Graphes sans circuits et applications

Les graphes sans circuits (**DAG**) sont essentiels en gestion de projets, ordonnancement et compilation.

Propriétés fondamentales

- Tout graphe fini sans circuits possède au moins un sommet sans prédécesseurs et un sans successeurs
- Tout sous-graphe partiel d'un graphe sans circuits est sans circuits
- Permettent d'introduire une notion de **rang** : $\text{rang}(u) < \text{rang}(v)$

Tri topologique (Kahn) - $O(n+m)$

- **But** : Ordonner les sommets en respectant les relations d'ordre (indispensable pour ordonnancement)
- **Applications** : Compilation, gestion projets, détection cycles
- **Principe** : Répéter jusqu'à épuisement des sommets
1. Identifier un sommet sans prédécesseurs dans le graphe résiduel
2. Le numéroter dans l'ordre croissant (rang topologique)
3. Le supprimer du graphe avec tous ses arcs sortants
- **Propriété** : Si le graphe contient un cycle, l'algorithme s'arrête avant d'avoir numéroté tous les sommets

Plus court/long chemin - Équation de Bellman

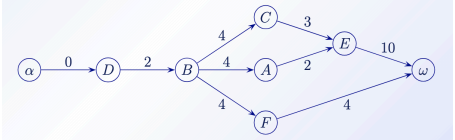
- **Applications** : Ordonnancement projets, optimisation, planification
- **Avantage DAG** : Traitement dans l'ordre topologique, pas d'itérations multiples comme Bellman-Ford classique
- **Plus court chemin** :
  - $\lambda_j = \min_{i \in \text{Pred}[j]}(\lambda_i + c_{ij})$ avec $\lambda_s = 0$
- **Plus long chemin** :
  - $\lambda_j = \max_{i \in \text{Pred}[j]}(\lambda_i + c_{ij})$ avec $\lambda_s = 0$
- **Algorithme** : Traiter sommets dans ordre topologique, appliquer équation
- **Complexité** : $O(n+m)$ (une seule passe suffit grâce au DAG)

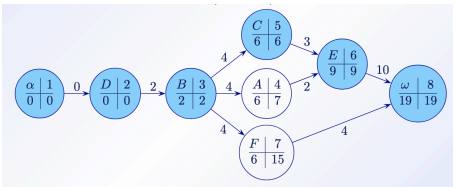
Graphes potentiels-tâches

Modélisation de projets

- **Sommets** : tâches du projet
- **Arcs** : contraintes de précédence (i précède j)
- **Poids** : durée d_i de la tâche i
- **Ajouts** : sommet début a et fin w (poids 0)

Tâche	Description	Durée (sem.)	Antériorités
D	Choix des stations	2	—
B	Accord administratif	4	D
C	Commande des décodeurs	3	B
A	Installation des antennes	2	B
E	Installation des décodeurs	10	C,A
F	Modification de la facturation	4	B





Méthode du chemin critique

- **But** : Identifier les tâches critiques dont tout retard retarde le projet entier
- **Applications** : Gestion de projets, planification industrielle, optimisation

Phase 1 - Calcul dates au plus tôt (forward pass) :
- $t_a = 0$ (début projet)
- $t_j = \max_{i \in \text{Pred}[j]}(t_i + d_i)$ pour chaque tâche j
- Traitement dans l'ordre topologique

Phase 2 - Calcul dates au plus tard (backward pass) :
- $T_w = t_w$ (durée minimale projet)
- $T_i = \min_{j \in \text{Succ}[i]}(T_j) - d_i$ pour chaque tâche i
- Traitement dans l'ordre topologique inverse

Résultats :
- **Tâche critique** : $t_i = T_i$ (marge libre nulle)
- **Chemin critique** : Succession de tâches critiques de début à fin
- **Durée projet** : t_w (date plus tôt de fin)
- **Marge libre tâche i** : $T_i - t_i$ (retard possible sans impact)

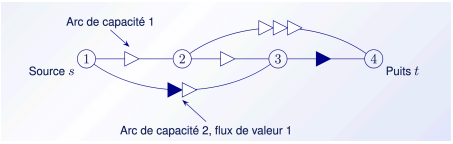
Composition d'un nœud

Nom des tâches	Numéros topologiques
Date de début au plus tôt	Date de début au plus tard

Flots dans un réseau

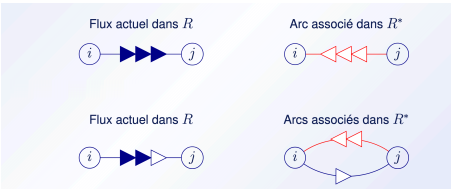
Concepts fondamentaux

- **Réseau** : $R = (V, E, c, u)$ avec capacités u_{ij} et coûts c_{ij}
- **Flot compatible** : Respecte capacités et conservation
- **Loi de conservation** : $\sum_{\text{entrant}} = \sum_{\text{sortant}}$ (sauf source/puits)



Réseau d'augmentation

Principe : Construire graphe permettant d'augmenter le flot
- **Arcs directs** : (i,j) si $x_{ij} < u_{ij}$, capacité résiduelle = $u_{ij} - x_{ij}$
- **Arcs inverses** : (j,i) si $x_{ij} > 0$, capacité = x_{ij} (annuler flot)



Algorithmes de flot maximum

Ford-Fulkerson - $O(mf*)$

- **But** : Trouver flot de valeur maximale de source s vers puits t
- **Applications** : Réseau transport, affectation ressources, couplage
- **Principe général** :
1. Partir d'un flot initial (souvent flot nul)
2. Construire réseau d'augmentation du flot actuel
3. Chercher chemin augmentant de s à t (DFS par exemple)
4. Si chemin existe : augmenter flot et retour étape 2
5. Si aucun chemin : flot actuel est optimal
- **Terminaison** : Algorithme se termine quand aucun chemin augmentant
- **Complexité** : $O(mf*)$ où $f*$ = valeur flot maximum (non polynomial)

Edmonds-Karp - $O(m^2n)$

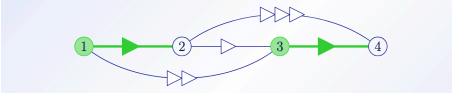
- **Amélioration de Ford-Fulkerson** : Choix du chemin augmentant
- **Stratégie** : Choisir plus court chemin (nombre d'arcs) via BFS
- **Avantages** :
- Complexité polynomiale garantie
- Évite cas pathologiques de Ford-Fulkerson
- Plus efficace en pratique sur graphes denses

Coupe et théorème max-flow min-cut

Coupe (S,T) : Partition de V avec $s \in S, t \in T$
Capacité coupe :

$$\sum_{(i,j):i\in S, j\in T} u_{ij}$$

Théorème Ford-Fulkerson : Valeur flot max = capacité coupe min



Flot maximum à coût minimum

Algorithme de Busacker-Gowen

- **But** : Flot de valeur maximale avec coût total minimal
- **Principe** : À chaque itération, saturer le plus court chemin (coût) dans réseau d'augmentation
- **Problème** : Arcs inverses ont coûts négatifs → impossibilité d'utiliser Dijkstra

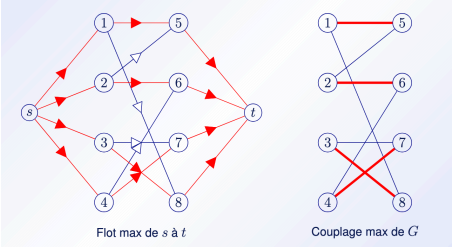
Fonction de potentiel (Edmonds-Karp)

- **Solution** : Transformer les coûts pour éliminer les valeurs négatives
- **Potentiel** : λ_i = distance depuis s dans réseau actuel
- **Coût réduit** : $c'_{ij} = c_{ij} + \lambda_i - \lambda_j$
- **Condition** : Réseau de base sans circuits de coût négatif

Applications des flots

Couplage maximum dans un graphe biparti

Transformation :
1. Orienter arêtes $A \to B$ (capacité 1)
2. Ajouter source s reliée à A (capacité 1)
3. Ajouter puits t relié depuis B (capacité 1)
4. Flot max = taille couplage max



Problème d'affectation linéaire

- **Contexte** : n personnes, n tâches, coût c_{ij} pour personne i sur tâche j
- **Objectif** : Affecter chaque personne à une tâche (coût minimum)
- **Méthode** : Couplage parfait de coût minimum → flot max-coût min

Problème de transbordement

Modélisation : Réseau $R = (V, E, c, u)$
- **Sources** : offre $b_i < 0$
- **Puits** : demande $b_i > 0$
- **Transit** : $b_i = 0$

Équation conservation :

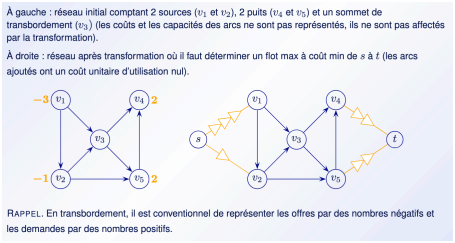
$$\sum_{j \in \text{Pred}(i)} x_{ji} - \sum_{j \in \text{Succ}(i)} x_{ij} = b_i$$

Condition équilibre :

$$\sum_{i \in V} b_i = 0$$

Transformation en flot max-coût min :
1. Source artificielle s → sources (coût 0, capacité = |offre|)
2. Puits → puits artificiel t (coût 0, capacité = demande)

Cas particuliers :
- **Transport** : graphe biparti complet (sources vers puits)
- **Affectation** : transport avec offres = demandes = 1

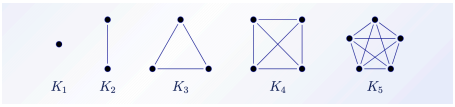


Autres types de graphes

Graphes complets et complémentaires

Graphe complet K_n : Graphe simple où toute paire sommets distincts reliée
- Nombre arêtes : $\binom{n}{2} = \frac{n(n-1)}{2}$
- Tous sommets ont degré $n-1$
- Exemple : K_4 a 6 arêtes, K_5 a 10 arêtes

Graphe complémentaire $\overline{G}$ de $G = (V, E)$:
- Mêmes sommets que G
- Arêtes = toutes arêtes possibles non présentes dans G
- $\overline{E} = \{\{u, v\} \mid \{u, v\} \notin E, u \neq v \text{ et } u, v \in V\}$
- Propriété : G et $\overline{G}$ forment partition complète des arêtes



Tournois

- **Définition** : Graphe orienté simple où chaque paire sommets reliée par exactement un arc
- **Construction** : Orientation complète d'un graphe complet
- **Propriétés fondamentales** :
- Graphe sous-jacent = graphe complet K_n
- Nombre total d'arcs = $\binom{n}{2}$
- Au plus 1 sommet sans prédécesseurs (source)
- Au plus 1 sommet sans successeurs (puits)

Caractérisation acyclique : Tournoi sans circuits $\iff$ matrice adjacence définit ordre strict total
Applications : Modélisation compétitions, classements, votes

Graphes bipartis

Définition : Graphe $G = (V, E)$ avec $V = A \cup B$ (A,B disjoints) tel que toute arête relie sommet de A à sommet de B **Notation** : $G = (A, B, E)$ ou $G = (A \cup B, E)$

Théorème caractérisation : Graphe biparti $\iff$ ne contient aucun cycle de longueur impaire

Graphes bipartis complets $K_{r,s}$:
- r sommets dans A, s sommets dans B
- Toute paire (a∈A, b∈B) reliée par arête
- Nombre arêtes = $r \times s$
- Applications : modélisation relations complètes entre deux ensembles

Recouvrements et transversaux

- **Recouvrement** : Sous-ensemble $R \subseteq E$ d'arêtes tel que chaque sommet du graphe est extrémité d'au moins une arête de R
- **Problème du recouvrement minimum** : Trouver recouvrement R de cardinal minimal
- **Transversal** : Sous-ensemble $T \subseteq V$ de sommets tel que chaque arête du graphe est incidente avec au moins un sommet de T
- **Problème du transversal minimum** : Trouver transversal T de cardinal minimal

Complexité algorithmique :
- Recouvrement minimum : **Polynomial** (problème "facile")
- Transversal minimum : **NP-difficile** (problème "difficile")

Couplages et chaînes augmentantes

Couplage : Ensemble M ⊆ E d'arêtes sans extrémités communes
- **Couplage parfait** : Sature tous les sommets du graphe
- **Couplage maximum** : Cardinal maximal parmi tous couplages possibles
- **Couplage maximal** : Ne peut être étendu ($\neq$ maximum)
- **Sommet saturé** : Incident à arête du couplage

Chaînes alternées (relativement à couplage M) : Chaîne dont arêtes alternent : dans M, hors M, dans M, hors M, …

Chaînes augmentantes (relativement à M) : Chaîne alternée avec extrémités NON saturées par M
Propriété clé : Permet augmenter taille couplage de 1

Théorème de Berge (1957) - Condition optimalité : Couplage M maximum $\iff$ graphe ne contient aucune chaîne augmentante relative à M

Démonstration :
- **($\Longrightarrow$)** Si C chaîne augmentante, alors $M' = M \triangle C$ (différence symétrique) est couplage de cardinal $|M| + 1$
- **($\Longleftarrow$)** Si M non maximum, ∃ couplage M' avec |M'| > |M|. Dans $M \triangle M'$, il existe une chaîne alternée avec plus d'arêtes de M' que de M → chaîne augmentante

Algorithme général de recherche couplage maximum :
1. Partir d'un couplage M (souvent vide)
2. Tant qu'il existe une chaîne augmentante C : a) Trouver chaîne augmentante C b) Remplacer M par $M \triangle C$ (différence symétrique)
3. M est maximum

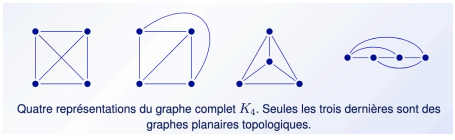
Cas graphes bipartis - Construction graphe orienté :
- Arêtes de M : orientées de B vers A
- Arêtes hors M : orientées de A vers B
- Explorer depuis sommets non saturés de A vers sommets non saturés de B
- Complexité : $O(mn)$ (Hopcroft-Karp : $O(m\sqrt{n})$)

Applications algorithmes : Base algorithmes hongrois, Blossom

Graphes planaires

Définitions et formule d'Euler

- **Planaire** : Représentable sur le plan sans croisements d'arêtes
- **Faces** : Régions délimitées par les arêtes (incluant face extérieure)
- **Formule d'Euler** : $n - m + f = 2$ (graphe connexe planaire)



Bornes et non-planarité

Inégalité générale (graphes simples connexes, $n \geq 3$) : $m \leq 3n - 6$

Démonstration :
- Chaque face bordée par ≥ 3 arêtes → $3f \leq 2m$
- Formule Euler : $f = 2 - n + m$
- Substitution : $3(2 - n + m) \leq 2m \rightarrow m \leq 3n - 6$

Inégalité bipartie (graphes bipartis simples connexes, $n \geq 4$) : $m \leq 2n - 4$

Démonstration :
- Graphe biparti : chaque face bordée par ≥ 4 arêtes → $4f \leq 2m$
- Même substitution → $m \leq 2n - 4$

Applications non-planarité :
- K_5 : $n = 5, m = 10$ mais $10\neg \leq 3(5) - 6 = 9 \rightarrow$ non planaire
- $K_{3,3}$: $n = 6, m = 9$ mais $9\neg \leq 2(6) - 4 = 8 \rightarrow$ non planaire

Théorème de Kuratowski (1930) :
- **Subdivision** : Graphe obtenu en insérant sommets au milieu d'arêtes
- **Théorème** : Graphe planaire $\iff$ ne contient aucune subdivision de K_5 ou $K_{3,3}$

Contraintes graphes potentiels-tâches