Je mets ce résumé à disposition pour votre inspiration, les résumés sont censés être personnels. Je vous déconseille d'imprimer simplement celui-là.

1 Définitions

1.1 Graphes non orientés

Définition 1.1. *Un **graphe non orienté** est une structure formée d'un ensemble V dont les éléments sont appelés les **sommets** ou les **nœuds** du graphe, d'un ensemble E (disjoint de V) dont les éléments sont appelés les **arêtes** du graphe et d'une fonction d'incidence qui associe à chaque arête de E une paire de sommets de V (pas forcément distincts) appelés les **extrémités** de l'arête.*

Si G est un graphe d'ensemble de sommets V et d'ensemble d'arêtes E, on notera $G = (V, E)$. De plus, si a et b sont les deux extrémités de l'arête e, on dira que e **relie les sommets** a **et** b, que les sommets a et b sont **adjacents**, qu'ils sont **incidents avec** e ou, encore, que l'arête e est **incidente avec** a et b.

Définition 1.2. *Un graphe non orienté est dit **simple** s'il ne possède ni arêtes parallèles ni boucles.*

Le plus petit graphe imaginable est celui ne possédant aucun sommet et, à plus forte raison, aucune arête. Il est appelé le **graphe nul**. Le graphe ne possédant qu'un seul sommet et aucune arête est appelé le **graphe trivial** alors qu'un **graphe vide** est un graphe ne possédant aucune arête (le graphe nul et le graphe trivial sont des exemples particuliers de graphes vides).

1.2 Graphes orientés

Définition 1.3. *Un **graphe orienté** est une structure formée d'un ensemble V dont les éléments sont appelés les **sommets** ou les **nœuds** du graphe, d'un ensemble E (disjoint de V) dont les éléments sont appelés les **arcs** du graphe et d'une fonction d'incidence qui associe à chaque arc de E un couple de sommets de V (c.-à-d. un élément de $V \times V$) appelés les **extrémités** de l'arc.*

Comme dans le cas non orienté, on notera $G = (V, E)$ le graphe défini par l'ensemble de sommets V et l'ensemble d'arcs E. De plus, si les extrémités de l'arc e correspondent au couple (a, b), on dira que e **va de** a **vers** b, que le sommet a est l'**extrémité initiale** de e et que le sommet b est son **extrémité finale** ou **terminale**. Les notions d'adjacence et d'incidence introduites dans la section précédente s'appliquent également à la situation présente.

Définition 1.4. *Un graphe orienté est dit **simple** s'il ne possède ni arcs parallèles ni boucles.*

À tout graphe orienté $G = (V, E)$ on peut associer un graphe non orienté $G' = (V, E')$, appelé graphe sous-jacent, obtenu en remplaçant chaque arc par une arête de mêmes extrémités.

1.3 Degrés

Définition 1.5. *Soit $G = (V, E)$ un graphe, le **degré** du sommet v de G, noté $deg(v)$, est égal au nombre d'arêtes incidentes à v, chaque boucle étant comptée deux fois.*

Théorème 1.1. $\sum_{v \in V} deg(v) = 2\,|E|$

Corollaire 1.1.1. *Dans tout graphe il y a un nombre pair de sommets de degré impair.*

Définition 1.6. *Soit $G = (V, E)$ un graphe orienté, le **degré extérieur** (ou **degré sortant**) du sommet v de G, noté $deg_+(v)$, est égal au nombre d'arcs issus de v alors que le **degré intérieur** (ou **degré entrant**) du sommet v, noté $deg_-(v)$, est égal au nombre d'arcs se terminant en v.*

Théorème 1.2.

$$\sum_{v \in V} deg_+(v) = \sum_{v \in V} deg_-(v) = |E|$$

1.4 Sous-graphes et graphes partiels

Un **graphe partiel** est créé en conservant tous les sommets, mais en ne retenant qu'un sous-ensemble des arêtes (ou des arcs).

Un **sous-graphe**, aussi appelé **sous-graphe induit** ou **engendré par** W, est défini en retenant un sous-ensemble W de sommets de G ainsi que toutes les arêtes (ou arcs) ayant leurs deux extrémités dans W.

Un **sous-graphe partiel** est obtenu en prenant un graphe partiel d'un sous-graphe de G.

1.5 Chaînes et cycles

Définition 1.7. *Dans un graphe $G = (V, E)$, une **chaîne** est une suite alternée de sommets et d'arêtes de G débutant et finissant par un sommet et telle que chaque arête de la chaîne est encadrée par ses deux extrémités.*

Définition 1.8. *Un **cycle** est une chaîne fermée comptant au moins une arête et dont les deux extrémités sont confondues.*

Définition 1.9. *La **longueur** d'une chaîne (ou d'un cycle) est égale au nombre d'arêtes qui la (le) constituent.*

Définition 1.10. *Une chaîne est **élémentaire** si chaque sommet y apparaît au plus une fois et elle est **simple** si chaque arête y apparaît au plus une fois.*
*De même, un cycle est **élémentaire** si chaque sommet, à l'exception de ses extrémités confondues, y apparaît au plus une fois et est **simple** si chaque arête y apparaît au plus une fois.*

Définition 1.11. *Un graphe est **sans cycles** ou **acyclique** s'il ne possède pas de cycles simples. Un graphe sans cycles est appelé une **forêt**.*

1.6 Chemins et circuits

Définition 1.12. *Dans un graphe orienté $G = (V, E)$, un **chemin** est une suite alternée de sommets et d'arcs de G débutant et finissant par un sommet et telle que chaque arc du chemin est précédé de son extrémité initiale et suivi de son extrémité finale.*

Définition 1.13. *Un **circuit** est un chemin fermé comptant au moins un arc et dont les deux extrémités sont confondues.*

Définition 1.14. *La **longueur** d'un chemin (ou d'un circuit) est égale au nombre d'arcs que le constituent.*

Définition 1.15. *Un chemin est **élémentaire** si chaque sommet y apparaît au plus une fois et il est **simple** si chaque arc y apparaît au plus une fois.*
*De même, un circuit est **élémentaire** si chaque sommet, à l'exception de ses extrémités confondues, y apparaît au plus une fois et est **simple** si chaque arc y apparaît au plus une fois.*

2 Représentation des graphes

2.1 Matrices d'adjacence et d'incidence

2.1.1 Cas non orienté

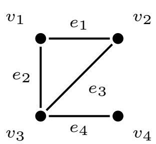
Définition 2.1. *La **matrice d'adjacence sommets-sommets** de G est la matrice A : $n \times n$ dont l'élément a_{ij}, associé aux sommets v_i et v_j, est égal à 1 si v_i et v_j sont adjacents (c.-à-d. s'ils sont reliés par une arête) et à 0 sinon.*

Définition 2.2. *La **matrice d'incidence sommets-arêtes** de G est la matrice $B : n \times m$ dont l'élément b_{ik}, associé au sommet v_i et à l'arête e_k, est égal à 1 si v_i est incident à e_k (c.-à-d. si v_i est une extrémité de e_k) et à 0 sinon.*

Exemple 2.1.

$$\boldsymbol{A} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}$$

$$\boldsymbol{B} = \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}$$



2.1.2 Cas orienté

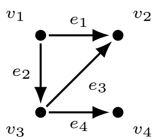
Définition 2.3. *La **matrice d'adjacence sommets-sommets** de G est la matrice A : $n \times n$ dont l'élément a_{ij}, associé aux sommets v_i et v_j, est égal à 1 s'il existe un arc reliant v_i et v_j et à 0 sinon.*

Définition 2.4. *La **matrice d'incidence sommets-arêtes** de G est la matrice $B : n \times m$ dont l'élément b_{ik}, associé au sommet v_i et à l'arc e_k, est égal à -1 si v_i est l'extrémité initiale de e_k, à 1 si v_i est l'extrémité finale de e_k et à 0 sinon.*

Exemple 2.2.

$$\boldsymbol{A} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}$$

$$\boldsymbol{B} = \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{bmatrix} -1 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}$$



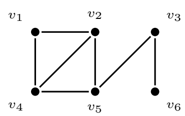
2.2 Listes d'adjacence et de successeurs

2.2.1 Cas non-orienté

Si $G = (V, E)$ est un graphe non orienté simple, on peut le représenter à l'aide de listes d'adjacence, un tableau Adj de $|V|$ listes, une pour chaque sommet de G. Pour le sommet $u \in V$, la liste $Adj[u]$ contient tous les sommets adjacents à u.

Exemple 2.3.

Sommet u	$Adj[u]$
v_1	(v_2, v_4)
v_2	(v_1, v_4, v_5)
v_3	(v_5, v_6)
v_4	(v_1, v_2, v_5)
v_5	(v_2, v_3, v_4)
v_6	(v_3)

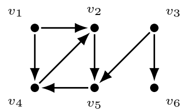


2.2.2 Cas orienté

Si $G = (V, E)$ est un graphe orienté simple, on peut le représenter à l'aide de listes d'adjacence associées à chaque sommet par des **listes de successeurs** ou des **listes de prédécesseurs**.

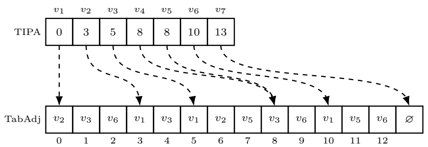
Exemple 2.4.

u	$Adj[u]$	$Pred[u]$
v_1	(v_2, v_4)	$\varnothing$
v_2	(v_5)	(v_1, v_4)
v_3	(v_5, v_6)	$\varnothing$
v_4	(v_2)	(v_1, v_5)
v_5	(v_4)	(v_2, v_3)
v_6	$\varnothing$	(v_3)

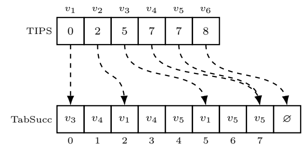


2.3 Tableaux compacts

2.3.1 Cas non-orienté



2.3.2 Cas orienté



3 Connexité et exploration des graphes

3.1 Graphe connexe et composantes connexes

Définition 3.1. *Un graphe G est **connexe** s'il existe une chaîne entre chaque paire de ses sommets.*

Définition 3.2. *Une **composante connexe** d'un graphe G est un sous-ensemble C de sommets de G, maximal au sens de l'inclusion, pour lequel il existe une chaîne entre chaque paire de ses sommets.*

3.2 Graphe/Composantes fortement connexe/s

Définition 3.3. *Un graphe orienté G est **fortement connexe** si pour chaque paire de sommets u, v de G il existe un chemin de u vers v et un chemin de v vers u.*

Définition 3.4. *Dans un graphe orienté G, une **composante fortement connexe** est un sous-ensemble C de sommets de G maximal au sens de l'inclusion et tel que pour toute paire u, v de sommets de cet ensemble il existe un chemin de u vers v et un chemin de v vers u.*

Définition 3.5. *Soit G un graphe orienté, le **graphe réduit** de G est le graphe orienté G_R dont les sommets correspondent aux composantes fortement connexes de G et où un arc relie le sommet a au sommet b s'il existe, dans G, un arc reliant un sommet de la composante a à un sommet de la composante b.*

Propriété 3.1. *Un graphe orienté G est fortement connexe si et seulement si son graphe réduit G_R est le graphe trivial (i.e. le graphe ne comptant qu'un seul sommet et aucun arc).*

Propriété 3.2. *Le graphe réduit G_R associé à un graphe orienté G est un graphe sans circuits.*

3.3 Exploration d'un graphe

3.3.1 Exploration en largeur

Complexité : $O(n + m)$.

3.3.2 Exploration en profondeur

Complexité : $O(n + m)$.

3.4 Calcul des composantes connexes

3.4.1 Algorithme de Tarjan

La structure n'est pas différente d'une exploration DFS. Cependant, en plus de numéroter les sommets dans l'ordre de leur découverte, l'algorithme calcule également, pour chaque sommet u, le numéro $low[u]$ du sommet situé le plus « haut » au-dessus de lui dans l'arbre DFS et accessible depuis u. Si, à la fin du traitement du sommet u, les valeurs $low[u]$ et $dfsnum[u]$ sont égales, cela signifie que tous les sommets de la composante fortement connexe de u sont situés en dessous de lui dans l'arbre DFS. L'algorithme utilise une pile, dans laquelle les sommets sont introduits au fur et à mesure de leur découverte, pour retrouver tous les sommets de la composante de u. La complexité de l'algorithme est identique à celle d'une exploration DFS, à savoir $O(n + m)$.

4 Arbres et arborescences

Définition 4.1. *Un **arbre** est un graphe sans cycles et connexe.*

Théorème 4.1. *Soit G un graphe simple sur n sommets. Les affirmations suivantes sont toutes équivalentes.*

1. *G est un arbre.*
2. *G est sans cycles et connexe.*
3. *G est sans cycles et comporte $n - 1$ arêtes.*
4. *G est connexe et comporte $n - 1$ arêtes.*
5. *Chaque paire de sommets distincts de G est reliée par une et une seule chaîne simple.*
6. *G est connexe et minimal pour cette propriété (aucun graphe partiel strict de G n'est connexe).*
7. *G est sans cycles et maximal pour cette propriété (tout ajout d'une arête à G crée exactement un cycle simple).*

Propriété 4.1. *Tout arbre sur 2 sommets ou plus possède au moins 2 feuilles.*

4.1 Arbres recouvrants de poids minimum

Définition 4.2. *Soit G un graphe. Un **arbre recouvrant** de G est un graphe partiel de G qui est un arbre.*

Définition 4.3. *Un graphe est connexe si et seulement s'il admet un arbre recouvrant.*

4.1.1 Algorithme de Kruskal

Complexité : $O(m \cdot \log n)$.

1. On part du graphe partiel vide, ne contenant aucune arête mais tous les sommets.
2. On cherche l'arête de plus petit poids reliant deux composantes connexes du graphe partiel courant et on l'ajoute à ce graphe.
3. On répète l'opération précédente jusqu'à obtenir un graphe partiel connexe.

4.1.2 Algorithme de Prim

Complexité : $O(m \cdot \log n)$.

1. On choisit un sommet au hasard dans G et on part de l'arbre T ne contenant que ce sommet.
2. À chaque itération, on ajoute à l'arbre T l'arête de plus petit poids reliant un sommet de T à un sommet n'appartenant pas encore à T.
3. On s'arrête dès que l'arbre T contient tous les sommets.

4.2 Chaînes de section maximale

Définition 4.4. *Dans un réseau $R = (V, E, c)$, la* ***section inférieure*** *d'une chaîne C non triviale est égale au minimum des poids des arêtes de C.*

Le *problème de la chaîne de section maximale* consiste à déterminer, parmi toutes les chaînes de R reliant deux sommets distincts s et t, celle dont la section inférieure est la plus grande.

Théorème 4.2. *L'arbre recouvrant de poids maximum contient des chaînes de section maximale entre toutes les paires de sommets distincts d'un réseau $R = (V, E, c)$.*

4.3 Arborescences

Définition 4.5. *Une* ***arborescence de racine*** *r est un arbre orienté dans lequel il existe un chemin du sommet r vers tous les autres.*

Définition 4.6. *Une* ***anti-arborescence d'anti-racine*** *r (ou simplement de racine r) est un arbre orienté dans lequel il existe un chemin de chaque sommet vers l'anti-racine r.*

Propriété 4.2. *Un arbre orienté est une arborescence de racine r si et seulement si le degré entrant de chaque sommet, à l'exception de la racine r, est égal à 1.*

Propriété 4.3. *Un arbre orienté est une anti-arborescence d'anti-racine r si et seulement si le degré sortant de chaque sommet, à l'exception de l'anti-racine r, est égal à 1.*

Définition 4.7. *Une arborescence recouvrante d'un graphe orienté connexe $G = (V, E)$ est un arbre recouvrant de G qui est une arborescence. Réciproquement, une anti-arborescence recouvrante d'un graphe orienté connexe $G = (V, E)$ est un arbre recouvrant de G qui est une anti-arborescence.*

Propriété 4.4. *Un graphe $G = (V, E)$ est fortement connexe si et seulement s'il possède une arborescence recouvrante de racine r pour tout $r \in V$.*

5 Plus courts chemins dans les réseaux

5.1 Propriétés

— **Plus courts chemins depuis une source** s : Calcul du plus court chemin d'un sommet source s vers chaque sommet du réseau

— **Plus court chemin de s à t** : Cette variante consiste à calculer un seul plus court chemin, d'un sommet s à un sommet t.

— **Plus courts chemins jusqu'à une destination** t : Plus court chemin jusqu'à un sommet destination t depuis chaque sommet du réseau. Calcul des plus courts chemins depuis une source donnée en inversant le sens de chaque arc du réseau

5.2 Principe d'optimalité de Bellman

Théorème 5.1. *Si P est un plus court chemin de s à t et u un sommet apparaissant dans P, alors les sous-chemins de s à u et de u à t contenus dans P sont également des plus courts chemins.*

$$\lambda_j \leq \lambda_i + c_{ij}, \quad \forall i \in Pred[j]$$

$$\lambda_j = \min_{i \in Pred[j]} (\lambda_i + c_{ij}), \quad \forall j \in V \setminus \{s\}$$

□

5.3 Plus courts chemins depuis une source unique

Le nombre de sommets de G est noté n ($n = |V|$) et son nombre d'arcs m ($m = |E|$).

5.3.1 Algorithme de Bellman-Ford

Complexité : $O(nm)$

5.3.2 Algorithme de Dijkstra

Complexité : $O(m \cdot \log n)$

5.4 Plus courts chemins entre tous les couples de sommets

5.4.1 Floyd-Warshall

L'algorithme de Floyd-Warshall utilise une décomposition limitant les numéros des sommets pouvant être utilisés comme étapes intermédiaires dans un chemin reliant i à j. À l'étape k, l'algorithme calcule les longueurs des plus courts chemins entre tous les couples de sommets mais en autorisant uniquement les sommets 1 à k comme étapes intermédiaires. Après n itérations, tous les sommets du réseau peuvent apparaître à l'intérieur d'un chemin et les dernières valeurs calculées correspondent aux longueurs des plus courts chemins entre tous les couples de sommets.

Complexité : $O(n^3)$

5.4.2 Dantzig

L'algorithme de Dantzig considère la suite de sous-graphes $G_k, k = 1, \ldots, n$ où G_k est le sous-graphe engendré par les sommets 1 à k uniquement. À l'étape k, l'algorithme calcule les longueurs des plus courts chemins, dans G_k, entre tous les couples de sommets de G_k. Après n itérations, le sous-graphe G_n est égal au graphe G de départ et les dernières valeurs calculées correspondent aux longueurs des plus courts chemins entre tous ses couples de sommets.

Complexité : $O(n^3)$

6 Graphes sans circuits

6.1 Fonction de rang et tri topologique

Définition 6.1. *Une* ***fonction de rang*** *est une fonction affectant, à chaque sommet v d'un graphe orienté $G = (V, E)$, un numéro, appelé le rang de v et noté $rang(v)$, tel que pour tout arc (u, v) de G on a $rang(u) < rang(v)$.*

Théorème 6.1. *Un graphe orienté $G = (V, E)$ est sans circuits si et seulement s'il admet une fonction de rang.*

Propriété 6.1. *Si $G = (V, E)$ est un graphe fini et sans circuits, alors G possède au moins un sommet sans prédécesseurs et au moins un sommet sans successeurs.*

Propriété 6.2. *Si $G = (V, E)$ est un graphe sans circuits, alors tout sous-graphe partiel de G est également sans circuits.*

6.1.1 Tri topologique

Algorithme de Kahn, complexité : $O(n + m)$.

6.2 Plus courts chemins dans un réseau sans circuits

Dans un réseau sans circuits, la présence d'arcs de poids négatifs n'est pas un problème et, dès qu'un tel réseau contient un chemin d'un sommet u à un sommet v, il en contient un plus court. Dans la plupart des applications, les réseaux sans circuits rencontrés possèdent une racine, c'est-à-dire un sommet à partir duquel il est possible d'atteindre tous les autres sommets par des chemins. Cette racine joue le rôle de sommet source et c'est à partir d'elle que l'on cherche à calculer des plus courts chemins vers tous les autres sommets du réseau.

6.3 Application à la gestion de projets

6.3.1 Graphes potentiels-tâches

À partir de l'ensemble des activités du projet et de la liste des contraintes d'antériorité, on construit un graphe où :

— chaque sommet correspond à une tâche du projet,

— un arc de poids d_i (égal à la durée de la tâche i) relie le sommet i au sommet j si l'exécution de la tâche i doit précéder celle de la tâche j.

Si le projet est réalisable, le graphe $G = (V, E)$ ainsi construit doit être sans circuits. Il possède donc au moins un sommet sans prédécesseurs et au moins un sommet sans successeurs. On le complète alors en ajoutant deux sommets, souvent notés α et ω, modélisant deux tâches fictives, de durée nulle, correspondant respectivement au début et à la fin des travaux. Le sommet α est ensuite relié, par des arcs de poids nul, à chacun des sommets sans prédécesseurs de G tandis que des arcs (i, ω), de poids d_i, sont ajoutés pour chaque sommet i sans successeurs dans G.

Dans un graphe potentiels-tâches, tout chemin de α à ω correspond à une suite d'activités qui ne peuvent avoir lieu simultanément et doivent se succéder. Ainsi, la longueur de n'importe quel chemin de α à ω, égal à la somme des durées des tâches qui le composent, fournit une borne inférieure sur la durée totale de réalisation du projet. La durée minimale nécessaire à l'exécution de toutes les tâches est alors égale à la longueur d'un plus long chemin de α à ω.

6.3.2 Méthode du chemin critique

La méthode du chemin critique est une adaptation de l'algorithme PCC_SansCircuits calculant pour chaque tâche j d'un projet :

— la date t_j de début au plus tôt de la tâche (en supposant que le projet commence le jour 0),

— la date T_j de début au plus tard de la tâche sous la contrainte que la durée totale du projet n'augmente pas.

La méthode du chemin critique détermine la durée minimale d'un projet, les dates de début et de fin de chaque tâche, et identifie les tâches prioritaires. Toute tâche avec une marge nulle ($T_j - t_j = 0$) est critique : tout retard ou allongement de durée augmente la durée totale du projet. Ces tâches critiques forment un ou plusieurs chemins critiques dans le graphe potentiels-tâches.

6.3.3 Généralisations

Les contraintes de succession prises en compte dans la modélisation des paragraphes précédents sont des contraintes de *type potentiel*.

$$t_j \geq t_i + c_{ij} \Leftrightarrow t_j - t_i \geq c_{ij}$$

Dans un graphe potentiels-tâches tel qu'il a été défini ci-dessus, chaque contrainte de ce type est modélisée par un arc (i, j) de poids c_{ij} égal à la durée d_i de la tâche i et représente la contrainte « l'activité j ne peut pas débuter avant la fin de la réalisation de l'activité i ».

Contraintes additionnelles

— « la tâche j peut débuter au plus tôt x jours après la fin de la tâche i » se modélise par un arc (i, j) de poids $c_{ij} = d_i + x$

— « la tâche j peut commencer au plus tôt x jours après le début de la tâche i » se modélise par un arc (i, j) de poids $c_{ij} = x$ ainsi qu'un arc (i, ω) de poids $c_{i\omega} = d_i$ (afin d'éviter que la fin du projet ne puisse débuter avant la fin de i)

— « la tâche i débute au plus tôt x jours après le commencement des travaux » se modélise par un arc (α, i) de poids $c_{\alpha i} = x$

— « la tâche j débute au plus tard x jours après le début de la tâche i » se modélise par un arc (j, i) de poids $c_{ji} = -x$

— « la tâche j doit commencer sitôt la tâche i terminée » se modélise par un arc (i, j) de poids $c_{ij} = d_i$ et un arc (j, i) de poids $c_{ji} = -d_i$

Il est important de noter que **si le graphe potentiels-tâches contient des circuits**, la planification du projet n'est possible que s'il n'y a **aucun circuit à coût positif**. Cette condition est essentielle pour l'existence d'un plus long chemin de α à ω. Dans un tel cas, il faut utiliser lalgorithme de Bellman-Ford.

7 Flots dans les réseaux

7.1 Flot de valeur maximale

On considère un réseau $R = (V, E, u)$ composé

— d'un graphe orienté $G = (V, E)$ simple et connexe avec un **sommet source** $s \in V$ et un **sommet puits** $t \in V$

— d'une fonction $u : E \to \mathbb{R}_+$ associant à chaque arc $e = (i, j)$ du graphe une **capacité** $u(e) = u_{ij}$ non négative

Définition 7.1. *Un* ***flot*** *x* ***de s à t*** *dans un réseau est une fonction associant à chaque arc $e = (i, j)$ du réseau un nombre $x(e) = x_{ij}$ vérifiant la loi de conservation : « à l'exception de la source s et du puits t, ce qui entre dans un sommet est égal à ce qui en ressort. »*
Les nombres x_{ij} sont appelés les ***quantités de flot*** *ou les* ***flux*** *transitant par les arcs du réseau.*
Un flot x de s à t est dit compatible ou admissible si, en plus de vérifier les équations de conservation précédentes, le flux passant par chaque arc du réseau est non négatif et ne dépasse pas la capacité de l'arc :

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in E$$

Définition 7.2. *La valeur f d'un flot de s à t est égale à l'excès de flux qui sort de la source s ou, de manière équivalente, à l'excès de flux qui arrive dans le puits t (car les équations de conservation du flot sont satisfaites pour tous les autres sommets)*

7.1.1 Algorithme de Ford-Fulkerson

Partant d'un flot compatible (par exemple le flot nul), si on trouve un chemin de s à t utilisant uniquement des arcs où le flux est inférieur à la capacité de l'arc (on parle alors d'un **chemin non saturé** ou **augmentant**), on peut augmenter le flot de s à t le long de ce chemin.
Afin d'éviter de se bloquer, on cherche des chemins non saturés dans un *réseau auxiliaire* $R^*(x)$ comportant toutes les augmentations possibles du flot x. Ce réseau d'augmentation (aussi appelé réseau résiduel) est construit à partir de l'observation qu'il est possible d'augmenter le flux de j à i en diminuant un flux existant de i à j.

Complexité (EK) : $O(m^2 n)$

7.1.2 Coupe de capacité minimale

Définition 7.3. *Soit A un sous-ensemble propre et non vide de sommets d'un graphe orienté $G = (V, E)$, la coupe $(A, \overline{A})$ définie par A est l'ensemble des arcs sortant de A, c'est-à-dire l'ensemble des arcs ayant leur extrémité initiale dans A et leur extrémité finale dans $\overline{A} = V \setminus A$. La capacité d'une coupe $(A, \overline{A})$ est la somme des capacités des arcs formant la coupe.*

Dans un réseau R, une coupe $(A, \overline{A})$ sépare le sommet s du sommet t si $s \in A$ et $t \in \overline{A}$ (on utilise parfois le terme de « s-t-coupe » pour désigner un tel ensemble d'arcs).

Lemme 7.1. *Dans un réseau, la valeur d'un flot compatible de s à t est toujours inférieure ou égale à la capacité d'une coupe séparant s de t.*

Théorème 7.1 (Ford-Fulkerson)**.** *La valeur maximale d'un flot compatible de s à t est égale à la capacité minimale d'une coupe séparant s de t.*

Théorème 7.2 (Valeurs entières)**.** *Dans un réseau où toutes les capacités maximales sont entières, il existe toujours un flot maximum de s à t entier, c'est-à-dire un flot maximum x où tous les flux x_{ij} sont entiers.*

7.1.3 Couplage maximum dans un graphe biparti

Soit $G = (V_1 \cup V_2, E)$ un graphe biparti non orienté, on peut ramener la recherche d'un couplage maximum dans G à la résolution d'un problème de flot maximum :

— on oriente toutes les arêtes de G de V_1 vers V_2, les arcs obtenus ont une capacité de 1 (ou restent sans capacité maximale, à choix),

— on ajoute une source s et des arcs de s vers chacun des sommets de V_1, ces arcs ont une capacité de 1,

— on ajoute un puits t et des arcs, de capacité 1, des sommets de V_2 vers t,

— on cherche, finalement, un flot maximum de s à t dans le réseau ainsi construit.

Les arcs de V_1 vers V_2 dans lequel le flux est égal à 1 dans le flot maximum définissent un couplage maximum dans le graphe initial.

7.2 Flots à coût minimum

On considère un réseau $R = (V, E, c, u)$ composé

— d'un graphe orienté $G = (V, E)$ simple et connexe avec un **sommet source** $s \in V$ et un **sommet puits** $t \in V$

— d'une fonction $c : E \to \mathbb{R}$ associant à chaque arc $e = (i, j)$ du graphe un **coût unitaire d'utilisation** $c(e) = c_{ij}$ (le plus souvent positif ou nul).

— d'une fonction $u : E \to \mathbb{R}_+$ associant à chaque arc $e = (i, j)$ du graphe une **capacité** $u(e) = u_{ij}$ non négative

Dans ce réseau on cherche un flot de valeur donnée (typiquement de valeur maximale) de s à t de coût total minimum.
Comme pour le problème du flot de valeur maximale, on peut déterminer un flot à coût minimum en partant d'un flot nul et en augmentant successivement la valeur du flot. À chaque itération le flot est à coût minimum mais sa valeur (la quantité transportée) augmente au fil des

itérations. Chaque augmentation se fait en saturant un *plus court chemin de s à t* dans le réseau d'augmentation R^* associé au flot courant
Dans le réseau d'augmentation R^*, les arcs renversés ont des coûts négatifs (si les coûts de départ sont positifs) car une diminution d'une unité du flux transitant de i à j correspond à une économie de c_{ij} (francs).

7.2.1 Problème de l'affectation linéaire

On considère n personnes et n tâches à réaliser. Pour chaque personne i et chaque tâche j, on connaît la durée c_{ij} que met la personne à réaliser la tâche. On cherche comment répartir les tâches entre les différentes personnes de manière à ce que chacune d'elles effectue une et une seule tâche et que la durée totale de réalisation de toutes les tâches soit minimale.
Le problème revient à chercher un couplage maximum (parfait) de coût minimum dans un graphe biparti. En utilisant la même technique que dans le cas d'un couplage maximum, ce problème se transforme facilement en un problème de flot maximum à coût minimum.

7.2.2 Problème du transbordement

On considère le réseau R :

— un graphe orienté $G = (V, E)$ dont les sommets sont soit des **sources** où un certain nombre d'unités sont disponibles ; des **puits** où un certain nombre d'unités sont désirées ; des **sommets de transit ou transbordement** où il n'y a ni offre ni demande

— des coûts unitaires d'utilisation (c_{ij} pour l'arc (i, j))

— des capacités maximales d'utilisation (u_{ij} pour l'arc (i, j))

On cherche un plan de transbordement permettant de satisfaire la demande des sommets puits à partir de l'offre des sommets sources, le tout en minimisant les coûts totaux de transport.

— On ajoute une source s et des arcs (s, v_i) de cette source vers chacun des sommets v_i où il y a une offre. Ces arcs ont un coût d'utilisation nul et leur capacité est égale à l'offre du sommet source auquel ils sont associés.

— On ajoute un puits t et des arcs (v_i, t) reliant chacun des sommets v_i où il y a une demande à t. Ces arcs ont un coût d'utilisation nul et leur capacité est égale à la demande du sommet puits auquel ils sont associés.

— On cherche dans ce nouveau réseau un flot de valeur maximale de s à t de coût total minimum.

8 Familles et problèmes classiques

8.1 Graphes complets

Définition 8.1. *Un graphe complet est un graphe simple, non orienté, où toute paire de sommets distincts est reliée par une arête. Un graphe complet sur n sommets est noté K_n.*

Le graphe complet K_n est donc le plus « grand » graphe simple sur n sommets, la grandeur d'un graphe étant, ici, mesurée par son nombre d'arêtes.

Propriété 8.1. *Le nombre m d'arêtes du graphe complet K_n est $m = \binom{n}{2} = \frac{n(n-1)}{2}$*

Définition 8.2. *Soit $G = (V, E)$ est un graphe simple non orienté, le **graphe complémentaire** de G est le graphe $\overline{G} = (V, \overline{E})$ possédant le même ensemble de sommets que G et dont l'ensemble d'arêtes E est formé de toutes les arêtes d'extrémités distinctes n'apparaissant pas dans E.*

$$\overline{E} = \{\{u, v\} | \{u, v\} \notin E, u \neq v \wedge u, v \in V\}$$

Le complémentaire d'un graphe simple est également simple et l'union des deux, c'est-à-dire le graphe obtenu en prenant l'union de leurs ensembles d'arêtes, est un graphe complet. Le complémentaire du graphe complet K_n est le graphe vide, parfois noté N_n, ne possédant aucune arête.

Définition 8.3. *Un **tournoi** est un graphe orienté dont le graphe sous-jacent (non orienté) est un graphe complet.*

Propriété 8.2. *Un tournoi possède au plus un sommet sans prédécesseur et au plus un sommet sans successeur.*

8.2 Graphes bipartis

Définition 8.4. *Un **graphe biparti** est un graphe $G = (V, E)$ dont l'ensemble V des sommets peut être partitionné en deux sous-ensembles V_1 et V_2 de telle manière que chaque arête (ou arc) de G ait une extrémité dans V_1 et l'autre dans V_2.*

Si on parcourt la suite des sommets d'un cycle d'un graphe biparti, on visite alternativement des sommets de V_1 et de V_2. La longueur d'un tel cycle doit donc être paire. Inversement, si un graphe ne contient aucun circuit de longueur impaire, on peut construire une bipartition de ses sommets en explorant le graphe, car la découverte d'une arête dont les deux extrémités appartiennent à un même sous-ensemble de la partition permettrait de construire un cycle de longueur impaire.

Propriété 8.3. *Un graphe est biparti si et seulement s'il ne possède aucun cycle de longueur impaire.*

Dans un graphe biparti $G = (V_1 \cup V_2, E)$ *simple*, il existe au plus une arête entre un sommet de V_1 et un sommet de V_2. Lorsque toutes ces arêtes existent, c'est-à-dire lorsque chaque sommet de V_1 est relié à chaque sommet de V_2, le graphe biparti est dit *complet*.

Définition 8.5. *Un **graphe biparti complet** est un graphe biparti simple possédant un nombre maximal d'arêtes. Un graphe biparti complet comptant r sommets dans un des ensembles de sa bipartition et s dans l'autre est noté $K_{r,s}$.*

8.3 Couplages

Définition 8.6. *Un **couplage**, dans un graphe simple non orienté $G = (V, E)$, est un sous-ensemble d'arêtes $M \subseteq E$ tel que deux arêtes quelconques de M n'ont pas d'extrémités communes.*

Si M est un couplage de $G = (V, E)$, le sommet $v \in V$ est dit saturé par M s'il existe une arête dans M incidente à v. Dans le cas contraire, le sommet v est dit non saturé par M.

Définition 8.7. *Un **couplage parfait** est un couplage qui sature tous les sommets d'un graphe.*

Définition 8.8. *Un **couplage maximum** est un couplage de cardinalité (taille) maximale.*

Si un graphe G possède n sommets, la taille maximale d'un couplage M de G est $\lfloor \frac{n}{2} \rfloor$ et, si n est impair, le graphe ne peut contenir un couplage parfait.

8.4 Graphes eulériens

8.4.1 Cas non-orienté

Définition 8.9. *Une **chaîne eulérienne** d'un graphe non orienté G est une chaîne passant une et une seule fois par chacune des arêtes de G. De manière similaire, un **cycle eulérien** de G est un cycle passant une et une seule fois par chaque arête de G.*

Définition 8.10. *Un graphe non orienté G est **eulérien** s'il possède un cycle eulérien, c.-à-d. un cycle passant une et une seule fois par chacune de ses arêtes*

Théorème 8.1. *Un graphe non orienté G connexe est eulérien (et possède donc un cycle passant une et une seule fois par chacune de ses arêtes) si et seulement si tous ses sommets sont de degré pair.*

Corollaire 8.1.1. *Un graphe non orienté G connexe possède une chaîne eulérienne si et seulement s'il possède deux sommets de degré impair car il suffit d'ajouter une arête entre ces deux sommets pour se ramener au cas précédent et obtenir un graphe eulérien.*

8.4.2 Cas orienté

Définition 8.11. *Dans un graphe orienté G un chemin, respectivement un circuit, est dit **eulérien** s'il passe une et une seule fois par chaque arc du graphe G.*

Définition 8.12. *Un graphe orienté G est eulérien s'il possède un circuit eulérien, c.à-d. un circuit passant une et une seule fois par chacun de ses arcs.*

Théorème 8.2. *Un graphe orienté G connexe est eulérien (et possède donc un circuit passant une et une seule fois par chacun de ses arcs) si et seulement si chacun de ses sommets a autant d'arcs entrants que d'arcs sortants autrement dit si et seulement s'il y a égalité entre les demi-degrés entrant et sortant de chaque sommet.*

Corollaire 8.2.1. *Un graphe orienté G connexe possède un chemin eulérien si et seulement si*

1. *G possède deux sommets de degré impair,*
2. *tous les sommets de degré pair de G ont autant d'arcs entrants que d'arcs sortants,*
3. *pour les deux sommets de degré impair, l'écart (absolu) entre les demi-degrés entrant et sortant est égal à 1.*

Le sommet de degré impair ayant un arc sortant de plus que d'arcs entrants correspond alors à l'extrémité initiale du chemin eulérien alors que celui ayant un arc entrant de plus que d'arcs sortants correspond à son extrémité finale.

9 Réponses types

9.1 Puits multiples

→ On considère un réseau $R = (V, E, u)$ possédant p sources $s_1, \ldots, s_p$ et q puits $t_1, \ldots, t_q$. Reformuler le problème de la recherche d'un flot de valeur maximale entre les sources et les puits de ce réseau comme un problème de flot maximum entre une source unique s et un puits unique t.
On ajoute au réseau R une source s que l'on relie à chaque source s_i par un arc (s, s_i) de capacité infinie.

9.2 Flot respectant les capacités aux sommets

→ On considère un réseau $R = (V, E, b)$ où b est une fonction associant à chaque sommet $i \in V$ (autre que s et t) une capacité b_i (de traitement par unité de temps par exemple). Reformuler le problème de la recherche d'un flot de valeur maximale de s à t respectant les capacités aux sommets comme un problème classique de flot maximum de s à t.
On remplace chaque sommet i (autre que s et t) par deux sommets i' (sommet d'entrée en i) et i'' (sommet de sortie de i). Tous les arcs entrant initialement en i entrent maintenant en i' (leur éventuelle capacité n'est pas modifiée). Tous les arcs partant initialement de i partent maintenant de i'' (leur capacité initiale, infinie ou non, n'est pas modifiée). On ajoute finalement un arc (i', i'') entre i' et i'' de capacité b_i. Le problème initial revient alors à calculer un flot compatible de valeur maximale de s à t dans ce réseau modifié.

9.3 Chemins disjoints

9.3.1 Chemins disjoints par les arcs

On affecte à chaque arc une capacité de 1 et on calcule un flot maximal de s à t dans le réseau obtenu. La valeur maximale du flot est égale au nombre maximum de chemins disjoints par les arcs reliant s à t.

9.3.2 Chemin disjoints par les sommets

On affecte à chaque sommet autre que s et t une capacité de 1, et à chaque arc une capacité de 1. On calcul le flot maximal de s à t respectant les capacités aux sommets. La valeur optimale de ce flot sera égale au nombre de chemins disjoints par les sommets reliant s à t.

9.4 Connexité du graphe orienté

→ Si $\deg_+(v) = \deg_-(v)$ pour chaque sommet v de G alors G est fortement connexe.
L'affirmation est correcte. En effet tout graphe orienté connexe dans lequel l'égalité des demi-degrés est vérifiée pour tous les sommets possède un circuit eulérien or un tel circuit passe forcément pour tous les sommets du graphe (il est connexe) est ce dernier est fortement connexe.
→ Si G est fortement connexe alors $\deg_+(v) = \deg_-(v)$ pour chaque sommet v de G
L'affirmation est fausse. Il suffit de prendre un graphe orienté connexe eulérien (donc où tous les sommets ont autant d'arcs entrants que sortants) et de lui ajouter un arc quelconque (mais pas une boucle). Le graphe est encore et toujours fortement connexe mais ne vérifie plus l'égalité des demi-degrés entrants et sortants.

9.5 Réseau des lignes de bus

On modélise le réseau avec $G = (V, E)$ avec

— chaque sommet $v \in V$ est un arrêt

— une ligne de bus a un ensemble d'arcs reliant les arrêts

— chaque arc reçoit un temps de trajet positif t_{ij}

1. BFS depuis s dans le graphe G afin de calculer une distance d_i en nombre d'arcs entre s et $i \in V$
2. Supprimer tous les sommets $i \neq t$ avec $d(i) \geq d(t)$
3. Supprimer tous les arcs (i, j) qui ne satisfont pas $d(j) = d(i) + 1$
4. PCC dans DAG obtenu pour s à t.

9.6 Attribution de tâches bipartie

n personnes se proposent : n_1 hommes, n_2 femmes, avec chacun leurs préférences. Nombre de postes est de $2m$, avec une parité homme-femme obligatoire.

1. Graphe biparti, deux ensembles de sommets avec candidats en V_1 et ministères en V_2,
2. Arc de capacité 1 candidat à ministère pour chaque compétence,
3. Sommet t relié par un arc de chaque ministère de capacité 1.
4. Sommet h et f,
5. Pour chaque candidat, arc de capacité 1 de h au candidat si homme, f si femme.
6. Sommet s et deux arcs (s, h) et (s, f) de capacité m,
7. Flot maximal de s à t.
8. La solution existe si la valeur du flot max est de $2m$, alors on examine les arcs entre candidats et ministères et ceux dont le flux est de 1 sont les candidats sélectionnés.

9.7 Réseau de communication et résistance

$G = (V, E)$ non orienté, pour deux sommets A et B, la résistance $\rho(A, B)$ est le nombre minimum de connexions à détruire pour couper la communication.
Comme le réseau est non orienté, on double chaque arête en deux arcs de sens opposés, ensuite on traite le nouveau graphe avec la méthode des chemins disjoints par les arcs.

9.8 Équilibrer un graphe pour un circuit eulerien

Établir un sous-graphe partiel avec les sommets au degré entrant différent du degré sortant, puis ajouter les arcs selon l'offre et la demande. Intégrer le résultat dans le graphe principal.

9.9 Routier en livraison

→ Routier peut rouler 8 heures, certains arrêts autorisent de passer la nuit. Itinéraire qui minimise le nombre de nuits.

1. Depuis la case A, déterminer à l'aide de W de Floyd-Warshall les cases autorisant la nuitée à une distance inférieure ou égale à 8.
2. On prend le sous-graphe partiel comportant uniquement les sommets autorisant la nuitée, on y ajoute une arête s'il existe un chemin de moins de 8 heures, le graphe aura donc des arêtes pour chaque trajet de moins d'une journée.
3. Exploration en BFS pour trouver le plus court chemin de A à B

9.10 Postier dans la ville

→ Coursier dans un réseau de route au poids par arc indiquant le temps. Trouver le chemin le plus court de s à t en passant par u (boite aux lettres).
Utiliser un algorithme de plus court chemin (p.ex. Dijkstra) depuis u pour déterminer le chemin de u à s qui sera transposé, et le chemin de u à t.

9.11 Questions variées

→ Soit (i, j) un arc appartenant à une coupe de capacité minimum séparant s de t dans un réseau $R = (V, E, u)$. Si on augmente d'une unité la capacité u_{ij} de l'arc (i, j) alors la valeur maximale d'un flot de s à t augmente également d'une unité.
Faux, car la valeur maximale d'un flot est égale à la capacité minimale de la coupe séparant s de t, en augmentant la capacité d'un des arcs de la coupe, ce n'est pas garanti que la valeur maximale du flot change.

→ Soit (i, j) un arc appartenant à une coupe de capacité minimum séparant s de t dans un réseau $R = (V, E, u)$. Si on diminue d'une unité la capacité u_{ij} de l'arc (i, j) alors la valeur maximale d'un flot de s à t diminue également d'une unité.
Vrai, car la valeur maximale d'un flot est égale à la capacité minimale de la coupe séparant s de t.

→ Dans un graphe connexe, il existe toujours un couplage saturant tous les sommets ou tous les sommets sauf un.
Faux, il peut y avoir plus d'un sommet non saturé.

10 Algorithme de Kahn

Algorithme TriTopologiqueKahn(G)
Données : Un graphe orienté $G = (V, E)$ sans circuits donné par des listes de successeurs et des listes de prédécesseurs.
Résultat : Un tableau *rang* contenant une numérotation des sommets de G compatible avec le rang.
Début
  $L := \varnothing$ // liste de sommets sans prédécesseurs non numérotés
  Pour chaque sommet $v \in V$ **faire**
    $degré := 0$
    Pour chaque prédécesseur $u \in Pred[v]$ **poser** $degré := degré + 1$
    $degré\_entrant[v] := degré$
    Si $degré\_entrant[v] = 0$ **faire** introduire v dans L
  $k := 1$
  Tant que $L \neq \varnothing$ **faire**
    Retirer un sommet u de L
    Poser $rang[u] := k$
    Incrémenter k
    Pour chaque successeur $v \in Succ[u]$ **faire**
      $degré\_entrant[v] := degré\_entrant[v] - 1$
      Si $degré\_entrant[v] = 0$ **faire** introduire v dans L
  Retourner le tableau *rang*
Fin

11 Tableaux types

11.1 BFS

	v_1	...
sommet	0	...
$d[i]$	0	...
$p[i]$	0	...

11.2 Algorithme de Tarjan

	v_1	...
dfsnum	0	...
low	0	...
scc	0	...

11.3 Algorithme de Kosaraju

	v_1	...
début	1	...
fin	n	...

11.4 Algorithme de Prim

Itér	Sommet retiré	Couples $(\lambda_j, p[j])$ v_1	v_2
0	−	$(0, -)$	$(\infty, -)$
1	v_1		
⋮			

11.5 Algorithme de Kruskal

k	e_k	$c(e_k)$	$\in T$
1	$\{1, 3\}$	1	✓ / ×
⋮	⋮	⋮	⋮

11.6 Algorithme de Bellman-Ford

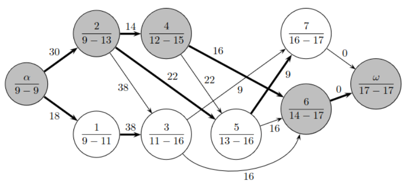
Itér. k	Arc testé	$c(e_k)$ 1	2	...	Valeur de *Continuer*
Valeurs de départ		$(0, -)$	$(\infty, -)$	...	faux
1	$(1, 2)$		$(1, 1)$		vrai
	⋮	⋮	⋮	⋮	
Valeurs de départ		$\{1, 3\}$	$(0, -)$	...	faux
2	$(1, 2)$				
	⋮				

11.7 Algorithme de Dijkstra

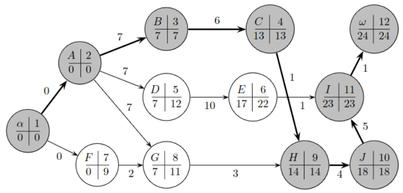
Itér	Sommet retiré	Couples $(\lambda_j, p[j])$ v_1	v_2
0	−	$(0, -)$	$(\infty, -)$
1	v_1		
⋮			

12 Graphes types

12.1 DAG représentant horaires



12.2 Potentiel-tâches



12.3 Problème de transbordement