Cours de la matière : Théorie des graphes
Pour les étudiants de la troisième année Licence Mathématiques Appliquée
Département de mathématiques
Centre Universitaire Abdelhafid Boussouf, Mila
Anné universitaire 2024/2025

Chapitre 3, Arbres et arborescences

Table des matières

Introduction

La recherche opérationnelle regroupe un ensemble de méthodes visant à optimiser la prise de décision dans des situations complexes. Elle permet de modéliser des problèmes issus du monde réel, d'identifier les approches de résolution les plus adaptées et d'exploiter les outils pertinents pour proposer des solutions efficaces. En tant que discipline appartenant à l'aide à la décision, elle fournit aux décideurs des techniques leur permettant d'opter pour les meilleures stratégies possibles. Comme toute théorie en constante évolution, la recherche opérationnelle ne cesse d'étendre son champ d'application à divers domaines. Certains problèmes de décision impliquent la prise en compte de plusieurs objectifs, nécessitant ainsi une analyse multicritère afin de sélectionner la solution la plus appropriée.

Parmi les outils fondamentaux de la recherche opérationnelle, la théorie des graphes joue un rôle central. En représentant des ensembles d'objets et leurs relations sous forme de graphes, elle permet d'aborder des problèmes d'optimisation majeurs tels que la recherche de chemins optimaux, l'ordonnancement des tâches, ainsi que l'étude des concepts de couverture et de domination dans les réseaux. Cette branche des mathématiques trouve des applications variées, notamment en logistique, en informatique, en télécommunications et dans le domaine des transports.

L'histoire de la théorie des graphes remonte au XVIII siècle avec les travaux de Leonhard Euler, notamment son célèbre problème des ponts de Königsberg : les habitants de la ville cherchaient à déterminer s'il était possible de traverser tous les ponts sans passer deux fois par le même et revenir au point de départ. D'autres problèmes historiques, tels que

la marche du cavalier sur l'échiquier ou le coloriage des cartes, ont également contribué à l'émergence de cette discipline.

Depuis son origine, la théorie des graphes a connu un développement considérable et s'est étendue à diverses disciplines telles que la chimie, la biologie et les sciences sociales. Dès le début du XX siècle, elle s'est imposée comme une branche à part entière des mathématiques, notamment grâce aux contributions de Konig, Menger, Cayley, Berge et Erdos.

De manière générale, un graphe permet de modéliser la structure et les connexions d'un système complexe en mettant en évidence les relations entre ses éléments. Il constitue un outil puissant pour représenter des réseaux de communication, des réseaux sociaux, des infrastructures routières, des interactions entre espèces en biologie, ou encore des circuits électriques. Aujourd'hui, la théorie des graphes demeure un domaine de recherche actif, mobilisant aussi bien des mathématiciens que des spécialistes de l'algorithmique, en raison de son importance dans la résolution de problèmes combinatoires et computationnels.

1

Arbres et arborescences

1.1 Arbre

Un arbre est un graphe connexe sans cycle.
D'autres définitions équivalentes sont possibles pour qu'un graphe G d'ordre n soit un arbre :

- G est connexe et possède $n-1$ arêtes.
- G est sans cycle et possède $n-1$ arêtes.
- G est connexe et minimal pour cette propriété.
- G est sans cycle et maximal pour cette propriété.
- Entre toute paire de sommets, il existe une unique chaîne les reliant.

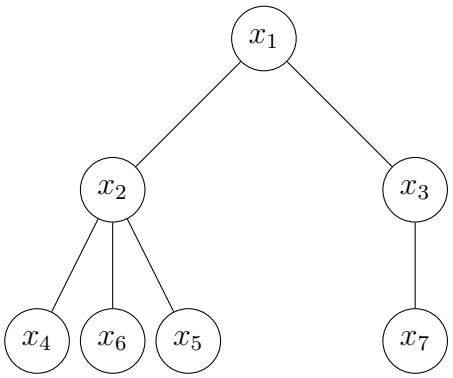

FIGURE 1.1 – Un arbre T où x_1 est sa racine, x_2 et x_3 sont des noeuds internes et x_4, x_5, x_6 et x_7 sont des feuilles.

1.1.1 Arbre n-aire complet

Pour tous entiers k ≥ 1 et t ≥ 2, on appelle arbre t-aire complet de profondeur k, le graphe $A_{k,t}$ défini inductivement comme suit :

- $A_{1,t}$ est le graphe biparti complet $K_{1,t}$.
- Pour k ≥ 2, $A_{k,t}$ est obtenu à partir de t copies disjointes de $A_{k-1,t}$ et d'un sommet relié par une arête à l'unique sommet de degré t de chacune des t copies de $A_{k-1,t}$.

L'unique sommet r de $A_{k,t}$ de degré t est la racine de $A_{k,t}$.
Soient k, $t \in \mathbf{N}^*$. Pour tout entier $0 \leq$ i $\leq$ k, on définit le i^{eme} *niveau* de $A_{k,t}$ par $V_i(A_{k,t}) = \{u \in V(A_{k,t}), d(u,r) = i\}$. En particulier, $V_k(A_{k,t})$ est l'ensemble des sommets pendants de $A_{k,t}$.
Pour un sommet $v \in V_i(A_{k,t})$, on écrit $l(v) = i$ et on dit que i est la profondeur de v. Enfin, pour tout sommet $u \in V(A_{k,t})$, on note par $A_{k,t}(u)$ l'unique sous-arbre binaire complet de $A_{k,t}$ de racine u et de profondeur $k - l(v)$.
Si t = 2, $A_{k,2}$ est appelé arbre binaire de profondeur k.

1.1.2 Arbre binaire complet

un arbre binaire enraciné est un arbre binaire complet ssi chaque sommet d'arbre possède exactement deux fils sauf les feuilles .

Exercice 1.1 *Calculer le nombre de sommets d'un arbre binaire complet en fonction de son profondeur k.*

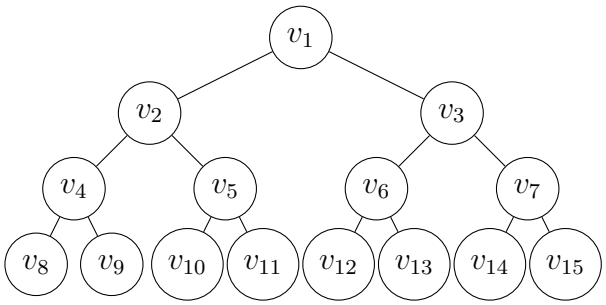

FIGURE 1.2 – Arbre binaire complet

1.2 Forêt

est un graphe non connexe et sans cycle.

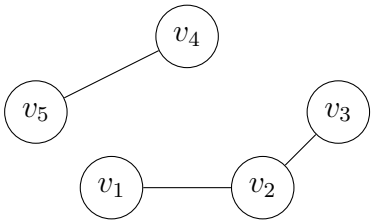

FIGURE 1.3 – Forêt

Dans cette section, nous présentons quelques algorithmes de cheminement dans les graphes orientés et les graphes non orientés. Nous commençons par le recherche d'un arbre couvrant de poids minimum dans un graphe non orienté.

1.3 Arbre couvrant de poids minimum

1.3.1 Arbre couvrant :

Un arbre couvrant (aussi appelé arbre maximal) est un graphe partiel qui est aussi un arbre. On distingue trois types de sommets dans un arbre enraciné :

- La racine.
- Les feuilles : ce sont les sommets de degré égal à 1.
- Les noeuds internes : ce sont les sommets de degré supérieur à 1.

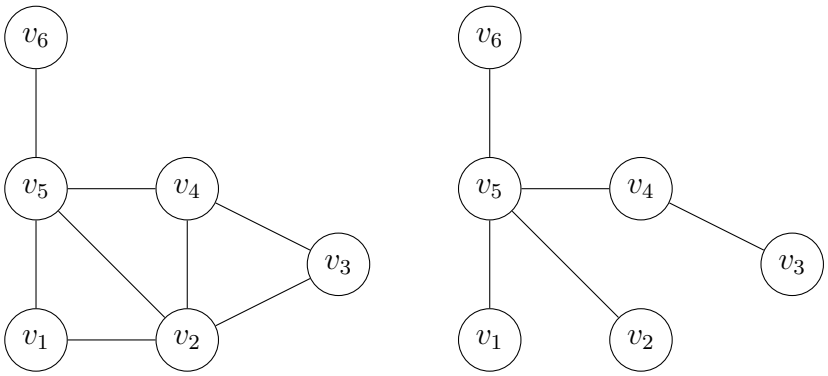

FIGURE 1.4 – Graphe G et arbre couvrant sur le graphe G

1.3.2 Arbre couvrant de poids minimum :

Soit le graphe G = (V,E) avec un poids associé à chacune de ses arêtes. On veut trouver, dans G, un arbre maximal A = (V,F) de poids total minimum.

Algorithme de Kruskal (1956) :

Données :

* Graphe $G = (V, E)$, ($|V| = n$, $|E| = m$)
* Pour chaque arête e de E , son poids $c(e)$.

Résultat : Arbre ou forêt maximale $A = (V, F)$ de poids minimum.

* Trier et renuméroter les arêtes de G dans l'ordre croissant de leur poids : $c(e_1) \leq c(e_2) \leq c(e_3)$ ... $\leq c(e_m)$.
* Poser F := $\emptyset$, k := 0
* Tant que $k < m$ et $|F| < n - 1$ faire
  Début
  si e_{k+1} ne forme pas de cycle avec F alors F := F $\bigcup$ $\{e_{k+1}\}$
  k := k + 1
  Fin

Exemple 1.1 *Appliquer l'algorithme de Kruskal pour déterminer un arbre couvrant de poids minimum sur le graphes G suivant :*

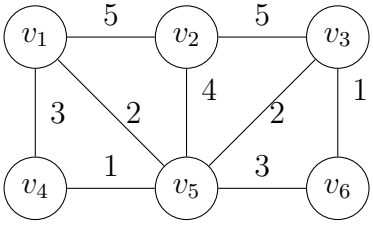

FIGURE 1.5 – G

1.4 Arborescences

En mathématiques, plus précisément dans la théorie des graphes, une arborescence est un arbre comportant un sommet particulier r, nommé racine de l'arborescence, à partir duquel il existe un chemin unique vers tous les autres sommets.
Structure arborescente de fichiers informatique En informatique, cette notion désigne souvent celle d'arbre de la théorie des graphes. Une arborescence désigne alors généralement une organisation des données en mémoire, de manière logique et hiérarchisée, utilisant une structure algorithmique d'arbre. Cette organisation rend plus efficace la consultation et la manipulation des données stockées. Les usages les plus courants en sont :
l'arborescence de fichiers, qui est l'organisation hiérarchique des fichiers sur une partition, et dans certains cas de partitions entre elles – par exemple : partitions virtuelles (« lecteurs logiques ») dans des partitions réelles ; le tri arborescent en mémoire ; les fichiers en mode séquentiel indexé.
La logique générale de l'arborescence coïncide avec le modèle relationnel du SQL : 1 vers N et réciproquement 1 vers 1. Un nœud peut posséder N feuilles, mais chaque feuille n'est possédée que par un seul nœud.

En informatique, elle désigne aussi un composant d'interface graphique qui présente une vue hiérarchique de l'information. Chaque élément (souvent appelé branche ou nœud) peut avoir un certain nombre de sous-éléments. Ceci est souvent représenté sous forme d'une liste indentée. Un élément peut être déplié pour révéler des sous-éléments, s'ils existent, et replié pour cacher des sous-éléments. La vue en arborescence apparaît souvent dans les applications de gestion de fichiers, où elle permet à l'utilisateur de naviguer dans les répertoires du système de fichiers. Elle est également utilisée pour présenter les données hiérarchiques, comme un document XML.
Usage pour la gestion des disques Structure arborescente de fichiers informatique à la base d'une arborescence se trouve un répertoire appelé la racine. Ce répertoire peut contenir des fichiers et des répertoires, qui eux-mêmes peuvent contenir la même chose. Si les fichiers et les répertoires sont placés de manière cohérente, la recherche de fichier est relativement aisée et rapide.

Forêt : Une forêt est un graphe qui contient plusieurs arbres ou arborescences distinctes.

Définition 1.1 *Étant donné un graphe orienté G=(V,E). Une arborescence est un graphe oriente sans circuit admettant une racine r telle que pour tout autre sommet si $v \in V$, il existe un chemin unique allant de r vers u. Si l'arborescence comporte n sommets, alors elle comporte exactement n - 1 arcs.*

Arborescence couvrante : On parcourt un graphe a partir d'un sommet donné r. Cette arborescence contient un arc (v_i, v_j) si et seulement si le sommet v_j a eté decouvert a partir du sommet v_i. L'arborescence associée a un parcours de graphe sera memorisée dans un tableau Π tel que $\Pi(v_j) = v_i$. Si v_j a été découvert a partir de v_i, et $\Pi(v_k) = nul$ si v_k est la racine, ou s'il n'existe pas de chemin de la racine vers v_k.

Parcours en largeur (Breadth First Search = BFS) :

Le parcours en largeur est obtenu en gérant la liste d'attente au coloriage comme une file d'attente (FIFO = First In First Out). Autrement dit, on enlève a 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 d'attente, en les coloriant en gris.

Structures de donnees utilisees : On utilise une file F, pour laquelle on suppose definies les operations :
$init - file(F)$ qui initialise la file F a vide,
$ajoute - fin - file(F, u)$ qui ajoute le sommet u a la fin de la file F,
$est - vide(F)$ qui retourne vrai si la file F est vide et faux sinon,
et $enleve - debut - file(F, u)$ qui enleve le sommet u au debut de la file F.
On utilise un tableau Π qui associe a chaque sommet le sommet qui a fait entrer dans la file, et un tableau *couleur* qui associe a chaque sommet sa couleur (blanc, gris ou noir).
On va en plus utiliser un tableau d qui associe a chaque sommet son niveau de profondeur par rapport au sommet de depart r (autrement dit, $d[v_i]$ est la longueur du chemin dans l'arborescence Π de la racine r jusqu'au sommet v_i). Ce tableau sera utilisé plus tard.

Algorithme
$init - file(F)$
pour tout sommet $v_i \in V$ faire
$\Pi(v_i) \longleftarrow nil$
$d[v_i] \longleftarrow \infty$
couleur$(v_i) \longleftarrow$ blanche
fin pour
$d[r] \longleftarrow 0$
$ajoute - fin - file(F, r)$
$couleur[r] \longleftarrow$ gris
tant que $est - vide(F) =$ faux faire
$enleve - debut - file(F, v_i)$
pour tout $v_i \in succ(v_i)$ faire
si $couleur[v_i] =$ blanche alors
$ajoute - fin - file(F, v_i)$

$couleur[v_i] \longleftarrow$ gris
$\Pi(v_j) \longleftarrow v_i$
$d[v_i] \longleftarrow d[v_i] + 1$
fin si
fin pour
$couleur[v_i] \longleftarrow$ noir
fin tant
fin BFS

Applications du parcours en largeur

depuis r. À la fin de l'exécution de $BFS(V; E; r)$, chaque sommet est soit noir soit blanc. Le parcours en largeur peut être utilisé pour rechercher l'ensemble des sommets accessibles sommets noirs sont ceux accessibles depuis r ; les sommets blancs sont ceux pour lesquels il n'existe pas de chemin/chaîne à partir de r. D'une façon plus générale, le parcours en largeur permet de déterminer les composantes connexes d'un graphe non orienté. Pour cela, il suffit d'appliquer l'algorithme de parcours en largeur à partir d'un sommet blanc quelconque. À la suite de quoi, tous les sommets en noirs appartiennent à la première composante connexe. S'il reste des sommets blancs, cela implique qu'il y a d'autres composantes connexes. Il faut alors relancer le parcours en largeur sur le sous-graphe induit par les sommets blancs, pour découvrir une autre composante connexe. Le nombre de fois où l'algorithme de parcours en largeur a été lancé correspond au nombre de composantes connexes. Le parcours en largeur peut aussi être utilisé pour chercher le plus court chemin (en nombre d'arcs ou arêtes) entre la racine r et chacun des autres sommets du graphe accessibles depuis r. Pour cela, il suffit de remonter dans l'arborescence Π du sommet concerné jusqu'à la racine r. L'algorithme 2 (récursif) affiche le plus court chemin pour aller de r à v_j.

Algorithme 2 : plus-court-chemin(r, v_j, Π)

Algorithme 2 : plus-court-chemin(r ; v_j ; Π)
Entrées : r sommet de départ, v_j sommet d'arrivée, Π arborescence couvrante. 1 si r = v_j alors
2 afficher(r)
3 sinon si $\Pi(v_j)$ = nil alors
4 afficher("pas de chemin")
5 sinon
6 plus-court-chemin(r, $\Pi(v_j)$, Π)
7 afficher v_j