bar{S} : u = (i,j) \in A, c_u = \phi_u \quad (1)$$

$$\forall i \in S, \forall j \in \bar{S} : u = (j,i) \in A, \phi_u = 0 \quad (2)$$

Or, si on somme sur S les équations de conservation de flot, on obtient (cf. preuve du théorème flot max - coupe min) :

$$\sum_{u \in \{S,\bar{S}\}} \phi_u - \sum_{u \in \{\bar{S},S\}} \phi_u = f_\phi$$

et (1) et (2) impliquent :

$$\sum_{u \in \{S,\bar{S}\}} c_u = f_\phi = v(S,\bar{S})$$

Le flot ϕ de valeur f_ϕ est donc égal à la capacité d'une coupe $v(S,\bar{S})$. ϕ est donc un flot de valeur maximale et $C(S,\bar{S})$ une coupe de valeur minimale.

A la dernière itération de l'algorithme de Ford et Fulkerson, on a déterminé un flot maximum, mais également une coupe minimum. Cette coupe est engendrée par la bipartition S, l'ensemble constitué des sommets marqués et, $\bar{S}$, l'ensemble des sommets non marqués à la dernière itération.

Théorème 7 *Si toutes les capacités sont entières, le problème du flot maximum a une solution optimale entière.*

Preuve: Le flot de départ est le flot nul. Comme toutes les capacités sont entières, la chaîne augmentante trouvée, si elle existe, dispose d'une capacité δ entière (car toutes les capacités résiduelles sont entières). Le nouveau flot est donc à coordonnées entières. Et ainsi de suite jusqu'à la dernière itération. Au pire cas, on augmente à chaque itération la valeur du flot d'une unité ($\delta = 1$) et, comme le flot maximum ne peut pas dépasser la valeur entière d'une coupe quelconque, l'algorithme termine en un nombre fini d'itérations.
■

La complexité de l'algorithme est maintenant évidente à calculer. Chaque itération comporte $O(m)$ opérations élémentaires car la méthode de marquage examine chaque arc et chaque sommet au plus une fois. En conséquence, la complexité totale est $O(m)$ fois le nombre d'augmentation. Si chaque capacité est entière et bornée par U, la capacité d'une coupe $(s-t)$ est au plus nU. Et la complexité totale est en $O(nmU)$. En conséquence, si par exemple $U = 2^n$, la complexité de l'algorithme est en $O(nm2^n)$, c'est-à-dire exponentielle sur le nombre de noeud du réseau.

7.3.3. Une application du problème de flot maximal : le couplage dans un graphe biparti

Un **couplage** dans un graphe $G = (X, U)$ est un sous-graphe de G où tous les sommets sont de degré 0 ou 1 (les arêtes n'ont pas d'extrémités communes). La cardinalité d'un couplage est égale à la cardinalité de l'ensemble U.

Un couplage est dit **parfait** lorsque tous les sommets sont de degré 1.

Un graphe est dit **biparti** si l'ensemble des sommets peut être partitionné en deux sous-ensembles X_1 et X_2 de façon à ce que chaque arête ait une extrémité dans X_1 et l'autre dans X_2.

Exemple : Le problème d'affectation
Lorsqu'on cherche à affecter des personnes à des tâches (une personne par tâche), on cherche un couplage dans un graphe simple où les sommets représentent les individus et les tâches, et les arêtes représentent les affectations (individu, tâches) possibles. Ce graphe, dit d'affectation, est un graphe biparti. On peut rechercher un couplage de cardinalité maximale lorsque l'on cherche à couvrir le plus de tâches possibles. On peut aussi rechercher un couplage maximal de poids minimal si le graphe d'affectation est valué par des coûts d'affectation.

Pour trouver un couplage maximal dans un graphe biparti il suffit de déterminer un flot de valeur maximale dans le graphe biparti transformé comme suit : on ajoute une source reliée à tous les sommets de X_1 et un puits relié à tous les sommets de X_2, ces nouveaux arcs ayant une capacité 1 (la capacité des arcs du graphe biparti peut valoir 1 ou $+\infty$).

7.4. Recherche d'un flot maximal à coût minimal

Soit f^* la valeur du flot maximal sur R. Le problème du flot maximal à coût minimal dans R peut s'écrire comme suit :

$$\begin{cases} min \sum_{u \in A} d_u \phi_u \\ \sum_{u \in \omega^+(i)} \phi(u) - \sum_{u \in \omega^-(i)} \phi(u) = 0 \quad \forall i \in N \backslash \{s,t\} \\ \sum_{u \in \omega^+(s)} \phi(u) - \sum_{u \in \omega^-(s)} \phi(u) = f^* \\ \sum_{u \in \omega^+(t)} \phi(u) - \sum_{u \in \omega^-(t)} \phi(u) = -f^* \\ 0 \leq \phi_u \leq c_u \quad \forall u \in A \end{cases}$$

On retrouve aussi la notion de coût dans le graphe d'écart. Soit ϕ un flot admissible sur R. Le graphe d'écart associé à ϕ sur R est le graphe $R^e(\phi) = [N, A^e(\phi)]$ défini comme suit : $\forall u = (i,j) \in A$

- Si $\phi(u) < c_u, (i,j) \in A^e(\phi)$ avec la capacité (résiduelle) $c'_{(i,j)} = c_u - \phi_u$ et le coût $d'_{ij} = d_u$
- Si $\phi(u) > 0, (j,i) \in A^e(\phi)$ avec la capacité $c'_{(j,i)} = \phi_u$ et le coût $d'_{ji} = -d_u$.

7.4.1. Condition d'optimalité

Théorème 8 *Une solution ϕ^* est une solution optimale du problème du flot maximal à coût minimal si et seulement si le graphe d'écart associé $R^e(\phi^*)$ ne comporte pas de circuit de longueur négative.*

En effet, s'il comporte un circuit de longueur négative cela signifie qu'il est possible de passer du flot ϕ^* à un nouveau flot de coût inférieur.

7.4.2. Algorithmes de détermination du flot maximal à coût minimal

Algorithme de Bennington De la condition d'optimalité présentée dans la section précédente, il est possible de définir un algorithme très simple de détermination de flot maximal à coût minimal comme suit :

Etablir un flot de valeur maximale
Tant que (le graphe réduit associé contient des circuits de valeur négative) **faire**
  Choisir un circuit de valeur négative avec δ la capacité résiduelle minimale du circuit
  Augmenter de δ unités le flot dans le circuit et remettre à jour le graphe réduit associé
Fait

Preuve: L'algorithme termine car il n'existe qu'un nombre fini de circuits élémentaires possibles et, à chaque étape, on diminue le coût du flot d'une valeur au moins égale au plus petit des coûts unitaires de ces circuits. Les flots obtenus à chaque étape sont des flots maximaux. Le graphe d'écart obtenu à la fin de l'algorithme ne comporte pas de circuit de coût négatif, par conséquent le flot obtenu est un flot de coût minimal. ■

La complexité de cet algorithme est exponentielle (nécessité de calculer les circuits élémentaires du graphe d'écart).

Algorithme de Roy Cet algorithme consiste à choisir, parmi les chemins de la source au puits, celui dont le coût unitaire (somme des coûts des arcs) est minimal et à augmenter le flot sur ce chemin. On répète le processus jusqu'à l'obtention d'un flot maximal.

Preuve: L'algorithme termine puisqu'à chaque étape on augmente le flot jusqu'à ce que l'on ait atteint un flot maximal. C'est la méthode de Ford-Fulkerson dans laquelle on force l'ordre des optimisations. Il est ensuite facile de montrer par récurrence que l'on obtient un flot de coût minimal parmi les flots de même valeur. ■

La complexité de l'algorithme de Roy est également exponentielle car on doit calculer des chemins acycliques (avec des coûts négatifs sur certains arcs) dans le graphe d'écart, ce qui donne, dans le pire des cas, une complexité de l'ordre de $|N|!$

Chapitre 8. Ordonnancement et graphes

Définition 3 *Un problème d'ordonnancement existe :*

- *quand un ensemble de travaux est à réaliser,*
- *que cette réalisation est décomposable en tâches,*
- *que le problème consiste à définir la localisation temporelle des tâches et/ou la manière de leur affecter les moyens nécessaires.*

Définition 4 *Les contraintes et les critères du problème ne doivent concerner que les tâches, leur localisation temporelle et les moyens nécessaires à leur réalisation.*

Un problème d'ordonnancement de projets a deux caractéristiques principales :
- la limitation plus ou moins grande des ressources (petite quantité ou illimitée),
- la dépendance plus ou moins grande entre les travaux (contraintes de précédence).

8.1. Méthode des potentiels

Soit un problème d'ordonnancement de projet décomposé en n tâches, chaque tâche $i = 1, \ldots, n$ ayant une durée d'exécution d_i.

La méthode des potentiels va permettre de déterminer un calendrier des n tâches qui composent le projet à réaliser. Ce calendrier dépend des durées des tâches et des contraintes d'antériorité à respecter pour commencer l'élaboration de chaque tâche.

Hypothèse : les tâches s'effectuent sans interruption. Si la tâche i de durée d_i débute à la date t_i, elle sera terminée à la date $t_i + d_i$.

On suppose qu'un calendrier est la donnée d'un ensemble des dates de début de chaque tâche du projet ainsi que des tâches "début" et "fin". $C = \{t_{debut}, t_{tache1}, \ldots, t_{tachen}, t_{fin}\}$.

Le graphe "potentiels-tâches" $G = (X, U)$ est un graphe sans circuit dont les sommets de X représentent l'ensemble des n tâches du projet plus deux tâches de durée nulle : "début" et "fin" et dont les arcs sont associés aux contraintes de potentiels (contraintes de précédence et de localisation temporelle) et sont valués : v_{ij} est le temps minimal qui doit s'écouler entre la date de début t_i de la tâche i et la date de début t_j de la tâche j.

8.1.1. Calendrier au plus tôt

Le calendrier $C^* = \{t^*_{debut}, t^*_{tache1}, \ldots, t^*_{tachen}, t^*_{fin}\}$ indique les dates de début au plus tôt de chacune des tâches. La date de début au plus tôt d'une tâche j, notée t^*_j, est la date avant laquelle il est impossible de débuter la tâche j compte tenues des contraintes à respecter. Le calendrier C^*

s'obtient en calculant les chemins de valeur maximale allant de la tâche "début" à la tâche "fin" en passant par les tâches i, $i = 1, \ldots, n$. Le graphe étant sans circuit, cela ne pose pas de difficulté.

$t^*_{debut} \leftarrow 0$
M : Ensemble des tâches marquées
Tant que $(M \neq X)$ **faire**
  //Sélectionner une tâche j non marquée dont tous les prédécesseurs sont marqués
  Sélectionner $j \in X \backslash M$ tel que $\Gamma^-(j) \subset M$
  $t^*_j \leftarrow \max_{i \in \Gamma^-(j)} \{t^*_i + v_{ij}\}$
  $M \leftarrow M \cup \{j\}$
Fait

Définition 5 *Le chemin critique est le chemin de valeur maximale allant de "début" à "fin". Il conditionne la durée minimale du projet.*

Définition 6 *Les tâches du chemin critique sont appelées tâches critiques car le moindre retard de démarrage de l'une d'entre elles retarde la date de fin de projet au plus tôt.*

8.1.2. Calendrier au plus tard

Pour une date de fin fixée, la date au plus tard t'_i d'une tâche i est la date après laquelle on ne peut débuter la tâche i compte tenu du respect des contraintes et de cette date de fin. La date au plus tard t'_i est calculée en retranchant à la date de fin fixée la valeur du chemin le plus long allant du sommet i au sommet "fin".

$t^*_{fin} \leftarrow$ date de fin fixée
M : Ensemble des tâches marquées
$M \leftarrow \{fin\}$
Tant que $(M \neq X)$ **faire**
  //Sélectionner une tâche i non marquée dont tous les successeurs sont marqués
  Sélectionner $i \in X \backslash M$ tel que $\Gamma^+(i) \subset M$
  $t'_i \leftarrow \min_{j \in \Gamma^+(i)} \{t^*_j - v_{ij}\}$
  $M \leftarrow M \cup \{i\}$
Fait

Définition 7 *La marge d'une tâche i, notée m_i est le retard maximal que l'on peut prendre sur la date de début au plus tôt de i sans affecter la date de fin de projet fixée. $m_i = t'_i - t^*_i$*

Références bibliographiques :

R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows : Theory, Algorithms, and Applications , *Prentice Hall*, 1993.
C. Berge, Graphes et Hypergraphes, *Dunod*, 1970.
M. Gondran, M. Minoux, Graphes et Algorithmes, *Eyrolles*, 1995.