République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderahmane Mira de Béjaia

Faculté des Sciences Exactes

Département de Recherche Opérationnelle

MÉMOIRE PRÉPARÉ

En vue

d'obtention du diplôme Master en Recherche Opérationnelle

Option : Modélisation Mathématique et évaluation des performance des réseaux

Thème

Connexité et Flots dans un réseau

Présenté par :

Mr ABDELMALEK Hichem   Mr BELGACEM Feiçal

Devant le jury composé de :

Président : Dr KABYL Kamal

Encadreur : Mr TALEM Djamel

Examinateur : Mr TAOUINET Smail

Examinateur : Mr BOUAROURI Wahib

Année Universitaire 2018-2019

Remerciements

Nous tenons à remercier :

Le bon Dieu de nous avoir donné la patience et la volonté pour accomplir ce travail,

Nos remerciements s'adressent également à :

Notre promoteur Mr TALEM pour ses conseils, ses orientations pour nous avoir transmis les renseignements nécessaires à la réalisation de ce travail, et son aide durant l'encadrement.

Nous remercions également :

Les membres de jury, pour l'honneur qu'ils nous font en acceptant de juger de lire et d'évaluer ce mémoire.

Nous tenons également à remercier

Tous les enseignants de notre département qui nous ont accompagnés au cours de notre formation et à tout le personnel de la bibliothèque de l'université.

Je dédie modeste travail à :

Mes parents que le dieu les acceuils dans son vaste paradis ;

Mes soeurs ;

Mes fréres ;

Toute ma famille ;

Tout mes amis ;

Hichem

Je dédie modeste travail à :

Mon trés cher pére ;

Ma trés cher mére ;

Mes soeurs ;

Mes fréres ;

Toute ma famille ;

Tout mes amis ;

Feiçal

Table des matières

Liste des figures v

Introduction Générale 1

1 Défitions et notations 2

1.1 Graphe non orienté et graphe orienté 2

1.2 Sous graphe 6

1.3 Représentation des graphes 8

2 Problème, algorithme et complexité 11

2.1 Problème et algorithme [8] 11

2.2 complexité temporelle d'un algorithme 13

2.3 complexité d'un problème 15

3 Connexité dans un graphe 19

3.1 Connexité dans un graphe non orienté 19

3.2 Connexité dans les graphes orientés 20

3.2.1 Algorithme de calcul des composantes connexes [11] 21

3.2.2 Algorithme de recherche des composantes fortement connexes [11] 22

3.3 Sommet-connexité et arête-connexité 23

3.3.1 Graphe sommet-connexe 23

3.3.2 Graphe arête-connexe 23

3.4 Parcours 25

3.4.1 Exploration des graphes (Parcours) 25

3.4.2 Parcours d'un graphe 25

3.4.3 Structure de données 26
3.4.4 Parcours en profondeur DFS (Depth-First Search) 26
3.4.5 Parcours en largeur BFS (Breadth-First Search) 33

4 Flots dans les réseaux 36
4.1 Les Flots 37
4.1.1 Opérations sur les flots 37
4.1.2 Flot élémentaire : 38
4.2 Propriétés fondamentales 38
4.2.1 Flot maximal et coupe minimale 38
4.2.2 Capacité d'une coupe 38
4.2.3 Graphe d'écart et chemin augmentant 40
4.3 Algorithme de Ford-Fulkerson [16] 41
4.3.1 Théorème 4.3.1 (Ford et Fulkerson [1956], Dantzig et Fulkerson [1956]) . 41
4.3.2 Description 41
4.3.3 Optimisation linéaire 43
4.4 Théorème de Menger 43
4.4.1 Théorème de Menger (versions arc et arête).[7] 44
4.4.2 Théorème de Menger (version sommet, 1927)[7] 44

5 Flots de coût minimum 47
5.1 Formulation du problème 47
5.2 Le problème du flot compatible 48
5.2.1 Une condition nécessaire d'existence 48
5.2.2 Théorème du flot compatible(Hoffman 1960) 49
5.3 Algorithme d'obtention d'un flot de coût minimum .[6] 50
5.3.1 Organigramme de d'obtention d'un flot de valeur ϕ_0 donnée, de coût minimum [22] 51

6 Mise en œuvre du programme 55
6.1 Présentation du logiciel Matlab/Simulink(V6) 55
6.2 Implémentation de L'algorithme de Ford-Felkerson 56
6.2.1 Algorithme Général de Flot Maximal 56
6.3 Application numériques 57

Conclusion Générale **63**

Bibliographie **63**

Table des figures

1.1 Graphe non orienté 3

1.2 (a)Un digraphe G ,(b) Un graphe sous-jacent de G 4

1.3 a) Un graphe complet K_4, (b) Un graphe biparti complet $K_{2,3}$. 5

1.4 (a) Arbres (b) Une arborescence de racine c 6

1.5 (a) Un graphe G, (b) Un sous graphe partiel, (c) Un sous graphe induit par $A = \{a, c, e, d\}$ 7

1.6 Un stable maximal et un stable maximum 8

1.7 Graphe orienté et sa matrice d'incidence 9

1.8 Graphe non orienté et sa matrice d'incidence 9

1.9 Graphe non orienté et sa matrice d'adjacence 10

3.1 Recherche des composantes connexes 22

3.2 Recherche des composantes fortement connexes 23

3.3 Le graphe complet K_6 est 5-sommet-connexe 24

3.4 Le graphe complet K_5 est 4-arête-connexe 24

3.5 Le graphe biparti complet $K_{(1,7)}$ est 1-arête-connexe 24

3.6 Arborescence associée au graphe G 33

4.1 Un réseaux 36

4.2 Transformation d'un graphe par insertion de sommets 46

5.1 Graphe d'écart 52

5.2 Flot initial et graphe d'écart 52

5.3 Flot à l'itération numéro 1 et graphe d'écart 53

5.4 Flot à l'itération numéro 2 et graphe d'écart 53

5.5 Flot à l'itération numéro 3 et graphe d'écart 54

6.1 L'exécution 58
6.2 La solution 58
6.3 Réseau de transport avec les capacités 59
6.4 Le nouveau réseau résiduel 62

Introduction générale

La théorie des graphes est, avec la combinatoire, une des pierres angulaires de ce qu'il est commun de désigner par mathématiques discrètes. Cependant, elle n'a reçu qu'assez tardivement une attention soutenue de la part de la communauté mathématique. En effet, bien que les graphes eulériens soient l'émanation du célèbre problème des sept ponts de Königsberg étudié par Euler en 1736, on peut dire que les premiers développements majeurs de la théorie des graphes datent du milieu du vingtième siècle (N. Biggs, C. Berge, W.T. Tutte, . . .). Ainsi, un des premiers ouvrages, si pas le premier, traitant de théorie des graphes ≪ Théorie Der Endlichen Und Unendlichen Graphen ≫ écrit par König remonte à 1936. Depuis cette époque, la théorie des graphes s'est largement développée et fait à présent partie du cursus standard en mathématiques de bon nombre d'universités.[4]

Les graphes (et par conséquent la théorie des graphes) sont utilisés dans de nombreux domaines. On peut donner quelques exemples :

- Les réseaux de communication : réseaux de routes représentés par une carte routière, réseaux de chemin de fer, de téléphone, de relais de télévision, réseaux électriques, réseaux des informations dans une organisation.
- La gestion de projet : graphes potentiels-étapes plus connu sous le nom de graphes PERT ≪ Programme Evaluation and Research Task ≫ ou ≪ Programme Evaluation Review Technique ≫.

Chapitre 1

Défitions et notations

Dans ce chapitre nous présentons certaines notions de bases de la théorie des graphes qui seront utilisées dans la suite de ce mémoire.

1.1 Graphe non orienté et graphe orienté

On distingue deux types de Graphes : les graphes orientés et les graphes non orientés.

Un **graphe non orienté et fini** G est défini par un couple de deux ensembles (V_G, E_G) : V_G est un ensemble non vide de **sommets** (vertices) de G et E_G désigne l'ensemble de ses **arêtes** (edges) ; on écrit $G = (V_G, E_G)$ ou simplement $G = (V, E)$ s'il n'y a pas d'ambiguité.un sommet v est représenté par un point $p(v)$ dans le plan ou dans l'espace, tandis que une arête e est définie par un couple de deux sommets u et v , elle est représentée par une courbe reliant les deux points $p(u)$ et $p(v)$; on la note $e = \{u, v\}$ ou $e = uv$. Notons que uv et vu représentent la même arête. Les sommets u, v sont les extrémités (endpoints) de l'arête $e = uv$; si $u = v$, l'arête $e = uu$ est une **boucle** (loop) ; si deux sommets sont reliés par plusieurs arêtes, G est un **multi graphe** (multigraph).[2]

Un graphe G est **simple** (simple graph) lorsque il ne contient ni boucle ni arêtes multiples. Les nombres de sommets et d'arêtes de G sont notés $n(G)$ et $m(G)$; ces deux paramétres fondamentaux sont appelés l**ordre** et la **taille** de G, respectivement.Tous les graphes utilisés dans ce mémoire sont simples et finis (les paramètres n et m sont finis). La figure ci- dessous presente le diagramme d'un graphe d'ordre 6, simple et non orienté. $V(G) = \{a, b, c, d, e, f\}$ et $E(G) = \{ac, ad, ae, bc, be, bf, ce, cf, df\}$.

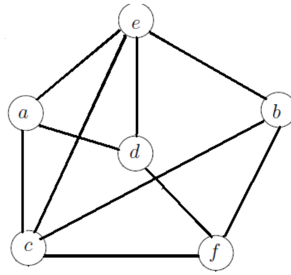


FIGURE 1.1 – Graphe non orienté

Les extrémités d'une arête sont dites **incidentes** à cette arête, et vice versa. Deux sommets incidents à une même arête sont **adjacents** (adjacent vertices) ou **voisins** (neighbors) ; de même, deux arêtes incidentes au même sommet sont dites adjacentes. L'ensemble des sommets adjacents à v, excepte lui-même, est appelé **voisinage ouvert** (open neighborhood) de v, il est noté $N(v) = \{u \in V, vu \in E\}$; Le voisinage **fermé** (closed neighborhood) d'un sommet v, noté $N[v] = N(v) \cup \{v\}$.

Le **degré** (degree) d'un sommet v, noté $d_G(v)$, est égal au nombre d'éléments dans $N(v)$, c'est-à-dire $deg(v) = |N(v)|$. Un sommet de degré nul est dit **isolé** (isolated vertex). Le plus petit degré d'un graphe G est noté $\delta(G) = min_{v \in V} d(v)$ et son plus grand degré est noté $\Delta(G) = max_{v \in V} d(v)$; ils sont, respectivement, égaux au degré d'un sommet ayant le moins et le plus de voisins. Un graphe est **régulier** si tous ses sommets ont le même degré, c'est-à-dire $\delta(G) = \Delta(G) = k$: on dit alors que le graphe est régulier de degré k ou k-régulier.

Dans la figure précédente, le sommet a est adjacent aux sommets c, d et e donc $N(a) = \{c, d, e\}$ et $N[a] = \{a, c, d, e\}$. les arêtes ac et cf sont adjacentes. $\Delta(G) = d(c) = 4$, $\delta(G) = d(a) = 3$.

Le **complémentaire** (complement) d'un graphe $G = (V, E)$ est le graphe $G^c = (V, E^c)$, c'est-à-dire G^c a les mêmes sommets que G et deux sommets sont adjacents dans G^c si et seulement s'il ne le sont pas dans G.

Un **graphe orienté** $\overrightarrow{G}$ ou **digraphe** (directed graph ou digraph) est un couple $(V_{\overrightarrow{G}}, U_{\overrightarrow{G}})$ formé d'un ensemble de sommets $V_{\overrightarrow{G}}$ et d'un ensemble de couples ordonnés $U_{\overrightarrow{G}} \subseteq V_{\overrightarrow{G}} \times V_{\overrightarrow{G}}$ dont les éléments sont appelés les arcs de $\overrightarrow{G}$. Ainsi, à tout arc a correspond un couple ordonné de deux sommets (u, v) tel que $a = (u, v)$; u est l'extrémité initiale de l'arc a, noté $I(a) = u$ et v son extrémité terminale, noté $T(a) = v$. Graphiquement, l'arc (u, v) est représenté par une courbe reliant les deux sommets u et v, orientée de u vers v (voir la figure 1.2). A tout digraphe

$\overrightarrow{G}$, nous pouvons associer un graphe G avec le même ensemble de sommets en remplaant chaque arc par une arête avec les mêmes extrémités. Ce graphe est le graphe **sous-jacent** de $\overrightarrow{G}$, noté $G(\overrightarrow{G})$. A l'inverse, tout graphe G peut être vu comme un digraphe, en remplaant chaque arête par deux arcs d'orientations opposées avec les mêmes extrémités ; ce digraphe est le digraphe associé à G, noté $\overrightarrow{G}(G)$.[2]

Tout concept valide pour les graphes non orientés, excepte ceux pour lesquels l'orientation joue un rôle essentiel, s'applique automatiquement aux digraphes. Par exemple, le degré d'un sommet u dans un digraphe $\overrightarrow{G}$ est simplement le degré de u dans le graphe sous-jacent de $\overrightarrow{G}$; un digraphe $\overrightarrow{G}$ est simple si son graphe sous-jacent G et simple.

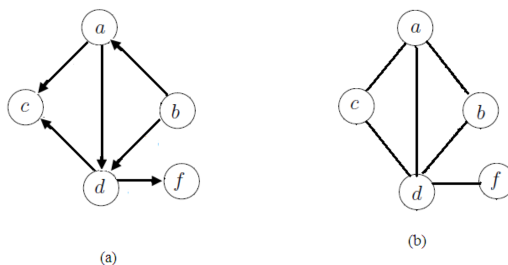


FIGURE 1.2 – (a)Un digraphe G ,(b) Un graphe sous-jacent de G

Pour tout sommet x de G, on associe :

- $d^+_G(x) = \{a \in U, I(a) = x\}$ le demi-degré extérieur de x, c'est le nombre d'arcs ayant x comme extrémité initiale.
- $d^-_G(x) = \{a \in U, T(a) = x\}$ le demi-degré intérieur de x, c'est le nombre d'arcs ayant x comme extrémité terminale.
- $d_G(x) = d^+_G(x) + d^-_G(x)$ le degré de x, c'est le nombre d'arcs ayant x comme extrémité.

Remarque 1.1.1. Comme pour les graphes orientés, on dit que l'arc $a = (x, y)$ est incident à x et y et que x et y sont adjacents. Une boucle est un arc a tel que $I(a) = T(a)$.

Un **chemin** dans un graphe non orienté $G = (V, E)$ est une séquence de sommets $\mu = (u_l, u_2, \ldots, u_q)$ telle que $u_i u_{i+1} \in E$ pour $i = 1, ..., q - 1$; on note $\mu[u_1, u_q]$ le chemin joignant le sommet u_1 au sommet u_q. Si $u_1 = u_q$, μ est un **circuit**. Une **chaîne** dans un graphe orienté $(\overrightarrow{G}, U)$ est une sequence de sommets $\mu = (u_1, u_2, ..., u_q)$ telle que $(u_i, u_{i+1}) \in U$ ou $(u_{i+1}, u_i) \in U$

pour $i = 1, ..., q-1$. Si $u_q = u_1$, μ est un **cycle**. Un chemin dans $(\overrightarrow{G}, U)$ est une sequence de sommets $\mu = (u_l, u_2, ..., u_q)$ telle que $(u_i, u_{i+1}) \in U$. Si $u_i = u_{i+1}$, μ est un circuit. Notons que dans un graphe non orienté, la notion d'une chaîne (resp. cycle) est la même que celle d'un chemin (resp. circuit) , car $\forall u, v \in V, uv = vu$.

Le nombre d'arcs de la séquence est la longueur de la chaîne (resp. chemin) μ. Une chaîne (resp. chemin, cycle, circuit) qui ne rencontre pas deux fois le même sommet est dite **élémentaire** ; une chaîne (resp. chemin, cycle, circuit) qui n'utilise pas deux fois la même arête (resp.arc) est dite **simple**.

Un graphe est **connexe** (connected) si, pour tout couple u, v de deux sommets, il y a une chaîne $\mu[u, v]$; dans le cas contraire, le graphe est séparé ou non connexe. Autrement dit, un graphe est séparé si son ensemble de sommets peut être partitionné en sous-ensembles connexes appelés **composantes connexes** (connected component) de G. Un sommet (resp. ensemble) d'**articulation** (cutvertex (resp. cutset)) est un sommet (resp. ensemble de sommets) tel que sa suppression augmente le nombre de composantes connexes. Parfois, on dit **séparateur** au lieu d'ensemble d'articulations.

Un graphe G est **complet** (complet graph) si entre deux sommets quelconques, il y a une arête ; K_n dénote un graphe complet d'ordre n. Un graphe **vide** est un graphe dans lequel l'ensemble d'arêtes est vide. Un graphe G est **biparti** (bipartite graph) si l'ensemble des sommets peut être partitionné en deux sous-ensembles V_1 et V_2 de sorte que toute arête ait une extrémité dans V_1 et une autre extrémité dans V_2 ; on le note $G = (V_1, V_2, E)$. Un graphe biparti est dit complet si tout sommet de V_1 est adjacent à tous les sommets de V_2. Un graphe biparti complet avec $|V_1| = p$, $|V_2| = q$ se dénote par $K_{p.q}$.

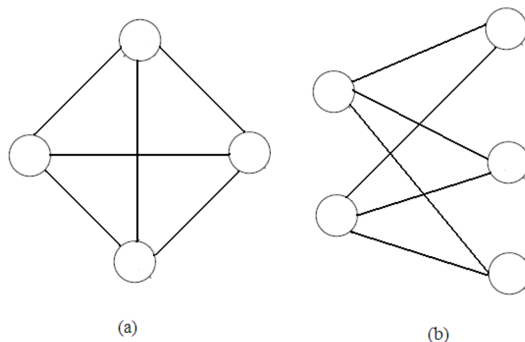


FIGURE 1.3 – a) Un graphe complet K_4, (b) Un graphe biparti complet $K_{2,3}$.

Un **arbre** (tree) est un graphe connexe sans cycle. La proposition suivante caractérise un arbre.

Proposition 1.1.1. les propriétés suivantes sont équivalentes :

1. G est un arbre ;
2. G sans cycle et posséde $n-1$ arêtes
3. G connexe et posséde $n-1$ arêtes
4. G sans cycle et si ajoutant une arêtes, on crée un et un seul cycle
5. G connexe et si on supprime une arête, il n'est plus connexe
6. Il existe une chaîne et une seule entre toutes paires de sommets.

Une **arborescence** est un graphe orienté sans cycle admettant une racine (ou une source) $s \in X$, c'est à-dire un sommet s vérifiant la propriété suivante : pour tout autre sommet $x \in X$,il existe un chemin unique allant de s vers x.

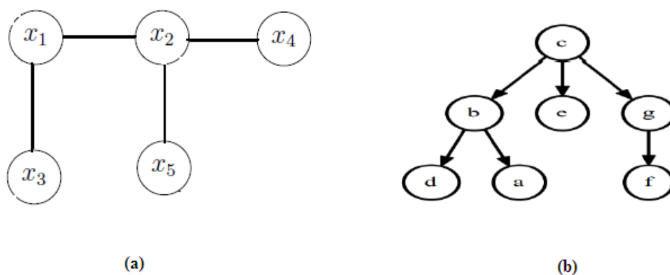


FIGURE 1.4 – (a) Arbres (b) Une arborescence de racine c

1.2 Sous graphe

Etant donné un graphe G, dans de nombreuses applications de théorie des graphes, on cherche à déterminer si un graphe donné a un **sous-graphe** (subgraph) avec certaines propriétés voulues. Il y a deux manières naturelles d'obtenir des graphes plus petits à partir de G : la suppression de sommets et la suppression des arêtes. Plus généralement, un graphe G' est un sous-graphe d'un graphe G si $V'_G \subseteq V_G$ et $E'_G \subseteq E_G$; G est alors un **sur-graphe** (supergraph) de G'. Ainsi nous disons que G contient G' ou que G' est contenu dans G.

Un graphe **partiel** (on dit aussi sous-graphe **couvrant**) (spanning subgraph) d'un graphe $G = (V, E)$ est un sous-graphe $G' = (V, E')$ tel que $E' \subset E$, c'est-à-dire G' est obtenu à partir

de G par suppressions d'arêtes uniquement. Si S est l'ensemble des arêtes supprimées, on écrit $G' = G \setminus S$. Par exemple, tout graphe simple est un sous-graphe couvrant d'un graphe complet, les chaînes et cycles Hamiltoniens (c'est-à-dire les chaînes, respectivement, les cycles qui passent par tous les sommets de G)sont des sous-graphes couvrants.

Un sous-graphe **induit** (induced subgraph) par un sous-ensemble de sommets A de V_G est le sous-graphe noté G_A dont l'ensemble de sommets est A et l'ensemble d'arêtes est constitué de toutes les arêtes de G qui ont leur deux extrémités dans A. Le sous-graphe induit G_A est le sous-graphe obtenu par la suppression consécutive des sommets de $V_G \setminus A$.

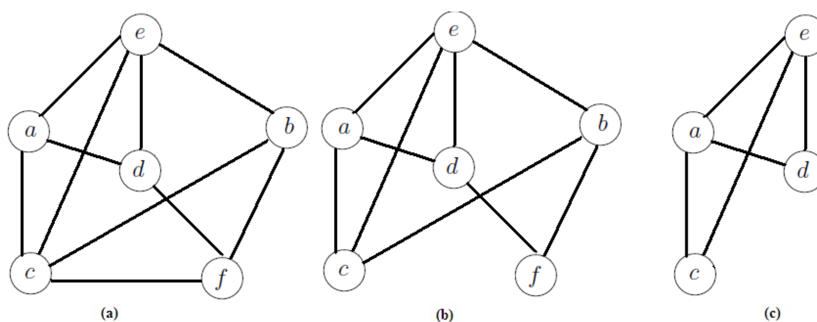


FIGURE 1.5 – (a) Un graphe G, (b) Un sous graphe partiel, (c) Un sous graphe induit par $A = \{a, c, e, d\}$

Un **stable** (independent set) dans G est un ensemble de sommets deux à deux non adjacents, c'est-à-dire un sous-graphe sans arêtes. Une **clique** (clique) dans G est un ensemble de sommets deux à deux adjacents, c'est-à-dire un sous-graphe complet. Une **biclique** (biclique) dans G est un sous graphe biparti complet.

Remarque 1.2.1. Ainsi un graphe complet (resp. graphe biparti-complet) est aussi une clique (resp.biclique). Une clique dans un graphe est un stable dans son complémentaire et inversement.

Soit $\mathfrak{F}$ une famille de sous-graphes d'un graphe G ($\mathfrak{F}$ peut être une famille de stables, ou de cliques ou de chaines ...). Un membre F de $\mathfrak{F}$ est **maximal** (resp.**minimal**) dans $\mathfrak{F}$ si aucun membre de $\mathfrak{F}$ ne contient proprement F (resp. n'est proprement contenue dans F) ; un membre F de $\mathfrak{F}$ est maximum (resp.minimum) dans $\mathfrak{F}$ si F est de cardinalité maximale (resp.minimale) dans $\mathfrak{F}$.

Notons que tout ensemble maximum est aussi maximal ; mais, un ensemble maximal n'est pas nécessairement maximum. En effet, dans la figure 1.6, l'ensemble des sommets $S = \{b, c\}$ induit un stable maximal d'ordre 2, car S n'est pas strictement inclue dans un autre stable ; cependant, ce dernier n'est pas maximum, car le stable induit par l'ensemble $S' = \{a, b, e\}$ est un stable d'ordre 3, donc contient plus d'éléments que S, et aucun autre stable ne peut avoir une cardinalité supérieure à 3 ; par conséquent S' est un stable maximum.

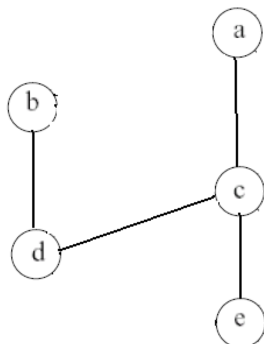


FIGURE 1.6 – Un stable maximal et un stable maximum

1.3 Représentation des graphes

Il existe deux façons classiques de représenter un graphe $G = (V, E)$: par un ensemble de listes d'adjacences, ou par une matrice d'adjacences. La représentation par listes d'adjacences est souvent préférée, car elle fournit un moyen peu encombrant de représenter les graphes peu denses (ceux pour qui m est trés inférieur à n^2).[2]

a) Matrice d'incidence sommet arc (Matrice d'incidence aux arcs)

La matrice d'incidence sommets-arcs est une matrice $n \times m$ associée à un graphe orienté et qui est définie par :

$$a_{ij} = \begin{cases} 1 & \text{si } x_i \text{ est l'extrémité initiale de } e_j \\ -1 & \text{si } x_i \text{ est l'extrémité terminal de } e_j \\ 0 & \text{sinon} \end{cases}$$

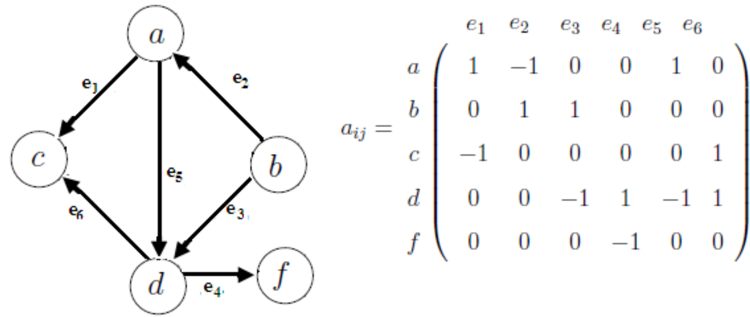


FIGURE 1.7 – Graphe orienté et sa matrice d'incidence

b) Matrice d'incidence sommets-arêtes d'un graphe non orienté

La matrice d'incidence sommet-arêtes est une matrice $n \times m$ associée à un graphe non orienté et qui est définie par :

$$b_{ij} = \begin{cases} 1 & \text{si le sommet } i \text{ est l'une des extrémité de l'arête } u_j \\ 0 & \text{sinon} \end{cases}$$

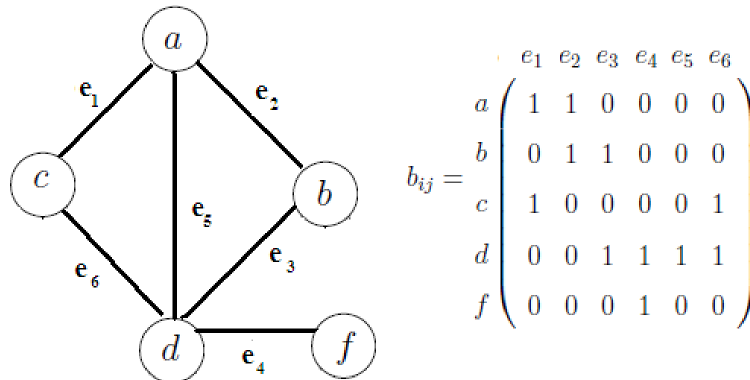


FIGURE 1.8 – Graphe non orienté et sa matrice d'incidence

c) Matrice d'adjacence (matrice d'incidence sommets - sommets) [19]

La matrice d'adjacence est une matrice carrée d'ordre n tel que chaque ligne et chaque colonne correspond à un sommet du graphe qui prennent les valeurs :

$$m_{ij} = \begin{cases} 1 & \text{s'il exite une arête } ij \text{ ; } i = \mathrm{I}(u) \text{ ; } j = \mathrm{T}(u) \\ 0 & \text{sinon} \end{cases}$$

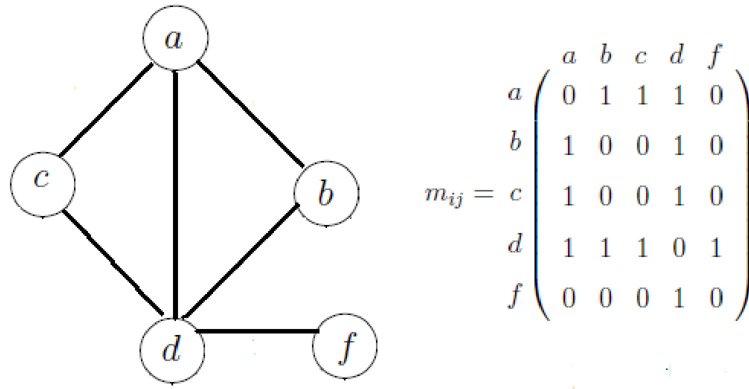


FIGURE 1.9 – Graphe non orienté et sa matrice d'adjacence .

Chapitre 2

Problème, algorithme et complexité

En général, dire qu'un problème admet une solution c'est equivalent à demontrer qu'il existe un programme qui le résout. Toutefois, rien n'implique que ce programme soit utilisable pratiquement : le temps et l'espace mémoire nécessaires à son exécution pourraient largement dépasser ce dont on peut espérer disposer.

Un algorithme utilisable se doit d'être relativement efficace ! Ce chapitre présente une théorie permettant de distinguer les problèmes solubles par un algorithme efficace de ceux qui ne le sont pas. Un algorithme est efficace s'il n'utilise qu'une quantité raisonnable des ressources nécessaires au calcul. Plus précisément, les ressources utilisées par un algorithme sont caractérisées par la fonction exprimant la quantité de ces ressources en termes de la taille de l'instance du problème traité (à savoir du mot représentant l'instance du problème).

Nous ferons alors l'hypothése que la frontirée entre (fonction de) complexité acceptable et inacceptable se situe à la limite entre fonction polynomiale et non polynomiale. Par exemple, la fonction $5n^2$ sera considéré comme acceptable alors que la fonction 2^n ne le sera pas.

Dans ce chapitre nous étudierons essentiellement la complexité en temps des algorithmes et n'arborderons pas la complexité en espace, cette dernière étant moins contrainte par l'explosion de la taille des mémoires des ordinateurs.

2.1 Problème et algorithme [8]

Un problème est une question génrique, c'est-à-dire qui s'applique à un ensemble d'éléments. Une instance (une donnée) du probleme est une question pose pour un élément particulier de cette ensemble.

Par exemple determiner si un entier naturel est premier ou calculer une clique de cardinalit maximal sont deux problemes. Dés qu'on fixe un entier, on aura une instance du premier probleme (connu sous le non du probléme de primalité) : 19 est-il un nombre premier ? est une instance ayant une reponse oui. De meme, la donnee d'un graphe G va constituer l'instance " calculer la clique maximum pour le graphe G".

Un probléme est donc composé de deux éléments : une entrée (ou instance) et une question ou une tâche à réaliser.

Les deux exemples précédents se reformulent ainsi :

1. Primalité :
   - entrée : un entier N ;
   - question : N est-il premier ?
2. clique de taille maximale :
   - entrée : Un graphe $G = (V, E)$;
   - tâche : Trouver une clique de G avec un maximum possible de sommets.

On distingue ainsi deux types de problèmes : ceux qui consistent à répondre par oui ou par non à une question donnée (dans l'exemple précédent, déterminer si un entier est premier), qu'on appelle problèmes de décision ; et ceux qui consistent à maximiser (ou minimiser) une certaine fonction sur un ensemble fini (dans l'exemple précédent, trouver $max(f)$ sur X, avec X est l'ensemble de toutes les cliques de G et f est une application sur X, qui à toute clique $C \in X$, $f(C)$ est le nombre de sommet de la clique C), qu'on appelle problèmes d'optimisation.

Ainsi, on obtient la définition suivante :

Définition 2.1.1

- Un problème d'optimisation combinatoire consiste à chercher le minimum (resp. le maximum) x^* d'une certaine application f sur un ensemble fini X à valeur le plus souvent entières ou réelles :

  $$f(x^*) = \min_{x \in X} f(x) \text{ (resp ; } f(x^*) = \max_{x \in X} f(x)).$$

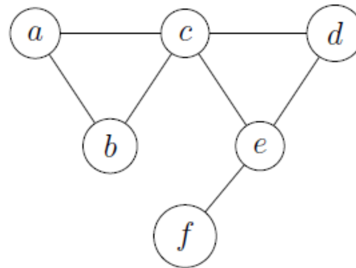
  La fonction f est une fonction-objectif ou économique, l'ensemble X est appelé ensemble des solutions réalisables.
- Un problème de décision (on dit aussi langage) consiste à chercher dans un ensemble fini X s'il y a un élément x vérifiant une certaine propriété P. Ainsi un problème de décision

est une application sur X à valeur dans $\{0,1\}$ telle que : $f(x) = 1$ si x vérifie P et $f(x) = 0$ sinon.

Notons que tout problème d'optimisation peut être transformé en un problème de décision équivalent. Par exemple, le problème du voyageur de commerce qui cherche, dans un graphe dont les arêtes sont étiquetées par des coûts, à trouver un cycle de coût minimum, passant une fois par chaque sommet, peut s'énoncer en un problème de dècision comme suit : Existe-t-il un cycle hamiltonien (passant une fois par chaque sommet) de coût inférieur à k ?

Comme on l'a signalé plus haut, résoudre un problème P signifie trouver un algorithme (ou programme) A qui prend en entrée une donnée d du problème P et produit en sortie un résultat $A(d)$.

Par exemple, un algorithme qui résout le problème du stable maximal, quand il s'applique sur le graphe de la figure ci-dessous, produit le résultat $S = \{b, e\}$ qui est un ensemble de sommets deux à deux non adjacents maximal.



Définition 2.1.2

Un algorithme est une méthode permettant de résoudre un problème donné en un temps fini. Il n'est pas un programme, il décrit une méthode qui sera ensuite implémentée dans un langage de programmation.

Si un problème doit être résolu par un algorithme, il faut que ses instances soient représentées d'une faon accessible à cet algorithme. Donc on a besoin de coder des entiers, des listes, des arbres, des graphes par des chaines de caractère qui soient compréhensibles par l'ordinateur. Le plus souvent, ces instances sont reprsentées par des chaines en 0 et 1.

2.2 complexité temporelle d'un algorithme

A tout algorithme est associée une fonction de complexité temporelle qui indique le temps de calcul à prévoir pour obtenir la solution. Pour une instance (une donnée) x d'un problème donné

p, le temps d'exécution de l'algorithme pour résoudre le probléme p sur l'entrée x ne s'exprime pas en seconde, mais il est donné par le nombre fini d'opérations élémentaires nécessaires jusqu'à l'affichage de la solution. Parmi les opérations élémentaires, on a les opérations arithmétiques ou logiques, les affectations, les comparaisons, ... Toute tâche qui se réalise en un temps constant par les calculateurs usuels indépendamment de leurs puissances est opération élémentaire.

En général, le temps d'exécution d'un algorithme sur l'entrée x dépend de la taille de x mais aussi de sa nature, et il peut varier sur des instances de même taille. En effet, rechercher une valeur dans un tableau demande plus de temps dans un tableau dont les éléments sont désordonnés que dans le même tableau trié . Pour cette raison, on définit la complexité d'un algorithme en considérant la pire instance possible parmi toutes les instances de taille n, c'est-à-dire celle demandant le plus de ressources.

Définition 2.2.1

Soient p un probléme et A un algorithme qui résout p. Notons $I_{(p,n)} = \{x/x$ est une instance de p et $|x| = n\}$ et φ_A est une application qui à toute instance x fait associer le temps d'exécution de A sur x .

La fonction de (ou tout court la complexité) de l'algorithme A est une application c_A définie sur l'ensemble des entiers naturels par :

$$c_A(n) = \max_{I_{(P,n)}} \varphi(n).$$

$c_A(n)$ est le nombre maximum d'opérations élémentaires pris par A sur les instances de taille n.

Remarque 2.2.1. Un algorithme de complexité polynômiale est quelquefois appelé bon ou efficace, et le problème qu'il résout est dit facile.

La complexité en temps d'un algorithme est habituellement exprimée à l'aide de la notation O. Sa définition est la suivante.

Définition 2.2.2 (Notation O)

Soient f et g deux fonctions $f; g : N \to R_+$. On dit que $f \in O(g)$ (on dit aussi f est un grand O de g) lorsqu'il existe un entier n_0 et une constante reelle c tel que pour tout $n \geq n_0$, $f(n) \leq cg(n)$.

Intuitivement, cela signifie que g devient plus grande que f à partir d'un certain entier n_0, à une constante multiplicative près. $O(g)$ est l'ensembles des fonctions d'ordre supérieur à g pour n assez grand. Par abus de langage, on écrira $f = O(g)$ là où on devrait ècrire $f \in O(g)$.[3]

Example :

Soit $f(n) = 6n^4 - 2n^3 + 5$. Choisissons $n_0 = 1$. Alors pour tout $n \geq n_0$, on a $6n^4 - 2n^3 + 5 \leq 6n^4 + 2n^4 + 5n^4 = 13n^4$ Ainsi, en prenant $c = 13$, on a $f = O(n^4)$. Autrement dit, à un facteur constant près, $f(n)$ ne croît pas plus rapidement que n^4. Il est facile de voir qu'un polynôme $P(n)$ de degré k est toujours en $O(n^k)$.

Définition 2.2.3 (Notations Ω, Θ) [3]

Soient f et g deux fonctions $f; g : \mathbb{N} \to \mathbb{R}_+$.

- On note $f = \Omega(g)$ lorsque il existe un entier n_0 et une constante reelle $c^{'}$ tel que pour tout $n \geq n_0$, $f(n) \geq c^{'}g(n)$.
- On note $f = \Theta(g)$ lorsque $f(n) = O(g)$ et $f(n) = \Omega(g(n))$, c'est-à-dire lorsque il existe un entier n_0 et deux constantes reelles c et $c^{'}$ tel que $cg(n) \leq f(n) \leq c^{'}g(n)$.

$\Omega(g)$ (respectivement $\Theta(g)$) est l'ensemble des fonctions d'ordre inférieur (respectivement équivalentes) à g pour n assez grand.

Example 2.2

Soit $f(n) = n^3 \sin n$, on a $\forall n$, $-n^3 \leq f(n) \leq n^3$. Ainsi $f(n) = \Theta(n^3)$.

Définition 2.2.4

Un algorithme A est dit polynômial si sa complexité est majorée par un polynôme en la taille des données, c'est-à-dire il existe un entier k tel que $c_A \in O(n^k)$. Dans le cas contraire, il est dit exponentiel.[12]

2.3 complexité d'un problème

Nous allons maintenant nous intéresser à l'étude de la difficulté que des problèmes de décision, ce que l'on appelle complexité des problèmes (non pas des algorithmes), et on va les classer selon la complexité des algorithmes les résolvant. Un grand nombre d'entre eux sont

des problémes faciles car on connait des algorithmes polynômiaux pour les résoudre. Cependant, il existe aussi un grand nombre de problèmes pour lesquels on ne connait pas d'algorithmes polynômiaux. On ne peut pas prouver qu'il n'en existe pas, mais on peut cependant montrer que l'existence d'un algorithme polynomial pour l'un dentre eux impliquerait l'existence d'un algorithme polynomial pour presque tous les problèmes.[12]

Définition 2.3.1

La complexité d'un probléme est la complexité du meilleur algorithme qui permet de le résoudre. Si cet algorithme est polynômial, le problème est dit facile, autrement le probléme est difficile.

Définition 2.3.2

La classe P est l'ensemble de tous les problèmes de décision pour lesquels il existe un algorithme polynômial.

Pour prouver qu'un problème est dans P, on décrit un algorithme polynômial résolvant ce problème.

Example 2.3 :

Soit le problème du plus court chemin entre deux sommets dans un graphe orienté et valué par des coûts positifs.

– Entrée : un graphe valué $G = (V, E, u)$, deux sommets s et t, et une constante k positive ;
– Sortie : exite-t-il un chemin allant de s à t de longueur au plus k.

Ce problème est dans la classe P car l'algorithme de Dijkstra, qui est un algorithme polynômial, peut résoudre ce problème.

Cependant, on ne sait pas si les problèmes de décision associés au problème du cycle hamiltonien au problème du stable maximum appartiennent à P ou pas. Nous allons maintenant introduire une autre classe, notée NP, qui contient ces deux problèmes.

Pour les problèmes de la classe NP, nous n'exigeons pas un algorithme polynômial, en revanche nous demandons qu'il y ait, pour chaque instance "oui", un certificat (une solution devinée) qui puisse être vérifié en temps polynômial.

Par exemple, pour le problème du cycle Hamiltonien, la solution est un cycle Hamiltonien, donc un cycle hamiltonien constitue un certificat. Comme il est facile de vérifier si un ensemble

d'arêtes donné forme un cycle hamiltonien, alors ce problème est dans NP.

La même chose pour le problème du stable de cardinalité au plus égale à k. En effet, ici la solution est un stable, et si l'on dispose d'un stable S (S est un certificat), on peut vérifier en un temps polynômial que S est un stable et aussi si $|S| \geq k$, ce qui implique que ce problème est dans NP.

Définition 2.3.3

La classe NP est l'ensemble de tous les problèmes de décision pour lesquels toute solution proposée est vérifiable par un algorithme polynômial.[10]

Remarque 2.3.1.

– Les problèmes de la classe NP sont ceux que l'on peut rèsoudre par énumération complète de toutes les solutions possibles (méthode "brutale") et en les testant à l'aide d'un algorithme polynômial.
– On a clairement $P \subseteq NP$. En effet, si on peut résoudre un problème par un algorithme polynômial, alors on peut aussi vérifier en temps polynômial que la solution fournie est bien une solution du même problème.
– La question de savoir si $P = NP$ est un problème ouvert, le plus important, de la théorie de la complexité. Cela revient à savoir si le fait de chercher une solution est aussi simple que de vérifier une solution. De nombreuses personnes pensent que $P \neq NP$.

Dans l'ensemble NP, on trouve un sous ensemble noté NP-complet constitué par les problèmes les plus difficiles de NP. Les problèmes $NP - complet$ sont tous équivalents en termes de difficultés. Pour affirmer que certains problèmes sont les plus difficiles, il faut pouvoir comparer les problèmes entre eux. On définit pour cela la notion de réduction polynômiale sur la classe NP.

Définition 2.3.4

Soient p et p' deux problèmes de décision. On dit que p se transforme (ou se réduit) polynômialement en p' s'il existe un algorithme polynômial transformant toute instance x de p en une instance x' de p' admettant la même réponse que x. On écrit alors $p \prec p'$. Autrement dit, les instances "oui" sont transformées en instances "oui", et les instances "non" sont transformées en instances "non".

L'importance du concept de réduction polynômiale est principalement justifiée par la proposition suivante :

proposition 2.3.1 [Optimisation combinatoire]

Si p se réduit polynômialement à p' et s'il existe un algorithme polynômial pour p', alors il existe un algorithme polynômial pour p.

- La relation $\prec$ est transitive et $p \prec p'$ signifie que p n'est pas plus difficile que p'.
- Ainsi, "p est polynômialement réductible à p'" signifie que si l'on connait un algorithme polynômial résolvant p', on en déduit que $p \in P$. En effet, on traduit les instances de p en instances de p', puis on résout p' et enfin on retraduit les solutions de p' en solutions de p, tout cela se fait en temps polynômial.

Définition 2.3.5 [13]

Un problème de decision q est dit $NP - complet$ si :

1. $q \in NP$
2. $\forall p \in NP$, $p \prec q$

Il est facile de voir que la restriction de la notion de "réduction polynômiale" sur les problèmes $NP - complet$ est une relation d'équivalence, donc s'il existe un algorithme polynômial pour un seul élément de la classe $NP - complet$, on pourait en déduire un algorithme polynômial pour n'importe quel autre élément dans la même classe, et d'après la proposition précédente, on aurait aussi $P = NP$!

Il a été demontré que de nombreux problèmes naturels réputés difficiles sont $NP - complets$. Cook [$Coo71$] et, indépendamment, Levin [$Lev73$] ont donné de tels problémes au début des années 1970, complétés ensuite par Karp [$Kar72$] notamment. Ici nous allons enoncé le théorème du problème SAT, la satisfaisabilité de formules booléennes.

Chapitre 3

Connexité dans un graphe

Une question importante est de savoir « à quel point » un graphe est connexe, c'est-à-dire combien de sommets ou d'arêtes on doit enlever pour déconnecter celui-ci. Cette notion de connexité est particulièrement pertinente pour analyser la « vulnérabilité » des réseaux.

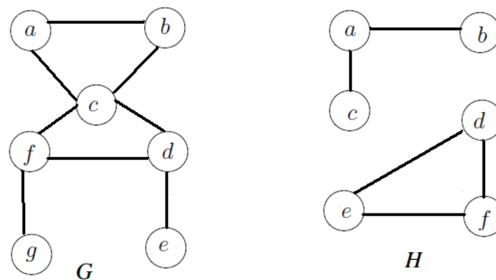
Exemple : un automobiliste doit se rendre de son domicile à son travail. Pour ce faire, il va emprunter le réseau routier ; une entreprise doit envoyer ses produits manufacturés sur le marché d'un pays donné, il utilisera divers réseaux : routier, maritime, aérien. . . Les réseaux sont des outils indispensables à notre société globalisée. Leur intérêt pratique dans différents domaines tels que les transports, les communications,la distribution est fondamental. Par conséquent une analyse mathématique de ces (di)graphes s'est développée, donnant naissance à la théorie des flots. Nous nous proposons ici d'introduire les propriétés élémentaires des réseaux. Dans tout le chapitre, les (di)graphes sont supposés **simples**, **d'ordre fini**.

3.1 Connexité dans un graphe non orienté

Définition 3.1.1

Un graphe non orienté est connexe s'il y a une chaîne entre n'importe quelle paire de sommets distincts du graphe.

Exemple 3.1.1



Le graphe G est connexe puisqu'il existe une chaîne entre n'importe quelle paire de sommets distincts.

Le graphe H n'est pas connexe ; car , il n'y a pas de chaîne entre les sommets a et d.

Définition 3.1.2

Un graphe qui n'est pas connexe est l'union de deux ou de plusieurs sous-graphes connexes, chaque paire de ceux-ci n'ayant pas de sommet en commun. **Les sous-graphes** connexes disjoints sont les **composantes connexes** du graphe.

Exemple 3.1.2

Dans la figure précédente, le graphe H contient deux composantes connexes : H_1 formé des sommets a, b et c et H_2 formé des sommets d, e et f.

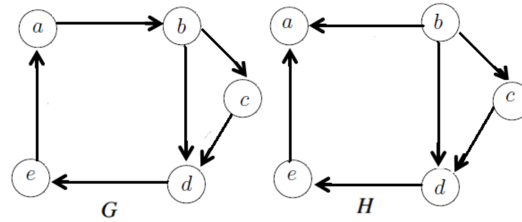
3.2 Connexité dans les graphes orientés

Définition 3.2.1

Un graphe orienté est **fortement connexe** s'il existe un chemin du sommet a au sommet b et du sommet b au sommet a, quels que soient les sommets représentés par a et b dans le graphe.

Un graphe orienté est **faiblement connexe** s'il y a une chaîne entre n'importe quelle paire de sommets dans le graphe si l'on ne considère plus l'orientation des arcs.

Exemple 3.2.1 :



Le graphe G est fortement connexe parce qu'il existe un chemin entre n'importe quelle paire de sommets dans ce graphe orienté . Par conséquent, G est également faiblement connexe.

Le graphe H n'est pas fortement connexe, car par exemple, il n'existe pas de chemin orienté de a vers b .

3.2.1 Algorithme de calcul des composantes connexes [11]

Soit $G(X, U)$ un graphe donné :

1. Initiation, $k = 0$, $W = X$ avec$X =\{$ ensemble de sommets de graphe G $\}$;
2. Choisir un sommet et marquer le d'un signe $(+)$, puis marquier tous ses voisins d'un singne $(+)$ (direct et indirect), continuer cette procédure jusqu'on ne puisse plus marquer d'autre sommets ;
3. Poser $k = k + 1$, et C_k l'ensemble des sommets marqués ;
4. Retirer de W les sommets C_k et poser $W = W - C_k$;
5. Tester si $W = \varnothing$
   - **si** oui terminer ;
   - **sinon** aller en (1).

Le nombre de composantes connexes est k et C_k sont les composantes connexes de G.
On applique l'algorithme précédent sur le graphe de la Figure 3.1 :

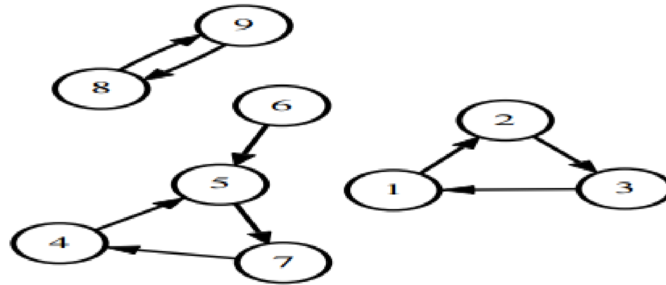


FIGURE 3.1 – Recherche des composantes connexes

- $k = 0$, $W = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $k = k + 1 = 1$,$C_1 = \{1, 2, 3\}$, $W = W - C_1 = \{4, 5, 6, 7, 8, 9\}$;
- $k = k + 1 = 2$, $C_2 = \{4, 5, 6, 7\}$,$W = W - C_2 = \{8, 9\}$;
- $k = k + 1 = 3$, $C_3 = \{8, 9\}$,$W = W - C_3 = \varnothing$, terminer.

Alors le nombre de composantes connexes de G est 3 et les composantes connexes sont : $C_1 = \{1, 2, 3\}$,$C_2 = \{4, 5, 6, 7\}$, $C_3 = \{8, 9\}$

3.2.2 Algorithme de recherche des composantes fortement connexes [11]

Soit $G = (X, E)$ un graphe orienté :

1. Initiation, $k = 0$, $W = X$ avec X = ensemble de sommets du graphe G ;
   Choisir un sommet et le marquer d'un signe (+) et (-).
2. Marquer tous ses succésseurs direct et indirect d'un singne $(+)$;
3. Marquer tous ses prédécesseurs direct et indirect d'un singne $(-)$;
4. Poser $k = k + 1$, C_k l'ensemble des sommets marqués d'un singne $(+)$ et $(-)$;
5. Poser $W = W - C_k$, et éffacer toutes les marques ;
6. Tester si $W = \varnothing$
   - **si** oui terminer ;
   - **sinon** pose $k = k + 1$ et aller en (1).

Le nombre de composantes fortement connexes est k et C_k sont les composantes fortement connexes de G.

On applique l'algorithme ci-déssus sur le graphe de la Figure 3.2, on obtient :

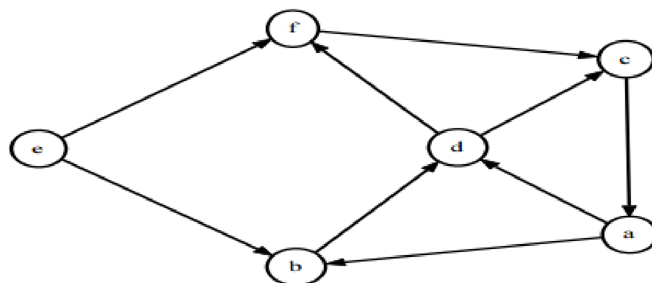


FIGURE 3.2 – Recherche des composantes fortement connexes

- $k = 0$, $W = \{a, b, c, d, e, f\}$
- $k = k + 1 = 1$, $C_1 = \{a, b, c, d, f\}$,$W = W - C_1 = \{e\}$,$W \neq \varnothing$,aller en (1) ;
- $k = k + 1 = 2$, $C_2 = \{e\}$, $W = W - C_2 = \varnothing$, terminer.

D'ou le nombre de composantes fortement connexes est 2 et fin sont : $C_1 = \{a, b, c, d, f\}$,$C_2 = \{e\}$.

3.3 Sommet-connexité et arête-connexité

3.3.1 Graphe sommet-connexe

Définition 3.3.1

un graphe connexe $G \ll$ est dit k-sommet-connexe s'il possède au moins $k + 1$ sommets et s'il reste encore connexe après en avoir ôté $k - 1$ sommets.

Un graphe régulier de degré k est au plus k-sommet-connexe et k-arête-connexe. S'il est effectivement k-sommet-connexe et k-arête-connexe, il est dit optimalement connecté

Exemple 3.3.1 :

- Pour tout n, le graphe complet K_n (régulier de degré n–1) est optimalement connecté.

3.3.2 Graphe arête-connexe

Définition 3.3.2

un graphe k-arête-connexe est un graphe connexe qu'il est possible de déconnecter en supprimant k arêtes et tel que ce k soit minimal. Il existe donc un ou plusieurs ensembles de k

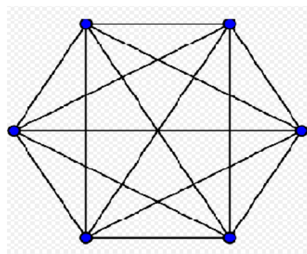


FIGURE 3.3 – Le graphe complet K_6 est 5-sommet-connexe

arêtes dont la suppression rende le graphe déconnecté, mais la suppression de $k-1$ arêtes, quelles qu'elles soient, le fait demeurer connexe.

Un graphe régulier de degré k est au plus k-arête-connexe et k-sommet-connexe. S'il est effectivement k-arête-connexe et k-sommet-connexe, il est qualifié de graphe optimalement connecté .

Exemple 3.3.2 :

- Pour tout n, le graphe complet K_n est optimalement connecté. Il est (n-1)-sommet-connexe, (n-1)-arête-connexe et (n-1)-régulier.

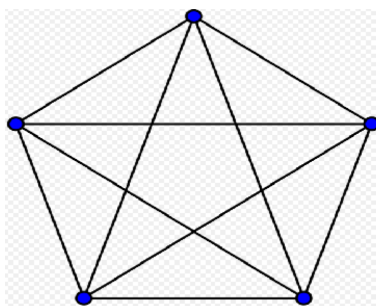


FIGURE 3.4 – Le graphe complet K_5 est 4-arête-connexe

- Le graphe biparti complet$K_{(1,n)}$ est 1-arête-connexe pour tout n.

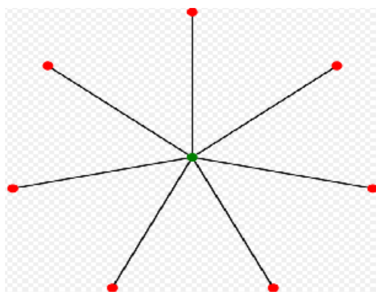


FIGURE 3.5 – Le graphe biparti complet $K_{(1,7)}$ est 1-arête-connexe

3.4 Parcours

Un réseau peut être représenté par un graphe non orienté. Pour assurer que deux noeuds (sommets) quelconques puissent communiquer, il faut évidemment que le graphe soit connexe. Il peut être utile de détecter d'éventuelles faiblesses d'un tel réseau. Si un noeuds cesse d'être opérationnel, il ne peut plus retransmettre les informations qui lui arrivent. Si elles peuvent prendre un autre chemin tout va bien. Mais si certaines informations doivent nécessairement passer par ce noeuds, il est à surveiller tout particulièrement.

3.4.1 Exploration des graphes (Parcours)

Les parcours servent comme outils pour étudier une propriété globale d'un graphe, comme la connexité et la forte connexité ...

Nous allons étudier dans cette section les deux principales stratégies d'exploration :

- Le parcours en largeur consiste à explorer les sommets du graphe niveau par niveau, à partir d'un sommet donné.
- Le parcours en profondeur consiste, à partir d'un sommet donné, à suivre un chemin le plus loin possible, puis à faire des retours en arrière pour reprendre tous les chemins ignorés précédemment

3.4.2 Parcours d'un graphe

On appelle **parcours** d'un graphe, tout procédé déterministe qui permet de choisir, à partir des sommets visités, le sommet suivant à visiter. Le problème de parcours consiste à déterminer un ordre sur les visites des sommets.

Racine

Le sommet de départ, fixé à l'avance, dont on souhaite visiter tous les descendants est appelé racine de l'exploration.

Propriété :

Un parcours de racine r est une suite L de sommets telle que :

- r est le premier sommet de L.

- Chaque sommet apparaît une fois et une seule dans L.
- Tout sommet sauf la racine est adjacent à un sommet placé avant lui dans la liste.

3.4.3 Structure de données

File

Une File est une structure de données basée sur le principe du "Premier entré, premier sorti $FIFO$, ce qui veut dire que les premiers éléments ajoutés à la $file$ seront les premiers à être récupérés.

Pile

Une pile est une structure de données basée sur le principe du "dernier arrivé, premier sorti $LIFO$, ce qui veut dire que les derniers éléments ajoutés à la *pile* seront les premiers à être récupérés.

Remarque :

La notion d'exploration peut être utilisée dans les graphes orientés comme non-orientés. Dans la suite, nous supposerons que le graphe est non-orienté. L'adaptation au cas des graphes orienté s'effectue sans aucune difficulté.

3.4.4 Parcours en profondeur DFS (Depth-First Search)

Principe :

Le principe du parcours en profondeur d'un graphe (orienté ou non) est celui du parcours d'un labyrinthe : on va de sommet en sommet en marquant au fur et à mesure les sommets visités. La visite se poursuit le plus loin possible tant qu'il reste des sommets accessibles non encore marqués. Quand on atteint un sommet v dont tous les voisins ont été déjà marqués alors on revient au sommet précédant v dans la visite.

Autrement dit on parcoure tous les sommets d'un graphe à partir d'un sommet v donné, à suivre un chemin le plus loin possible, puis à faire des retours en arrière pour reprendre tous les chemins ignorés précédemment.

Remarque :

Dans un parcours en profondeur, on applique la règle "Dernier marqué, premier exploré". i,e : on explore les sommets dans l'ordre inverse de celui utilisé pour les marquer.

Enoncé :

Dans l'algorithme de parcours en profondeur on procède au coloriage des sommets, on utilise une $pile(P)$.

- Initialement, tous les sommets $v \in V$ sont coloriés à blanc (traduisant le fait qu'ils n'ont pas encore été découverts "parcourus").
- Lorsqu'un sommet v_i est "découvert" (autrement dit, quand on arrive pour la première fois sur ce sommet), il est colorié en gris.

  Le sommet reste gris tant qu'il reste des successeurs de ce sommet qui sont blancs (qui n'ont pas encore été découverts).
- Un sommet v_i est colorié en noir lorsque tous ses successeurs sont gris ou noirs (lorsqu'ils ont tous été découverts).

Pratiquement on va utiliser une pile au coloriage en noir dans laquelle on va stocker tous les sommets gris : un sommet est mis dans la pile dès qu'il est colorié en gris. Un sommet gris dans la pile peut faire rentrer dans la liste ses successeurs qui sont encore blancs (en les coloriant en gris). Quand tous les successeurs d'un sommet gris de la pile sont soit gris soit noirs, il est colorié en noir et il sort de la pile.

La structures des données utilisées :

- On utilise une pile P, pour laquelle on suppose définir les opérations :
  - $initpile(P)$ qui initialise la pile P à vide.
  - $empile(P, v)$ qui ajoute v au sommet de la pile P.
  - $depile(P, v)$ qui enlève v des sommets de la pile P.
  - $estvide(P)$ qui retourne vrai si la pile P est vide et faux sinon.
  - $sommet(P)$ qui retourne le sommet v au sommet de la pile P.
- On utilise deux tableaux :
  - Un tableau T qui associe à chaque sommet le sommet qui l'a fait entrer dans la pile,
  - Un tableau couleur qui associe à chaque sommet sa couleur (blanc, gris ou noir).
- On va en plus mémoriser pour chaque sommet si :

- $dec[v_i]$ = date de découverte de v_i (passage en gris).
- $fin[v_i]$ = date de fin de traitement de v_i (passage en noir) où l'unité de temps est un itération.
- *tps* la variable dont La date courante est mémorisée.

Programme du parcours en profondeur DFS :

Algorithm de Depth First Search DFS algorithm [10]

Result :

T arborescence couvrante ; dec tableau des dates de découverte ;

fin tableau des dates de fin de traitement ;

Input :

– couleur tableau des couleurs des sommets ; tps date courante ;

– P pile ;

– $initpile(P)$;

– $i \leftarrow 0$;

while (tout sommet $v_i \in V$) **do**

$T[v_i] \leftarrow nil$; $couleur[v_i] \leftarrow blanc$; $d[v_i] \leftarrow 1$; $i \leftarrow i + 1$;

end

$tps \leftarrow 0$; $dec[v_0] \leftarrow tps$; $empile(P, v_0)$; $couleur[v_0] \leftarrow gris$;

while ($estvide(P) = faux$) **do**

$tps \leftarrow tps + 1$;

$v_i \leftarrow sommet(P)$;

if ($\exists u_j \in succ(v_i)$ tel que $couleur[v_j] = blanc$) **then**

$empile\ (P\,;\, v_j)$;

$couleur[v_j\ ] \leftarrow gris$;

$T[v_j] \leftarrow v_i$;

$dec[v_j] \leftarrow tps$;

else

/* tous les successeurs de v_i sont gris ou noirs */

$depile(P\,;\, v_i)$;

$couleur[v_j] \leftarrow noir$;

$fin[v_i] \leftarrow tps$;

end

end

Complexité de programme du parcours en profondeur DFS

- Espace : Représentation du graphe : $O(n+m)$.

  structure pile : $O(n)$.

  tableau marque :$O(m)$

  $\Longrightarrow$: complexité dans l'espace : $O(n+m)$.

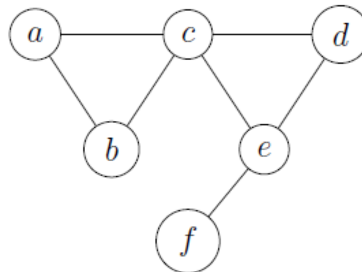
- Temps :marquage : n opérations.

  Exploration : chaque sommet x nécessite d_x opérations, donc en tout $\sum_{x \in V} d_x = 2|E| = 2m$

  $\Longrightarrow$: complexité en temps : $O(n+m)$. [12]

Exemple d'application

Soit le graphe $G = (X, E)$:



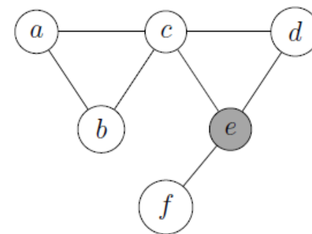
Itération 01

$T = \{e : Non\}$

$P = [e]$

Découverts **(gris ou noirs)**=$[e]$

Fermés = $[]$



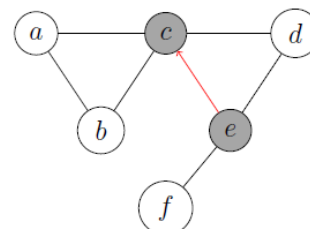
Itération 02

$T = \{e : Non, c : e\}$

$P = [e, c]$

Découverts **(gris ou noirs)**= $[e, c]$

Fermés = $[\varnothing]$



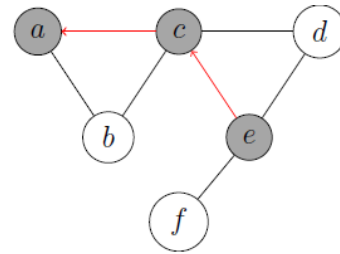
Itération 03

$T = \{e : Non, c : e, a : c\}$

$P = [e, c, a]$

Découverts **(gris ou noirs)**$= [e, c, a]$

Fermés $= [\varnothing]$



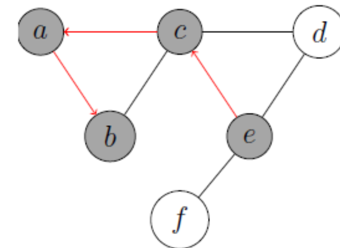
Itération 04

$T = \{e : Non, c : e, a : c, b : a\}$

$P = [e, c, a, b]$

Découverts **(gris ou noirs)**$= [e, c, a, b]$

Fermés $= [\varnothing]$



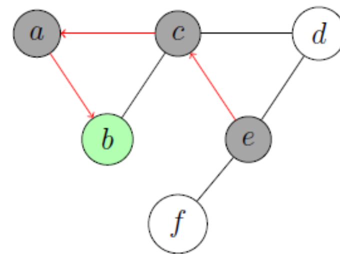
Itération 05

$T = \{e : Non, c : e, a : c, b : a\}$

$P = [e, c, a]$

Découverts **(gris ou noirs)**$= [e, c, a, b]$

Fermés $= [b]$



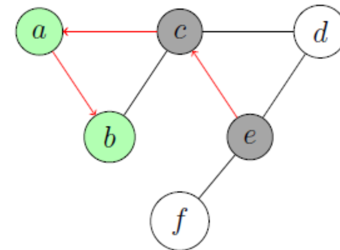
Itération 06

$T = \{e : Non, c : e, a : c, b : a\}$

$P = [e, c]$

Découverts **(gris ou noirs)**$= [e, c, a, b]$

Fermés $= [b, a]$



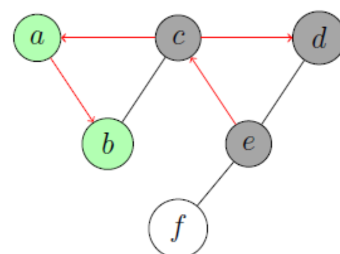
Itération 07

$T = \{e : Non, c : e, a : c, a : b, d : c\}$

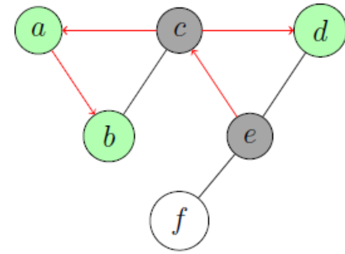
$P = [e, c, d]$

Découverts **(gris ou noirs)**$= [e, c, a, b, d]$

Fermés $= [b, a]$



Itération 08



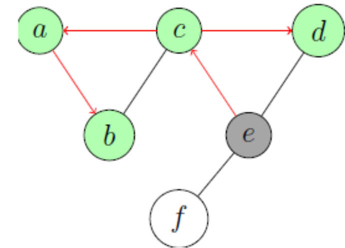
$T = \{e : Non, c : e, a : c, a : b\}$

$P = [e, c]$

Découverts **(gris ou noirs)**= $[e, c, a, b, d]$

Fermés = $[b, a, d]$

Itération 09



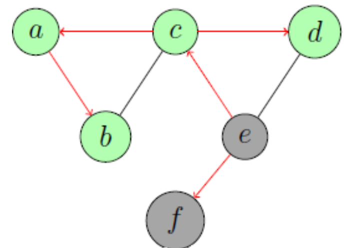
$T = \{e : Non, c : e, a : c, a : b, f : e\}$

$P = [e, f]$

Découverts **(gris ou noirs)**= $[e, c, a, b, d, f]$

Fermés = $[b, a, d, c]$

Itération 10



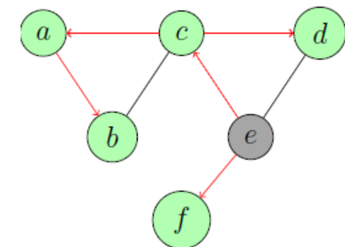
$T = \{e : Non, c : e, a : c, a : b, f : e\}$

$P = [e, f]$

Découverts **(gris ou noirs)**= $[e, c, a, b, d, f]$

Fermés = $[b, a, d, c]$

Itération 11



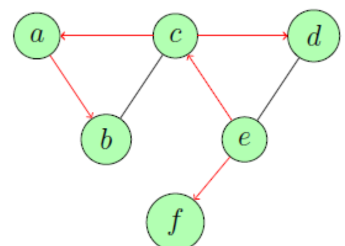
$T = \{e : Nonc, c : e, a : c, a : b, f : e\}$

$P = [e]$

Découverts **(gris ou noirs)**= $[e, c, a, b, d, f]$

Fermés = $[b, a, d, c, f]$

Itération 12



$T = \{e : Non, c : e, a : c, a : b, f : e\}$

$P = [\varnothing]$

Découverts **(gris ou noirs)**= $[e, c, a, b, d, f]$

Fermés = $[b, a, d, c, f, e]$

Arborescence associée au parcours :

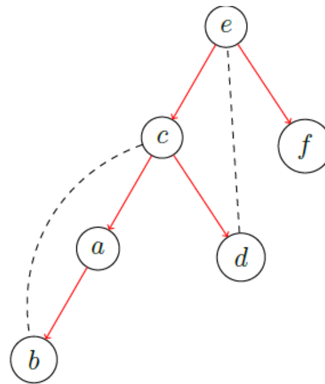


FIGURE 3.6 – Arborescence associée au graphe G

3.4.5 Parcours en largeur BFS (Breadth-First Search)

Principe :

Le principe est de visiter tous les voisins d'un sommet avant de visiter le sommet suivant qui sera le premier voisin à avoir été visité auparavant. Autrement dit on parcoure tous les sommets d'un graphe à partir d'un sommet de départ v (racine), on commence par visiter tous les successeurs de v avant de visiter les autres.

Le parcours en largeur est obtenu en gérant la liste d'attente au coloriage comme une file d'attente. Autrement dit, on enlève à chaque fois le plus vieux sommet gris dans la file d'attente, et on introduit tous les successeurs blancs de ce sommet dans la file, en les coloriant en gris.

Enoncé et structures de données utilisées :

On utilise une file F, pour laquelle on suppose définir les opérations $initfile(F)$ qui initialise la file F à vide, $ajoutefinfile(F; v)$ qui ajoute le sommet v à la fin de la file F, $estvide(F)$ qui retourne vrai **si** la file F est vide et **faux sinon**, et $enlevedebutfile(F; s)$ qui enlève le sommet v au début de la file F.

On utilise un tableau T qui associe à chaque sommet le sommet qui l'a fait entrer dans la file, et un tableau couleur qui associe à chaque sommet sa couleur **(blanc, gris ou noir).**[11]

Complexité de programme du Parcours en largeur BFS (Breadth-First Search)

- Espace : Représentation du graphe : $O(n+m)$.

  structure file : $O(n)$.

  tableau marque :$O(m)$

  $\Longrightarrow$: complexité dans l'espace : $O(n+m)$.

- Temps :marquage : n opérations.

  Exploration : chaque sommet x nécessite d_x opérations, donc en tout $\sum_{x \in V} d_x = 2|E|2m$

  $\Longrightarrow$: complexité en temps : $O(n+m)$.[12]

Exemple d'application :

On prend le graphe $G = (V, E)$ précédent :

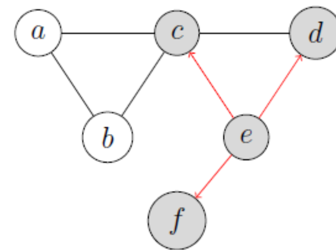
Itération 01

$T = \{e : Nonc, d : e, c : e, f : e\}$

$F = [e, d, f, c]$

Découverts **(gris ou noirs)**$= [e, d, f, c]$

Fermés $= [\varnothing]$



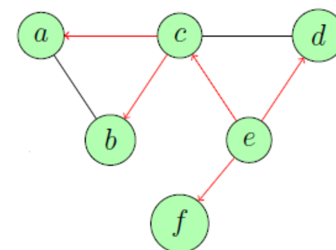
Itération 02

$T = \{e : Non, d : e, c : e, f : e, a : c, b : c\}$

$P = [\varnothing]$

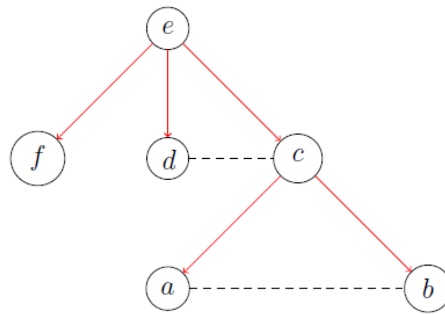
Découverts **(gris ou noirs)**$= [e, c, a, b, d, f]$

Fermés $= [e, d, f, c, a, b]$



Les sommets sont visités depuis e dans l'ordre $\{e, c, a, b, d, f\}$

Arborescence associée :



Conclusion

Ce chapitre a été consacré aux méthodes de résolution et différents algorithmes ainsi que quelques exemples pour mieux comprendre leurs résolution.

Chapitre 4

Flots dans les réseaux

Intuitivement un « flot » dans un réseau peut-être vu comme l'écoulement d'une substance le long des arêtes d'un graphe. Ainsi la circulation de pétrole dans un système de pipelines, d'eau dans des conduites, d'électricité dans les câbles, d'appels téléphoniques, de courriers électroniques, d'informations dans l'internet ou de véhicules peut être modélisée par un flot dans un réseau. La théorie des flots est ainsi devenue un aspect important de la théorie des graphes. Comme nous le verrons, cette théorie a aussi des applications dans d'autres domaines.

Définition 4 (Les réseaux)[16]

Un réseau est un graphe orienté $G = (X, U)$ avec une valuation positive de ses arcs. La valuation d'un arc (x, y), notée $c(x, y)$, est appelée la capacité de l'arc.

On distingue sur un réseau G deux sommets particuliers : un sommet dit source s et un autre dit destination t.

Les arcs de capacité nulle ne sont pas représentés sur le réseau.

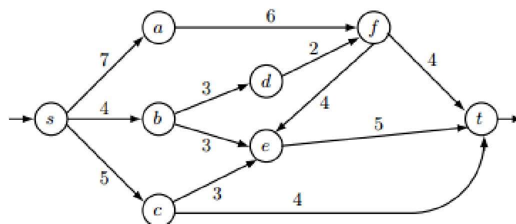


FIGURE 4.1 – Un réseaux

4.1 Les Flots

Un flot représente l'acheminement d'un flux, de matière par exemple, depuis une source s vers un puits t.

Un flot vérifie la loi de conservation analogue aux lois de Kirshoff (loi de conservation aux nœuds) est vérifiée, c'est à dire que le flux entrant à une nœud (ce qui arrive) doit être égal au flux sortant de ce nœud (ce qui repart).

Il n'est donc pas possible de stocker ou de produire de la matière aux nœuds intermédiaires .

Définition 4.1 (Le flot)

Un flot φ sur un réseau $G = (X, U)$ est une valuation positive φ des arcs, φ est une application définie de U dans $\mathbb{R}^+$, telle que : pour tout sommet $X - \{s, t\}$,

$$\sum_{u \in W^+(i)} \varphi_u = \sum_{u \in W^-(i)} \varphi_u \tag{4.1}$$

Le flux transitant sur chacun des arcs du réseau doit être inférieure à la capacité de cet arc (flot compatible ou admissible).

Un flot est dit compatible si pour tout arc $(x, y) \in U$, $0 \leq \varphi(x, y) \leq c(x, y)$

La valeur du flot est définie comme le flux net sortant de $\sum_{u \in W^+(s)} \varphi_u)$ ou entrant dans $(\sum_{u \in W^-(t)} \varphi_u)$.

4.1.1 Opérations sur les flots

Soient φ, φ_1 etφ_2 des flots sur G et $K \in \mathbb{R}$

Lemme 4.2.1

1. $K.\varphi$ est un flot sur G.
2. $\varphi_1 + \varphi_2$ est un flot sur G.
3. $\varphi_1 - \varphi_2$ est un flot sur G, si $\varphi_1 \geq \varphi_2$.

4.1.2 Flot élémentaire :

Soit C un circuit élémentaire sur G (c'est-à-dire, il passe au plus une fois par un sommet).

On appelle $\underline{v}$ le vecteur constitué par les éléments v_i tels que :

1. $v_i = 1$ si $u_i \in C$; $i = 1, ..., m$
2. $v_i = 0$ sinon

$\underline{v}$ est un flot cyclique élémentaire sur G.

Définition 4.1.1 (Flot maximum)

Un flot maximum dans un réseau est un flot compatible de valeur maximale.[16]

4.2 Propriétés fondamentales

4.2.1 Flot maximal et coupe minimale

Dans un réseau de transport $G = (X, U)$, soit S un sous-ensemble de X et T son complément. Une coupe $(s-t)$ se définit par une partition $X = S \cup T$ des sommets telle que $s \in S$ et $t \in T$. Les arcs de la coupe sont alors les arcs (x, y) ayant leur extrémité initiale dans S et leur extrémité terminale dans T.

4.2.2 Capacité d'une coupe

Définition 4.2.1 [15]

On appelle coupe séparant s et t, un ensemble d'arcs de la forme :

$w^+(A)$ où $A \subset X$ est un sous-ensemble de sommets tel que $s \in A$ et $t \in X \backslash A$.

On définit la capacité de la coupe $w^+(A)$ comme la somme des capacités des arcs qui la constituent :

$$C(A, X\backslash A) = \sum_{u \in w^+(A)} c_u$$

Lemme 4.2.1

La valeur maximale d'un flot de s et t compatible avec les capacités c_u ne dépasse jamais la capacité d'une coupe séparant s et t.

Preuve

Soit $A \subset X$ quelconque tel que $s \in A$ et $t \in X \backslash A$.

Soit $\varphi' = \varphi_0, \varphi_1, \varphi_2, ..., \varphi_m$ un flot quelconque dans G^0. D.après la loi de conservation aux nœuds pour φ on a :

$$\sum_{u \in w^-(A)} \varphi_u = \sum_{u \in w^+(A)} \varphi_u$$

Or l'arc u_0 est un arc de $w^-(A)$, on peut écrire :

$$\varphi_0 + \sum_{(u \in w^-(A)) u \neq 0} \varphi_u = \sum_{u \in w^+(A)} \varphi_u$$

d'où

$$\varphi_0 = \sum_{u \in w^+(A)} \varphi_u - \sum_{(u \in w^-(A)) u \neq 0} \varphi_u$$

Comme $0 \leq \varphi_u \leq c_u$; $\forall u \in U$, on a :

$$\sum_{u \in w^+(A)} \varphi_u \leq \sum_{u \in w^+(A)} c_u, et \sum_{u \in w^-(A)} \varphi_u > 0$$

on peut écrire :

$$\varphi_0 \leq \sum_{u \in w^+(A)} \varphi_u \leq \sum_{u \in w^+(A)} c_u$$

Ceci est vrai $\forall A \subset X$, pour tout flot φ compatible avec les capacités c_u et, en particulier pour le flot donnant à φ_0 une valeur maximale. Donc :

$$\max \varphi_0 \leq \min \left\{ \sum_{u \in w^+(A)} c_u \right\}$$

Corollaire 4.2.1

Si un flot φ et une coupe $w^+(A)$ sont tels que la valeur φ_0 du flot est égale à la capacité de la coupe, alors φ est un flot maximum de s à t et $w^+(A)$ est une coupe de capacité minimale séparant s et t.

Corollaire 4.2.2

Une condition nécessaire et suffisante pour que le problème du flot maximum de s à t dans G ait une solution de valeur finie, est qu'il n'existe pas de chemin de capacité infinie entre s et t.

Corollaire 4.2.3

Soit φ un flot et K une coupe. Si $\varphi = cap(K)$, alors φ est un flot maximum et K est une coupe minimum.

Théorème 4.2 (Théorème du flot-max et de la coupe-mini)

La valeur maximale d'un flot de s à t dans $G = (X, U)$ muni des capacités c_u avec $u \in U$ est égale à la capacité d'une coupe de capacité minimale séparant s et t.[**?**]

Proposition 4.1

Le Probléme Du Flot Maximum a toujours une solution optimale.[18]

4.2.3 Graphe d'écart et chemin augmentant

Définition 4.2.1 (Graphe d'écart)

Soit $\varphi = (\varphi_1, \varphi_2, ..., \varphi_m)^T$ un flot entre s et t dans $G = (X, U)$ compatible avec les contraintes de capacité $0 \leq \varphi_u \leq C_u$, $(\forall u = 1, ..., m)$.

Le graphe d'écart associé à φ est le graphe $\overline{G}(\varphi) = (X, U(\varphi))$ ayant le même ensemble de sommets que G et dont l'ensemble des arcs, $\overline{U}(\varphi)$ est constitué de la façon suivante : à chaque arc $u = (i, j) \in U$ de G on associe au plus deux arcs de $G(\varphi)$

$$u^+ = (i, j) \text{ si } \varphi_u < C_u; u^-(j, i) \text{ si } \varphi_u > 0$$

On associe également à chaque arc de $\overline{G}(\varphi)$ une capacité (capacité résiduelle) égale à $C_u - \varphi_u = 0$ dans le premier cas à $\varphi_u > 0$ dans le second cas.

Soit φ un flot admissible sur G et $G'(\varphi) = (X, U'(\varphi))$le graphe d'écart associé. Soit μ un chemin allant de s à t dans $G'(\varphi)$ et $\varepsilon = \min_{u \in \mu} c(u)$. Ce chemin est appelé **chemin augmentant** car il est possible d'augmenter la valeur du flot sur G de ε de la façon suivante $\forall (i, j) \in \mu$:

– Si $u = (i, j) \in U$, alors $\varphi(u) = \varphi(u) + \varepsilon$
– Si $u = (j, i) \in U$, alors $\varphi(u) = \varphi(u) - \varepsilon$

Théorème 4.3 [18]

Un flot φ de s à t est maximum si et seulement s'il n'existe pas de chemin φ-augmentant.

4.3 Algorithme de Ford-Fulkerson [16]

L'algorithme de Ford-Fulkerson construit un flot et détermine une coupe qui vérifient le critère d'optimalité.

4.3.1 Théorème 4.3.1 (Ford et Fulkerson [1956], Dantzig et Fulkerson [1956])

Si les capacités sont entiéres, alors la valeur de l'ALGORITHME DE FORDFULKERSON est toujours entier. Puisqu'il existe un flot maximum de valeur finie (proposition 4.1), l'algorithme se termine aprés un nombre fini d'itérations. Nous avons donc l'importante conséquence suivante :

Corollaire 4.3.1 (Dantzig et Fulkerson [1956])

Si les capacités du réseau sont entiéres, il existe un flot maximum entier.

Ce corollaire – appelé quelquefois le théoréme du flot entier – peut également se démontrer en utilisant la totale unimodularité de la matrice d'incidence d'un graphe orienté .

4.3.2 Description

La coupe s'obtient en réalisant une partition (dynamique) des sommets en sommets marqués et non marqués.

Obtention d'un flot complet Le flot est construit par améliorations successives jusqu'à l'obtention d'un flot complet .

Définition 4.4.1

Un flot est dit complet si tout chemin allant de s à t comporte au moins un arc saturé Tant qu'il existe un chemin de a à b n'ayant aucun arc saturé, on peut améliorer le flot sur ce chemin.

On le détermine de façon systématique en balayant les arcs depuis chaque nœud, dans un ordre convenu (arbitraire).

Marquage des sommets

Le marquage des sommets tente de construire une chaîne allant de s à t. Sur cette chaîne,certains arcs seront parcourus dans le bon sens (arc progressifs) et d'autres en sens contraire (régressifs), choisis d'après la règle :

- Un arc progressif v doit vérifier : $\varphi(v) < C(v)$
- Un arc régressif w doit vérifier : $\varphi(w) > 0$

- Si on arrive à marquer t, la chaîne ainsi construite est dite améliorante, car on peut augmenter le flot :
  - En ajoutant une unité sur tout arc progressif v,
  - En retranchant une unité sur tout arc régressif w

  La loi de Kirchhoff reste vérifiée sur les sommets de la chaîne, sauf en s et en t, où on a ajouté une unité.
- S'il est impossible de marquer b, cette partition des sommets (en marqués et non marqués)détermine une coupe d'arcs (d, f) ayant, par construction, la propriété suivante :
  - Si c'est d qui est marqué, alors $\varphi(u) = C(u)$,
  - Si c'est f , alors $\varphi(u) = 0$.

    i.e. :

$$\varphi(K) = C(K)$$

Théorème 4.4.2 :

Quand l'algorithme se termine, φ est un flot de s à t maximum.

Preuve.

φ est un flot de s à t puisqu'il n'y a pas de sommets actifs, il n'existe pas de chemin augmentant, φ est maximum.

En résumé

L'algorithme est fini Il y a un arrêt des itérations, car :

- Le nombre de sommets est fini (et le graphe supposé sans circuit),
- On ne peut augmenter le flot que jusqu'à la limite des capacités.

L'arrêt définit une coupe qui est l'ensemble K des arcs ayant une seule marque : soit à leur origine (d), soit à leur extrémité (f).

Le flot est maximum par obtention d'une coupe minimale.

Remarque 4.4.1 La coupe ainsi déterminée n'est pas forcément unique.

C'est à dire qu'il ne sert à rien d'augmenter les capacités des arcs de cette coupe : l'optimum ne changera pas s'il se produit dans d'autres.

4.3.3 Optimisation linéaire

Formulation primale [17]

Le problème du flot maximum est un problème d'optimisation linéaire sous contraintes linéaires et on peut se proposer de le résoudre comme tel.

Soient :

$$\Phi = (\Phi_1, \Phi_2, ..., \Phi_m)$$
$$M(n, m) = \text{matrice d'adjacence}$$

alors le problème s'écrit :

$$\max \varphi = (M\Phi | e_i)$$
$$\Phi \geq 0$$
$$\Phi \leq C$$

4.4 Théorème de Menger

Nous sommes maintenant en mesure de donner une version en terme de cardinal minimal d'ensembles arc-séparants (respectivement arête-séparants) du théorème de Menger. Rappelons que les (di)graphes considérés sont supposés sans boucle. On dit que deux chaînes sont arête-disjointes si elles n'ont aucune arête commune.

4.4.1 Théorème de Menger (versions arc et arête).[7]

i) soit $D = (X, E)$ un digraphe simple et s, t deux sommets non adjacents de D. Alors le nombre maximum de chemins élémentaires arc-disjoints entre s et t est égal au cardinal d'un ensemble arc-séparant minimum pour s et t.

ii) Soit $G = (X, E)$ un graphe simple et s, t deux sommets non adjacents de G. Alors le nombre maximum de chaînes élémentaires arête-disjointes entre s et t est égal au cardinal d'un ensemble arête-séparant minimum pour s et t.

Démonstration.

i) S'il n'existe aucun chemin entre s et p, le résultat est immediat. Sinon, on définit une capacité c en posant $c(a) = 1$ pour tout $a \in E$;
(X, E, s, t, c) n'est pas forcement un réseau, car s et t ne sont pas nécessairement une source et un puits, mais il suffit d'ajouter, si nécessaire, deux sommets s_0 et t_0 ainsi que deux arcs reliant respectivement s_0 à s et t à t_0 pour obtenir réellement un réseau .

Soit k le nombre maximum de chemins élémentaires arc-disjoints entre s et t, chemins que l'on designe par $C_1, ..., C_k$ et, soit l le cardinal minimum d'un ensemble arc-séparant pour s et t.

ii) Le cas des graphes se raméne au cas des digraphes de la maniére suivante : on remplace toutes les arêtes du graphe G par deux arcs de directions opposees. On obtient ainsi un digraphe note $\overrightarrow{G}$. Un systeme de chaines élémentaires arête-disjointes entre s et t induit un systeme de chemins élémentaires arc-disjoints entre s et t. De maniére réciproque, à tout systeme de chemins élémentaires arc-disjoints entre s et t dans $\overrightarrow{G}$ correspond un systeme de chaînes élémentaires arête-disjointes entre s et t dans G.

Soit k le nombre maximal de chaînes élémentaires arête-disjointes entre s et t dans G. Alors k est également le nombre maximum de chemins élémentaires arc-disjoints entre s et t. Il existe donc un ensemble arc-séparant A dans $\overrightarrow{G}$. de cardinal égal à k. Cet ensemble A est également un ensemble arête-séparant dans G.

4.4.2 Théorème de Menger (version sommet, 1927)[7]

Soit D un digraphe (respectivement un graphe).. Soient s et t deux sommets non adjacents de D. Alors le nombre maximum de chemins (respectivement de chaines) élémentaires sommet-

disjoints entre s et t est égal au cardinal d'un ensemble sommet-separant minimum pour s et t.

Démonstration.

Nous démontrons le résultat dans le cas d'un digraphe. Le cas d'un graphe G en découle facilement en appliquant le résultat a un digraphe dont le graphe sous-jacent est G.

Soit $D = (X, U)$ un digraphe. Nous allons transformer ce digraphe afin de pouvoir utiliser le théorème 4.4.1.

Définissons un digraphe D' de la maniere suivante :

- **les sommets de D' sont :** s, t et tout sommet $x \neq s$, p de G est dédoublé en deux sommets x' et x" ;
- **les arcs de D' sont :** tout arc de s à x dans D est remplace par un arc de s à x' dans D'. Tout arc de x à t dans D est remplace par un arc de x" à t dans D'. Tout arc de x à y dans D avec $x, y \neq s$, t est remplacé par un arc de x" à y'. Enfin entre chaque paire de nouveaux sommets x' et x", on rajoute un arc.

Une illustration de cette transformation est donnee en Figure 4.2. Il est clair que des chemins élémentaires sommet-disjoints de D deviennent des chemins élémentaires arc-disjoints de D. Réciproquement, deux chemins élémentaires non sommet-disjoints de D induisent deux chemins élémentaires de D' qui ne sont pas arc-disjoints.

Par consequent, d'apres le Théorème 4.4.1, le nombre maximum de chemins élémentaires sommet-disjoints entre s et t dans D est égal au cardinal d'un ensemble arc-separant minimum A entre s et t dans D. En remplacant chaque arc de A de x" vers y' par un arc de x' vers x", on obtient un ensemble A' qui reste arc-separant entre s et t et de même cardinal que A par minimalite.

Ainsi on peut se restreindre aux ensembles arc-separants minimums A' de D' ne contenant que des arcs de la forme $(x', x")$. Ceux-ci correspondent naturellement aux ensembles sommet-separants de D en identifiant les arcs de x' à x" aux sommets x dont ils proviennent. L'objectif c'est de demontrer T.Menger, En utilisant un flot max et coup min.

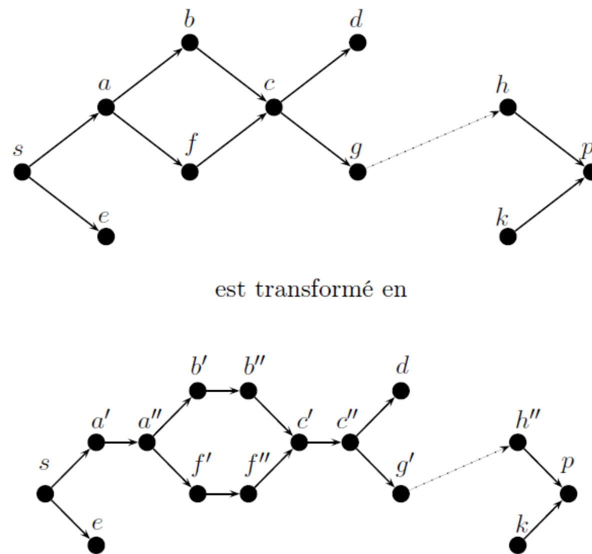


FIGURE 4.2 – Transformation d'un graphe par insertion de sommets

Les chemins (a, b, c, d) et $(a, f, c, g, ..., h)$ ont un sommet commun c dans D. Ils deviennent $(a', a", b', b", c', c", d)$ et $(a', a", f', f", c', c", g', ..., h")$ dans D' et ont une arête commune $(c', c")$.

Chapitre 5

Flots de coût minimum

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

L'outil de base reste la chaîne améliorante. Nous la simplifions en introduisant la notion de graphe d'écart qui permet de se ramener à la recherche de chemins améliorants.

Définition 5.1 (Problème de flot à coût minimum)[19]

On appelle problème de flot à coût minimum le programme linéaire :

$$\sum_{i,j)\in U} c_{ij}x_{ij}$$

avec :

c_{ij} :capacité d'arc

x_{ij} :coût

Le problème de flot à coût minimum admet plusieurs sous-problèmes importants.

5.1 Formulation du problème

Soit $R = (X, U, C)$ un réseau avec capacité. Supposons qu'à chaque arc $(i, j) \in U$ est associé un nombre réel f_{ij} qui représente le coût unitaire de mouvement de la matière le long de cet

arc. Le coût total d'un flot réalisable φ, noté $f(\varphi)$ est :

$$f(\varphi) = \sum_{(i,j)\in U} f_{ij}\varphi_{ij}$$

Considérons l'ensemble des flots réalisables Φ de même valeur ϕ_0, $0 \leq \phi_0 \leq \phi_M$ où $\phi_M = \max\{\varphi_0, \varphi \in \Phi\}$. Un flot de cet ensemble est dit de coût minimal s'il minimise le coût total $f(\varphi)$. Nous présentons ci-dessous un algorithme permettant l'obtention d'un tel flot, pour une valeur ϕ_0 donnée. Cet algorithme est une extension directe de l'algorithme de Ford et Fulkerson.[15]

5.2 Le problème du flot compatible

Définition 5.2.1

Soit un graphe $G = (X, U)$ connexe. A chaque arc $u \in U$ de G on affecte deux nombres b_u et c_u tels que : $b_u \leq c_u$. Le problème est de trouver un flot φ dans G compatible avec les contraintes :

$$b_u \leq \varphi_u \leq c_u\,;\ \forall u \in U$$

Remarque 5.2.1

le flot nul n'est pas nécessairement compatible.

5.2.1 Une condition nécessaire d'existence

Si φ est un flot dans G, on peut écrire, pour tout sous ensemble de sommets $A \subset X$:

$$\sum_{u\in W^+(A)} \varphi_u - \sum_{u\in W^-(A)} \varphi_u = 0$$

Si φ est un flot compatible c'est-à-dire $b_u \leq \varphi_u \leq c_u\,;\ \forall u \in U$ alors :

$$\sum_{u\in W^+(A)} c_u - \sum_{u\in W^-(A)} b_u \geq 0. \tag{5.1}$$

Le premier membre de (5.1) est appelé la capacité de la coupe associée à A.

Théorème d'optimalité :

Un flot φ_0 compatible est de coût minimal si et seulement si $G(\varphi_0)$ ne contient pas de circuit de coût strictement négatif.

5.2.2 Théorème du flot compatible(Hoffman 1960)

Etant donné un graphe $G = (X, U)$ et pour chaque arc $u \in U$ deux nombres b_u et c_u tels que : $b_u \leq c_u$ une condition nécessaire et suffisante pour qu'il existe un flot φ vérifiant :

$$b_u \leq \varphi_u \leq c_u (\forall u \in U).$$

est que :

$$\sum_{u \in W^+(A)} C_u - \sum_{u \in W^-(A)} b_u \geq 0. \tag{5.2}$$

Pour tout cocycle $W(A) = W^+(A) \cup W^-(A)$.

Remarque 5.2.2

Le Théorème de Hoffman peut être considéré comme une généralisation du Théorème du flot maximum et de la coupe minimale. En effet, ce dernier apparait comme un cas particulier dans lequel : $b_u = 0 (\forall u \in U)$ La condition (5.2) devient alors :

$$\sum_{u \in W^+(A)} C_u - \varphi_0 \geq 0 (\forall A \subset X; s \in A; t \notin A).$$

et par conséquent il existe un flot de valeur φ_0 entre s et t si et seulement si :

$$\min_{A \subset X, s \in A, t \notin A} \{ \sum_{u \in W^+(A)} C_u \} \geq \varphi_0$$

Un flot maximum est donc tel que :

$$\varphi_0 = \min_{A \subset X, s \in A, t \notin A} \{ \sum_{u \in W^+(A)} C_u \}$$

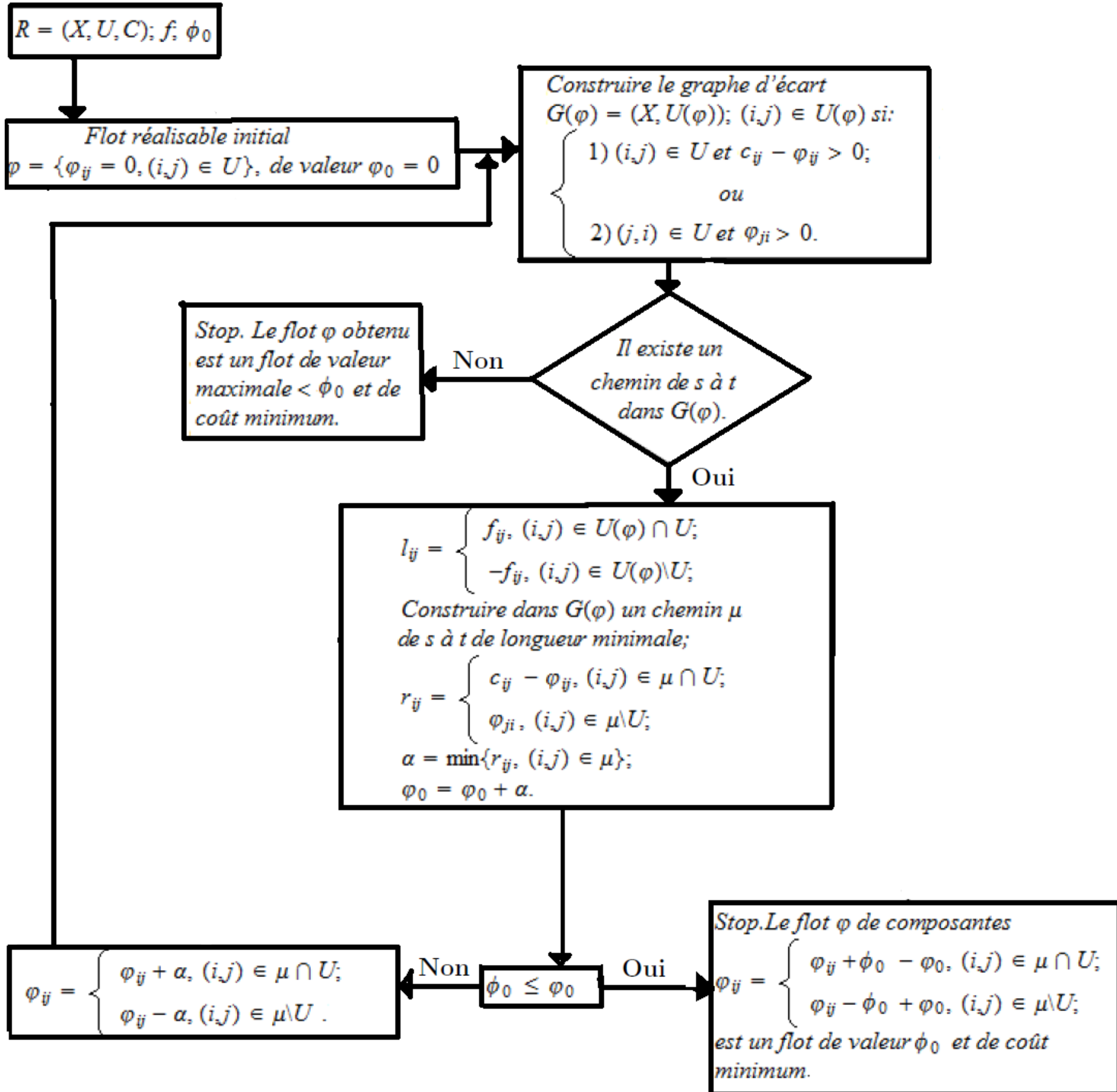
5.3 Algorithme d'obtention d'un flot de coût minimum .[6]

Principe de l'algorithme :

Rappelons que chaque étape de l'algorithme de Ford et Fulkerson consiste à rechercher un chemin μ de s à t dans le graphe d'écart $G(\varphi)$:
Pour minimiser les coûts de circulation de matière, il suffit donc de déterminer, à chaque itération un chemin de coût minimum.

En assimilant le coût unitaire f_{ij} sur l'arc (i, j) à une longueur l_{ij} ; le problème revient à obtenir, à chaque étape, un chemin de longueur minimale, et cela jusqu'à ce que la valeur du flot ainsi construit atteigne le niveau ϕ_0 : Remarquons qu'il convient généralement de choisir ici comme flot réalisable initial le flot de valeur nulle.

5.3.1 Organigramme de d'obtention d'un flot de valeur ϕ_0 donnée, de coût minimum [22]



Exemple :

Déterminons un flot de valeur ϕ_0 de coût minimal de 1 à 7 dans le réseau ci-dessous, où les couples de nombres associés aux arcs représentent, respectivement, les capacités et les coûts unitaires : On commence par le flot nul $\varphi = (0,0,0,0,0,0,0,0,0,0)$ de valeur $\varphi_0 = 0$;

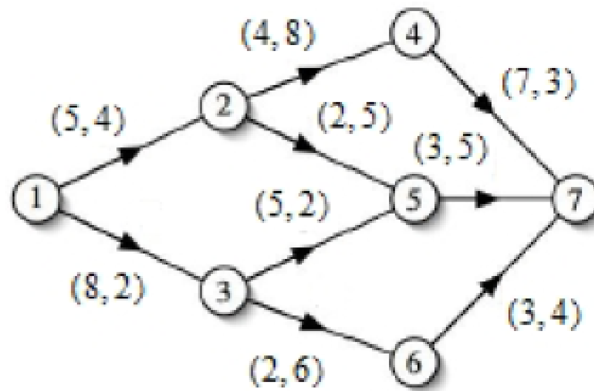


FIGURE 5.1 – Graphe d'écart

Etape 1 :

- Graphe d'écart : comme φ est nul, alors le graphe d'écart $G(\varphi)$ correspond au graphe initial.
- $\mu = ((1,3),(3,5),(5,7))$ est un chemin μ de longueur minimale dans $G(\varphi)$.
- $\alpha = \min(8,5,3) = 3$; le flot initial peut être amélioré de la quantité $\alpha = 3$.
- Un flot de valeur ϕ_0, où $0 < \phi_0 \leq 3$; de coût minimal.

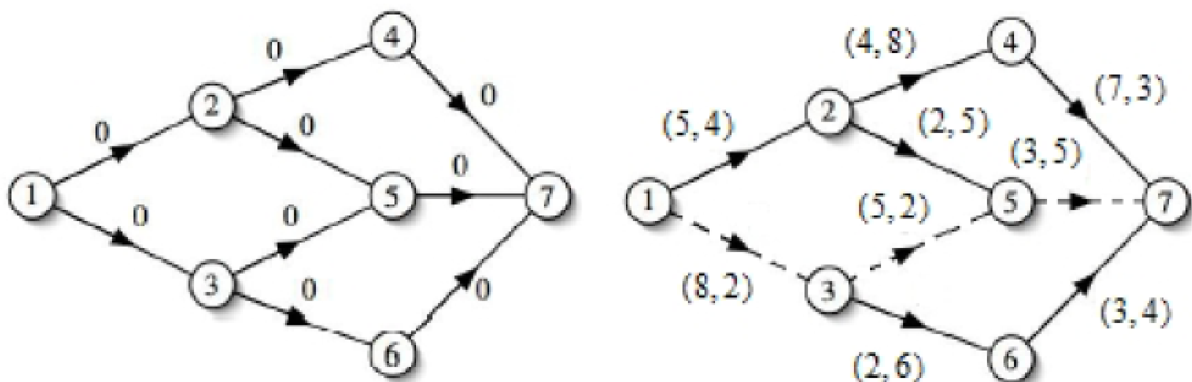


FIGURE 5.2 – Flot initial et graphe d'écart

Si la valeur de flot choisie $\phi_0 \geq 3$ alors $\phi_0 = 3$

Etape 2 :

- Graphe d'écart $G(\varphi)$: donné ci-dessous.
- $\mu = ((1,3),(3,6),(6,7))$ est un chemin μ de longueur minimale dans $G(\varphi)$.
- $\alpha = \min(5,2,3) = 2$; le flot initial peut être amélioré de la quantité $\alpha = 2$.
- Un flot de valeur ϕ_0, où $3 < \phi_0 \leq 5$; de coût minimal.

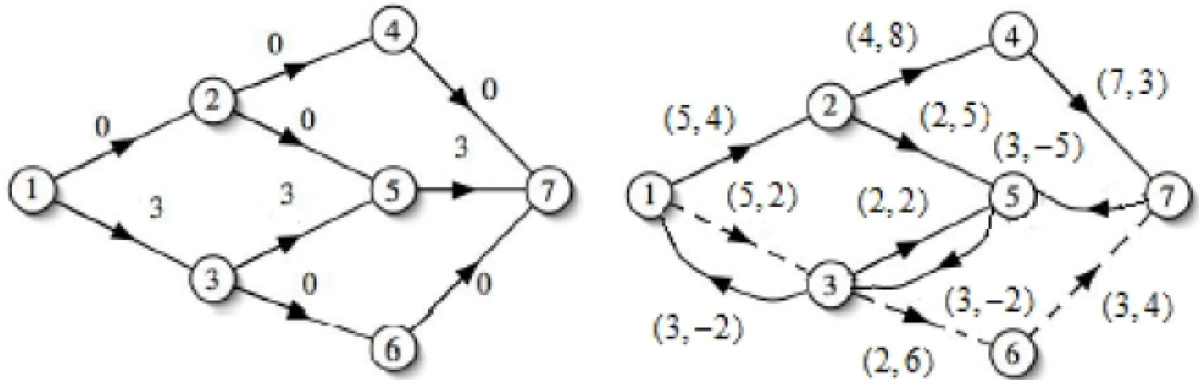


FIGURE 5.3 – Flot à l'itération numéro 1 et graphe d'écart

Si la valeur de flot choisie $\phi_0 \geq 5$ alors $\phi_0 = 5$.

Etape 3 :

- Graphe d'écart $G(\varphi)$: donné ci-dessous.
- $\mu = ((1,2),(2,4),(4,7))$ est un chemin μ de longueur minimale dans $G(\varphi)$.
- $\alpha = \min(5,4,5) = 4$; le flot initial peut être amélioré de la quantité $\alpha = 4$.
- Un flot de valeur ϕ_0, où $5 < \phi_0 \leq 9$; de coût minimal.

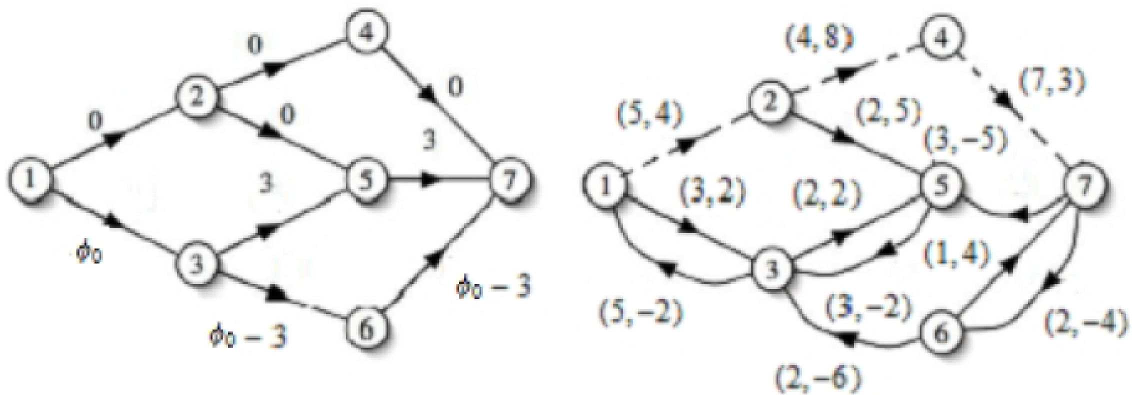


FIGURE 5.4 – Flot à l'itération numéro 2 et graphe d'écart

Si la valeur de flot choisie $\phi_0 \geq 9$ alors $\phi_0 = 9$.

Etape 4 :

– Graphe d'écart $G(\varphi)$: donné ci-dessous.

– Il n'y a pas de chemin de 1 à 7 dans $G(\varphi)$.

Alors, le flot $\phi_0 = 9$ obtenu à l'étape 3 est de valeur maximale et de coût minimal. Ce flot est donc :

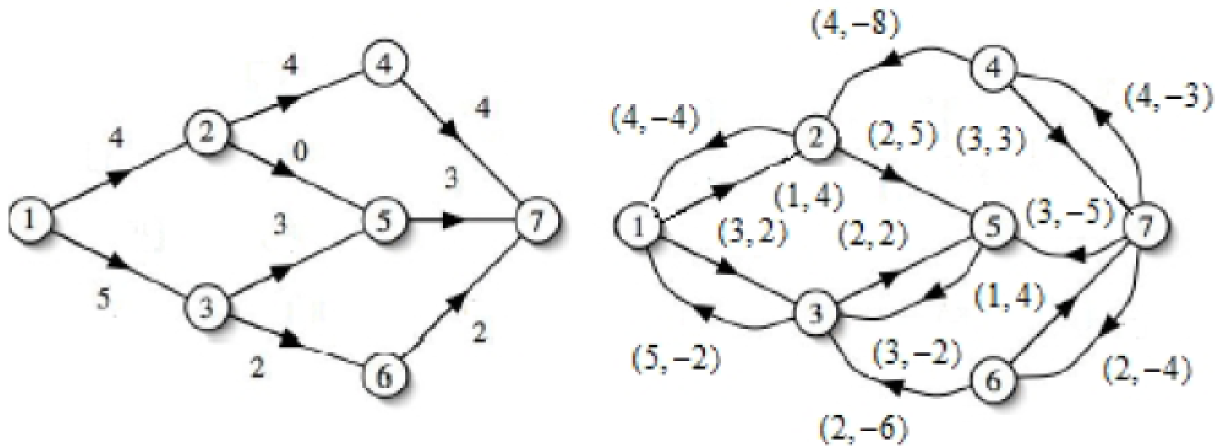


FIGURE 5.5 – Flot à l'itération numéro 3 et graphe d'écart

et le coût total correspondant est égale à

$(4 \times 4) + (4 \times 8) + (4 \times 3) + (5 \times 2) + (2 \times 6) + (2 \times 4) + (0 \times 2) + (3 \times 2) + (3 \times 5) = 111.$

Remarque 5.3.1

L'exécution de cet algorithme nécessite la recherche d'un chemin de longueur minimale. *A* cet effet, il existe des algorithmes permettant de trouver un tel chemin. Citons à titre d'exemple, l'algorithme de Dijkstra, l'algorithme de Ford,...

Chapitre 6

Mise en œuvre du programme

Introduction

Partout dans le monde, des millions d'ingénieurs et de scientifiques utilisent MATLAB pour analyser et concevoir les systèmes et produits de demain. MATLAB est présent dans des systèmes automobiles de sécurité active, des véhicules spatiaux, des appareils de surveillance médicale, des réseaux électriques intelligents et des réseaux mobiles LTE. Il est utilisé dans les domaines de l'apprentissage automatique, le traitement du signal, la vision par ordinateur, les communications, la finance computationnelle, la conception de contrôleurs, la robotique et bien plus .

6.1 Présentation du logiciel Matlab/Simulink(V6)

Le logiciel Matlab est un logiciel de manipulation de données numériques et de programmation dont le champ d'application est essentiellement les sciences appliquées. Son objectif, par rapport aux autres langages, est de simplifier au maximum la transcription en langage informatique d'un problème mathématique, en utilisant une écriture la plus proche possible du langage naturel scientifique.

Le logiciel fonctionne sous Windows et sous Linux. Son interface de manipulation HMI utilise les ressources usuelles du multi-fenêtrage. Son apprentissage n'exige que la connaissance de quelques principes de base à partir desquels l'utilisation des fonctions évoluées est très intuitive grâce à l'aide intégrée aux fonctions.

6.2 Implémentation de L'algorithme de Ford-Felkerson

6.2.1 Algorithme Général de Flot Maximal

Algorithme de Ford-Felkerson [9]

Marquage de la source s

Tant qu'on marque des sommets

pour tout sommet marque i

Marquer les sommets j non marquer tel que

$\varphi(i, j) \leq C(i, j)$ ou $\varphi(j, i) \geq 0$

Fin pour

Fin tant que

Si le puits "t" n'est pas marqué **alors**

Le flot est maximal

Sinon

Amélioration du flot [14]

1) trouver une chaîne qui a permis de marquer t et calculer .

$\varepsilon = \min(\varepsilon_1, \varepsilon_2) > 0$ avec

$\varepsilon_1 = \min_{u \in \mu^+}\{C_u - \varphi_u\}$

$\varepsilon_2 = \min_{u \in \mu^-}\{\varphi_u\}$

2) trouver le nouveau flot φ'

3)En effet, on peut améliorante la valeur du flot de ε unites sur les arcs avant de $\mu(\mu^+)$ et diminuer de ε unites sur les arc arrieres de $\mu(\mu^-)$.

L'algorithme se termine lorsque on arrive pas a trouver **une chaine augmentante**.

Fin Si

Complexité de L'algorithme de Ford-Felkerson

- Capacités à valeurs entières

  Pour des capacités à valeurs entières, l'algorithme de Ford-Fulkerson converge en un nombre fini d'opérations.

  Pour un graphe avec n sommets et m arêtes :

  - $O(m)$ opérations pour la recherche d'une chaîne améliorante et l'amélioration du flot.
  - La capacité d'une coupe est au plus en $O(n)$. Dans le pire des cas, le flot augmente d'une seule unité à chaque fois. Il y a donc au plus $O(n)$ améliorations.

  $\Longrightarrow O(n \times m \times C_{max})$ opérations pour l'algorithme de Ford-Fulkerson.

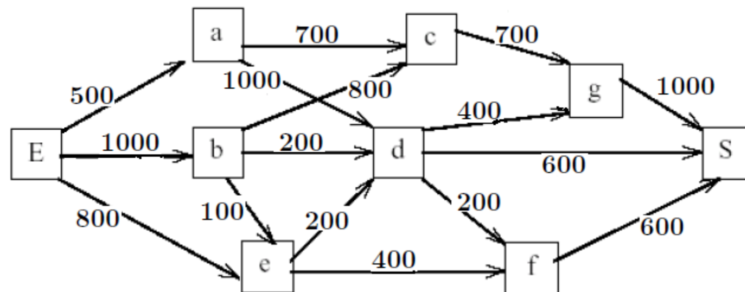
- Pour des capacités à valeurs réelles et un parcours en largeur (BFS),l'algorithme converge également

6.3 Application numériques

Soit le problème suivant :

Exemple d'application : Avant d'établir un projet de construction d'autoroute, on désire étudier la capacité du réseau routier, représenté par le graphe ci-dessous, reliant la ville E à la ville S.

Pour cela, on a évalué le nombre maximal de véhicules que chaque route peut écouler par heure, compte tenu des ralentissements aux traversées des villes et villages, des arrêts aux feux etc... Ces évaluations sont indiquées en centaines de véhicules par heure sur les arcs du graphe. Les temps de parcours entre villes sont tels que les automobilistes n'emprunteront que les chemins représentés par le graphe.



Le problème à résoudre est la recherche d'un flot maximal dans un réseau de transport. On peut utiliser l'algorithme de marquage des sommets de Ford-Fulkerson. Plutôt que de démarrer du flot nul, construisons un flot en étudiant toutes les chaînes de E à S.

Résolution du problème par l'algorithme de ford -Fulkerson

Implémente sous Matlab

```
MATLAB R2012a
File  Edit  Debug  Parallel  Desktop  Window  Help
Current Folder: C:\Documents and Settings\Administrateur\Bureau
Shortcuts  How to Add  What's New
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

Donne le sommet de source  s  =1
Donne le sommet de puit t  =9
Donne le flot   =0
Donne  La matrice d"adjacence de graphe  =[0  500  1000  0    0    800  0    0    0 ;
                                           0  0    0     700  1000 0    0    0    0  ;
                                           0  0    0     800  200  100  0    0    0 ;
                                           0  0    0     0    0    0    0    700  0 ;
                                           0  0    0     0    0    0    200  400  600 ;
                                           0  0    0     0    200  0    400  0    0 ;
                                           0  0    0     0    0    0    0    0    600 ;
                                           0  0    0     0    0    0    0    0    1000 ;
                                           0  0    0     0    0    0    0    0    0];
Le flot maximal est =  2000
le Graph Residual :
     0     0   100     0     0   200     0     0     0
   500     0     0   700   500     0     0     0     0
   900     0     0   100     0   100     0     0     0
     0     0   700     0     0     0     0     0     0
     0   500   200     0     0   200     0   300     0
   600     0     0     0     0     0     0     0     0
     0     0     0     0   200   400     0     0     0
     0     0     0   700   100     0     0     0   200
     0     0     0     0   600     0   600   800     0
```

FIGURE 6.1 – L'exécution

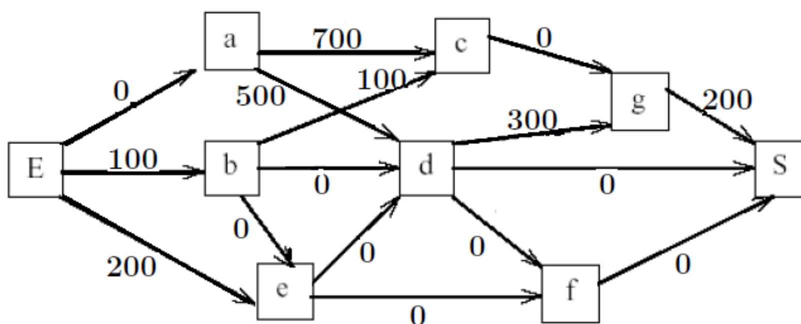


FIGURE 6.2 – La solution

Le débit total maximal de véhicules susceptibles de s'écouler entre les villes E et S est de 2000 véhicules par heure.

Exemple 2 :

Considérons le graphe G orienté valué dont la représentation sagittale est la suivante :

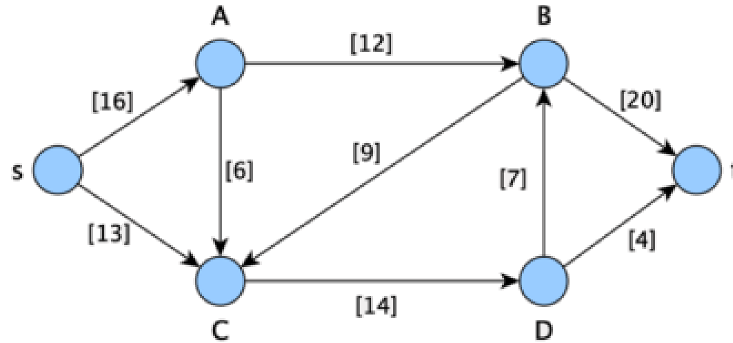
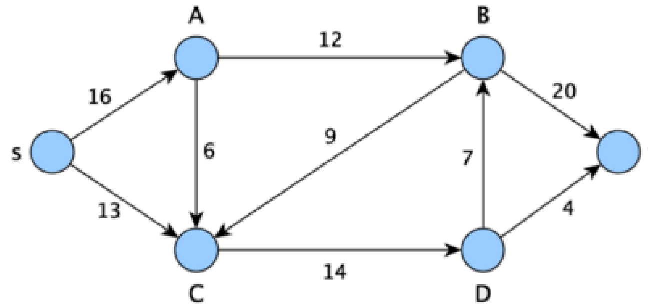


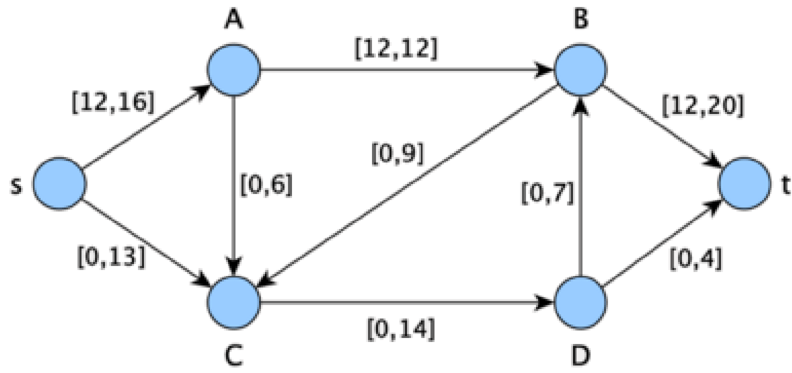
Figure 6.3 – Réseau de transport avec les capacités

Nous avons pour l'instant un flot nul, dont le graphe d'écart correspond est bien sûr :

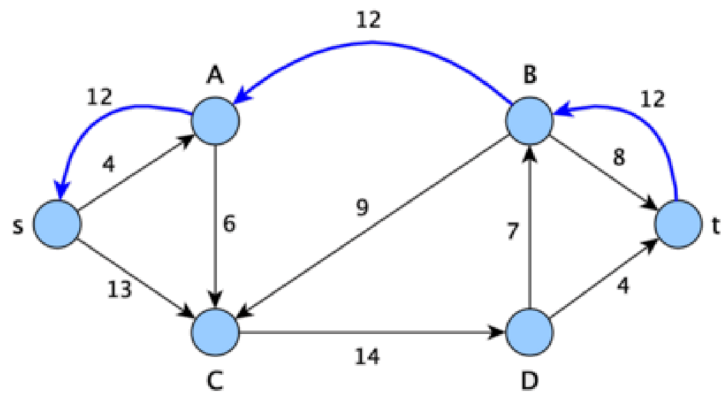


Marquage 1 :

Considérons le chemin(s, A, B, t). La valuation minimale de ses arcs est égale à 12, on va donc augmenter le flot de cette valeur. Tous les arcs de ce chemin appartiennent bien au réseau donc on y augmente le flot courant. Cela donne :

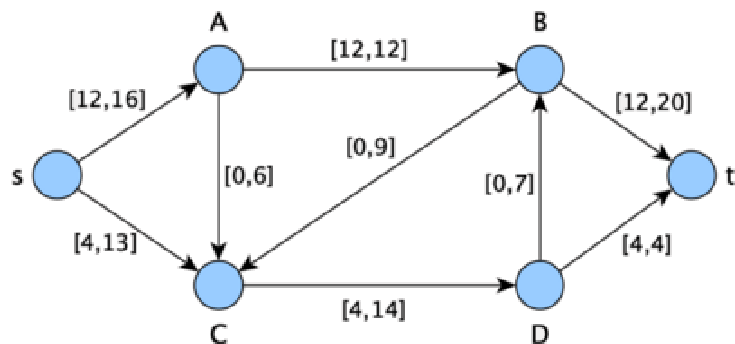


Mettons à jour le graphe d'écart :

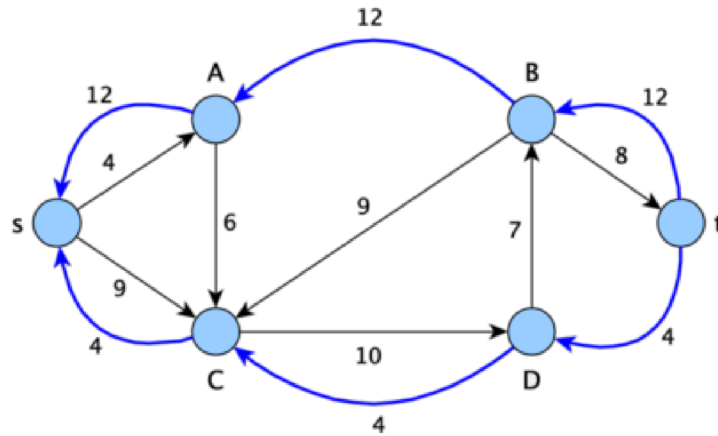


Marquage 2 :

Considérons maintenant le chemin (s, C, D, t). La valuation minimale de ses arcs est égale à 4, on va donc augmenter le flot de cette valeur. Tous les arcs de ce chemin appartiennent bien au réseau donc on y augmente le flot courant. Cela donne :

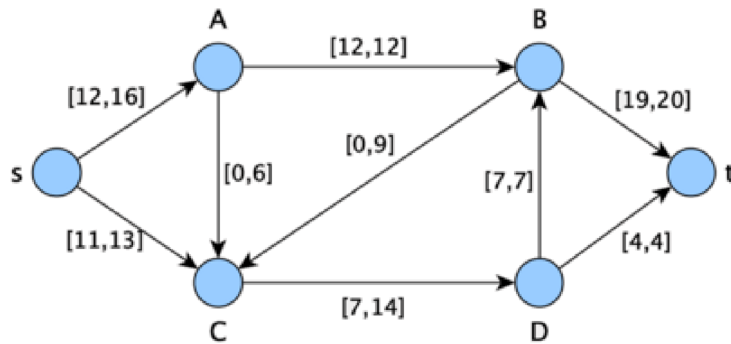


Mettons à jour le graphe d'écart :

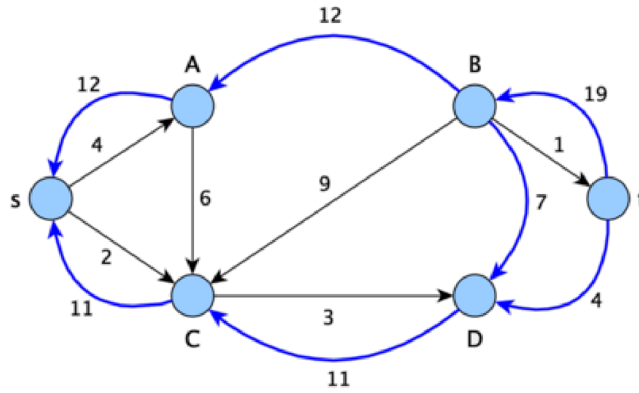


Marquage 3 :

Considérons maintenant le chemin (s, C, D, B, t). La valuation minimale de ses arcs est égale à 7, on va donc augmenter le flot de cette valeur. Tous les arcs de ce chemin appartiennent bien au réseau donc on y augmente le flot courant. Cela donne :



Mettons à jour le graphe d'écart :



Marquage 4 :

Il n'y a plus de chemins allant de la source vers le puits dans le graphe d'écart ,Arcs de la coupe $(A, B)(D, B)(D, T)$.

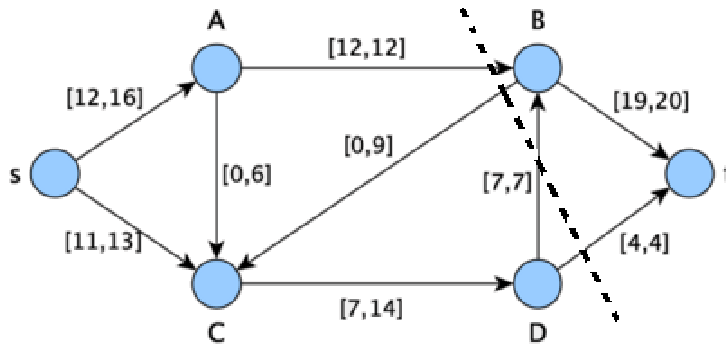


FIGURE 6.4 – Le nouveau réseau résiduel

Capacité $= 12 + 7 + 4 = 23$, donc La valeur du flot maximal est ainsi de 23.

Conclusion Générale

Notre travail consiste à la présentation de Flot et de Connexité la présentation de flot et de connexité qui ont comme principe pour modéliser diverses situations par des graphes pondérés connexes pour résoudre des problémes d'optimisation, par exemple une société livre par train à Alger des containers de marchandises remplis à Annaba. L'entreprise a le droit d'envoyer un nombre limité de containers par jour sur chaque tronçon de voie. Expédier un maximum de containers chaque jour, donc le probléme est de modéliser la situation par un graphe sur forme de flot, afin de trouver la valeur optimale en faisant appel à certaines techniques d'optimisation dans les réseaux.

Dans notre travail nous avons présenter quelques caractéristiques de flot et connexité , puis nous avant implémenté le probléme de flot sur le logiciel MatLab.

Comme perspective, nous allons modélise une situation réelle sur forme d'un graphe de connexité, pour lequel nous allons appliquer un algorithme afin de trouver la flot maximum

Bibliographie

[1] Bretto.A, A.Faisant, F.Hennecart. Éléments de théorie des graphes. 15/05/2012

[2] Bondy.A, et U.S.R. Murty , «Graph theory». 2008.

[3] B. W. KERNIGHAN et S. LIN : An efficient heuristic procedure for partitioning graphs. bell system technical journal, 49(2) : 291–307, 1970.

[4] C.Berge, Théorie des graphes et ses applications, Dunod, 1958.

[5] E. YARACK : An evaluation of move-based multi-way partitioning algorithms. In proceedings of the 2000 IEEE International Conference on Computer Design : VLSI in Computers and Processors, page 363, 2000.

[6] É. Tardos. A strongly polynomial minimum cost circulation algorithm. Combinatorica, 5(3) :247-255, 1985.

[7] Éléments de théorie des graphes ,Alain Bretto, Alain Faisant,Université de Caen – Campus II,Boulevard Maréchal Juin BP 5186, 14032 Caen.

[8] François Queyroi. Partitionnement de grands graphes : mesures, algorithmes et visualisation. Université Sciences et Technologies - Bordeaux I, 2013. Français.

[9] Ford, L. R., Jr. ; Fulkerson, D. R. (1956),Maximal flow through a network,Canadian Journal of Mathematics 8 : 399–404.

[10] G. KARYPIS et V.KUMAR : Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing, 48(1) : 96–129, 1998.

[11] Gandron.Michel. et Minoux.Michel, "Graphes et algorithmes", Editions Eyrolles, Paris VI 1986.

[12] H. S. Wilf. Algorithms and Complexity. A.K. Peter, 1985.

[13] H.D. SIMON et S-H.TENG : ≪ How good is recursive bisection ≫. SIAM Journal on scientific computing. Vol. 18, No. 5, Society for Industrial and Applied Mathematics. 1997.

[14] J.-F. Scheid, Flot maximal dans un graphe ,TELECOM Nancy 2A , 2010

[15] KHELFAOUI Fahima Et OUCHENE Siham ,Problème de flot. Application à un réseau de transport,mémoire de fin de cycle, université de Béjaia, 2012-2013.

[16] MATHÉMATIQUES 2 ,Pascal Laurent, Théorie des graphes ,Boussad Mammeri 2007.

[17] M. Sakarovitch, Graphes et programmation linéaire, Hermann, 1984.

[18] Optimisation combinatoire ,Théorie et algorithmes,Bernhard Korte ,Jens Vygen, Traduit de l'anglais par Jean Fonlupt et Alexandre Skoda,2006.

[19] Roseaux. Exercices et problèmes résolus de recherche opérationnelle : Tome 3 : Programmation linéaire et extensions - Problèmes classiques. DUNOD, 1985.

[20] Tekaya.Wajdi, "Problème d'arbre couvrant de poinds minimum ", these de doctorat, Université Paris dauphine.

[21] West.D. Introduction to graph theory, (2nd edition),prentice hall. 2000.

[22] F. Droesbeke, M. Hallin et C. Lefevre, Les graphes par l'exemple, Ellipses, 1987.

Résumé

Dans ce travail, nous avons élaboré le flot et la connexité dans un graphe, aprés avoir rappelé les concepts mathématiques importants qui jouent un rôle essentiel en théorie des graphes dans les chapitres.

Une implémentation de flot citées ci-dessus dans un seul programme a été faite sous MatLab, afin d'assurer que notre programme soit applicable quelque soit le réseau de transport. Nous avons réalisé un exemple par l'algorithme de Ford-Fulkerson citées.

Abstract

In this work, we have developed the flow and the connectivity in a graph, after recalling the important mathematical concepts which play an essentiel role in graph theory in the first chapters.

An implementation of flow cited above in a signle program was made under MatLab, to ensure that our program is applicable regardless of the transportation system. We have made an example by the Ford-Fulkerson algorithm cited.