Cours de

GRAPHES

Pierre Lopez

LAAS–CNRS

30 novembre 2005

http://www.laas.fr/~lopez/cours/GRAPHES/graphes.html

Résumé

Ce cours constitue une introduction à la théorie des graphes.
Le contenu est composé de quatre chapitres. Le premier « Eléments de théorie des graphes » présente les concepts généraux. Le deuxième « Le problème du plus court chemin » aborde certainement l'un des plus fameux sujets de la théorie des graphes, en présentant les principaux algorithmes de recherche de chemins de longueur minimale dans un graphe. Le troisième, « Chemins, parcours hamiltoniens, arbres », propose des méthodes générales concernant l'énumération et l'existence de chemins, de circuits hamiltoniens et d'arbres à coût minimum. Le quatrième « Flots dans les réseaux » parle plus particulièrement des réseaux de transport et introduit les méthodes fondamentales de recherche d'un flot maximum.

Avertissement : De manière générale en théorie des graphes, il faut se méfier de l'apparente facilité de certaines notions, définitions ou concepts. Dans la réalité, un graphe n'est pas forcément un dessin sur lequel vous pourrez déduire empiriquement les résultats attendus. Il peut être donné par une liste de milliers d'arcs, stockée dans un fichier. Aussi, veillez à appliquer les méthodes qui vous seront proposées pour atteindre les objectifs, même si cela vous semble trivial à l'œil nu...

Table des matières

Chapitre 1

Eléments de théorie des graphes

La théorie des graphes est née en 1736 quand Euler démontra qu'il était impossible de traverser chacun des sept ponts de la ville russe de Königsberg (aujourd'hui Kaliningrad) une fois exactement et de revenir au point de départ. Les ponts enjambent les bras de la Pregel qui coulent de part et d'autre de l'île de Kneiphof. Dans la figure suivante, les nœuds représentent les rives.

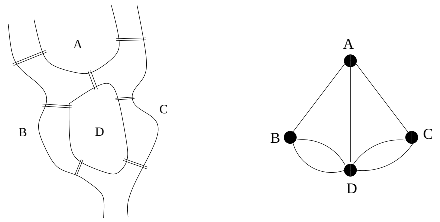


Fig. 1.1: Les sept ponts de Königsberg

La théorie des graphes constitue un domaine des mathématiques qui, historiquement, s'est aussi développé au sein de disciplines diverses telles que la chimie (modélisation de structures), la biologie (génome), les sciences sociales (modélisation des relations) ou en vue d'applications industrielles (problème du voyageur de commerce). Elle constitue l'un des instruments les plus courants et les plus efficaces pour résoudre des problèmes discrets posés en Recherche Opérationnelle (RO).
De manière générale, un graphe permet de représenter simplement la structure, les connexions, les cheminements possibles d'un ensemble complexe comprenant un grand nombre de situations, en exprimant les relations, les dépendances entre ses éléments (*e.g.*, réseau de communication, réseaux ferroviaire ou routier, arbre généalogique, diagramme de succession de tâches en gestion de projet, ...).
En plus de son existence purement mathématique, le graphe est aussi une structure de données puissante pour l'informatique.

1.1 Définition et concepts de base

1.1.1 Concepts orientés

Dans beaucoup d'applications, les relations entre éléments d'un ensemble sont orientées, *i.e.*, un élément x peut être en relation avec un autre y sans que y soit nécessairement en relation avec x. On parle alors de graphe orienté (en Anglais *directed graph* ou plus simplement *digraph*).
Un *graphe* $G = (X, U)$ est déterminé par :
– un ensemble $X = \{x_1, x_2, \ldots, x_n\}$ dont les éléments sont appelés *sommets* ou *nœuds* (ce dernier terme est plutôt dans le contexte des réseaux) ;
– un ensemble $U = \{u_1, u_2, \ldots, u_m\}$ du produit cartésien $X \times X$ dont les éléments sont appelés *arcs*.
Pour un arc $u = (x_i, x_j)$, x_i est l'*extrémité initiale*, x_j l'*extrémité finale* (ou bien *origine* et *destination*). L'arc u *part* de x_i et *arrive* à x_j.
Graphiquement, l'arc u se représente de la manière suivante (*diagramme sagittal*) :

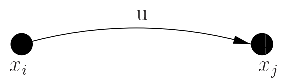


Fig. 1.2: Arc $u = (x_i, x_j)$

Un arc (x_i, x_i) est appelé une *boucle*.

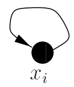


Fig. 1.3: Boucle

Un *p-graphe* est un graphe dans lequel il n'existe jamais plus de p arcs de la forme (i, j) entre deux sommets quelconques.

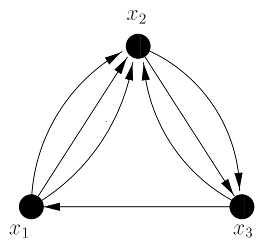


Fig. 1.4: 3-graphe

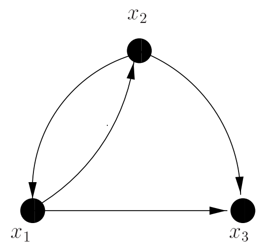


Fig. 1.5: 1-graphe ≡ graphe

La densité d'un 1-graphe est donnée par le quotient m/n^2, rapport du nombre effectif d'arcs sur le nombre maximal théorique.

Applications multivoques

x_j est *successeur* de x_i si $(x_i, x_j) \in U$; l'ensemble des successeurs de x_i est noté $\Gamma(x_i)$. x_j est *prédécesseur* de x_i si $(x_j, x_i) \in U$; l'ensemble des prédécesseurs de x_i est noté $\Gamma^{-1}(x_i)$.
L'application Γ qui, à tout élément de X, fait correspondre une partie de X est appelée une *application multivoque*.
Pour un 1-graphe, G peut être parfaitement déterminé par (X, Γ), notation à la base d'une représentation informatique très utilisée, les *listes d'adjacence* (cf. 1.2.3).

Exemple : $X = \{x_1, x_2, x_3, x_4, x_5\}$; $\Gamma(x_1) = \{x_2\}$; $\Gamma(x_2) = \{x_1, x_3\}$; $\Gamma(x_3) = \{x_4, x_5\}$; $\Gamma(x_4) = \{x_5\}$; $\Gamma(x_5) = \{x_1, x_5\}$.

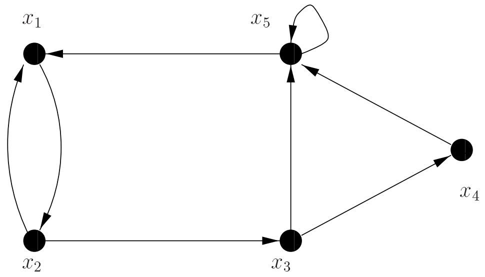


FIG. 1.6: Graphe déterminé par (X, Γ)

Remarque : on peut également définir une fonction ω telle que $\omega^+(x_i)$ représente l'ensemble des arcs sortant de x_i et réciproquement pour $\omega^-(x_i)$ et les arcs entrants.

1.1.2 Concepts non orientés

Lors de l'étude de certaines propriétés, il arrive que l'orientation des arcs ne joue aucun rôle. On s'intéresse simplement à l'existence d'arc(s) entre deux sommets (sans en préciser l'ordre). Un arc sans orientation est appelé *arête*. U est constitué non pas de couples, mais de paires de sommets non ordonnés. Pour une arête (x_i, x_j), on dit que u est *incidente* aux sommets x_i et x_j.
Remarque : Dans le cas non-orienté, au lieu de noter $G = (X, U)$ et $u = (x_i, x_j)$, on préfère souvent $G = (X, E)$ et $e = [x_i, x_j]$.
Un *multigraphe* $G = (X, E)$ est un graphe pour lequel il peut exister plusieurs arêtes entre deux sommets.
Un graphe $G = (X, E)$ est *simple* :

1. s'il n'est pas un multigraphe ;
2. s'il n'existe pas de boucles.

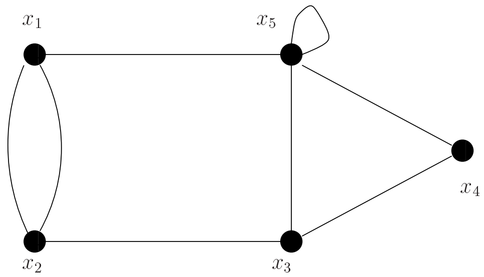


FIG. 1.7: Graphe non orienté associé au graphe orienté de la Fig. 1.6

1.1.3 Principales définitions

- *Adjacence*
  - Deux sommets sont adjacents (ou voisins) s'ils sont joints par un arc.
  - Deux arcs sont adjacents s'ils ont au moins une extrémité commune.
- *Degrés*
  - Le *demi-degré extérieur* de x_i, $d^+(x_i)$, est le nombre d'arcs ayant x_i comme extrémité initiale ; $d^+(x_i) = |\omega^+(x_i)|$.
  - Le *demi-degré intérieur* de x_i, $d^-(x_i)$, est le nombre d'arcs ayant x_i comme extrémité finale ; $d^-(x_i) = |\omega^-(x_i)|$.
  - Le *degré* de x_i est $d(x_i) = d^+(x_i) + d^-(x_i)$. Le degré d'un sommet d'un graphe non orienté est le nombre d'arêtes qui lui sont incidentes.

  Remarque : dans le cas d'un 1-graphe, on peut tout aussi bien définir le degré d'un sommet à l'aide de l'application multivoque Γ puisque $|\omega^+(x_i)| = |\Gamma(i)|$ et $|\omega^-(x_i)| = |\Gamma^{-1}(i)|$.

  Une boucle augmente de deux unités le degré du sommet concerné.

  Exemple : [cf. Fig. 1.6]
  $d^+(x_2) = 2;\ d^-(x_2) = 1;\ d(x_2) = 3.$
  $d^+(x_5) = 2;\ d^-(x_5) = 3;\ d(x_5) = 5.$
- *Graphe complémentaire*
  $G = (X, U)$ et $\overline{G} = (X, \overline{U})$. $(x_i, x_j) \in U \Rightarrow (x_i, x_j) \notin \overline{U}$ et $(x_i, x_j) \notin U \Rightarrow (x_i, x_j) \in \overline{U}$. $\overline{G}$ est le *graphe complémentaire* de G.
- *Graphe partiel*
  $G = (X, U)$ et $U_p \subset U$. $G_p = (X, U_p)$ est un *graphe partiel* de G. (On peut ainsi obtenir des sommets isolés...)
- *Sous-graphe*
  $G = (X, U)$ et $X_s \subset X$. $G_s = (X_s, V)$ est un *sous-graphe* de G, où V est la restriction de la fonction caractéristique de U à X_s. $V = \{(x, y)/(x, y) \in U \cap X_s \times X_s\}$. $\forall x_i \in X_s,\ \ \Gamma_s(x_i) = \Gamma(x_i) \wedge X_s$.
- *Sous-graphe partiel*
  Combine les deux définitions précédentes.

Exemple : *Réseau routier*
- *que les autoroutes : graphe partiel*
- *que la région Midi-Pyrénées : sous-graphe*
- *que les autoroutes de Midi-Pyrénées : sous-graphe partiel*

- Graphe *réflexif* : $\forall x_i \in X,\ \ (x_i, x_i) \in U$.
- Graphe *irréflexif* : $\forall x_i \in X,\ \ (x_i, x_i) \notin U$.
- Graphe *symétrique* : $\forall x_i, x_j \in X,\ \ (x_i, x_j) \in U \Rightarrow (x_j, x_i) \in U$.
- Graphe *asymétrique* : $\forall x_i, x_j \in X,\ \ (x_i, x_j) \in U \Rightarrow (x_j, x_i) \notin U$ (si G est asymétrique, G est irréflexif).
- Graphe *antisymétrique* : $\forall x_i, x_j \in X,\ \ (x_i, x_j) \in U$ et $(x_j, x_i) \in U \Rightarrow x_i = x_j$ (si G est asymétrique, G est aussi antisymétrique).
- Graphe *transitif* : $\forall x_i, x_j \in X,\ \ (x_i, x_j) \in U, (x_j, x_k) \in U \Rightarrow (x_i, x_k) \in U$.
- Graphe *complet* : $\forall x_i, x_j \in X,\ \ (x_i, x_j) \notin U \Rightarrow (x_j, x_i) \in U$.
- *Clique* : ensemble des sommets d'un sous-graphe complet. Soit $\mathcal{C} \subset X$ une clique de G non orienté : $\forall (x_i, x_j) \in \mathcal{C}, (x_i, x_j) \in U$ (2 sommets distincts de G sont toujours adjacents). Notons qu'un graphe complet et antisymétrique s'appelle un « tournoi », car il symbolise le résultat d'un tournoi où chaque joueur est opposé une fois à chacun des autres joueurs.

Exemple :

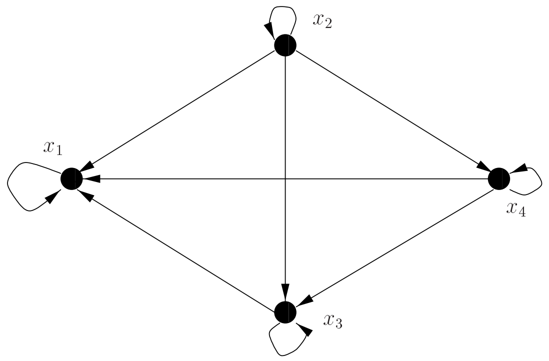


FIG. 1.8: Graphe réflexif, antisymétrique, transitif et complet

1.1.4 Notion de complexité des algorithmes

Les problèmes de graphe se rattachent à la grande classe des problèmes d'optimisation combinatoire. Tous ces problèmes se répartissent en deux catégories : ceux qui sont résolus optimalement par des algorithmes efficaces (rapides), et ceux dont la résolution peut prendre un temps exponentiel sur les grands cas. On parle respectivement d'algorithmes *polynomiaux* et *exponentiels*.

Pour évaluer et classer les divers algorithmes disponibles pour un problème de graphe, il nous faut utiliser une mesure de performance indépendante du langage et de l'ordinateur utilisés. Ceci est obtenu par la notion de *complexité* d'un algorithme, qui consiste à mettre en évidence les possibilités et les limites théoriques du processus calculatoire, en évaluant le nombre d'opérations caractéristiques de l'algorithme dans le pire des cas. Elle est notée O (*e.g.*, $O(n^2)$ pour une fonction qui augmente dans le carré de la taille des données). On rencontre aussi la notation Θ (*e.g.*, $\Theta(n^2)$) qui donne une borne asymptotique par excès et par défaut (alors que O ne donne que la borne asymptotique par excès).

Fin 1ère séance

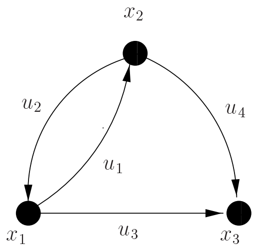
1.2 Représentations d'un graphe

Un certain nombre de représentations existent pour décrire un graphe. En particulier, elles ne sont pas équivalentes du point de vue de l'efficacité des algorithmes. On distingue principalement la représentation par matrice d'adjacence, par matrice d'incidence sommets-arcs (ou sommets-arêtes dans le cas non orienté) et par listes d'adjacence.

1.2.1 Matrice d'adjacence

On considère un 1-graphe. La matrice d'adjacence fait correspondre les sommets origine des arcs (placés en ligne dans la matrice) aux sommets destination (placés en colonne). Dans le formalisme *matrice booléenne*, l'existence d'un arc (x_i, x_j) se traduit par la présence d'un 1 à l'intersection de la ligne x_i et de la colonne x_j ; l'absence d'arc par la présence d'un 0 (dans un formalisme dit *matrice aux arcs* les éléments représentent le nom de l'arc).

Exemple :



	x_1	x_2	x_3	← destination
x_1	0	1	1	
x_2	1	0	1	
x_3	0	0	0	
↑				
origine				

FIG. 1.9: 1-graphe

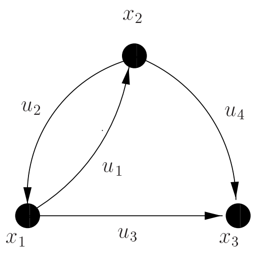
Place mémoire utilisée : n^2 pour un graphe d'*ordre* $|X| = n$ (*i.e.*, n sommets).

Remarque : pour des graphes *numérisés* (ou *valués* ou *pondérés*), remplacer « 1 » par la valeur numérique de l'arc.

1.2.2 Matrice d'incidence sommets-arcs

ligne $\leftrightarrow$ sommet
colonne $\leftrightarrow$ arc
Si $u = (i, j) \in U$, on trouve dans la colonne u : $a_{iu} = 1$: $a_{ju} = -1$; tous les autres termes sont nuls.

Exemple :



	u_1	u_2	u_3	u_4
x_1	1	-1	1	0
x_2	-1	1	0	1
x_3	0	0	-1	-1

FIG. 1.10: 1-graphe

Place mémoire utilisée : $n \times m$.

Remarques : la somme de chaque colonne est égale à 0 (un arc a une origine et une destination) ; la matrice est totalement *unimodulaire*, *i.e.,* toutes les sous-matrices carrées – extraites de la matrice – ont pour déterminant $+1, -1$ ou 0.

1.2.3 Listes d'adjacence

Pour un 1-graphe, l'avantage de la représentation par listes d'adjacence (grâce à l'application multivoque Γ), par rapport à celle par matrice d'adjacence, est le gain obtenu en place mémoire ; ce type de représentation est donc mieux adapté pour une implémentation. Le but est de représenter chaque arc par son extrémité finale, son extrémité initiale étant définie implicitement. Tous les arcs émanant d'un même sommet sont liés entre eux dans une liste. A chaque arc sont donc associés le nœud destination et le pointeur au prochain sommet dans la liste.
On crée deux tableaux LP (tête de listes) de dimension $(n + 1)$ et LS (liste de successeurs) de dimension m (cas orienté) ou $2m$ (cas non orienté). Pour tout i, la liste des successeurs de i est dans le tableau LS à partir de la case numéro $LP(i)$.

1. On construit LS par $\Gamma(1), \Gamma(2), \ldots, \Gamma(n)$.
2. On construit LP qui donne pour tout sommet l'indice dans LS où commencent ses successeurs.
3. Pour un sommet $i \rightarrow$ 1er suivant : $LS(LP(i))$; 2ème suivant : $LS(LP(i) + 1)$. L'ensemble des informations relatives au sommet i est contenue entre les cases $LP(i)$ et $LP(i + 1) - 1$ du tableau LS.

4. Si un sommet i n'a pas de successeur, on pose $LP(i) = LP(i + 1)$ (liste vide coincée entre les successeurs de $i - 1$ et ceux de $i + 1$).
   Pour éviter des tests pour le cas particulier $i = n$ (le sommet $i + 1$ n'existant pas), on « ferme » par convention la dernière liste en posant $LP(n+1) = m+1$.
   Exemple :

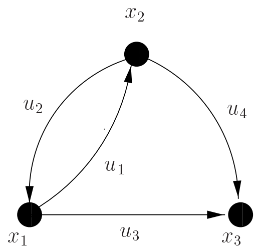


FIG. 1.11: 1-graphe

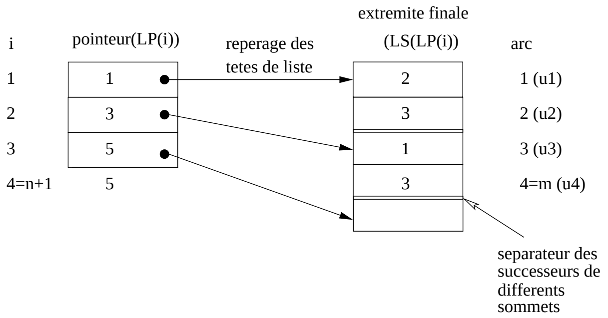


FIG. 1.12: Listes d'adjacence

Place mémoire utilisée : $(n + 1) + m$.

1.3 Coloration des sommets d'un graphe

Définition 1 *Soit $G = (X, U)$ un graphe non orienté. Un sous-ensemble $S \subset X$ est un ensemble* stable *(ou plus simplement* un stable*) s'il ne comprend que des sommets non adjacents deux à deux : $\forall\, i, j \in S \Rightarrow (i, j) \notin U$.*

Comme tout sous-ensemble d'un ensemble stable est un ensemble stable, il est naturel de chercher le cardinal maximum d'un ensemble stable (un *stable maximal*). Ce nombre, noté $\alpha(G)$, est le *nombre de stabilité*.

Définition 2 *La* coloration *des sommets d'un graphe consiste en une affectation de couleurs à tous les sommets du graphe de telle sorte que deux sommets adjacents ne soient pas porteurs de la même couleur.*

Exemple :

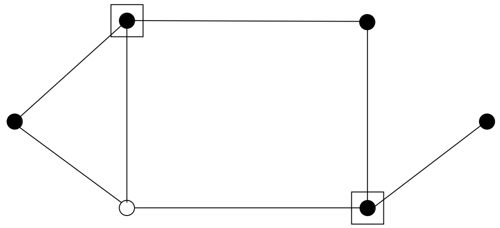


FIG. 1.13: Coloration en 3 couleurs

Le plus petit k pour lequel G est k-colorable est le *nombre chromatique* de G noté $\gamma(G)$; le nombre chromatique est donc défini comme le nombre minimum de couleurs distinctes nécessaires à la coloration des sommets de G.
Une *k-coloration* des sommets est une partition $(S_1, S_2, \ldots, S_k)$ de l'ensemble des sommets en k ensembles stables.
Borne inférieure du nombre chromatique :

$$\alpha(G).\gamma(G) \geq n(G) \Rightarrow \gamma(G) \geq \lceil \frac{n(G)}{\alpha(G)} \rceil$$

$n(G)$ étant le nombre de sommets du graphe.

Une borne supérieure est donnée par $deg+1$ avec deg plus grand degré d'un sommet, d'où :

$$\lceil \frac{n(G)}{\alpha(G)} \rceil \leq \gamma(G) \leq deg + 1$$

Applications (voir [Prins] p.321) :

1. problèmes d'emploi du temps ;
2. coloriage d'une carte géographique (10 sommets et 17 arcs).

$$7 \geq \gamma(G) \geq \lceil \frac{10}{4} \rceil = 3$$

On peut également se pencher sur le problème de la coloration des arêtes d'un graphe $G = (X, E)$. On a alors la définition de l'indice chromatique, duale de celle du nombre chromatique pour le cas de la coloration des sommets d'un graphe.
L'*indice chromatique* $q(G)$ est défini comme le nombre minimum de couleurs distinctes nécessaires à la coloration des arêtes de G.

Fin 2ème séance

1.4 Connexité dans les graphes

1.4.1 Chaîne – Cycle

Une *chaîne* est une séquence d'arcs telle que chaque arc ait une extrémité commune avec le suivant. Un *cycle* est une chaîne qui contient au moins une arête, telle que toutes les arêtes de la séquence sont différentes et dont les extrémités coïncident.

Exemple :

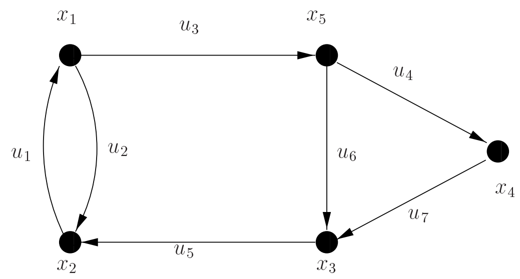


FIG. 1.14: $< u_2, u_5, u_6, u_4 >$ est une chaîne de x_1 à x_4. $< u_4, u_7, u_6 >$ est un cycle.

1.4.2 Chemin – Circuit

Ce sont les mêmes définitions que les précédentes mais en considérant des concepts orientés.

Exemple : [cf. Fig. 1.14]

$< u_1, u_3, u_4, u_7 >$ *est un chemin de* x_2 *à* x_3.
$< u_1, u_3, u_6, u_5 >$ *est un circuit.*

Le sous-ensemble de sommets atteignables à partir d'un sommet donné, grâce à des chemins, est appelé *fermeture transitive* de ce sommet.
Le terme de *parcours* regroupe les chemins, les chaînes, les circuits et les cycles. Un parcours est :
– *élémentaire* : si tous les sommets qui le composent sont tous distincts ;
– *simple* : si tous les arcs qui le composent sont tous distincts ;
– *hamiltonien* : passe une fois et une seule par chaque sommet du graphe ;
– *eulérien* : passe une fois et une seule par chaque arc du graphe[1] ;
– *préhamiltonien* : passe au moins une fois par chaque sommet du graphe ;
– *préeulérien* ou *chinois* : passe au moins une fois par chaque arc du graphe.
Remarque : le *problème du voyageur de commerce* est voisin du problème hamiltonien. Il consiste à trouver un circuit hamiltonien de coût minimal dans un graphe valué.

[1] Théorème d'Euler : un graphe connexe est eulérien si et seulement si tous ses sommets sont de degré pair

1.4.3 Connexité

Un graphe $G = (X, U)$ est *connexe* si $\forall\, i, j \in X$, il existe une chaîne entre i et j.
On appelle *composante connexe* le sous-ensemble de sommets tels qu'il existe une chaîne entre deux sommets quelconques.
Un graphe est connexe s'il comporte une composante connexe maximale et une seule.
Chaque composante connexe est un graphe connexe.

Exemple :

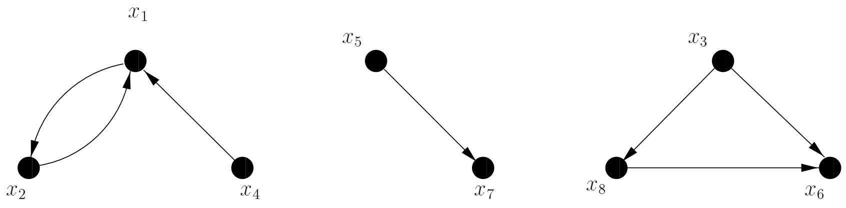


FIG. 1.15: Graphe ayant trois composantes connexes

Application : vérification de la connexité aux réseaux téléphoniques ou électriques.

1.4.4 Forte connexité

Un graphe $G = (X, U)$ est *fortement connexe* si $\forall\, i, j \in X$, il existe un chemin entre i et j.
Une *composante fortement connexe (cfc)* est un sous-ensemble de sommets tel qu'il existe un chemin entre deux sommets quelconques. Une *cfc maximale (cfcm)* est un ensemble maximal de cfc. Les différentes cfcm définissent une partition de X.
Un graphe est fortement connexe s'il comporte une seule cfcm.

Recherche de cfcm

Algorithme 1
– Répéter

1. Partir d'un sommet quelconque n'appartenant pas à une cfcm ; le marquer $\pm$.
2. Sur l'ensemble des sommets non marqués $\pm$ et tant qu'on peut marquer un sommet faire :
   (a) Marquer $+$ tout sommet suivant d'un sommet marqué $+$.
   (b) Marquer $-$ tout sommet précédent d'un sommet marqué $-$.

   Tout sommet marqué $\pm$ appartient à une cfcm.

– Jusqu'à ce que tout sommet appartienne à une cfcm.
Complexité : $O(nm)$.

Exemple :

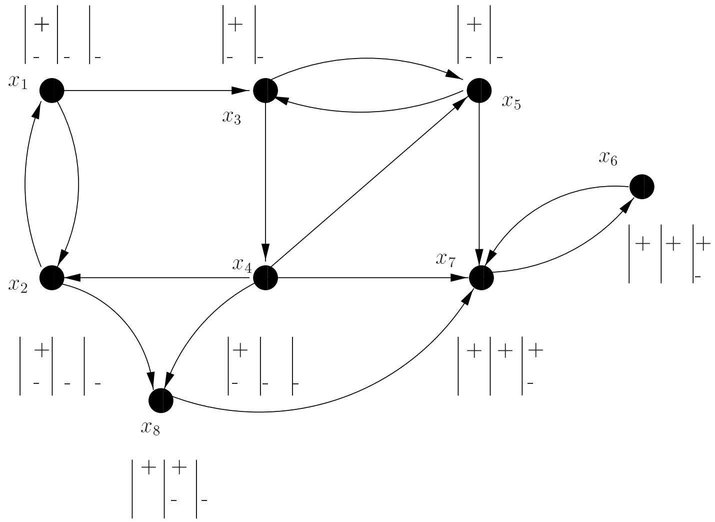


FIG. 1.16: Trois cfcm : $\{x_1, x_2, x_3, x_4, x_5\}$; $\{x_8\}$; $\{x_6, x_7\}$

Algorithme 2 A partir de la matrice des fermetures transitives (cf. Chap.3 §3.1.2).

Remarque : Il existe des algorithmes pour trouver l'ensemble des cfc (algorithme de Tarjan en $O(m)$ par exemple). Dans l'exemple de la figure 1.16, un tel algorithme trouverait quatre cfc associées à la cfcm $\{x_1, x_2, x_3, x_4, x_5\}$: $\{x_1, x_2, x_3, x_4\}$; $\{x_1, x_2\}$; $\{x_3, x_5\}$; $\{x_3, x_4, x_5\}$.

1.5 Graphes particuliers

1.5.1 Graphes sans circuit

On les rencontre lors de la représentation d'une relation d'ordre sur des éléments :
– à partir de certains sommets, on ne peut atteindre qu'un sous-ensemble de sommets ;
– aucun des sommets de ce sous-ensemble ne peut atteindre les sommets de départ. Les différents sous-ensembles constituent des *niveaux* ordonnés par un *rang* (rang = distance maximale d'un nœud à la racine). Il n'existe pas de chemin allant d'un sommet à un autre situé à un même niveau ou dans un niveau de rang inférieur.

Algorithme 1
A chaque itération, on recherche tous les sommets qui n'ont pas de précédent. Ils constituent un niveau. On efface ces sommets avant de rechercher le niveau suivant. AJOUTER et PRELEVER signifient respectivement insérer un élément en queue de

file et supprimer un élément en tête de file, la file étant définie par un accès FIFO (First In First Out).

Décomposition-En-Niveaux-GSC

```
M,S : files de sommets
k ← 0 ; M ← ∅
pour i=1 à n
  si d⁻(x_i) = 0
    AJOUTER(x_i,M)
tant que M ≠ ∅
  S ← M ; M ← ∅
  tant que S ≠ ∅
     PRELEVER(x,S)
     pour tout y dans Γ(x)
        d⁻(y) ← d⁻(y)-1
        si d⁻(y)=0
          AJOUTER(y,M)
     rang(x) ← k
   k ← k+1
```

Initialement, M contient les sommets sans prédécesseur. Puis, on la bascule dans S et on remplit de nouveau M.

Remarque : Cet algorithme peut être utilisé pour dédtecter si un graphe possède un circuit. En effet, si c'est le cas, on arrive à une étape de l'algorithme où aucun sommet sans prédécesseur ne peut être trouvé alors que l'ensemble des sommets n'est pas totalement parcouru.

Exemple :

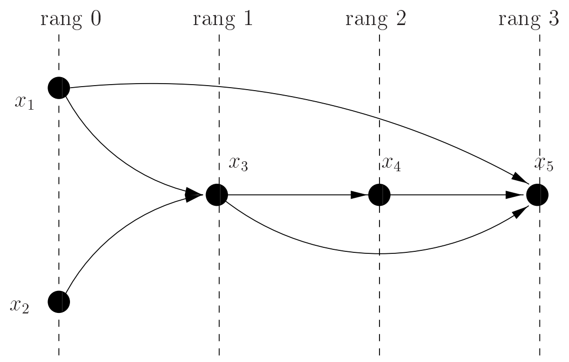


FIG. 1.17: Décomposition en niveaux

Complexité : $O(m)$ (tout sommet est mis dans M, puis enlevé de M et tous ses successeurs sont balayés. Ceci revient donc à examiner tous les arcs).

Algorithme 2

A partir de la matrice d'adjacence, on supprime, à chaque itération, tous les sommets qui correspondent à une colonne nulle, donc qui n'ont pas de précédents.

Exemple :

	x_1	x_2	x_3	x_4	x_5
x_1	0	0	1	0	1
x_2	0	0	1	0	0
x_3	0	0	0	1	1
x_4	0	0	0	0	1
x_5	0	0	0	0	0

niveaux= $\{x_1, x_2\}, \{x_3\}, \{x_4\}, \{x_5\}$

Fin 3ème séance

Application : recherche de chemin critique en *ordonnancement de projet* (*e.g.*, construction d'un bâtiment). Cette recherche passe par l'élaboration d'un *diagramme PERT*.

En gestion de projet, on est en présence d'une suite d'*activités* (ou *tâches*) concurrentes ou ordonnées dans le temps. L'objectif est de déterminer les dates de début au plus tôt des différentes tâches et la durée minimale du projet.

tâche	durée	précédents	rang
1 (début)	0	–	0
2	4	1	1
3	10	1	1
4	6	2	2
5	2	2	2
6	11	2	2
7	22	4, 5	3
8	3	5	3
9	17	3, 6, 8	4
10 (fin)	0	7, 9	5

Représentation par *graphe potentiels-tâches* :

– date de début au plus tôt de i = longueur du plus long chemin du début à i :

$$\lambda_i = \max_{j \in \Gamma^{-1}(i)} (\lambda_j + a_{ji})$$

– date de début au plus tard de i = longueur du plus long chemin de la destination à i :

$$\lambda'_i = \min_{j \in \Gamma(i)} (\lambda'_j - a_{ij})$$

– durée minimale du projet = longueur du plus long chemin du début à la fin (longueur du *chemin critique*). Elle correspond à la borne inférieure du temps total nécessaire à l'exécution de toutes les activités du projet.

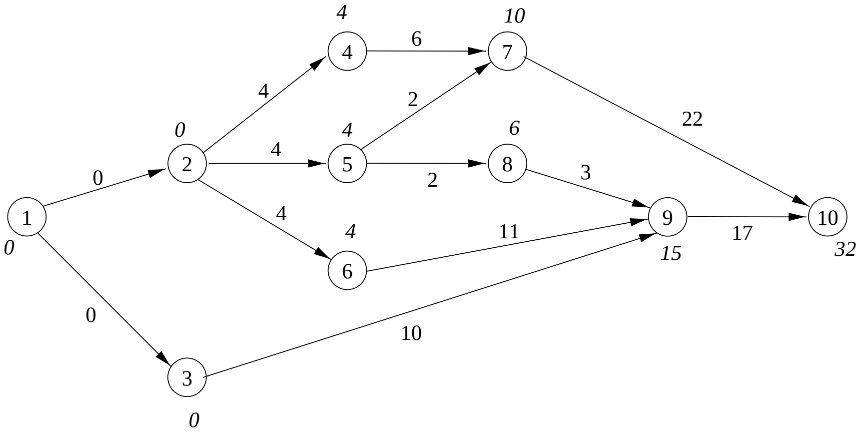


FIG. 1.18: Réseau PERT

1.5.2 Autres graphes

– *Graphe biparti*
$X = X_1 \cup X_2$ et si $\forall\, u = (i, j),\ \ i \in X_1 \Rightarrow j \in X_2,\ \ i \in X_2 \Rightarrow j \in X_1$.
Un graphe biparti complet est noté $K_{r,s}$ (pour le topologiste polonais Kuratowski) avec $r = card(X_1)$ et $s = card(X_2)$.

Exemple :

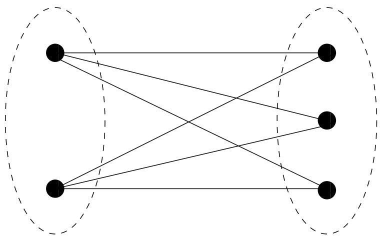


FIG. 1.19: Graphe biparti complet $K_{2,3}$

Propriétés
– Un graphe biparti est 2-coloriable.
– Un graphe biparti ne possède aucun cycle impair (*i.e.*, à nombre impair d'arêtes).

Proposition L'indice chromatique d'un graphe biparti est égal au maximum des degrés des sommets.
Démonstration : elle fait appel à la notion de couplage dans un graphe biparti.

Un *couplage* C d'un graphe simple $G = (X, E)$ est un sous-ensemble d'arêtes deux à deux sans extrémité commune. Un sommet $x \in X$ est dit *saturé* dans un couplage C si x est l'extrémité d'une des arêtes de C. Un couplage est *parfait* si ses arêtes contiennent tous les sommets du graphe (en d'autres termes, un couplage sature tous les sommets de X). Un couplage parfait a un cardinal maximal. Un *problème d'affectation* désigne la recherche d'un couplage maximal dans un graphe biparti.

Exemple : *[A. Sache] A l'oral du Bac, un ensemble d'élèves E doivent se présenter chacun devant certains des professeurs constituant un ensemble P, pour une épreuve orale d'une demi-heure. Le problème est d'organiser l'examen pour terminer le plus tôt possible.*

On forme le graphe biparti ayant E et P pour ensembles de sommets, l'arête (e, p) signifiant que l'élève e doit passer une épreuve avec le professeur p. Les colorations des arêtes de G et l'attribution à chaque épreuve d'une demi-heure de la journée se correspondent à une permutation près des demi-heures : on cherche donc l'indice chromatique $\gamma(G)$. La proposition précédente nous indique que la durée totale de l'examen est égale au maximum de la somme des durées des épreuves concernant un élève ou un professeur.

– *Graphe planaire*
C'est un graphe qui peut être représenté sur un plan (ou une sphère) tel que deux arcs (ou arêtes) ne se coupent pas. Tous les graphes de moins de 5 sommets sont planaires (K_5) ainsi que tous les graphes bipartis de moins de 6 sommets ($K_{3,3}$). Pour les autres, on connaît des algorithmes pour déterminer si un graphe est planaire.
On trouve par exemple des applications en conception de circuits électriques.

Théorème de coloration (Appel & Haken, 1977) : Tout graphe planaire est 4-chromatique.
Démonstration : par un programme d'exploration nécessitant plusieurs heures de calcul sur ordinateur.

– *Hypergraphe*
C'est un graphe non orienté où chaque *hyperarête*, au lieu de relier deux sommets, relie un sous-ensemble arbitraire de sommets de cardinal non imposé.

– *Arbre*
C'est un graphe non orienté, connexe, acyclique.

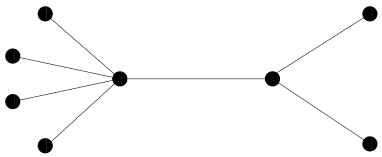


FIG. 1.20: Arbre

Un arbre comprend $n-1$ arêtes. L'addition à un arbre d'une arête entre deux sommets crée un cycle et un seul.

– *Forêt*
C'est un graphe non orienté acyclique (pas forcément connexe). Chaque composante connexe d'une forêt est un arbre.
– *Arborescence*
C'est un graphe orienté où chaque sommet possède un seul précédent sauf un qui n'en a pas : la *racine*. $\forall x \in X$, $\exists$ un chemin unique de la racine à x. On considère un nœud x d'une arborescence T, de racine r. Un nœud y quelconque sur le chemin unique de r à x est appelé *ancêtre* de x ; x est un *descendant* de y. Si le dernier arc sur le chemin de r vers x est (y, x), alors y est le *père* de x, x est un *fils* de y. Si deux nœuds ont le même père, ils sont *frères*. Un nœud sans fils est une *feuille*. La longueur du chemin entre r et x est la profondeur de x dans T. La plus grande profondeur de T est la *hauteur* de T.
Si chaque nœud a au maximum deux fils, on parle d'*arborescence binaire*.

Exemple : *Arborescence de 12 nœuds, de hauteur 4 (profondeur du sommet 9), avec la racine 7. Dans cette arborescence, 3 et 12 sont des ancêtres de 1 ; 12 est le père de 1 ; 5 est un fils de 8 ; 6 et 5 sont frères ; 6, 9, 1, 10, 11 et 2 sont des feuilles.*

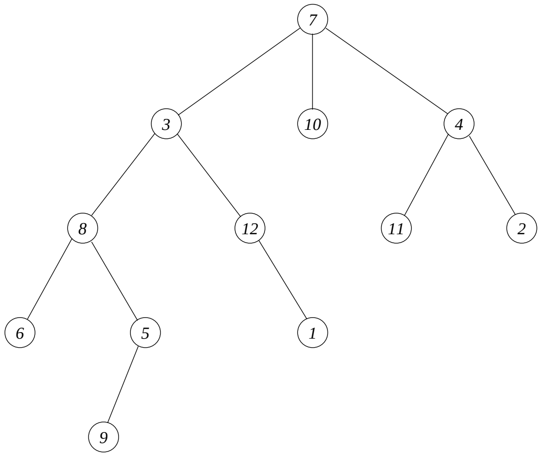


Fig. 1.21: Arborescence

– *Point d'articulation et isthme*
Un point d'articulation (resp. isthme) est un sommet (resp. une arête) dont la suppression augmente le nombre de composantes connexes du graphe.

Exemple : *Le graphe de la figure 1.22 a 10 points d'articulation et 8 isthmes.*

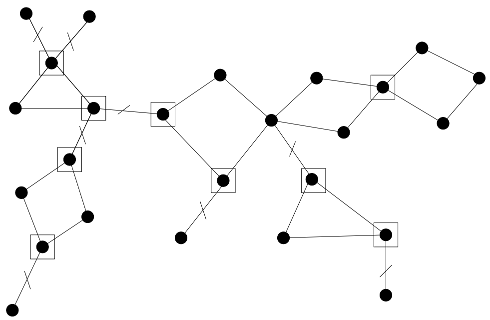


Fig. 1.22: Cactus

– *Corde et graphe d'intervalles*
Une *corde* est une arête entre deux sommets non consécutifs dans un cycle. Un cycle de longueur 3 n'admet pas de corde, alors qu'un cycle de longueur 4 peut avoir 0, 1 ou 2 cordes.

Soit $a = (A_1, A_2, \ldots, A_n)$ une famille d'intervalles sur une droite. Le graphe représentatif de a est tel que chaque sommet a_i est associé à un intervalle A_i et (a_i, a_j) est un arc ssi $A_i \cap A_j \neq \emptyset$.

<u>Théorème</u> : Un graphe d'intervalles est « triangulé », c'est-à-dire que tout cycle de longueur supérieure à 3 admet au moins une corde. Son complémentaire est un « graphe de comparabilité ».
Une propriété importante des graphes triangulés est que le nombre de cliques[2] maximales est toujours borné par le nombre de sommets du graphe. Sinon, on trouve des algorithmes de recherche des cliques maximales en $O(n+m+deg)$ où deg représente le degré maximal d'un sommet (Tarjan, 84).

[2]clique = ensemble des sommets d'un sous-graphe complet

Chapitre 2

Le problème du plus court chemin

Les problèmes de cheminement dans les graphes (en particulier la recherche d'un plus court chemin) comptent parmi les problèmes les plus anciens de la théorie des graphes et les plus importants par leurs applications.

2.1 Définition

Soit $G = (X, U)$ un graphe valué ; on associe à chaque arc $u = (i, j)$ une longueur $l(u)$ ou l_{ij}. Le problème du plus court chemin entre i et j est de trouver un chemin $\mu(i, j)$ de i à j tel que :

$$l(\mu) = \sum_{u \in \mu} l(u) \text{ soit minimale.}$$

Interprétation de $l(\mu)$: coût de transport, dépense de construction, temps nécessaire de parcours, ...

Remarque : la recherche du plus court chemin est analogue à la recherche du plus long chemin.

Les algorithmes seront différents suivant les propriétés des graphes :
- $l(u) \geq 0, \forall\, u \in U$
- $l(u)$ égales $\Leftrightarrow l(u) = 1, \forall\, u \in U$ (problème du plus court chemin en nombre d'arcs)
- G sans circuit
- G et $l(u)$ quelconques

et suivant le problème considéré :
- recherche du plus court chemin d'un sommet à tous les autres,
- recherche du plus court chemin entre tous les couples de sommets.

2.2 Chemins minimaux dans les graphes

2.2.1 Principe d'optimalité

Le principe d'optimalité énonce le fait que les sous-chemins des plus courts chemins sont des plus courts chemins (la programmation dynamique repose sur ce principe fondamental).

Lemme : Soient un graphe $G = (X, U)$ et une fonction de pondération $l : X \times X \to \mathbb{R}$, soit $C =< X_1, X_2, \ldots, X_k >$ un plus court chemin de X_1 à X_k et $\forall\, (i, j)$ tel que $1 \leq i \leq j \leq k$, soit $C_{ij} =< X_i, X_{i+1}, \ldots, X_j >$ un sous-chemin de C allant de X_i à X_j. Alors C_{ij} est un plus court chemin de X_i à X_j.
Démonstration : Si l'on décompose C en $< X_1, X_i, X_j, X_k >$ alors $l(C) = l(C_{1i}) + l(C_{ij}) + l(C_{jk})$. Supposons qu'il existe C'_{ij} avec $l(C'_{ij}) < l(C_{ij})$. Dans ce cas $< X_1, X_i, X_j, X_k >$ a pour poids $l(C_{1i}) + l(C'_{ij}) + l(C_{jk}) < l(C)$, en contradiction avec la prémisse qui veut que C soit un plus court chemin entre X_1 et X_k. $\square$ Fin 4ème séance

Principe des algorithmes de recherche de chemins minimaux :
- une distance $dist(i)$ est associée à x_i ;
- en fin d'algorithme, cette distance représente la longueur d'un plus court chemin de l'origine au sommet considéré.

2.2.2 D'un sommet à tous les autres

Ce problème est aussi appelé le problème de recherche du *plus court chemin à origine unique*. Beaucoup d'autres problèmes peuvent être résolus par l'algorithme avec origine unique :
- plus court chemin à destination unique (inversion du sens de chaque arc du graphe) ;
- plus court chemin pour un couple de sommets donné ;
- plus court chemin pour tout couple de sommets (algorithme à origine unique à partir de chaque sommet).

Remarque : si dans le graphe il existe un circuit de longueur négative, la recherche d'un plus court chemin de l'origine au sommet considéré est sans objet. On peut utiliser le circuit ω une infinité de fois (présence d'un *circuit absorbant* qui entraîne une diminution perpétuelle de la longueur du chemin [cf. Fig. 2.1]).

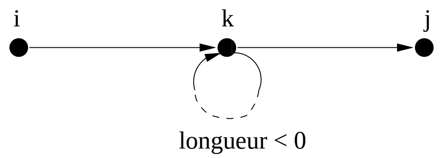


FIG. 2.1: Circuit absorbant

Les algorithmes présentés utilisent une procédure d'initialisation que nous appellerons SOURCE-UNIQUE-INITIALISATION(G, s) et une procédure dite de relâchement, appelée RELACHER(i, j, l). Le relâchement d'un arc $u = (i, j)$ consiste à déterminer s'il est possible, en passant par u, d'améliorer le plus court chemin jusqu'à j et, si oui, de mettre à jour $dist(j)$ et $pred(j)$.

Source-Unique-Initialisation(G,s)

```
pour i=1 à n
  dist(i) ← ∞
  pred(i) ← 0
dist(s) ← 0
```

Relacher(i,j,l)

```
si dist(j) > dist(i) + l(i,j)
  alors dist(j) ← dist(i) + l(i,j)
        pred(j) ← i
```

Dans l'algorithme de Dijkstra-Moore, chaque arc est relâché exactement une fois. Dans celui de Bellman-Ford, chaque arc est relâché plusieurs fois.

Algorithme de Dijkstra-Moore (1959)

Souvent les longueurs des arcs sont non-négatives ($l(u) \geq 0$), *e.g.*, carte routière, et on utilise alors cet algorithme glouton. A chaque sommet p, on associe trois éléments $[marq(p), pred(p), dist(p)]$. s est le sommet origine et t le sommet destination.
$marq(p)$ = marquage booléen
$pred(p)$ = sommet précédant p sur le plus court chemin de l'origine à p (sur le sous-graphe des sommets marqués)
$dist(p)$ = plus courte distance de l'origine à p (sur le sous-graphe des sommets marqués).
Principe de l'algorithme :

1. Etat initial : origine $[1, 0, 0]$; autres sommets $[0, 0, \infty]$.
2. X = ensemble des sommets non marqués ; marquer le sommet k tel que $dist(k) = \min_{x \in X} dist(x)$.
   Si $k = t$ FIN.
3. Considérer chaque sommet y non marqué suivant de k. RELACHER(k, y, l) ; aller en 2.

Dijkstra-Moore(G,l,s)

```
SOURCE-UNIQUE-INITIALISATION(G,s)
pour i=1 à n
  marq(i) ← 0
marq(s) ← 1
recent ← s        % sommet qui vient d'être marqué
tant que marq(t)=0 faire
```



```
  pour chaque successeur i de recent
    si marq(i)=0
      RELACHER(recent,i,l)
  soit y tel que dist(y)=min_{x∈X} dist(x)
  marq(y) ← 1
  recent ← y
```

Complexité : $O(n^2)$ ou $O(m \log n)$ avec une structure de tas, intéressante si le graphe est *peu dense* (*i.e.*, $m \lll n^2$).

L'algorithme de Dijkstra-Moore utilise une stratégie *gloutonne* lorsqu'il choisit le sommet le moins coûteux à chaque étape. On démontre que dans le cas de cet algorithme, cette stratégie conduit à un résultat global optimal.

Exemple :

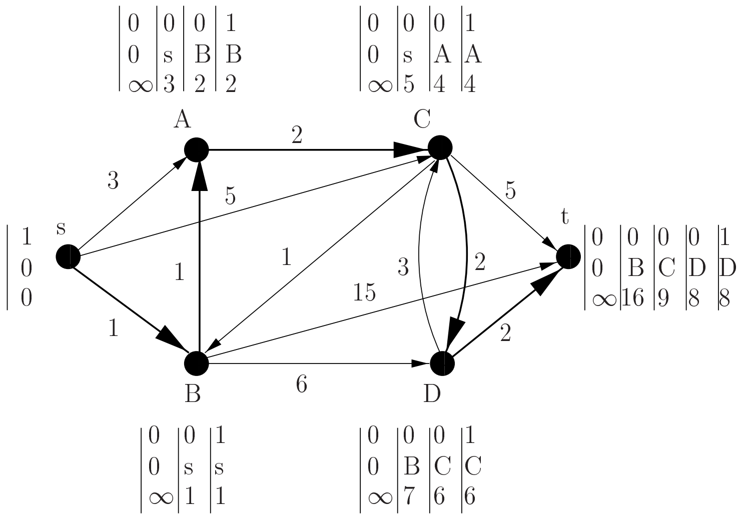


FIG. 2.2: de s à t : B, A, C, D de longueur 8

Algorithme de Bellman-Ford (1958-1962)

La présence de longueurs de signes différents ($l(u)$ quelconques) permet par exemple de modéliser des coûts et des profits. L'algorithme de Dijkstra-Moore ne permet pas de considérer les arcs négatifs, car une fois qu'un sommet est marqué on ne peut changer ce marquage lors des itérations suivantes. L'algorithme de Dijkstra-Moore est ainsi dit *à fixation d'étiquettes*. On considère donc ici un algorithme qui permet un marquage qui n'est pas définitif tant que le programme n'est pas déterminé (le marquage est modifié itérativement). Ce type d'algorithme est appelé *à correction* *d'étiquettes.*

Principe de l'algorithme :
- Source-Unique-Initialisation(G,s)
- Répéter
  Relacher(i,j,l)
- Jusqu'à ce qu'aucun arc ne permette plus de diminuer $dist(i)$.

Théorème : Les valeurs finales des distances sont obtenues en au plus $n-1$ itérations principales (consistant à consulter les prédécesseurs de tous les sommets).
Démonstration : En l'absence de circuit absorbant, un plus court chemin de s à tout autre sommet est un chemin élémentaire, c'est-à-dire un chemin d'au plus $n-1$ arcs.

Conséquence : si au bout de n itérations, les valeurs $dist(i)$ continuent à être modifiées, c'est que le graphe possède un circuit de longueur strictement négative.

Bellman-Ford(G,l,s)

```
SOURCE-UNIQUE-INITIALISATION(G,s)
k ← 0
répéter
   stable ← VRAI
   pour tout j ≠ s
    pour tout i ∈ Γ^{-1}(j)
      si dist(j) > dist(i) + l(i,j) alors
         dist(j) ← dist(i) + l(i,j)
         pred(j) ← i
         stable ← FAUX
   k ← k+1
jusqu'à stable = VRAI ou k > n-1
si k > n-1 alors présence d'un circuit de poids négatif
```

Complexité : $O(nm)$, soit $O(n^3)$ pour des graphes *denses*, *i.e.*, des graphes tels que $m \simeq n^2$.

Exemple :

Fin 5ème séance

Graphes sans circuit

Lorsque le graphe est sans circuit il peut être décomposé en niveaux. La longueur du chemin de la racine à tous les autres sommets est obtenue en considérant les sommets dans l'ordre des rangs croissants et en calculant

$$dist(i) = \min_{j \in \Gamma^{-1}(i)} (dist(j) + l(j,i))$$

avec $dist(s) = 0$.
On applique en fait une version simplifiée de l'algorithme de Bellman-Ford.

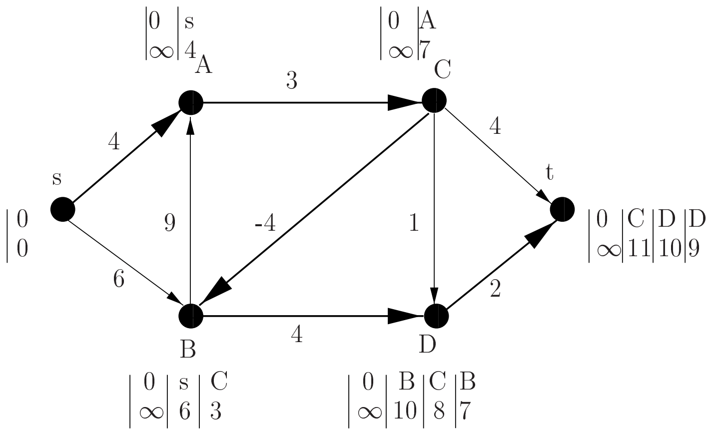


Fig. 2.3: de s à t : A, C, B, D de longueur 9

Plus-Courts-Chemins-GSC(G,l,s)

```
trier les sommets par rangs croissants
SOURCE-UNIQUE-INITIALISATION(G,s)
pour i=1 à n
   pour chaque successeur j de i
      RELACHER(i,j,l)
```

Complexité : $\Theta(n+m)$ (ou même $O(m)$ puisque cela revient à balayer les successeurs de chaque sommet).

Exemple :

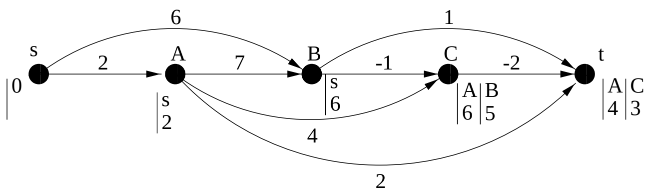


Fig. 2.4: de s à C : B de longueur 5 ; de s à t : B, C de longueur 3

Application : détermination d'un chemin critique dans un diagramme PERT :
- soit en prenant l'opposé des poids d'arcs et en exécutant Plus-Courts-Chemins-GSC ;
- soit en exécutant Plus-Courts-Chemins-GSC en remplaçant ∞ par $-\infty$ dans Source-Unique-Initialisation et $>$ par $<$ dans Relacher.

2.2.3 Entre tous les couples de sommets (algorithme matriciel)

On va ainsi calculer un *distancier* $n \times n$. Si tous les arcs sont tous de longueur positive ou nulle ($l(u) \geq 0$), on peut appliquer n fois l'algorithme de Dijkstra-Moore pour chaque sommet i. Si le graphe comporte des arcs de longueur strictement négative, on peut appliquer n fois l'algorithme de Bellman-Ford. L'algorithme de Floyd constitue une autre approche qui peut être avantageuse principalement par rapport à la seconde solution, qui nécessite un temps d'exécution en $O(n^4)$ pour des graphes denses. Contrairement aux algorithmes à origine unique qui supposent que le graphe est représenté par une liste d'adjacence, l'algorithme de Floyd-Warshall (algorithme de programmation dynamique) utilise une représentation par matrice d'adjacence.

Soient les matrices : $L = [l_{ij}]$; $P = [p_{ij}]$
$l_{ij} = l(i,j)$ si $(i,j) \in U$
$= \infty$ sinon

A l'initialisation :
$l_{ii} = 0$
$p_{ij} = 0$ si $l_{ij} = \infty$
$= i$ sinon

En fin d'algorithme :
l_{ij} = longueur du plus court chemin entre i et j
p_{ij} = prédécesseur de j sur le plus court chemin de i à j

L'algorithme utilisé est celui de Floyd (dans une application de recherche de la fermeture transitive d'un graphe, cet algorithme a été développé par Warshall la même année – 1962 ; d'où souvent l'appellation d'algorithme de Floyd-Warshall). Il consiste en trois boucles imbriquées sur i, j, k de 1 à n et on calcule :

$$l_{ij} \leftarrow \min(l_{ij}, l_{ik} + l_{kj})$$

Floyd-Warshall(L,P)

```
pour k=1 à n
  pour i=1 à n sauf k
    si l_ik + l_ki < 0 FIN        % présence d'un circuit négatif
    si l_ik ≠ ∞ alors
      pour j=1 à n sauf i
      γ = l_ik      % pour éviter le calcul d'adresse de la variable
indicée à chaque itération de j
        si γ + l_kj < l_ij
          l_ij ← γ + l_kj
          p_ij ← p_kj
```

Complexité : $O(n^3)$.

Exemple :

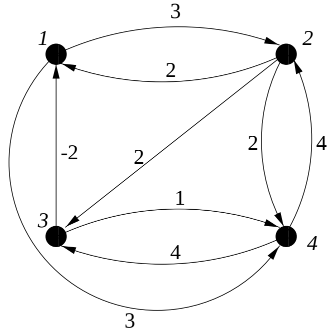


Fig. 2.5: Procédure de Floyd-Warshall

$$L^{(0)} = \begin{pmatrix} 0 & 3 & \infty & 3 \\ 2 & 0 & 2 & 2 \\ -2 & \infty & 0 & 1 \\ \infty & 4 & 4 & 0 \end{pmatrix} \qquad P^{(0)} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 0 & 3 & 3 \\ 0 & 4 & 4 & 4 \end{pmatrix}$$

$$L^{(1)} = \begin{pmatrix} 0 & 3 & \infty & 3 \\ 2 & 0 & 2 & 2 \\ -2 & \mathbf{1} & 0 & 1 \\ \infty & 4 & 4 & 0 \end{pmatrix} \qquad P^{(1)} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & \mathbf{1} & 3 & 3 \\ 0 & 4 & 4 & 4 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & \mathbf{5} & 3 \\ 2 & 0 & 2 & 2 \\ -2 & 1 & 0 & 1 \\ \mathbf{6} & 4 & 4 & 0 \end{pmatrix} \qquad P^{(2)} = \begin{pmatrix} 1 & 1 & \mathbf{2} & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 1 & 3 & 3 \\ \mathbf{2} & 4 & 4 & 4 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & 5 & 3 \\ \mathbf{0} & 0 & 2 & 2 \\ -2 & 1 & 0 & 1 \\ \mathbf{2} & 4 & 4 & 0 \end{pmatrix} \qquad P^{(3)} = \begin{pmatrix} 1 & 1 & 2 & 1 \\ \mathbf{3} & 2 & 2 & 2 \\ 3 & 1 & 3 & 3 \\ \mathbf{3} & 4 & 4 & 4 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 3 & 5 & 3 \\ 0 & 0 & 2 & 2 \\ -2 & 1 & 0 & 1 \\ 2 & 4 & 4 & 0 \end{pmatrix} = \hat{L} \qquad P^{(4)} = \begin{pmatrix} 1 & 1 & 2 & 1 \\ 3 & 2 & 2 & 2 \\ 3 & 1 & 3 & 3 \\ 3 & 4 & 4 & 4 \end{pmatrix} = \hat{P}$$

Chapitre 3

Chemins, parcours hamiltoniens, arbres

Nous étudions dans ce chapitre l'énumération de chemins, d'arbres et de circuits hamiltoniens dans un graphe.

3.1 Chemins

3.1.1 Enumération de chemins

La recherche de chemins s'effectue à partir de la matrice booléenne A associée au graphe. Pour rechercher les chemins de longueur p, on utilise le produit matriciel tel qu'il est défini en algèbre linéaire et on calcule A^p ; en effet :
$A = \{a_{ij}\};\ a_{ij} = 1 \Rightarrow \exists$ un arc entre i et j
$a^2_{ij} = a_{i1}.a_{1j} + a_{i2}.a_{2j} + \ldots + a_{in}.a_{nj};\ a^2_{ij} = 1 \Rightarrow \exists$ au moins un chemin d'exactement 2 arcs reliant i à j

Remarques :
- les opérations sont logiques ;
  '.' $\Rightarrow$ arcs bout à bout (ET logique)
  '+' $\Rightarrow$ chemins en parallèle (OU logique)
- si le graphe est réflexif ($a_{ii} = 1$), l'existence de A^p (qu'on écrit alors $\mathcal{A}^p$) matérialise l'existence de chemins de $1, 2, \ldots, p$ arcs.
  Exemple : $a^2_{24} = a_{21}a_{14} + a_{22}a_{24} + a_{23}a_{34} + a_{24}a_{44}$, *i.e.*, chemins de 1 ou 2 arcs, sans les boucles ; si le graphe n'est pas réflexif $a^2_{24} = a_{21}a_{14} + a_{23}a_{34}$, *i.e.*, chemins de 2 arcs exactement.

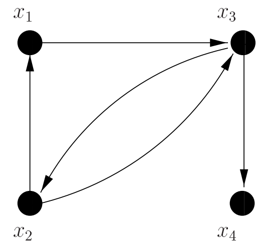
3.1.2 Existence de chemins

Pour avoir plus rapidement l'existence de tous les chemins dans un graphe, on rend le graphe réflexif ($\mathcal{A} = \mathbb{I} + A$), puis on calcule $\mathcal{A}^2, \mathcal{A}^4, \ldots, \mathcal{A}^p, \mathcal{A}^{2p}$. Lorsque $\mathcal{A}^p = \mathcal{A}^{2p}$,

on a matérialisé tous les chemins du graphe ; on obtient ainsi la matrice des fermetures transitives.

Remarques :
- on cherche l'égalité $\mathcal{A}^p = \mathcal{A}^{2p}$ pour converger plus vite. Ainsi, si un graphe possède des chemins de longueur 8, on arrête la procédure après avoir calculé $\mathcal{A}^2, \mathcal{A}^4, \mathcal{A}^8$ au lieu de calculer $\mathcal{A}^2, \mathcal{A}^3, \mathcal{A}^4, \mathcal{A}^5, \mathcal{A}^6, \mathcal{A}^7, \mathcal{A}^8$;
- comme cela a été annoncé précédemment, le calcul de $\mathcal{A}^p$ donne les chemins de longueur inférieure à p $(1, 2, \ldots, p)$. Pour trouver les chemins de longueur p uniquement, on ne peut s'affranchir de calculer A^p.

Exemple :



$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

FIG. 3.1: Existence de chemins

$$\mathcal{A} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \mathcal{A}^2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \mathcal{A}^4$$

Remarque : recherche de cfc à partir de la matrice des fermetures transitives.

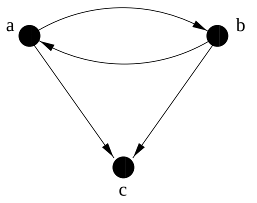
$$\mathcal{A}^2 = \left(\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \Rightarrow \begin{pmatrix} cfcm_1 \\ cfcm_2 \end{pmatrix} = \begin{pmatrix} \{x_1, x_2, x_3\} \\ \{x_4\} \end{pmatrix}$$

Pour trouver les cfc, il faut isoler des matrices carrées maximales ne contenant que des « 1 ».
Exercice : retrouver ces cfc par l'algorithme du marquage.

3.1.3 Construction de chemins

On remplace la matrice booléenne par une matrice comportant le nom des arcs (matrice aux arcs).

Exemple :



$$\mathcal{A} = \begin{pmatrix} aa & ab & ac \\ ba & bb & bc \\ 0 & 0 & cc \end{pmatrix}$$

FIG. 3.2: Enumération de chemins

$$\mathcal{A}^2 = \begin{pmatrix} - & - & - \\ - & - & ba.ac + bb.bc + bc.cc \\ - & - & - \end{pmatrix}$$

FIG. 3.3: 3 chemins en parallèle de b à c

3.2 Parcours hamiltoniens

On traite maintenant du problème de l'énumération de parcours hamiltoniens (par exemple les circuits – CH) dans un graphe.

On choisit un arc « a » du graphe initial. On distingue deux types de circuits :
– cat.1 : ceux qui contiennent l'arc a ;
– cat.2 : ceux qui ne le contiennent pas.
Remarque (simplification du graphe) :
Si $(x, y) \in CH$, les arcs sortant de x et les arcs entrant dans y n'appartiennent pas à CH. Il advient une suppression de ces arcs-là.

Exemple :

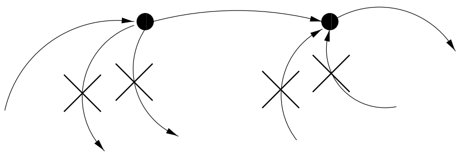


FIG. 3.4: Procédure de simplification

$G1$ est le graphe obtenu par contraction de a et simplification. $G2$ est le graphe obtenu par suppression de a.
Les circuits de cat.1 contiennent a et forment un circuit sur $G1$. Les circuits de cat.2 forment un circuit sur $G2$.

Exemple : [cf. Fig. 3.5]

Remarques :

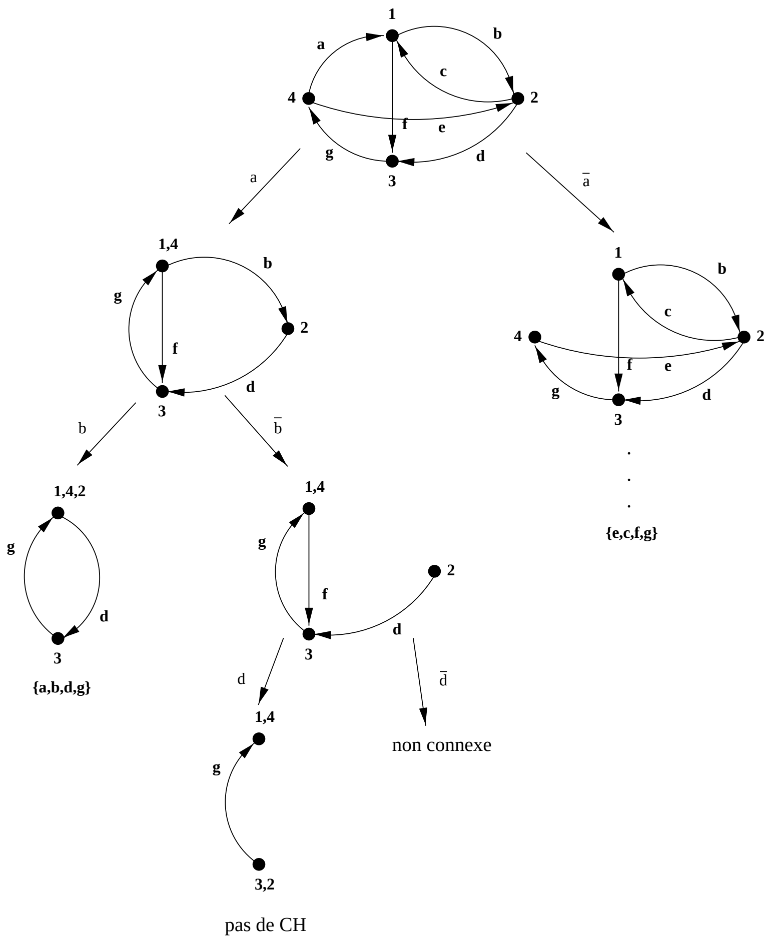


FIG. 3.5: Enumération de circuits hamiltoniens

1. Une autre technique pour la détermination de CH passe l'application de la méthode de *composition latine*. Celle-ci consiste à construire la matrice aux arcs dans laquelle la diagonale ne comprend que des zéros (M_1). On supprime ensuite chaque lettre initiale (c-à-d qu'on enlève le sommet origine de la définition de l'arc – ou du chemin) : on obtient $\widetilde{M}_1$. Puis on opère à la multiplication (latine) des matrices : $M_1 \otimes \widetilde{M}_1 = M_2$, où l'on ne retient que les séquences ne contenant pas de répétition de lettre. L'élévation en puissance est effectuée jusqu'à obtenir des chemins de longueur $n-1$ (M_{n-1}). Pour la détermination des circuits hamiltoniens, on calcule $M_n = M_{n-1} \odot \widetilde{M}_1$ en conservant les séquences ayant des répétitions de lettre, à condition qu'elles se trouvent sur la diagonale.
2. Lorsque le graphe comporte un CH, il constitue un graphe fortement connexe (la matrice $\mathcal{A}$ ne comporte que des 1). En revanche, le fait que $\mathcal{A}$ ne comporte que des 1 n'implique pas l'existence d'un CH dans le graphe.
3. D'autre part, il existe plusieurs théorèmes renseignant sur l'existence et/ou le nombre de parcours hamiltoniens dans un graphe, par exemple :

   Théorèmes :
   – Si un graphe admet 2 cycles hamiltoniens sans arêtes communes, il admet au moins 3 cycles hamiltoniens.
   – Un 1-graphe complet fortement connexe admet un circuit hamiltonien.
   – Dans un 1-graphe complet antisymétrique et transitif, il existe un chemin hamiltonien et un seul.
   – Dans un 1-graphe complet et antisymétrique, le nombre de ses chemins hamiltonien est impair.

   et d'autres encore, très souvent liés aux degrés ou aux demi-degrés des sommets...

3.3 Arbres

Ce paragraphe concerne l'énumération des arbres dans un graphe, ainsi que la recherche d'un arbre couvrant de poids minimum.

3.3.1 Enumération d'arbres

On choisit une arête « a » sur le graphe initial G. On trouve deux catégories d'arbres :
– cat.1 : ceux qui contiennent l'arête a ;
– cat.2 : ceux qui ne la contiennent pas.
$G1$ est le graphe obtenu par contraction de a. $G2$ est le graphe obtenu par suppression de a.
Chaque arbre de cat.1 est composé de a et d'un arbre sur $G1$. Chaque arbre de cat.2 est constitué d'un arbre sur $G2$.
La cat.2 est vide si a est un *pont*, *i.e.*, si sa suppression rend $G2$ non connexe.

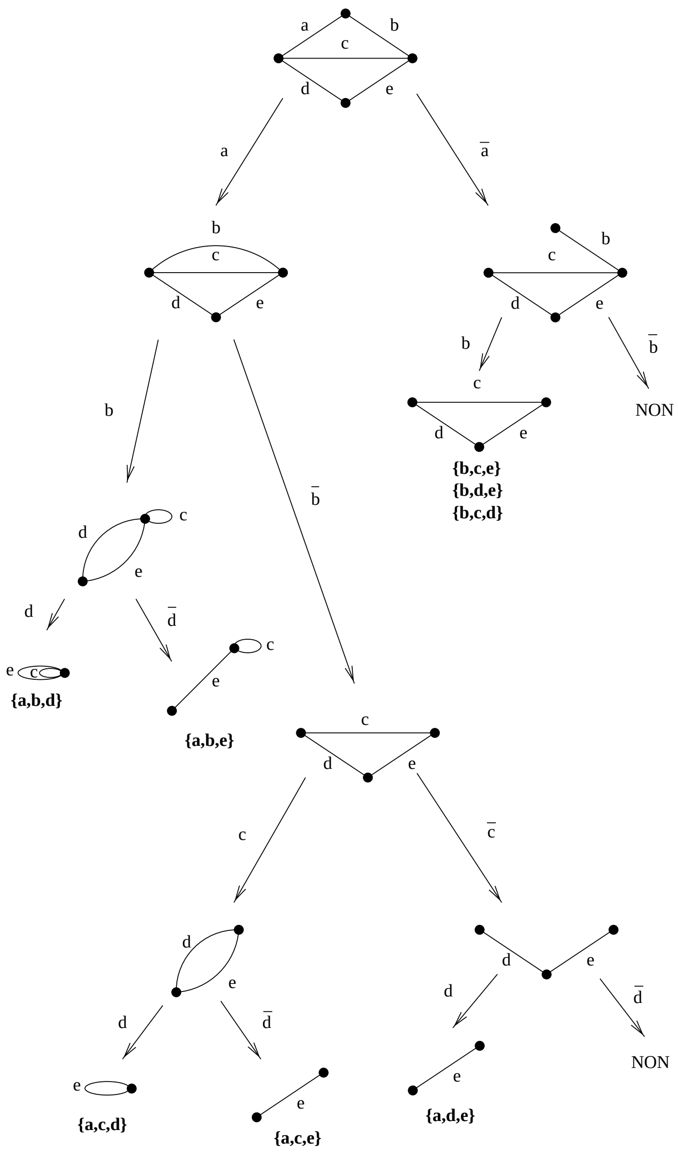


FIG. 3.6: Enumération d'arbres

Exemple : [cf. Fig. 3.6]

Applications :

3.3.2 Arbre couvrant de poids minimum

On parle ainsi d'un arbre qui « couvre » le graphe G (il connecte tous ses sommets) et dont le coût de connexion est minimal.

Applications : optimisation de réseaux (lignes à haute tension, oléoducs, ...), câblage de circuits électroniques, etc.

Algorithme de Kruskal (1956)
En reprenant l'algorithme d'énumération des arbres, on descend l'arborescence à gauche (contraction de l'arête) en choisissant l'arête de longueur minimale à chaque étape. On s'arrête lorsque tous les sommets du graphe sont connectés — ou, ce qui revient au même, lorsque le nombre d'arêtes retenues égale $n-1$. C'est un algorithme *glouton*, *i.e.*, il fait un choix optimal localement dans l'espoir que ce choix mènera à la solution optimale globalement. Ici, il rajoute à chaque étape l'arête de poids minimal à la forêt qu'il construit. L'arbre obtenu est unique si toutes les arêtes sont initialement de valeurs différentes.

Complexité : $O(m \log m)$.

Exemple : [cf. Fig. 3.7]

Algorithme de Prim (1957)
Principe :
– on part d'un arbre initial A réduit à un seul sommet s (*e.g.*, $s = 1$) ;
– ensuite, à chaque itération, on augmente l'arbre A en le connectant au « plus proche » sommet libre au sens des poids.

Complexité : $O(m \log n)$.

Exemple : [cf. Fig. 3.8]

Remarque : Sur cet exemple, on retrouve l'arbre à coût minimum calculé par l'algorithme de Kruskal. Dans le cas général, on peut trouver un arbre différent, mais de même poids.

Fin 7ème séance

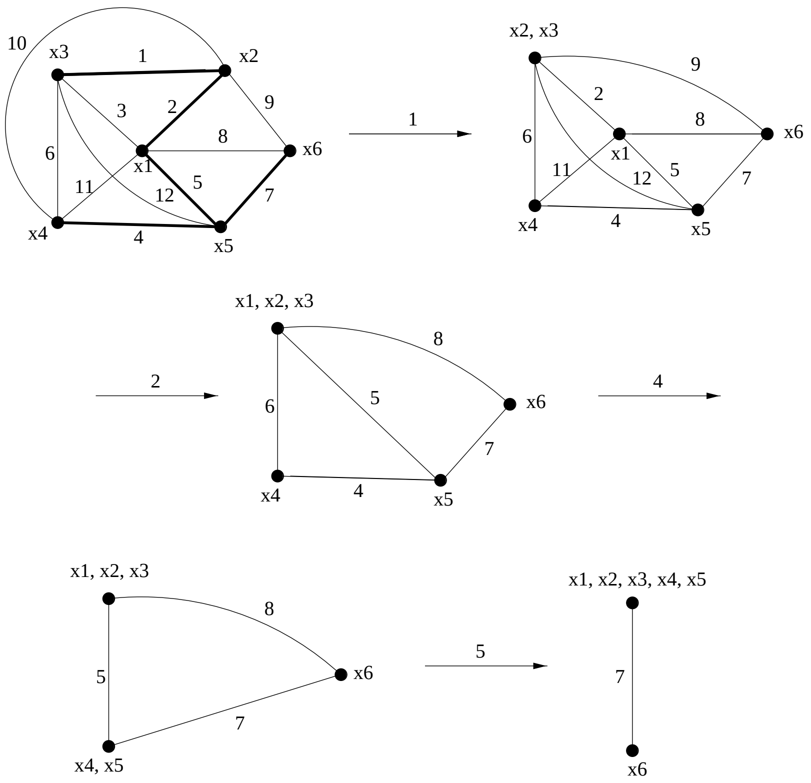


FIG. 3.7: Recherche d'un arbre à coût minimum par Kruskal

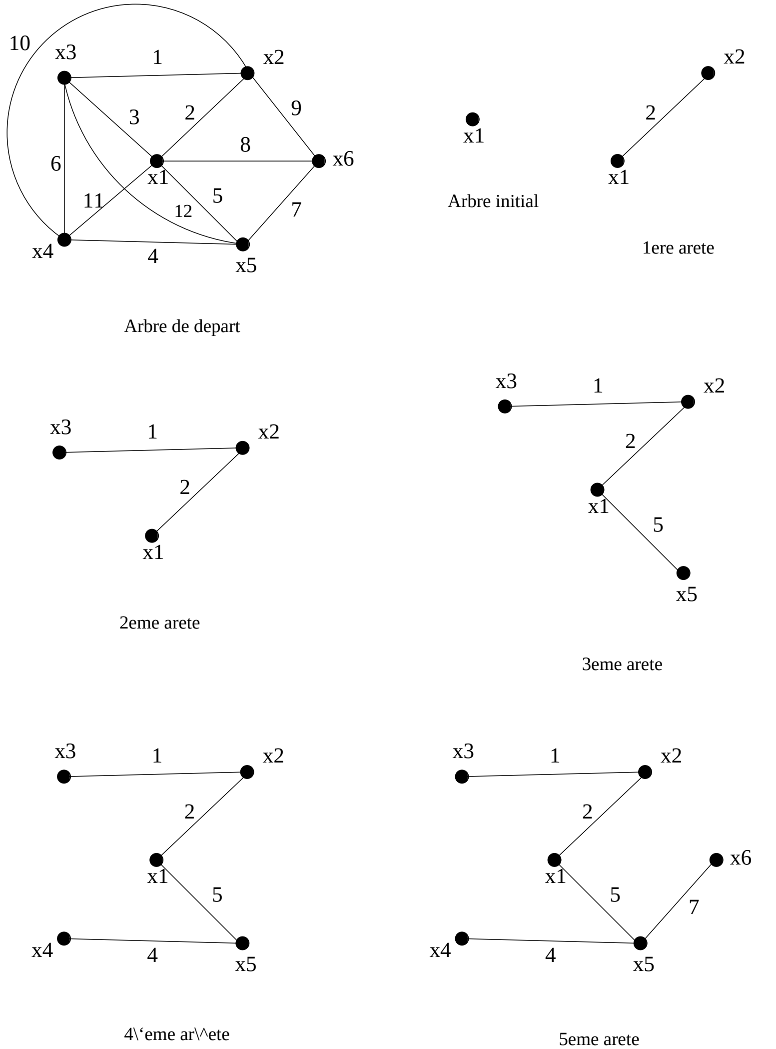


FIG. 3.8: Recherche d'un arbre à coût minimum par Prim ($s = 1$)

Chapitre 4

Flots dans les réseaux

Le problème des flots dans les réseaux concerne la circulation de matière sur les arcs d'un graphe. Parmi les nombreuses applications qui relèvent de ce problème, on trouve les réseaux de transport de marchandises (urbains, ferroviaires ou aériens) de différents points distributeurs à différents points consommateurs ; l'écoulement de liquides à l'intérieur de tuyaux ; le courant dans les réseaux électriques ; l'informatique à travers les réseaux de communication ; le coût de réalisation d'un projet en ordonnancement ; etc.

4.1 Définitions et propriétés

On considère des *réseaux*, *i.e.*, des graphes connexes, sans boucle et asymétriques, possédant une entrée et une sortie. Soit $S = \{s_{ij}\}$, la matrice d'incidence sommets-arcs de $G = (X, U)$, $u = 1, 2, \ldots, m$.

4.1.1 Flot dans un réseau

4.1.1.1 Définition

Un *flot* sur un graphe $G = (X, U)$ est un vecteur ligne $\underline{\varphi} = [\varphi_1, \varphi_2, \ldots, \varphi_m] \in \mathbb{R}^m$ à m composantes et tel que :

– $\varphi_j \geq 0 \ \forall j \in 1, \ldots, m \iff \underline{\varphi} \geq \underline{0}$
– en tout sommet $i \in X$, la 1ère loi de Kirchhoff est vérifiée (loi de conservation aux nœuds) :

$$\sum_{j \in \omega^+(i)} \varphi_j = \sum_{j \in \omega^-(i)} \varphi_j$$

φ_j est la quantité de flot ou *flux* sur l'arc j.
La 2ème condition peut également s'écrire (pour $i = 1$ à n) :

$$\sum_{j=1}^{m} s_{ij} \times \varphi_j = 0 \iff S.\underline{\varphi}^t = \underline{0}$$

4.1.1.2 Opérations sur les flots

Soient $\underline{\varphi}, \underline{\varphi}_1, \underline{\varphi}_2$ des flots sur G, $k \geq 0$.
Lemme : $k.\underline{\varphi}$ est un flot sur G. $\underline{\varphi}_1 + \underline{\varphi}_2$ est un flot sur G. $\underline{\varphi}_1 - \underline{\varphi}_2$ est un flot sur G si $\underline{\varphi}_1 \geq \underline{\varphi}_2$.
Démonstration (partie) : $k.\underline{\varphi} \geq \underline{0}$ et $S.(k.\underline{\varphi})^t = k.S.\underline{\varphi}^t = \underline{0}$. □

4.1.1.3 Flot élémentaire

Soit γ un circuit élémentaire sur G (*i.e.*, il passe au plus une fois par un sommet).
On appelle $\underline{v}$ le vecteur constitué par les éléments v_i tels que :
– $v_i = 1$ si $u_i \in \gamma$, $i = 1..m$
– $v_i = 0$ sinon
$\underline{v}$ est un flot cyclique élémentaire sur G.

Théorème : tout flot $\underline{\varphi}$ se décompose en une somme de flots cycliques élémentaires linéairement indépendants :

$$\underline{\varphi} = \lambda_1.\underline{v}_1 + \lambda_2.\underline{v}_2 + \ldots + \lambda_k.\underline{v}_k, \quad \lambda_i \geq 0$$

Exemple :

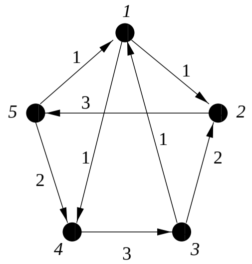


FIG. 4.1: Flot initial

Dans cet exemple, il existe deux possibilités de décomposition en flots cycliques élémentaires :

4.1.1.4 Capacité des arcs

Un *réseau de transport* est un réseau où à chaque arc $j \in U$ est associé une *capacité* $c_j \geq 0$ (et éventuellement un coût d_j) (exemple de capacité : tonnage ou débit maximum). C'est la limite supérieure du flux admissible sur j. Un flot est admissible ssi $\varphi_j \leq c_j \ \forall j = 1, \ldots, m \iff \underline{\varphi} \leq \underline{c}$.

4.1.2 Graphe d'écart

Soit $\underline{\varphi}$ un flot admissible sur G. Le graphe d'écart associé à $\underline{\varphi}$ est le graphe $G^e(\underline{\varphi}) = [X, U^e(\underline{\varphi})]$ où $U^e(\underline{\varphi})$ est tel que pour tout $j \in U$, on associe deux arcs de $G^e(\underline{\varphi})$:

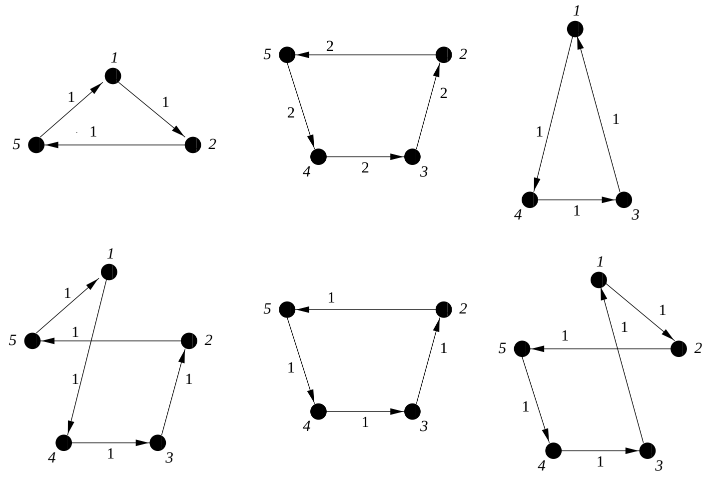


FIG. 4.2: Flots cycliques élémentaires

– j^+ de même sens et de capacité résiduelle $c_j^+ = c_j - \varphi_j \geq 0$;
– j^- de sens opposé et de capacité résiduelle $c_j^- = \varphi_j \geq 0$.
La *capacité résiduelle* correspond à la quantité de flot net supplémentaire qu'il est possible d'ajouter sur l'arc j sans dépasser la capacité c_j.

Exemple :
Par convention, les arcs de capacité nulle ne sont pas représentés sur le graphe.

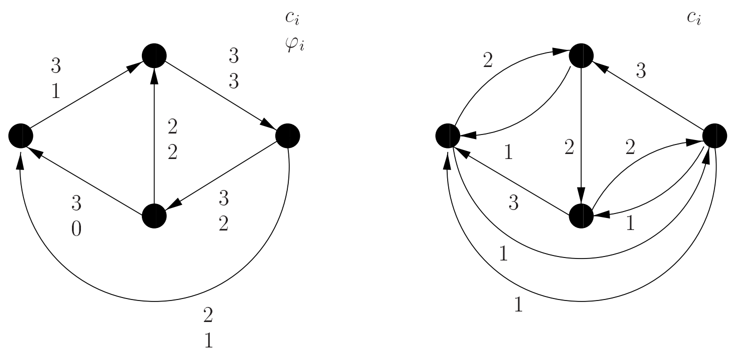
Le graphe d'écart représente la modification que l'on peut faire subir au flot $\underline{\varphi}$ tout en lui conservant la propriété de flot admissible.
$\underline{\varphi}^e = [\underline{\varphi}^+ | \underline{\varphi}^-]$
$\underline{c}^+ = \underline{c} - \underline{\varphi}$
$\underline{c}^- = \underline{\varphi}$

$\underline{0} \leq \underline{\varphi}^+ \leq \underline{c} - \underline{\varphi}$
$\underline{0} \leq \underline{\varphi}^- \leq \underline{\varphi}$
$S.(\underline{\varphi}^+ - \underline{\varphi}^-)^t = \underline{0}$

4.2 Problème du flot maximum dans un réseau de transport

4.2.1 Définition

Soient deux sommets fictifs source (s) et puits (t) permettant de modéliser des entrées et des sorties de matières à différents sommets du graphe. $G^0(X, U^0)$ est



$G(\underline{\varphi})$ $\qquad$ $G^e(\underline{\varphi})$

FIG. 4.3: Graphe et son graphe d'écart

déduit de G en rajoutant l'arc (t, s) appelé *arc de retour du flot*, noté u_0 :

$$\sum_{j\in\omega^+(s)} \varphi_j = \sum_{j\in\omega^-(t)} \varphi_j = \varphi_0$$

où φ_0 = valeur du flot.

Le problème du flot maximum de s à t dans G consiste à déterminer un flot $\underline{\varphi}'$ dans G^0 vérifiant les contraintes de capacité et maximisant φ_0.

4.2.2 Circuit d'incrémentation

On appelle *circuit d'incrémentation* sur $G^e(\underline{\varphi})$ un circuit traversant u_0^+ et pas u_0^- dont tous les arcs ont une capacité différente de 0.

Théorème : $\underline{\varphi}$ est maximum ssi $G^e(\underline{\varphi})$ ne contient pas de circuit d'incrémentation.

4.2.3 Coupes

Soit $\{X', X''\}$ une partition de X, *i.e.*, $X' \cup X'' = X$ et $X' \cap X'' = $. L'ensemble des arcs ayant leur extrémité initiale dans X' et leur extrémité finale dans X'' forme une *coupe*. La capacité de la coupe est :

$$C(X', X'') = \sum_{j \,\in\, coupe(X',X'')} c_j$$

Théorème : $\varphi_0 \leq$ capacité de toute coupe séparant s et t.

Théorème de Ford-Fulkerson (max-flow min-cut) : la valeur d'un flot maximum est égale à la plus petite capacité des coupes séparant s et t.

Ceci entraîne notamment que la valeur d'une coupe quelconque est une borne supérieure pour la valeur du flot.

4.2.4 Algorithme de recherche d'un flot maximum

Méthode de Ford-Fulkerson (1956)

Etape 1
- Constitution d'un flot initial admissible (nul ou *complet*, *i.e.*, un flot tel que tout chemin de s à t possède au moins un arc *saturé* – $\varphi_i = C_i$).
- $k \leftarrow 0$.

Etape 2
- Construire $G^e(\underline{\varphi}_k)$.
- Chercher un circuit d'incrémentation γ.
  S'il n'existe pas, FIN, le flot est maximum.

Etape 3
- Soit δ la plus petite capacité de γ.
  Pour j^+ de γ, augmenter φ_j de δ.
  Pour j^- de γ, diminuer φ_j de δ.
- $k \leftarrow k + 1$.
- Aller en Etape 2.

Complexité : par une bonne implémentation (*e.g.*, algorithme de Edmonds-Karp), on arrive à un temps d'exécution en $O(nm^2)$.

Exemple :

Il n'existe pas de chemin de s à $t \Rightarrow \underline{\varphi}^1$ est un flot maximum :

$$\varphi_0 = \varphi_0^1 = 60$$

4.3 Problème du flot maximum à coût minimum

4.3.1 Position du problème

On associe un coût unitaire de transport d_j à chaque arc u_j du réseau.

$$\text{coût total } = \underline{d} \times \underline{\varphi}^t = \sum_{j=1}^{m} d_j.\varphi_j$$

On cherche un flot maximum de coût minimum sur $G^e(\underline{\varphi})$:
- pour un arc j^+, on associe un coût d_j ;
- pour un arc j^-, on associe un coût $-d_j$.

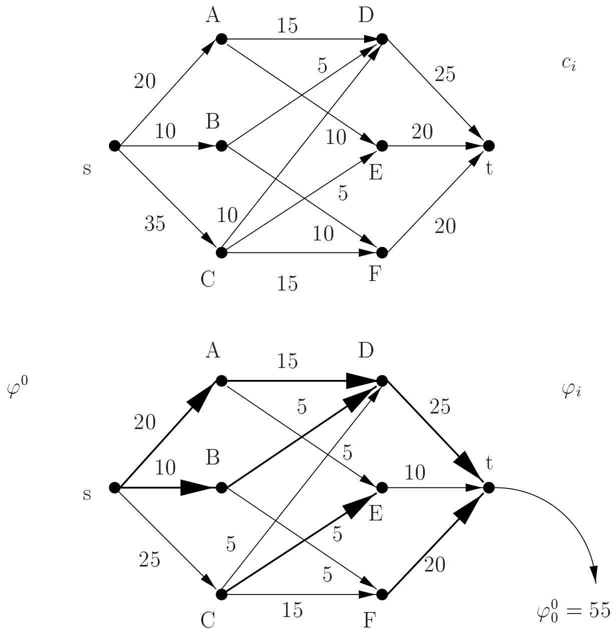


FIG. 4.4: Procédure de Ford-Fulkerson à partir d'un flot complet

4.3.2 Algorithme de construction d'un flot maximum à coût minimum

On présente ici une méthode par augmentation de flot, à partir d'un flot initial nul. Une autre approche consiste à appliquer une méthode par diminution du coût, à partir d'un flot initial complet (Ford-Fulkerson 1962).
$\underline{\varphi}$ est un flot à coût minimum de G, γ un circuit d'incrémentation à coût minimum sur $G^e(\underline{\varphi})$ et δ sa plus petite capacité.

Comme dans l'algorithme de Ford-Fulkerson, on introduit un vecteur $\underline{\varphi'}$ défini par $\varphi' =$
- $\varphi + \delta$ si $j^+ \in \gamma$;
- $\varphi - \delta$ si $j^- \in \gamma$;
- φ si $j^+, j^- \notin \gamma$.

$\underline{\varphi'}$ est un flot à coût minimum sur G de valeur plus grande de δ, que celle de $\underline{\varphi}$.
L'algorithme est identique à celui de Ford-Fulkerson pour la recherche d'un flot maximum aux différences suivantes près :

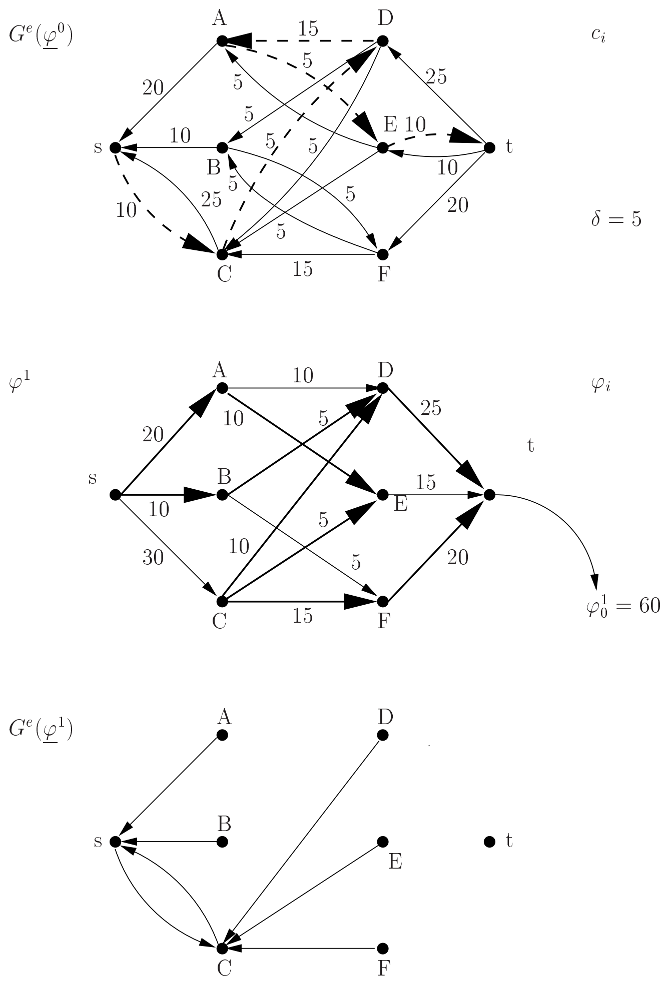


FIG. 4.5: Flot maximum

Algorithme de Busacker-Gowen (1961)

Etape 1. Construction d'un flot initial `à coût minimum` (flot nul).
Etape 2. Chercher un circuit d'incrémentation `à coût minimum` z.
Etape 3. La valeur du coût total est incrémentée par δz.

Complexité : suivant les implémentations $O(n^4)$, $O(n^3)$ (Edmonds & Karp), $O(n^{2,5})$ (Hopcroft & Karp).

Exemple :

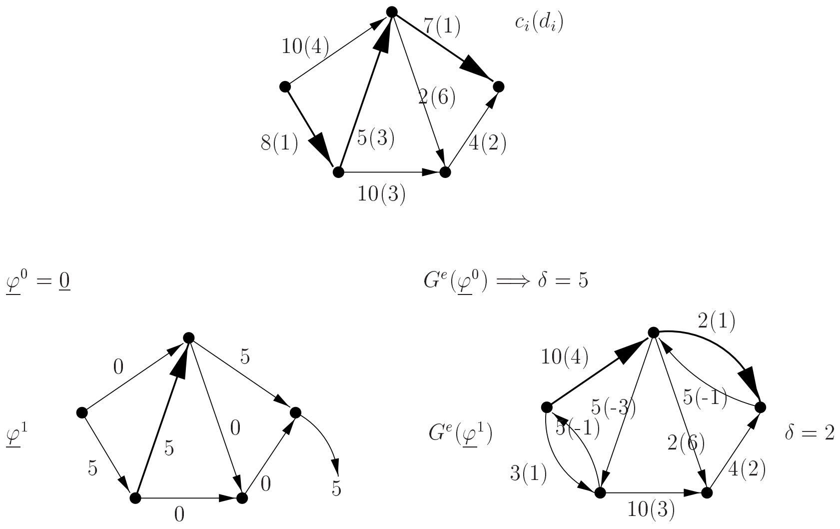


FIG. 4.6: Procédure de Busacker-Gowen

Remarque : Sur cet exemple, on peut obtenir la solution optimale en 2 itérations (au lieu de 4), en sélectionnant, en cas de choix multiples, la chaîne améliorante (circuit d'incrémentation) de coût minimum ET de capacité minimale la plus grande (afin d'améliorer plus rapidement la valeur du flot).

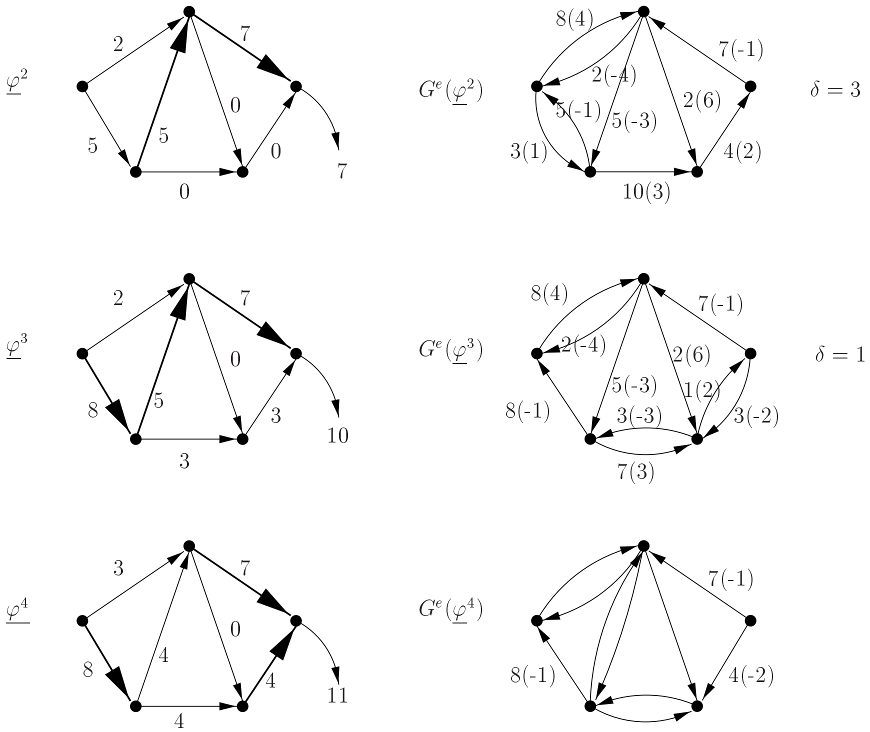


FIG. 4.7: Flot maximum à coût minimum (flot max = 11 ; coût total = $5 \times 5 + 2 \times 5 + 3 \times 6 + 1 \times 6 = 59$)

Chapitre 5

Couplages

Un *couplage* C d'un graphe simple $G = (X, E)$ est un sous-ensemble d'arêtes deux à deux sans extrémité commune. Le cardinal d'un couplage est le nombre d'arêtes composant le couplage.

Un sommet $x \in X$ est dit *saturé* par un couplage C si x est l'extrémité d'une des arêtes de C. Dans le cas contraire, le sommet est dit *insaturé*.

Un couplage est *parfait* si ses arêtes contiennent tous les sommets du graphe (en d'autres termes, un couplage sature tous les sommets de X).

Un couplage *maximal* de G est un couplage de cardinal maximal. Un couplage parfait est un couplage maximal. Un *problème d'affectation* désigne la recherche d'un couplage maximal dans un graphe biparti.

5.1 Problème du couplage maximal

Soit C un couplage de G. Une chaîne est dite *alternée* relativement au couplage C si elle emprunte alternativement des arêtes de C et des arêtes de $E \backslash C$. Une arête unique est une chaîne alternée de longueur 1.

Une chaîne alternée *augmentante* ou *améliorante* (CAA) joint deux sommets insaturés par C. Une arête non dans C est l'exemple le plus simple de CAA.

Exemple :

A gauche, un couplage parfait. A droite, un couplage non parfait; $[C, D]$ *et* $[H, A, B, C]$ *sont des CAA.*

Une opération de *transfert* sur une CAA consiste à échanger le rôle des arêtes du couplage ($\in C$) et celles qui ne font partie du couplage ($\notin C$). Une telle opération augmente la cardinalité du couplage d'une unité.

Exemple :

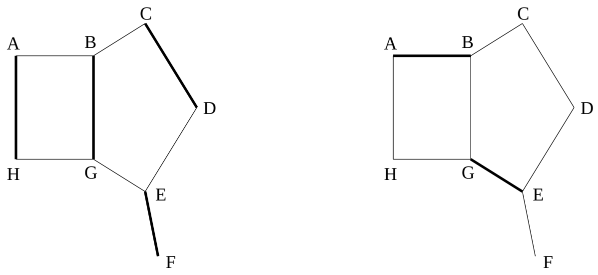


Fig. 5.1: CAA

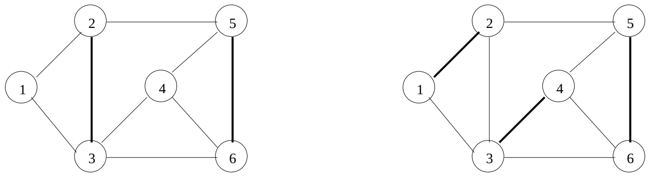


Fig. 5.2: Opération de transfert

<u>Théorème</u> : Un couplage C est maximal s'il n'existe pas de CAA relativement à C.

Algorithme de recherche d'un couplage maximal

1. $C \leftarrow \emptyset$
2. Répéter
   (a) <u>A partir d'une CAA</u> de C, effectuer un transfert. On obtient un couplage C' de cardinal $|C'| = |C| + 1$
   (b) $C \leftarrow C'$
3. jusqu'à ce qu'il n'existe plus de CAA.
4. C est un couplage maximal.

Le problème que l'on a est, partant d'un couplage donné, de <u>construire une CAA</u>. On construit pour cela une *arborescence alternée* :

1. On choisit comme racine de l'arborescence un sommet insaturé relativement au couplage courant.
2. Répéter
   (a) Au niveau i impair, on insère dans l'arborescence un sommet adjacent à l'un des sommets insérés au niveau $i - 1$ par le biais d'une arête **n'appartenant pas au couplage** courant, ainsi que cette arête.
   (b) Au niveau i pair, on insère dans l'arborescence un sommet adjacent à l'un des sommets insérés au niveau $i - 1$ par le biais d'une arête **appartenant au couplage** courant, ainsi que cette arête.

3. jusqu'à insertion d'un sommet insaturé à un niveau impair ou ce qu'il n'y ait plus de sommet à insérer.

   Exemple :

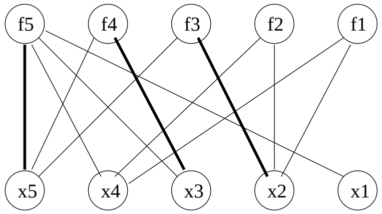


FIG. 5.3: Couplage courant

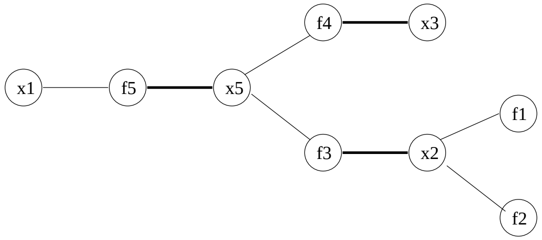


FIG. 5.4: Arborescence alternée de racine x_1

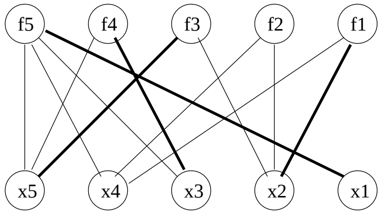


FIG. 5.5: Couplage augmenté

5.2 Conversion en problème de flot

Pour un graphe biparti $G = (X, Y, E)$, on peut convertir les problèmes de couplage en problèmes de flot dans un réseau $R = (X, Y, U, C, s, t)$ avec :

– on oriente les arêtes de G en arcs de X vers Y et on leur attribue une capacité infinie ;

– on ajoute une entrée s reliée à tout sommet de X par un arc de capacité 1 et une sortie t à laquelle tout sommet de Y est relié par un arc de capacité 1.

Exemple :

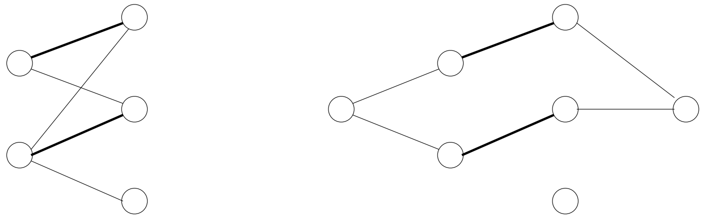


FIG. 5.6: D'un couplage à un flot

On peut notamment ramener le problème du couplage maximal en un problème de flot maximal dans un réseau de transport (résolu par exemple par l'algorithme de Busacker-Gowen).
On fait passer une unité de flot sur les arêtes du couplage et sur les arêtes correspondantes vers s et t.

Exemple :

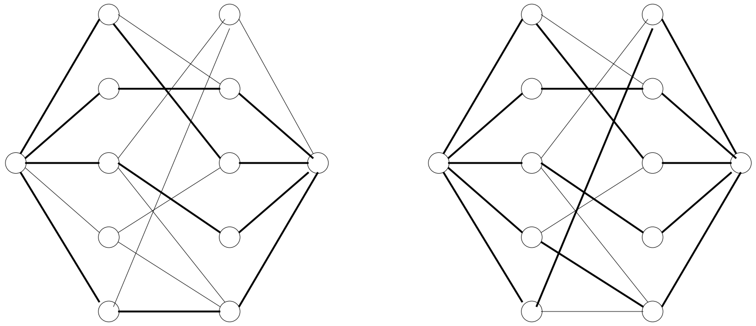


FIG. 5.7: Flots complet et maximal

5.3 Problème d'affectation

Etant donné n tâches à réaliser sur n machines, et sachant que le coût de réalisation de t_i sur m_j est C_{ij}, on cherche une permutation σ de $\{1, 2, \ldots, n\}$ telle que le coût total $\sum_{i=1}^{n} C_{i,\sigma(i)}$ soit minimum.
Le problème d'affectation peut être vu comme un problème de couplage parfait de poids minimum dans un graphe biparti.
Un algorithme spécifique, dit *algorithme Hongrois*, a été proposé par Kuhn.

Bibliographie

Claude Berge
Théorie des graphes et ses applications
Dunod, Paris, 1958.

Claude Berge
Graphes
Gauthiers-Villars, Paris, 1983.

Bernard Carré
Graphs and Networks
Clarendon Press, Oxford, 1979.

Thomas Cormen, Charles Leiserson et Ronald Rivest
Introduction à l'algorithmique
Dunod, Paris, 1994.

Gérard Desbazeille
Exercices et problèmes de recherche opérationnelle
Dunod, Paris, 1976.

Stuart E. Dreyfus
An appraisal of some shortest-path algorithms
Operations Research, 1969, 17, pp.395–412.

Salah E. Elmaghraby
Activity Networks
John Wiley & Sons, New-York, 1977.

Robert Faure
Précis de recherche opérationnelle
Dunod, Paris, 1979.

Gérard Fontan
Combinatoire et Ordonnancement
Notes de Cours, DEA Automatique et Informatique Industrielle, Université Paul Sabatier, Toulouse, 1987.

Michel Gondran et Michel Minoux
Graphes et algorithmes
Eyrolles, Paris, 1984.

Philippe Lacomme, Christian Prins, Marc Sevaux
Algorithmes de graphes
Eyrolles, Paris, 2003.

Didier Maquin
Eléments de théorie des graphes
Cours de l'Ecole Nationale d'Electricité et de Mécanique, INPL, 2003.

Christian Prins
Algorithmes de graphes (avec programmes en Pascal)
Eyrolles, Paris, 1994.

Bernard Roy
Algèbre moderne et théorie des graphes
Tome II, Dunod, Paris, 1984.

Aimé Sache
La théorie des graphes
Que sais-je ?, N^o 1554, Presses Universitaires de France, Paris, 1974.

M.M. Sysło, N. Deo and J.S. Kowalik
Discrete Optimization Algorithms (with Pascal programs)
Prentice Hall, 1983.