Institut National des Sciences Appliquées et
de Technologie

RECHERCHE OPÉRATIONNELLE

Mohamed Khalil Labidi

mohamed.khalil.labidi@gmail.com

Année universitaire : 2014 – 2015

Plan global

1. Introduction générale
2. Introduction à la théorie des graphes
3. Complexité des problèmes, méthodes de résolution
4. Programmation linéaire

Introduction générale

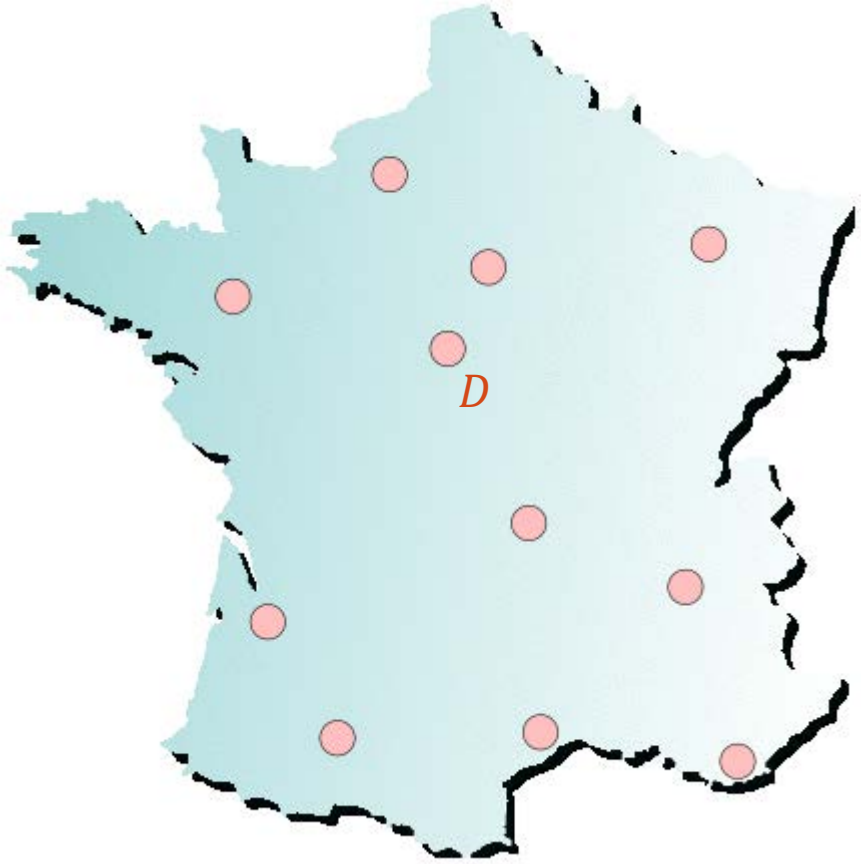
Introduction générale

1. Motivation
2. Définitions
3. Formalisation
4. Domaines d’application

Motivation

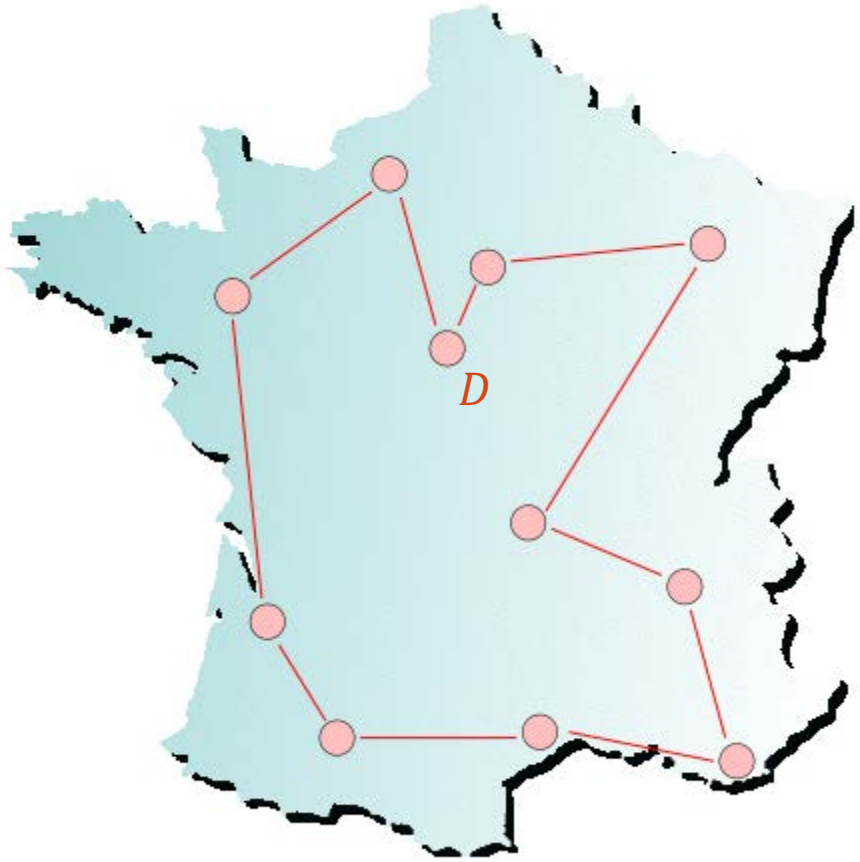
- Un voyageur de commerce, basé à Toulon, doit visiter ses clients à travers la France

- Il souhaite effectuer la tournée la plus courte possible



Motivation

- Données:
  - n villes avec une matrice de distances
- Solution:
  - Tournée visitant chaque ville et revenant au point de départ



Motivation


Villefranche-sur-Saône
Annecy
Sallanches
Champex
Tarare
Rillieux-la-Pape
Culoz
Megève
Saint-Gervais-les-Bains
Lyon
Meyzieu
Courmayeur
Belley
Aix-les-Bains
Vénissieux
Serrières
Veyrins
Bourg-Saint-Maurice
Bourgoin-Jallieu
Chambéry
Vallandry
Saint-Etienne
Virieu
Pontcharra
Moûtiers
Tignes
Méribel-les-Allues
Voiron
Tullins
Tencin
Saint-Jean-de-Maurienne
Vinay
Grenoble
Saint-Michel-de-Maurienne
Saint-Vallier
Jaume
D902
Susa
Rivoli
Tournon
Tournon-sur-Rhône
les Deux-Alpes
Valence
la Mure
Grenoble
Vallouise
Briançon
Pinerolo
Vivarais
la Voulte-sur-Rhône
la Chalp
Briançon
Abriès
Torre Pellice
Privas
Crest
Die
Lalley
Paesana
Orcières
Saint-Marcellin
Montélimar
Veynes
Gap
Embrun
Saint-Paul
Murazzo
Establet
Argentera
Donzère
Serres
Gap
Barcelonnette
Pierrelatte
Nyons
Bollène
Laragne-Montéglin
Vinadio
Vaison-la-Romaine
Auron
Isola
Sisteron
Orange
Avignon
Digne-les-Bains
Valdeblore
Carpentras
Daluis
Forcalquier
Avignon
Scaffarels
Rognonas
la Brillanne
Apt
Castellane
Nîmes
Cavaillon
Manosque
Vence
Nice
Montpellier
Sénas
Pertuis
Grasse
Salon-de-Provence
Montmeyan
Antibes
Miramas
Aix-en-Provence
Fréjus
Istres
Lorgues
Vitrolles
Trets
Saint-Raphaël
les Pennes-Mirabeau
la Garde-Freinet
Sausset-les-Pins
Marseille
Saint-Tropez
la Ciotat
Toulon
Cavalaire-sur-Mer
Sanary-sur-Mer
Hyères
le Lavandou
la Bergerie
Toulon

Motivation

- Pouvons-nous fournir des méthodes pour répondre à un type précis de problème, c'est-à-dire à élaborer une démarche universelle pour un type de problème qui aboutit à la ou les solutions les plus efficaces?

Définitions

- *Operational research UK, US operations research, Or simply OR*
- Cambridge Dictionary:
  - The systematic study of how best to solve problems in business and industry
- Wikipedia:
  - Is the use of mathematical models, statistics and algorithms to aid in decision-making
- Roadef (société française de recherche opérationnelle et d'aide à la décision):
  - Approche scientifique pour la résolution de problèmes de gestion de systèmes complexes

Formalisation

Face à un problème pratique de décision

- Aspects mathématiques
  - Contraintes, objectifs, simplifications
- Modélisation
  - Graphes, programmation linéaire, PPC…
- Analyse des modèles et résolution
  - Etude de complexité: que peut-on espérer pour le temps disponible?

Formalisation

Face à un problème pratique de décision

- Mise au point d'algorithmes
- Implémentation et analyse des résultats
  - Valider par rapport à la demande
  - Itérer avec le demandeur si nécessaire
- Déploiement des solutions
  - Intégration logicielle

Domaines d’application

- Conception, configuration et exploitation de systèmes techniques complexes (réseaux de communication, systèmes d'information…)
- Gestion de la chaîne logistique (transports, production, stocks...)
- Gestion stratégique d'investissements
- Et aussi: santé, instruction publique, voirie, ramassage et distribution de courrier, production et transport d‘énergie, télécommunications, banques, assurances...

Domaines d’application

- **Production:** maximiser le profit selon la disponibilité de la main d‘œuvre, demande du marché, capacité de production, prix de revient du matériau brut…

- **Transport:** minimiser la distance totale parcourue selon les quantités de matériaux à transporter, capacité des transporteurs, points de ravitaillement en carburant…

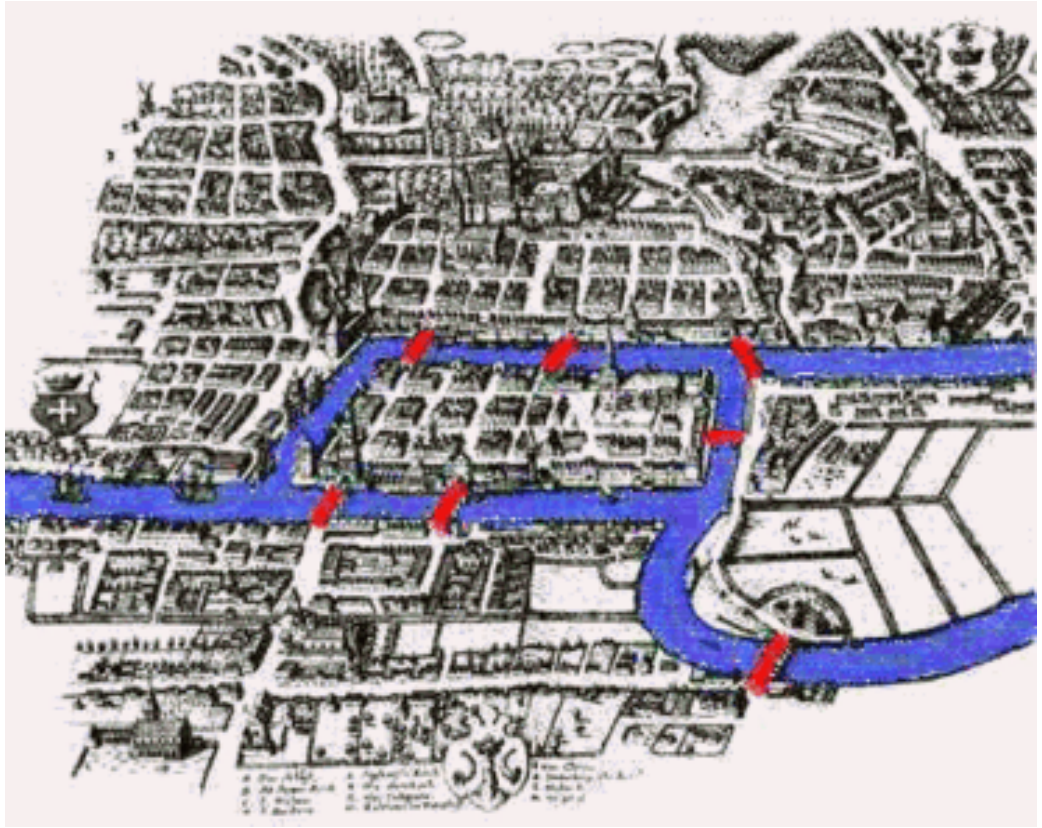
Introduction à la théorie des graphes

Introduction à la théorie des graphes

1. Introduction
2. Notions fondamentales sur les graphes
3. Représentation d'un graphe
4. Arbres, forêts et arborescences
5. Problèmes
    1. Le problème du plus court chemin
    2. Le problème de l'arbre couvrant minimum
    3. Le problème de flot maximum

Introduction

- XVIII^{ème} siècle
- Ville de Königsberg



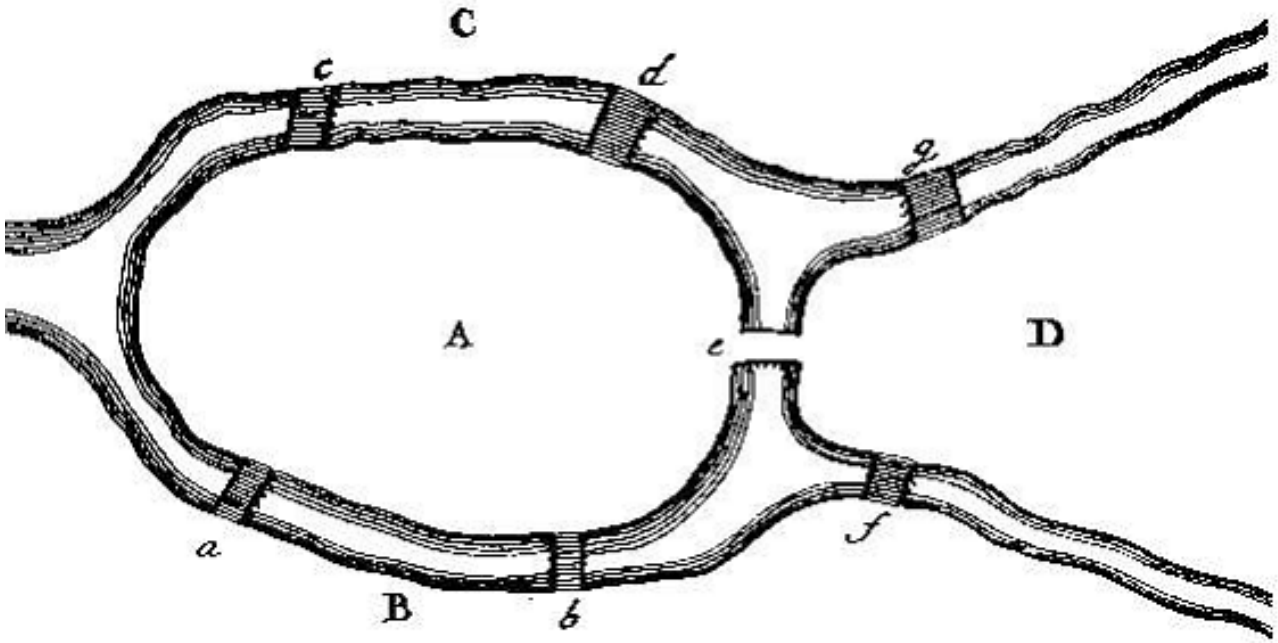
Introduction

- Peut-on traverser d'une seule traite les 7 ponts de la ville sans avoir à repasser par l'un d'entre eux ?

Introduction

- Euler (1707-1783)
- Résolution de l’énigme en 1736
- Représentation de cette situation à l'aide d'un « dessin »

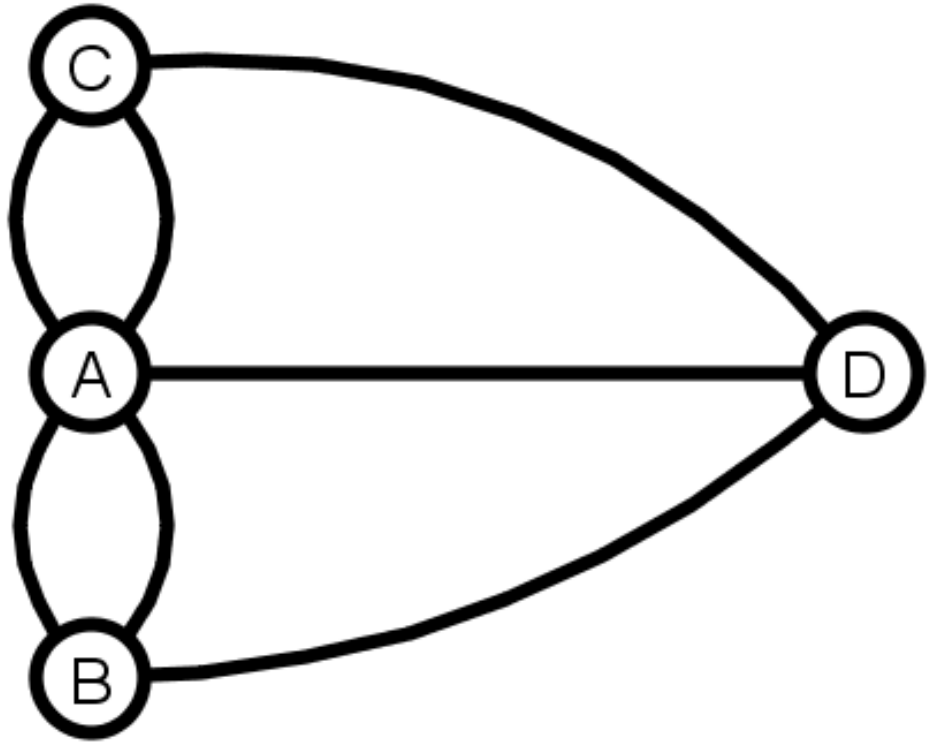
Introduction



Introduction

- Les sommets représentent les terres et les arêtes les ponts

Introduction



Notions fondamentales sur les graphes

- Un graphe $G = (X, U)$ est déterminé par la donnée :
  - D'un ensemble X non vide et fini dont les éléments sont appelés sommets ou nœuds. Si $n = |X|$ est leur nombre, on dira que G est d'ordre $\boldsymbol{n}$
  - D'un ensemble $U \subseteq X \times X$ dont les éléments sont des couples ordonnés de sommets appelées arcs ou arêtes. On notera souvent $|U| = \boldsymbol{m}$

Notions fondamentales sur les graphes

- Si $u = (i, j)$ est un arc de G, i et j sont les extrémités de u et sont dites adjacentes. Si u est orienté de i vers j, alors i est l'extrémité initiale de u et j est l'extrémité terminale de u
- Si tous les nœuds de G sont mutuellement connectés, on aura $m = \frac{n(n-1)}{2}$ et G sera dit complet
- Le degré $d(i)$ d'un sommet i est égal au nombre d'arcs dont i est une extrémité

Notions fondamentales sur les graphes

- Dans un graphe orienté, le demi-degré entrant $d^+(i)$ (resp. sortant $d^-(i)$d'un sommet i est égal au nombre d'arcs dont i est une extrémité terminale (resp. initiale) et nous avons $d(i) = d^+(i) + d^-(i)$
- Un graphe est simple s'il ne contient ni boucle ni arêtes multiples, par boucle on sous-entend une arête dont ses deux extrémités sont identiques et par arêtes multiples les différentes arêtes reliant les mêmes extrémités initiale et terminale

Notions fondamentales sur les graphes

- On dira aussi qu'un graphe $G' = (X', U')$ est un sous-graphe d'un graphe $G = (X, U)$ lorsque $X' \subseteq X$ et $U' \subseteq U$
- Un sous-graphe recouvrant d'un graphe $G = (X, U)$ est un sous-graphe $G' = (X, U')$ c'est-à-dire un sous-graphe dont sont sommets tous les sommets de G
- Une chaine de longueur q (de cardinalité q) est une séquence de q arcs : $L = \{u_1, u_2, \dots, u_q\}$ telle que chaque arc U_r de la séquence $(2 \leq r \leq q - 1)$ ait une extrémité commune avec l'arc u_{r-1} $(u_{r-1} \neq u_r)$ et l'autre extrémité commune avec l'arc u_{r+1} $(u_{r+1} \neq u_r)$

Notions fondamentales sur les graphes

- L'extrémité i de u_1 non adjacente à u_2, et l'extrémité j de u_q non adjacente à u_{q-1} sont appelées les extrémités de la chaine L
- Une chaîne dont le nœud de départ et le nœud d'arrivée sont identiques s'appelle cycle
- Un chemin est une séquence finie et alternée de sommets et d'arcs, débutant et finissant par des sommets, telle que chaque arc est sortant d'un sommet et incident au sommet suivant dans la séquence
- Un chemin dont le nœud de départ et le nœud d'arrivée sont identiques s'appelle circuit

Notions fondamentales sur les graphes

- Un graphe est dit connexe si, pour tout couple de sommets i et j, il existe une chaine joignant i et j. La relation:

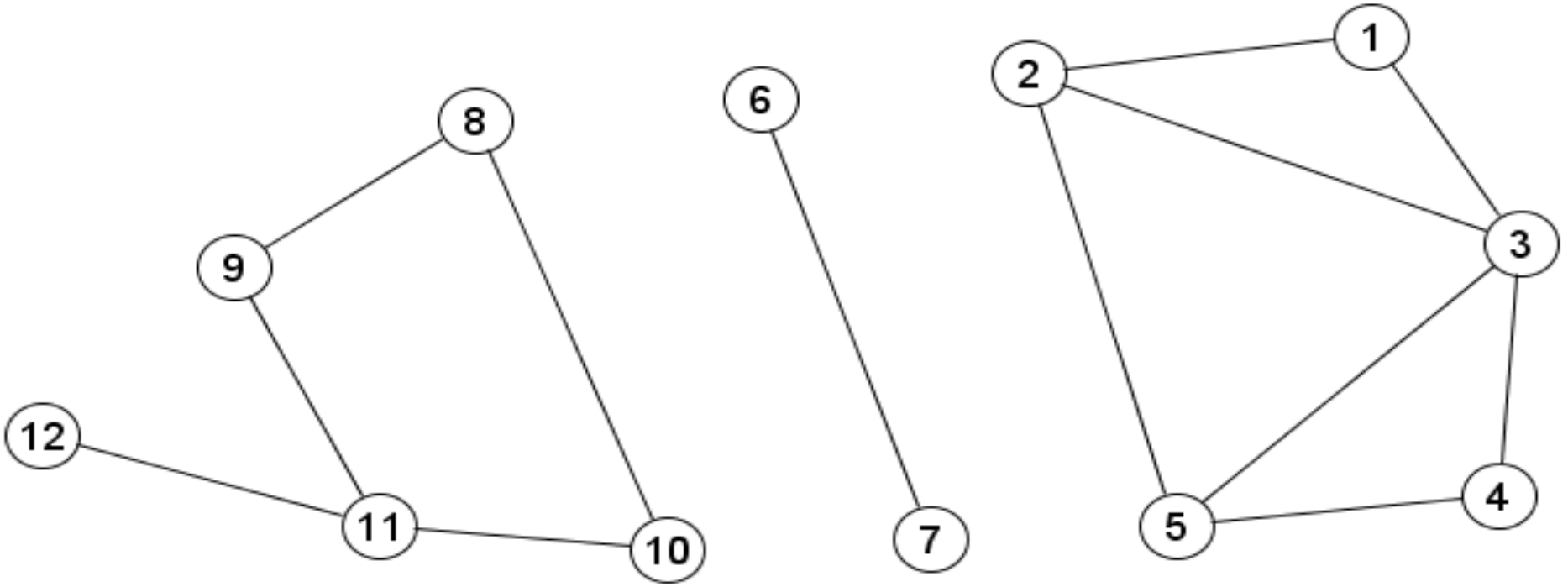
$$i \mathcal{R} j \Leftrightarrow \begin{cases} soit\ i = j \\ soit\ il\ existe\ une\ chaine\ joigant\ i\ et\ j \end{cases}$$

est une relation d'équivalence (réflexive, symétrique, transitive)

Notions fondamentales sur les graphes

- Les classes d'équivalence induites sur X par cette relation forment une partition de X en : $X_1, X_2, \dots, X_p$. Le nombre de classes d'équivalence distinctes, noté p, est appelé le nombre de connexité du graphe. Un graphe est dit connexe si et seulement si son nombre de connexité p est égal à 1.
- Un sous-graphe G_A engendré par $A \subset X$ dont les sommets sont les éléments de A, et les arêtes sont les arêtes de G ayant leurs deux extrémités dans A, les sous-graphes $G_1, G_2, \dots, G_p$ engendrés par les sous-ensembles $X_1, X_2, \dots, X_p$ sont appelés composantes connexes (CC) du graphe G

Notions fondamentales sur les graphes



Notions fondamentales sur les graphes

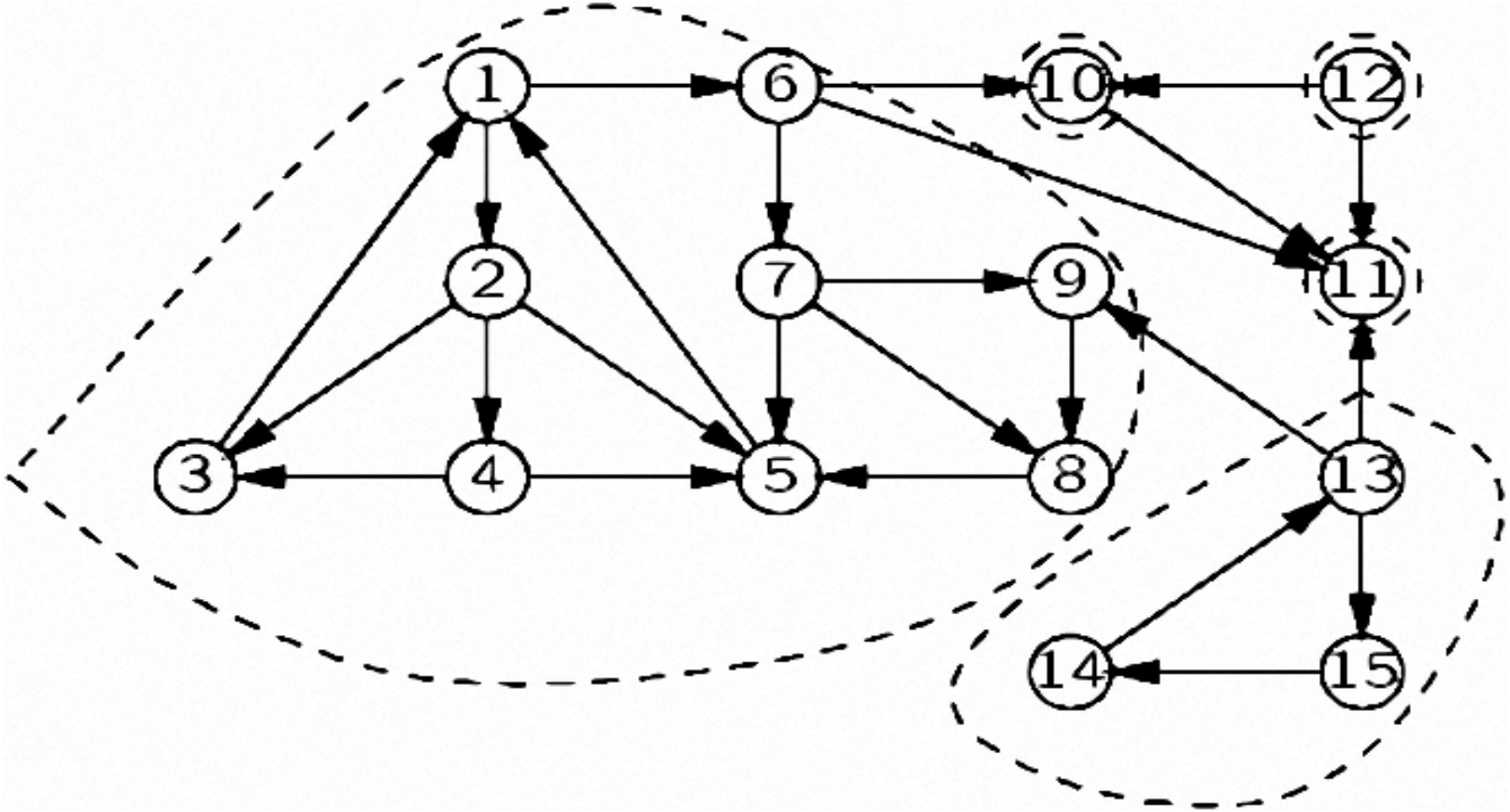
- Un graphe est fortement connexe est un graphe orienté dont toutes les paires de sommets peuvent être reliées par un chemin. La relation:

$$i \mathcal{R} j \Leftrightarrow \begin{cases} soit\ i = j \\ soit\ il\ existe\ un\ chemin\ allant\ de\ i\ \text{à}\ j \\ \boldsymbol{et}\ un\ chemin\ de\ j\ \text{à}\ i \end{cases}$$

est une relation d'équivalence.

- Les classes d'équivalence induites sur X par cette relation forment une partition de X en : $X_1, X_2, \dots, X_p$. Les sous-graphes $G_1, G_2, \dots, G_p$ engendrés par les sous-ensembles $X_1, X_2, \dots, X_p$ sont appelés les composantes fortement connexes (CFC) de G

Notions fondamentales sur les graphes



Notions fondamentales sur les graphes

- La distance entre deux sommets d'un graphe connexe (ou entre 2 sommets d'une même composante connexe) est égale à la longueur (nombre d'arêtes) de la plus courte chaine les reliant

- On appelle ainsi diamètre d'un graphe G, noté $\delta(G)$, la distance maximale entre deux sommets du graphe G

Représentation d’un graphe

- Comment stocker un graphe d’une manière efficace ?

Représentation d’un graphe

espace mémoire

la complexité d’accès

Représentation d'un graphe

- Aucune représentation n'est parfaite
- Choix de la structure à adopter en fonction du graphe et l'algorithme
- Trois structures très utilisées:
  - Matrice d'incidence nœud-arc
  - Matrice d'adjacence nœud-nœud
  - Liste d'adjacence

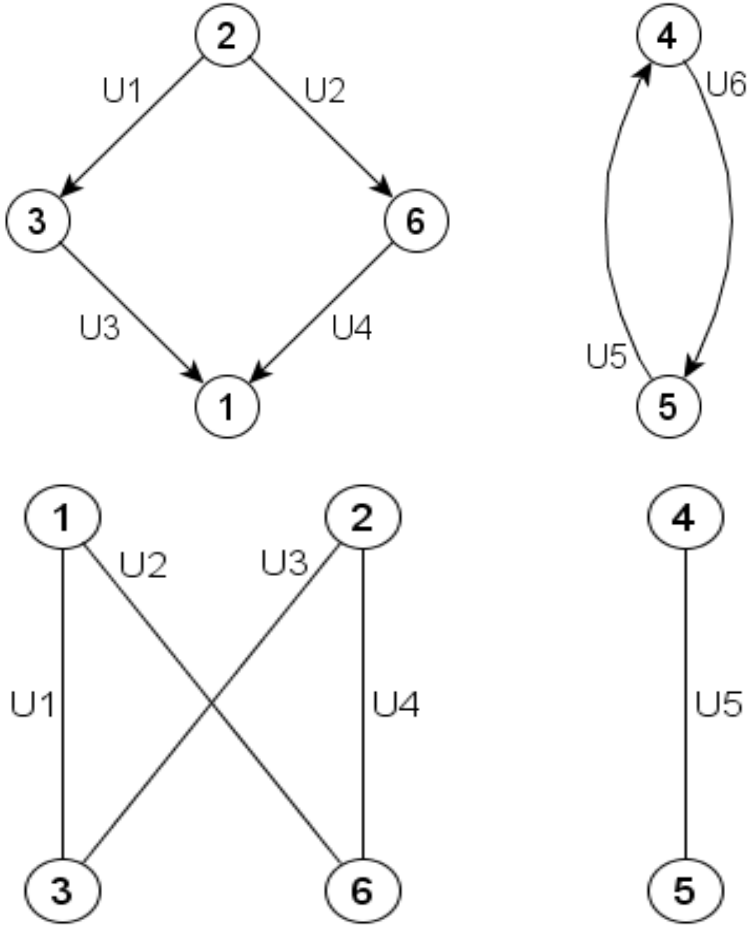
Représentation d'un graphe

Matrice d'incidence nœud-arc:

- Matrice $\mathcal{A}(G)$ de taille $n \times m$

$$a_{ik} = \begin{cases} 1 & si\ le\ sommet\ i\ est\ l'origine\ de\ l'arc\ k \\ -1 & si\ le\ sommet\ i\ est\ la\ destination\ de\ l'arc\ k \\ 0 & partout\ ailleurs \end{cases}$$

- Dans le cas non orienté, les coefficients valent 1 si le sommet en est une extrémité de l'arête et 0 sinon

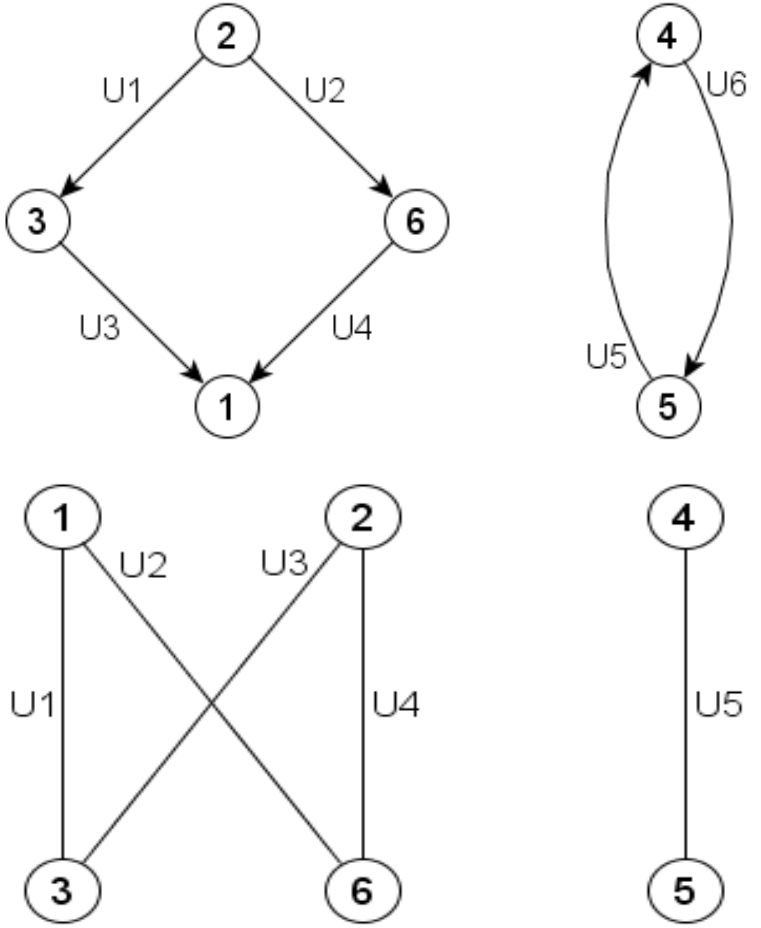


Représentation d'un graphe

Matrice d'incidence nœud-arc:

- Cas orienté:

	u_1	u_2	u_3	u_4	u_5	u_6
1	0	0	-1	-1	0	0
2	1	1	0	0	0	0
3	-1	0	1	0	0	0
4	0	0	0	0	-1	1
5	0	0	0	0	1	-1
6	0	-1	0	1	0	0

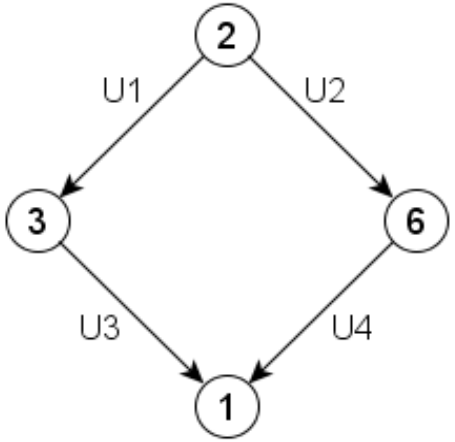


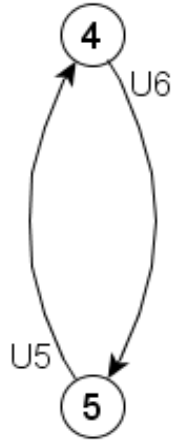
Représentation d'un graphe

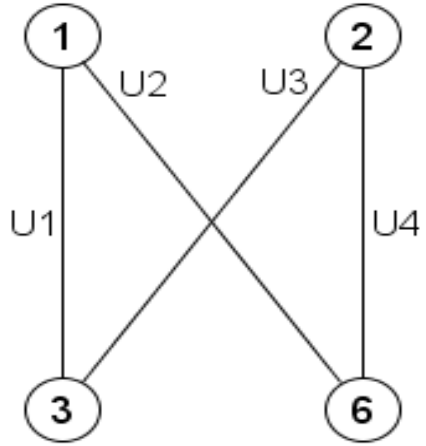
Matrice d'incidence nœud-arc:

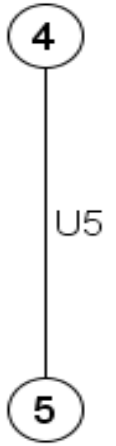
- Cas non orienté:

	u_1	u_2	u_3	u_4	u_5
1	1	1	0	0	0
2	0	0	1	1	0
3	1	0	1	0	0
4	0	0	0	0	1
5	0	0	0	0	1
6	0	1	0	1	0









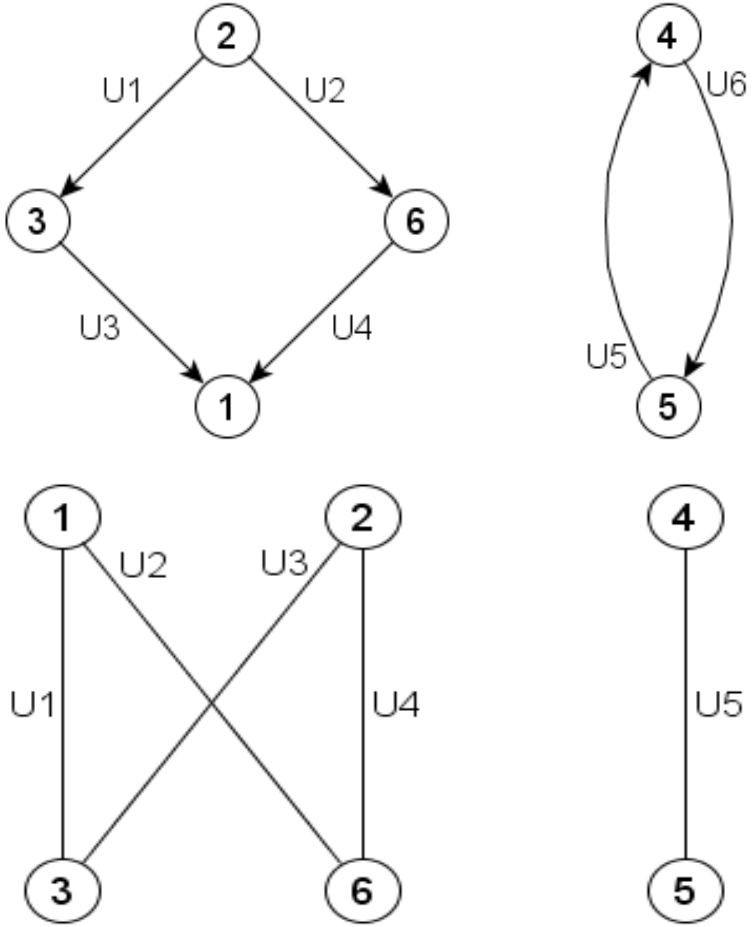
Représentation d'un graphe

Matrice d'adjacence nœud-noeud:

- Matrice $\mathcal{B}(G)$ de taille $n \times n$

$$b_{ij} = 1 \; si \; et \; seulement \; si \; (i,j) \in U \; (b_{ij} = 0 \; sinon)$$

- **Remarque:** G ne peut contenir ni d'arêtes multiples, ni plus d'une boucle par sommet

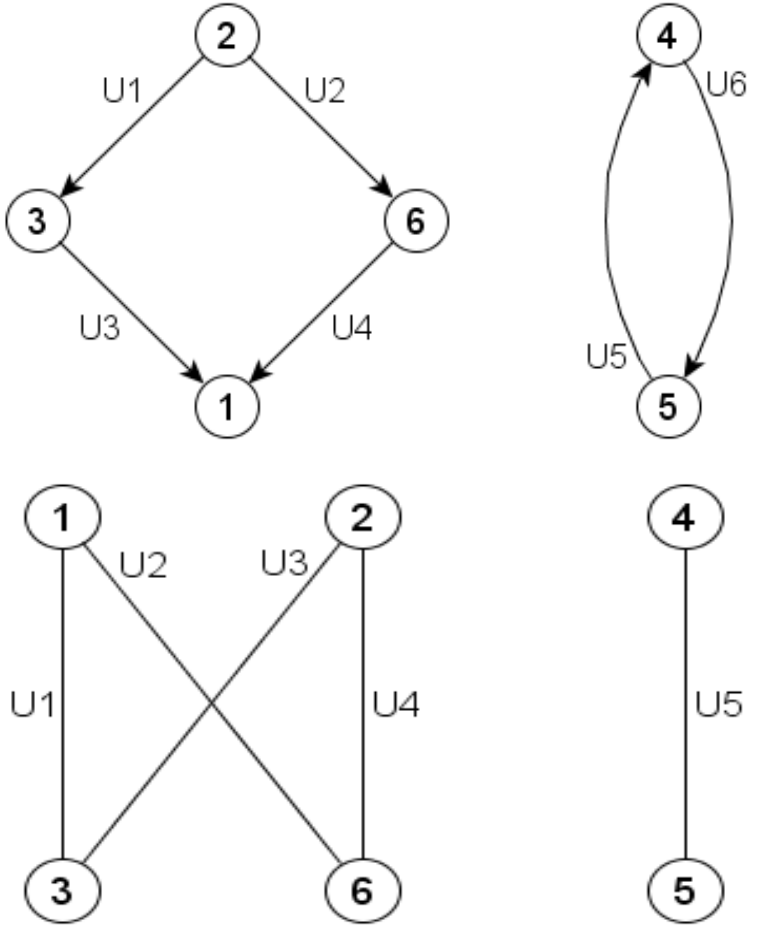


Représentation d'un graphe

Matrice d'adjacence nœud-noeud:

- Cas orienté:

	1	**2**	**3**	**4**	**5**	**6**
1	0	0	0	0	0	0
2	0	0	1	0	0	1
3	1	0	0	0	0	0
4	0	0	0	0	1	0
5	0	0	0	1	0	0
6	1	0	0	0	0	0

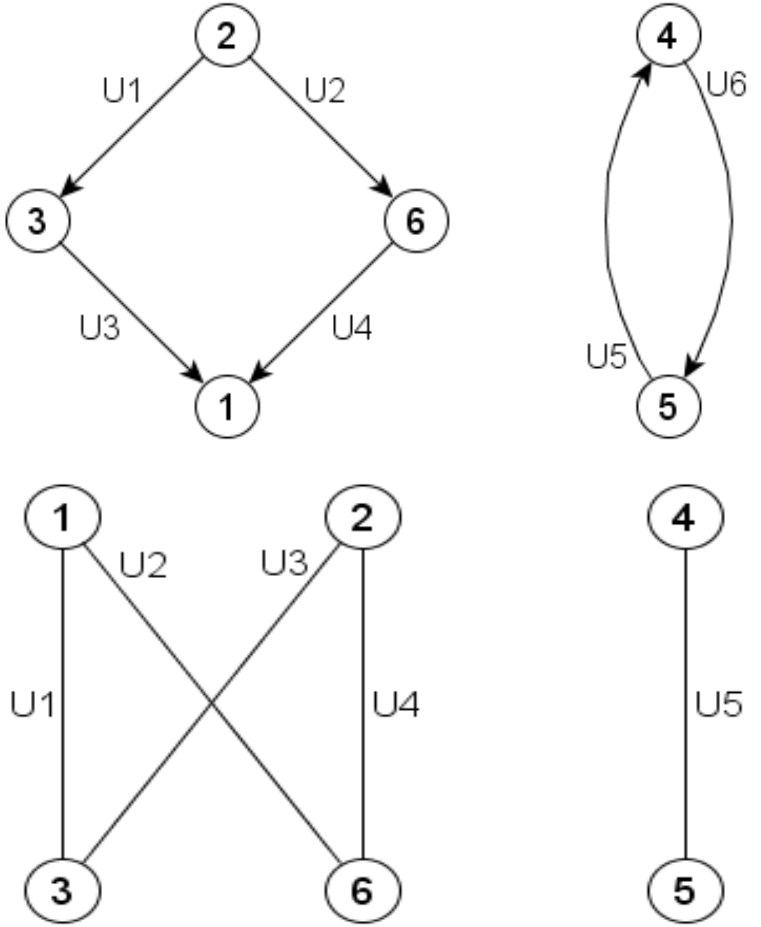


Représentation d'un graphe

Matrice d'adjacence nœud-noeud:

- Cas non orienté:

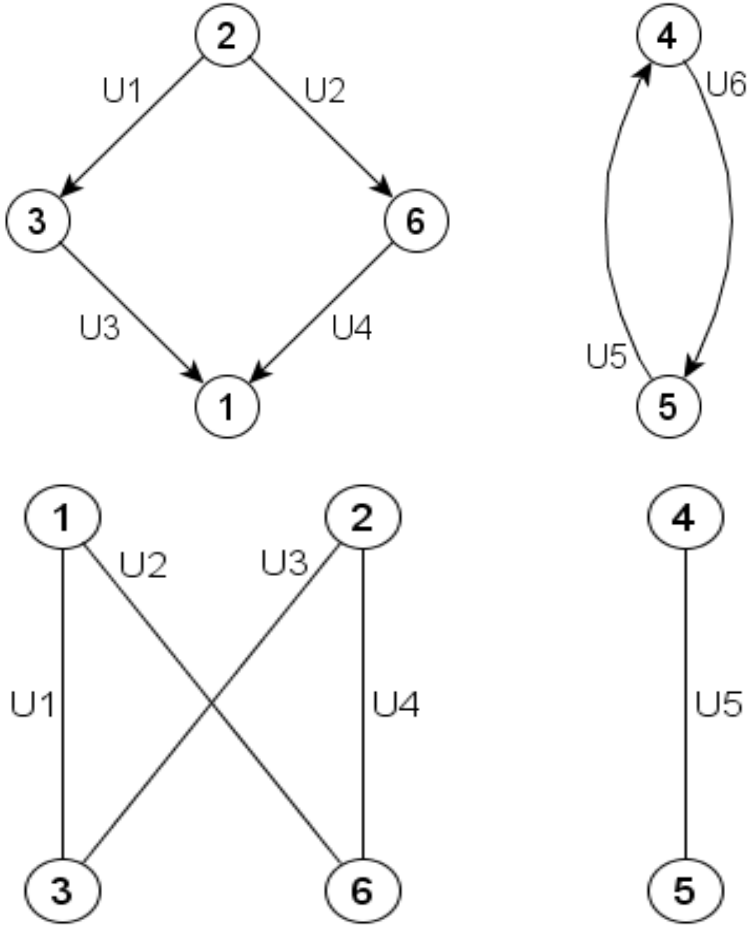
	1	**2**	**3**	**4**	**5**	**6**
1	0					
2	0	0				
3	1	1	0			
4	0	0	0	0		
5	0	0	0	1	0	
6	1	1	0	0	0	0



Représentation d'un graphe

Liste d'adjacence:

- Chaque élément de la première représente un nœud qui à son tour comporte une structure à une dimension qui énumèrera les successeurs du sommet en question

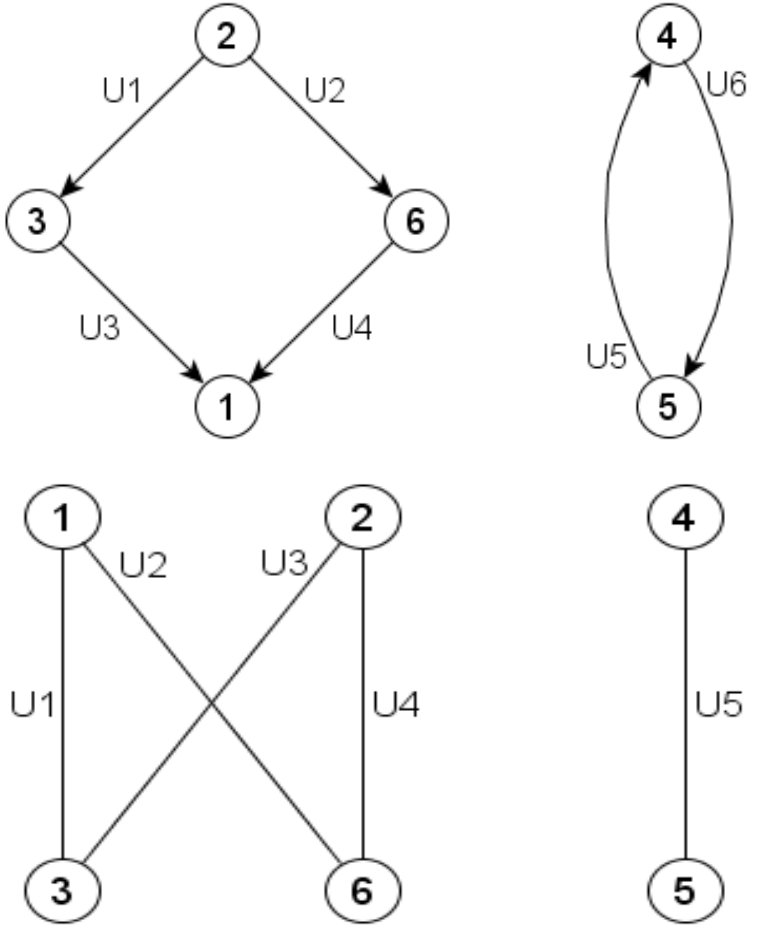


Représentation d'un graphe

Liste d'adjacence:

- Cas orienté:

1	→	∅	
2	→	3	6
3	→	1	
4	→	5	
5	→	4	
6	→	1	

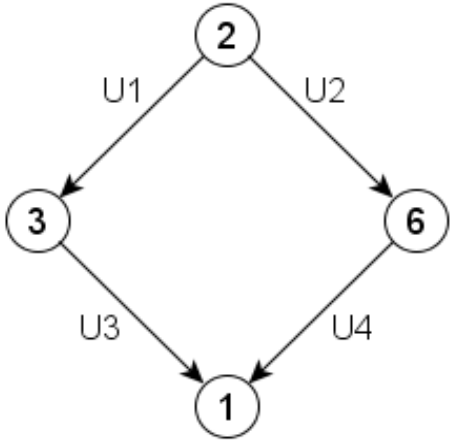


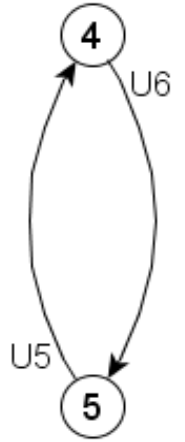
Représentation d'un graphe

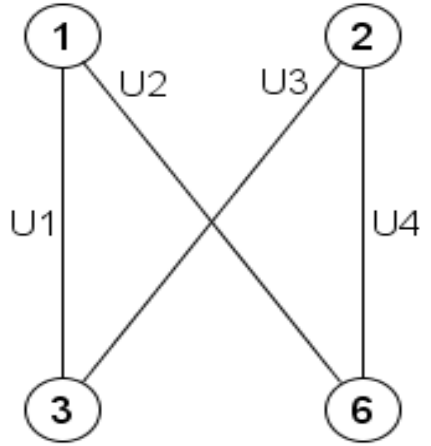
Liste d'adjacence:

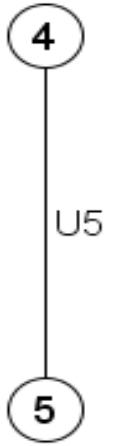
- Cas non orienté:

1	→	3	6
2	→	3	6
3	→	1	2
4	→	5	
5	→	4	
6	→	1	2









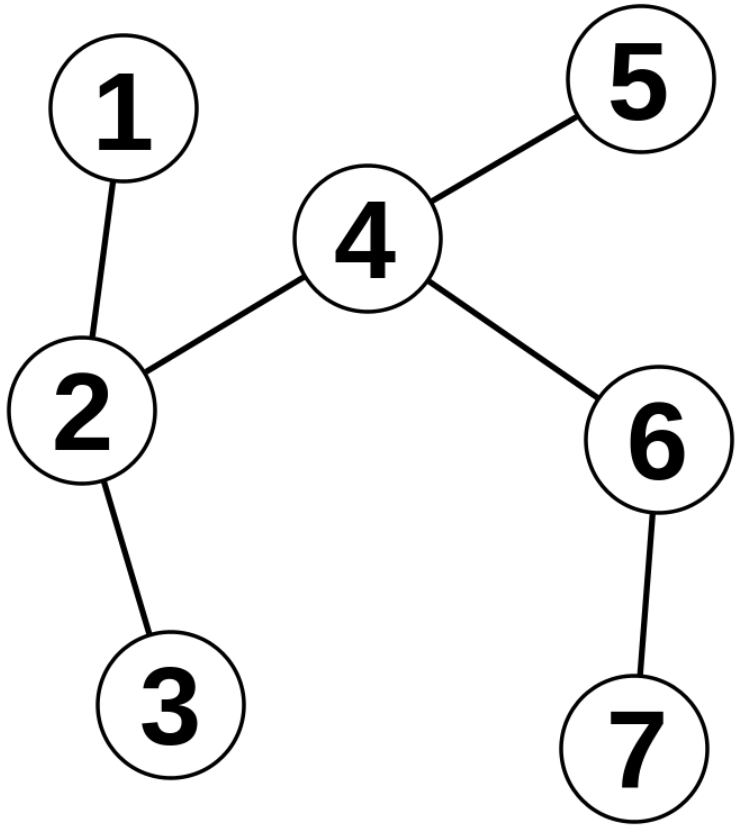
Arbres, forêts et arborescences

- Un arbre peut être définit de plusieurs manières, nous citons quelques unes.

Un graphe G est un arbre si et seulement si:

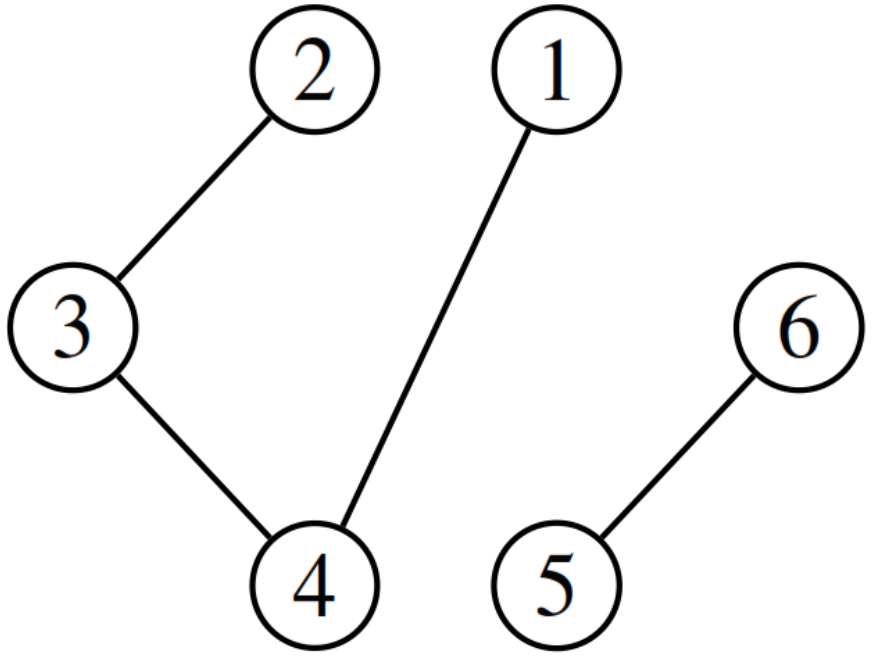
- Il est connexe et sans cycle
- Il est connexe minimum au sens des arêtes, c'est-à-dire qu'il n'est plus connexe si on lui supprime l'une quelconque de ses arêtes
- Il est sans cycles et maximum au sens des arêtes, c'est-à-dire qu'on crée un cycle en ajoutant une arête rendant adjacents deux quelconques de ses sommets qui ne l'étaient pas
- Il est connexe (ou sans cycles) et possède $n - 1$ arêtes

Arbres, forêts et arborescences



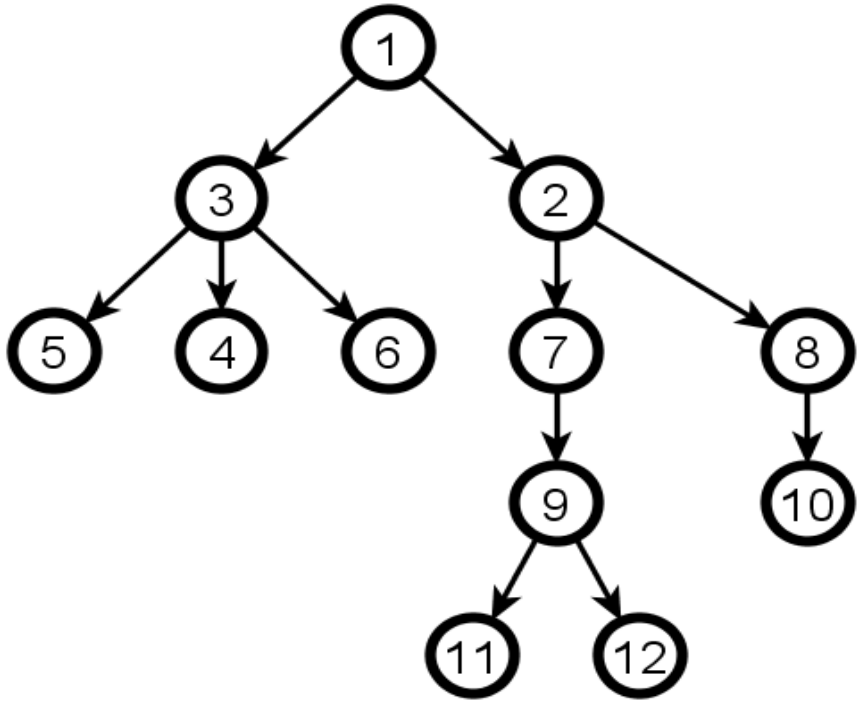
Arbres, forêts et arborescences

- Une forêt est un graphe sans cycle (ou un ensemble d'arbre)



Arbres, forêts et arborescences

- Un graphe G est une arborescence s'il existe un sommet r appelé racine de G tel que pour tout sommet s de G, il existe un chemin et un seul de r vers s



Problèmes: Le problème du plus court chemin

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

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

Problèmes: Le problème du plus court chemin

- **Lemme de Koenig:** Un plus court chemin entre 2 sommets est élémentaire
  - Nombre de chemins devient fini
  - De l'ordre de $n!$

- **Principe de sous-optimalité:** Si $\mu = (i, .., j)$ est un plus court chemin entre i et j, alors pour tout sommet x sur le chemin μ:
  - Le sous-chemin de μ jusqu'à x, $(i, .., x)$, est le plus court chemin de i à x
  - Le sous-chemin de μ depuis x, $(x, .., j)$, est le plus court chemin de x à j

Problèmes: Le problème du plus court chemin

- **Lemme de Koenig:** Un plus court chemin entre 2 sommets est élémentaire
  - Nombre de chemins devient fini
  - De l'ordre de $n!$

- **Principe de sous-optimalité:** Si $\mu = (i, .., j)$ est un plus court chemin entre i et j, alors pour tout sommet x sur le chemin μ:
  - Le sous-chemin de μ jusqu'à x, $(i, .., x)$, est le plus court chemin de i à x
  - Le sous-chemin de μ depuis x, $(x, .., j)$, est le plus court chemin de x à j

Problèmes: Le problème du plus court chemin

Algorithme de Dijkstra:

- Permet de calculer le plus court chemin entre un sommet et tous les autres
- Graphe G à longueurs positives
- Numérotation des sommets de G de 1 à n
- Recherche des chemins partant du sommet 1
- Construction d'un vecteur $\lambda = (\lambda(1); \lambda(2); \dots; \lambda(n))$ tel que $\lambda(j)$ soit égale à la longueur du plus court chemin allant de 1 au sommet j
- Construction d'un vecteur p pour mémoriser le chemin pour aller du sommet 1 au sommet voulu. La valeur $p(j)$ donne le sommet qui précède j dans le chemin

Problèmes: Le problème du plus court chemin

Algorithme de Dijkstra:

- λ est initialisé à c_{1j}, tel que:

$$c_{ij} = \begin{cases} 0 & si\ i = j \\ \infty & si\ i \neq j\ et\ (i,j) \notin E \\ l(i,j) & si\ i \neq j\ et\ (i,j) \in E \end{cases}$$

- $p(j)$ initialement est égal à NULL, ou 1 si $(1,j) \in E$
- Deux ensembles de sommets sont considérés, S initialisé à $\{1\}$ et T à $\{2, \dots, n\}$
- À chaque itération, on ajoute à S un sommet de T de telle sorte que le vecteur λ donne à chaque étape la longueur minimale des chemins de 1 aux sommets de S

Problèmes: Le problème du plus court chemin

Algorithme de Dijkstra:

Initialisations

$\lambda(j) = c_{1j}$ et $p(j) = NULL$, pour $1 \leq j \leq n$

Pour $2 \leq j \leq n$ **faire**

Si $c_{1j} < \infty$ **alors**

$p(j) = 1$

$S = \{1\}\,; T = \{2, \dots, n\}$

Problèmes: Le problème du plus court chemin

Algorithme de Dijkstra:

Itérations

Tant que T n'est pas vide **faire**

choisir i dans T tel que $\lambda(i)$ est minimum

retirer i de T et l'ajouter à S

Pour chaque successeur j de i, avec j dans T, **faire**

Si $\lambda(j) > \lambda(i) + l(i,j)$ **alors**

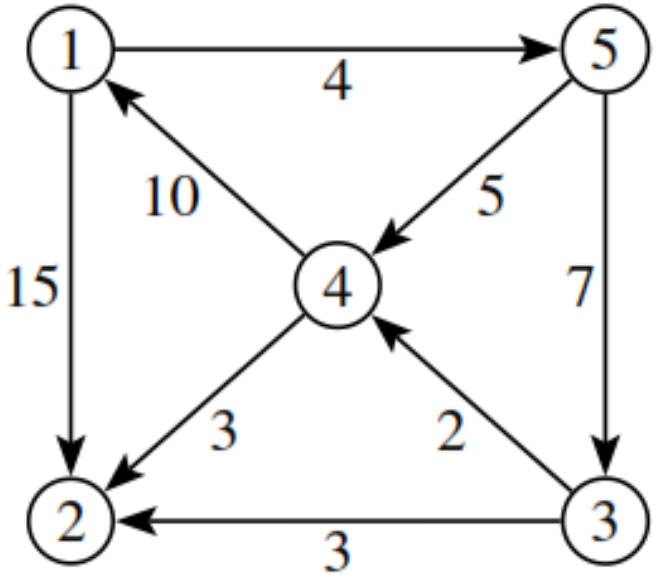
$\lambda(j) = \lambda(i) + l(i,j)$

$p(j) = i$

Problèmes: Le problème du plus court chemin

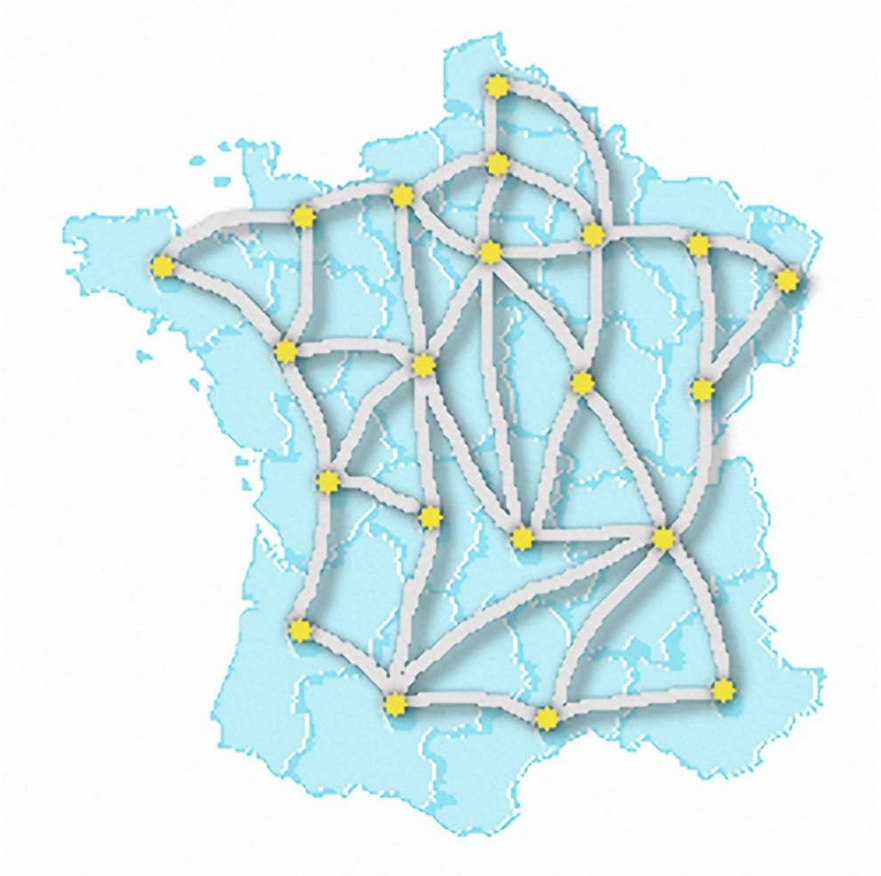
Algorithme de Dijkstra:

Exp:



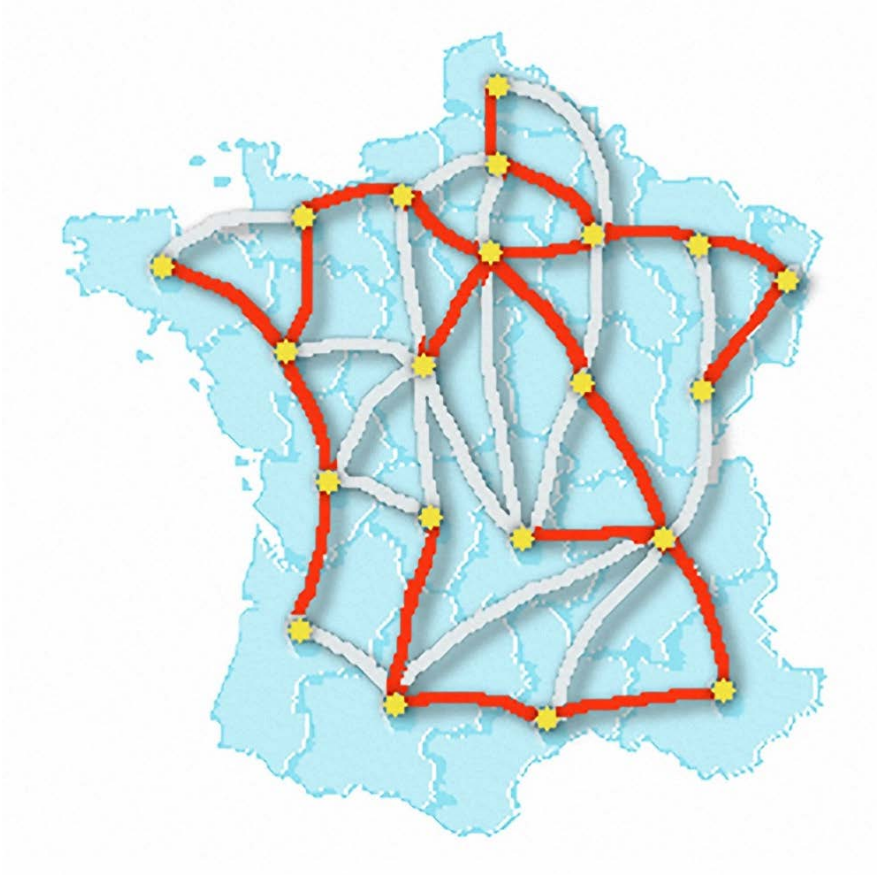
Problèmes: Le problème de l'arbre couvrant minimum

- On souhaite connecter des villes entre elles avec un nouveau réseau très haut débit
- Un certains nombre de connexions directes point à point entre les villes sont techniquement possibles
- Il nous faut choisir lesquelles parmi ces connexions nous allons effectivement mettre en place



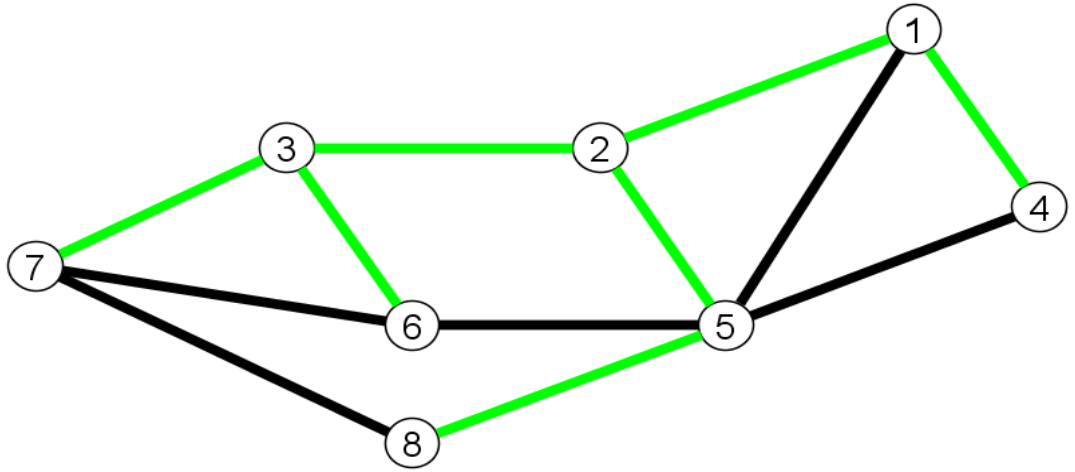
Problèmes: Le problème de l’arbre couvrant minimum

- La distance entre 2 villes dans le réseau final a peu d'importance au vu des débits prévus
- Les coûts d'installation des liaisons ne sont pas les mêmes
- Comment connecter toutes les villes en minimisant le coût total du réseau?



Problèmes: Le problème de l'arbre couvrant minimum

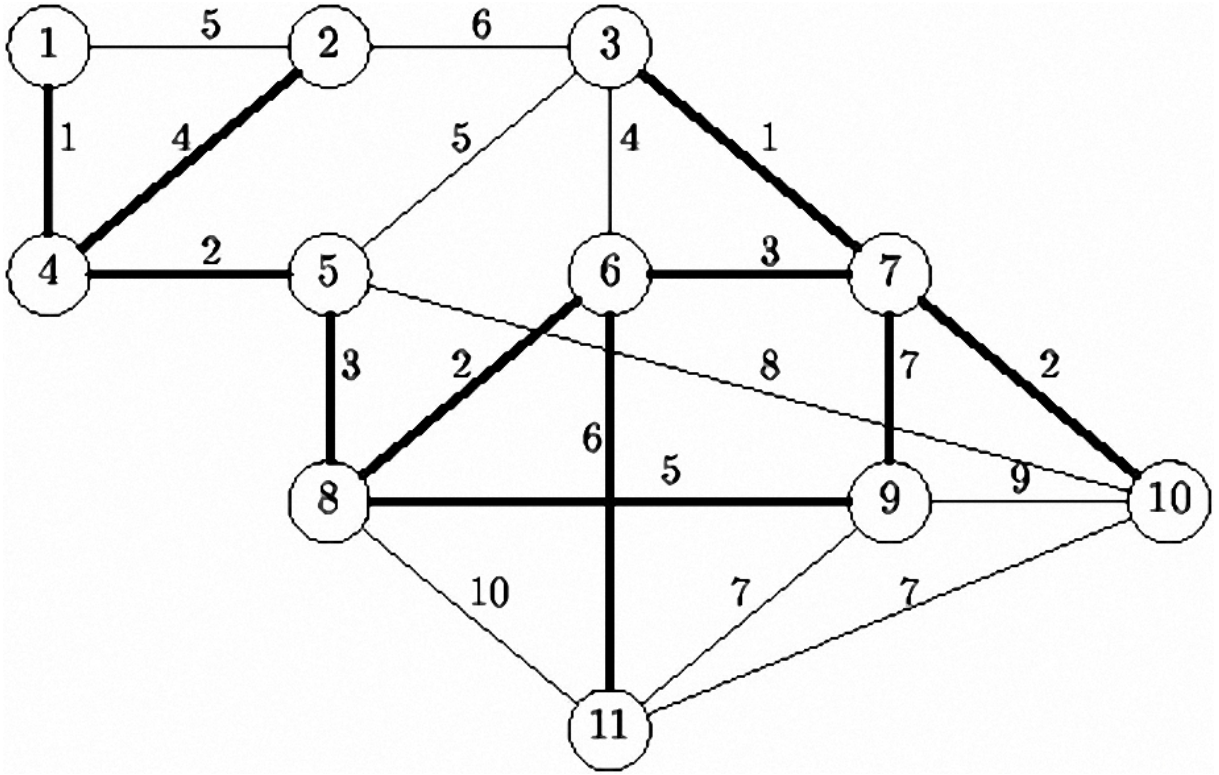
- **Arbre de recouvrement:** Un arbre de recouvrement (AR) $G_{AR} = (X, U^*)$ de G est un sous-graphe recouvrant de G qui est un arbre, les arêtes de G_{AR} sont appelées branches



Problèmes: Le problème de l'arbre couvrant minimum

- **Formulation:** étant un graphe $G = (X, U)$ non orienté, connexe et pondéré par une fonction l attachée aux arêtes. On souhaite trouver dans G un AR de poids total minimum

Problèmes: Le problème de l'arbre couvrant minimum



Problèmes: Le problème de l'arbre couvrant minimum

- Plusieurs algorithmes de résolution
- Exploitation des caractéristiques des arbres
  - Graphes connexes minimaux
  - Graphes acycliques (sans cycles) maximaux
- Les deux algorithmes les plus connus:
  - Prim
  - Kruskal

Problèmes: Le problème de l’arbre couvrant minimum

Algorithme de Prim:

- Exploite la première caractéristique
- Dans un arbre couvrant T, il existe nécessairement une arête qui relie l'un des sommets de T avec un sommet en dehors de T
- L'idée est de maintenir un sous-graphe partiel connexe, en le connectant à un nouveau sommet à chaque étape
- Pour construire un arbre couvrant de poids minimum, il suffirait de choisir à chaque fois parmi les arêtes sortantes celle de poids le plus faible pour que l’augmentation soit la plus économique possible

Problèmes: Le problème de l'arbre couvrant minimum

Algorithme de Prim:

Initialisations

$F = VIDE$ /* F est l'ensemble des arêtes de l'arbre */

marquer arbitrairement un sommet

Itérations

Tant Que il existe un sommet non marqué adjacent à un sommet marqué **faire**

Sélectionner un sommet y non marqué adjacent à un sommet marqué x
tel que (x, y) est l'arête sortante de plus faible poids
$F = F \cup \{(x, y)\}$
marquer y

Problèmes: Le problème de l'arbre couvrant minimum

Algorithme de Kruskal:

- Exploite la deuxième caractéristique
- L'ajout d'une arête à un arbre T crée un cycle
- Construction de l'arbre couvrant en ne raisonnant que sur les arêtes
- Construction par ajout à chaque fois d'une arête, si et seulement si le graphe reste acyclique
- On a intérêt à ajouter les arêtes de poids le plus faible
- Parcours d'une liste des arêtes triée dans l'ordre des poids croissants

Problèmes: Le problème de l'arbre couvrant minimum

Algorithme de Kruskal:

Initialisations

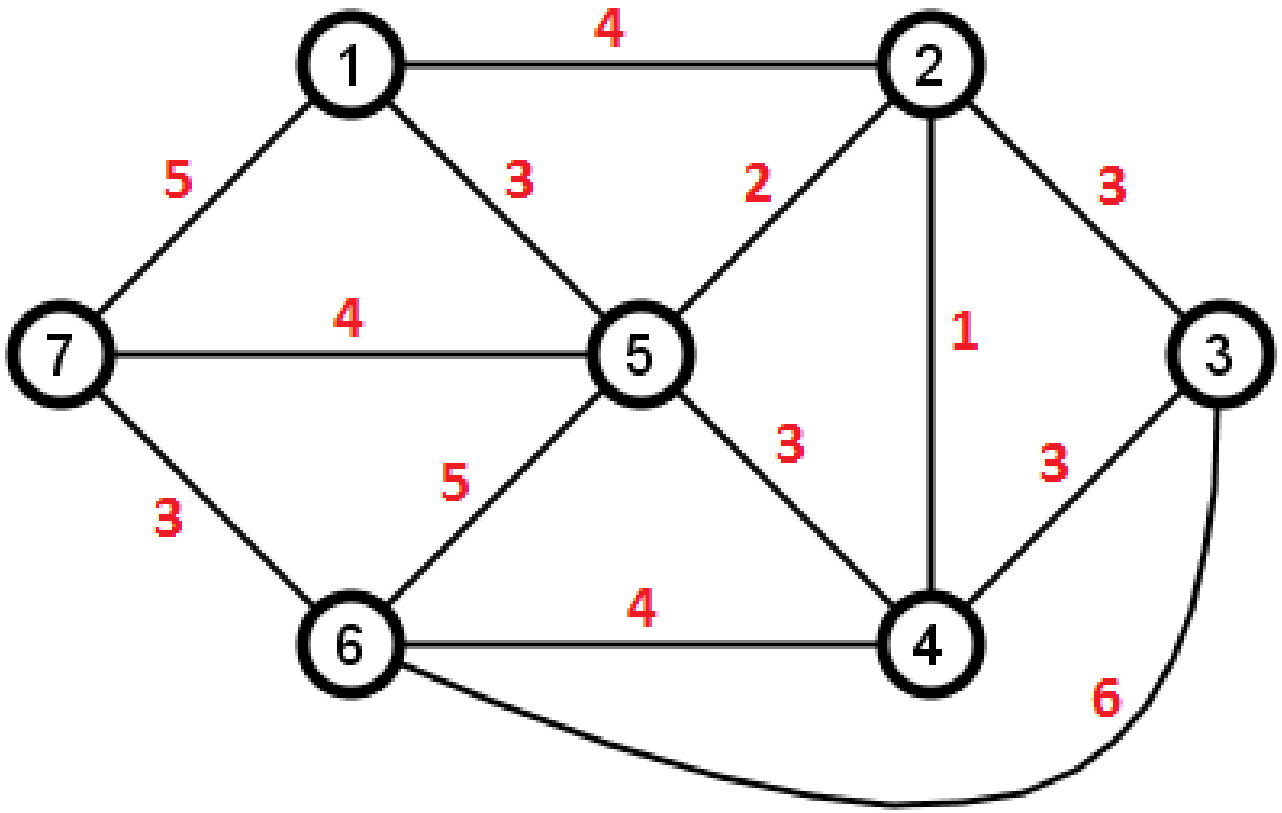
trier L_U dans l'ordre des poids croissants
$F = VIDE$ et $K = 0$

/ L_U *est l'ensemble des arêtes* */*
/ F *est l'ensemble des arêtes de l'arbre* */*

Itérations

Tant Que $k < m$ et $|F| < n - 1$ **faire**
  $k = k + 1$
  Si l_k ne forme pas de cycle avec F **alors**
    $F = F \cup \{l_k\}$

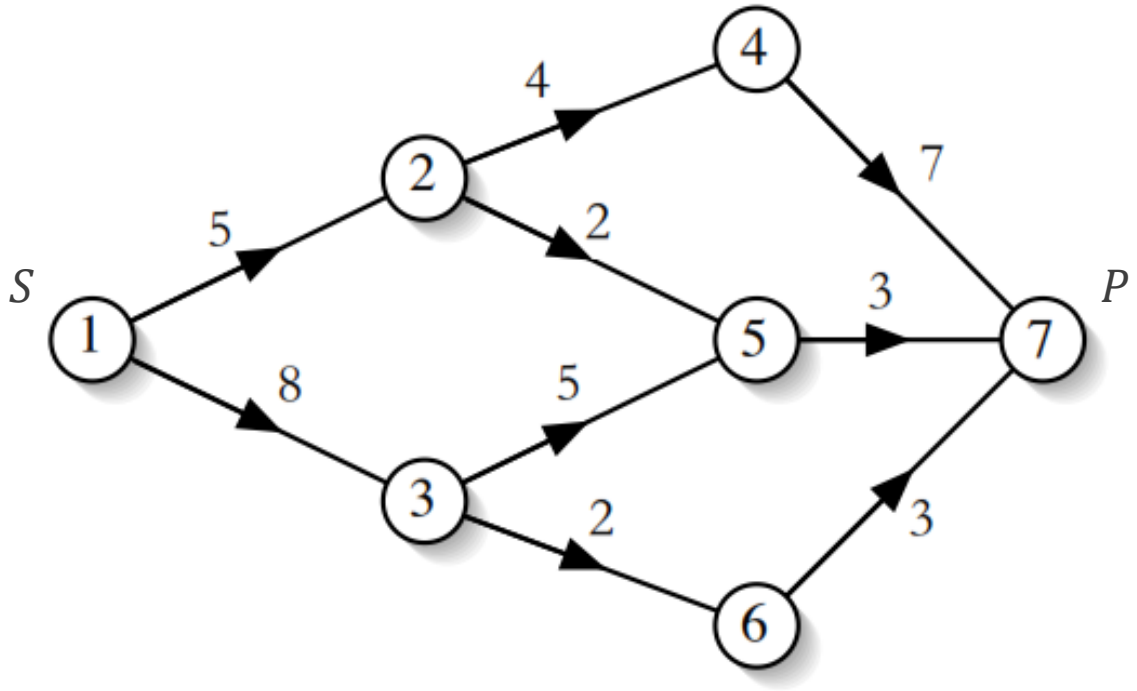
Problèmes: Le problème de l'arbre couvrant minimum



Problèmes: Le problème de flot maximum

- Un réseau de transport est un graphe, sans boucle, où chaque arc est associé à un nombre $c(u) \geq 0$, appelé "capacité" de l'arc u. En outre, un tel réseau vérifie les hypothèses suivantes
  - Il existe un seul nœud S qui n'a pas de prédécesseurs, tous les autres en ont au moins un. Ce nœud est appelé l'entrée du réseau, ou la source
  - Il existe également un seul nœud P qui n'a pas de successeurs, tous les autres en ont au moins un. Ce nœud est appelé la sortie du réseau, ou le puits

Problèmes: Le problème de flot maximum



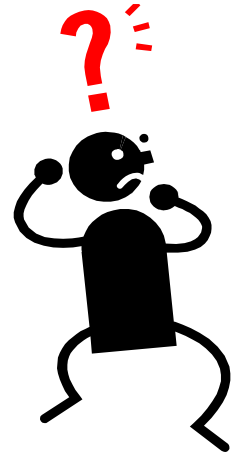
Problèmes: Le problème de flot maximum

- **Définition**: Un flot dans un graphe $G = (X, U)$ est un vecteur $\varphi = [\varphi_1, \varphi_2, \dots, \varphi_m]$

Problèmes: Le problème de flot maximum

- Le problème du flot maximal consiste à déterminer un flot dont la valeur en un certain lieu est maximale

- Mais comment sait-on si un flot est maximal?



Problèmes: Le problème de flot maximum

- Pour un flot φ dans un réseau de transport G, on dit qu'un arc u est saturé si l'on a $\varphi(u) = c(u)$

- Un flot est dit complet si tout chemin allant de S à P contient au moins un arc saturé

Problèmes: Le problème de flot maximum

- Soit un réseau de transport $G = (X, U, C)$ possédant un flot complet φ. On appelle graphe d'écart ou réseau résiduel, le réseau $G'(\varphi) = (X, U', C')$ tel que:
  - Si $u \in U$ et $\varphi(u) < C(u)$ alors $u \in U'$ et $C'(u) = C(u) - \varphi(u)$
  - Si $u \in U$ et $\varphi(u) = C(u)$ alors $u \notin U'$
  - Si $u = (x, y) \in U$ et $\varphi(u) > 0$ alors $u^{-1} = (y, x) \in U'$ et $C'(u^{-1}) = \varphi(u)$

Problèmes: Le problème de flot maximum

- Le réseau résiduel indique le long de quels arcs on peut augmenter ou diminuer le flot.

- **Théorème:** Soit φ un flot de G (de S à P) et $G'(\varphi)$ le réseau résiduel associé à φ. Une condition nécessaire et suffisante pour que le flot φ soit maximal est qu'il n'existe pas de chemin de S à P dans $G'(\varphi)$

Problèmes: Le problème de flot maximum

Algorithme de Ford-Fulkerson:

Initialisations

Partir d'un flot initial φ^0 compatible avec les contraintes de capacité
par exemple $\varphi^0 = (0,0,0,\dots,0)$
$k = 0$

Itérations */* soit φ^k le flot courant */*

Rechercher un chemin μ^k de S à P dans le graphe d'écart $G'(\varphi^k)$
Si il n'en existe pas **alors**
FIN */* le flot φ^k est maximal */*
Soit ε^k la capacité résiduelle du chemin μ^k (minimum des capacités résiduelles)

Problèmes: Le problème de flot maximum

Algorithme de Ford-Fulkerson:

Définir le flot φ^{k+1} par:

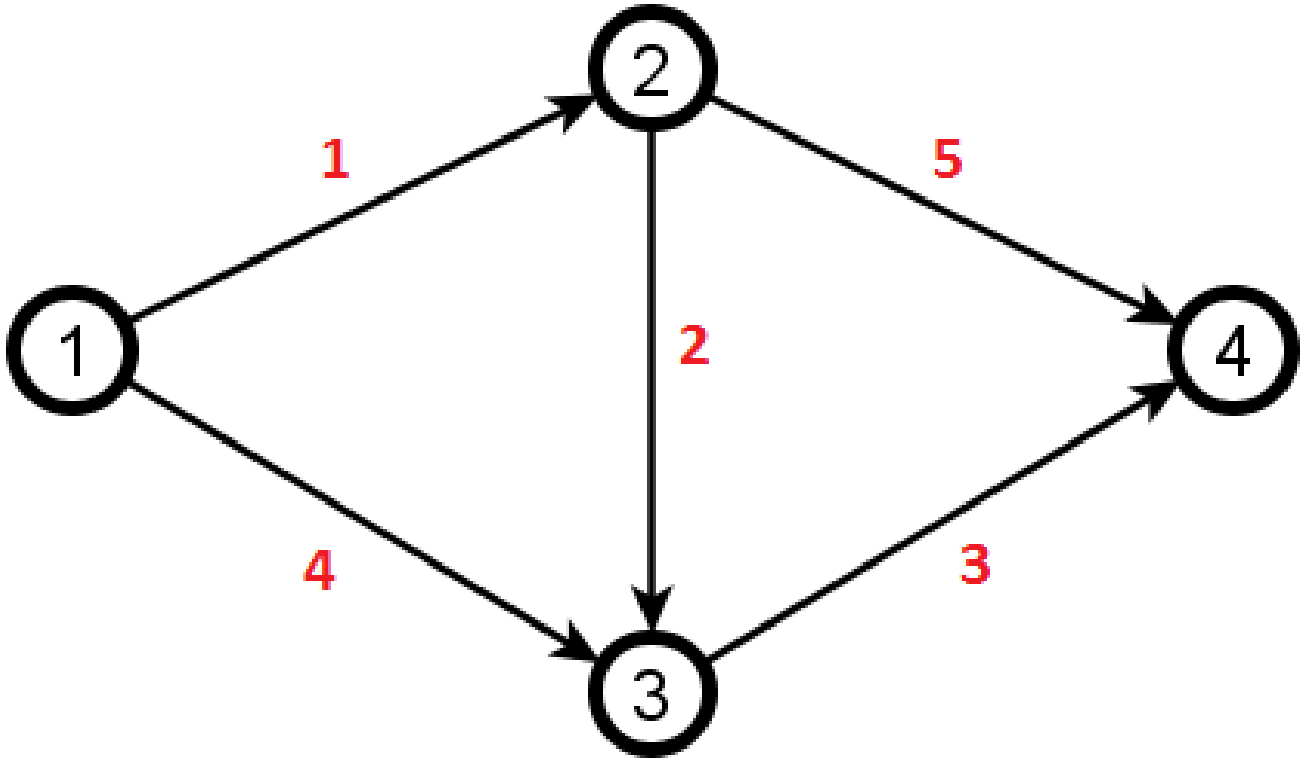
$\varphi_u^{k+1} = \varphi_u^k + \varepsilon^k$ si $u \in \mu^k$ et si u est orientée dans le sens de μ^k

$\varphi_u^{k+1} = \varphi_u^k - \varepsilon^k$ si $u \in \mu^k$ et si u est orientée dans le sens contraire de μ^k

$k = k + 1$

Jump To Début itération

Problèmes: Le problème de flot maximum



Complexité des problèmes, méthodes de résolution

Complexité des problèmes, méthodes de résolution

1. Introduction
2. Classification des problèmes
3. Recherche opérationnelle et optimisation
4. Méthodes de résolution

Introduction

- Malgré la révolution informatique de ces dernières années, certains problèmes d'intérêt pratique demeurent hors de portée de nos ordinateurs, même les plus rapides

- Pour certains problèmes, la difficulté est telle qu'il faut des siècles pour que tous les ordinateurs modernes fonctionnant en parallèles aboutissent à la solution

Introduction

- Ces problèmes sont-ils intrinsèquement difficiles ou y-a-t-il un raccourci pour les résoudre?

Classification des problèmes

La classe P

- Un problème est dit *polynomial* (appartenant à la classe P) s'il peut être résolu par un algorithme déterministe de complexité temporelle polynomiale en la taille du problème

Classification des problèmes

La classe NP

- La classe NP possède une définition moins naturelle que celle de la classe P (nom trompeur) NP signifie *polynomial non déterministe*
- Extension de la classe P en autorisant des choix non déterministes
- Pour ce type de problème il est possible de vérifier une solution efficacement (en temps polynomial)

Classification des problèmes

La classe NP-complet (NP-Complete)

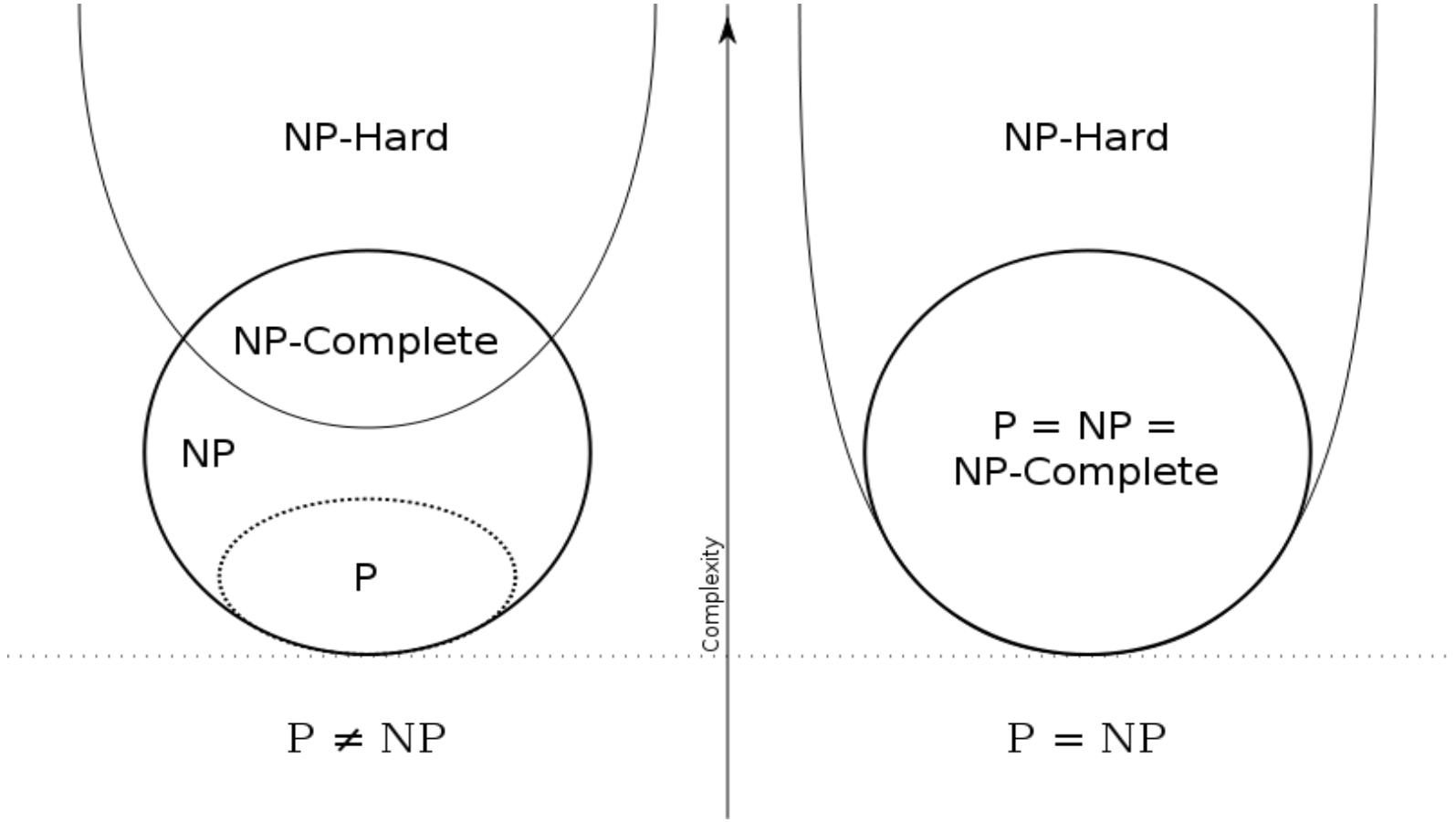
- La théorie de la NP-complétude concerne la reconnaissance des problèmes les plus durs de la classe NP
- Un problème est dit NP-complet s'il vérifie les propriétés suivantes:
  - Il est possible de vérifier une solution en temps polynomial
  - Tout problème Q de NP lui est réductible; cela signifie que le problème est au moins aussi difficile que tous les autres problèmes de la classe NP

Classification des problèmes

La classe NP-difficile (NP-Hard)

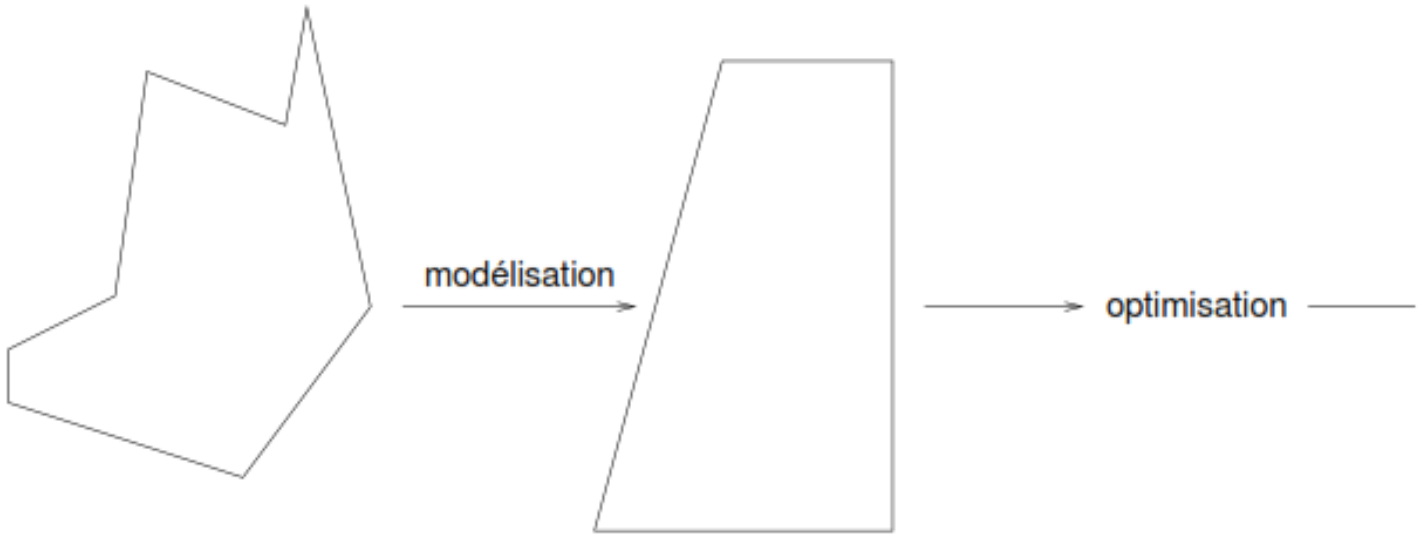
- Cette classe contient les problème les plus dur à résoudre (en terme de complexité temporelle)
- Un problème est dit NP-difficile si tout problème Q de NP lui est réductible
- Si on savait résoudre un quelconque de ces problèmes en temps polynomial, alors on saurait résoudre tous les problèmes NP en temps polynomial

Classification des problèmes



Recherche opérationnelle et optimisation

- Un lien très étroit lie la recherche opérationnelle à l'optimisation



Recherche opérationnelle et optimisation

- Très souvent, on ramène une aide à la décision à la résolution d'un problème d'optimisation
- **Définition:** Soit n un entier strictement positif et soient $D \subset \mathbb{R}^n$ un sous ensemble non vide de $\mathbb{R}^n$ et $f: D \longrightarrow \mathbb{R}$ une application sur D à valeurs réelles. Un problème d'optimisation (PO) consiste à déterminer, lorsqu'il existe, un extremum (minimum ou maximum) de f sur D

Recherche opérationnelle et optimisation

- Classification des PO selon la nature des variables de décision:
  - Optimisation continue: D est continu
  - Optimisation discrète ou optimisation combinatoire: D est discret ($D \subset \mathbb{Z}^n$, fini)
- Classification des PO selon la nature des contraintes:
  - Optimisation sans contraintes: Pas de contraintes ou contraintes faciles à satisfaire
  - Optimisation sous contraintes: il est difficile de trouver un point satisfaisant les contraintes

Méthodes de résolution

- Le problème à résoudre est-il facile ou difficile?
- Si le problème est « facile »: exhiber un algorithme efficace
- Si le problème est « difficile »:
  - Et de « petite taille »: chercher la solution optimale
  - Et de « grande taille »: chercher une solution approchée et essayer de garantir la valeur de cette solution

Méthodes de résolution

Exemples de PO faciles:

- Problème du plus court chemin
- Router un flot maximal entre 2 sommets sous des contraintes de capacités
- Problème de recherche d'une chaîne eulérienne dans un graphe
- Ordonnancer des tâches sous des contraintes de précédence

Méthodes de résolution

Exemples de PO difficiles:

- Problème du voyageur de commerce
- Router un flot maximal entre plusieurs paires de sommets sous des contraintes de capacités
- Problème de recherche d'une chaîne hamiltonienne dans un graphe

Méthodes de résolution

Principales (classes de) méthodes de résolution:

- Algorithmes polynomiaux
- Programmation dynamique
  - Certains problèmes ont de bonnes caractéristiques qui permettent de les résoudre à l'aide d'une formule de récurrence. Les méthodes de programmation dynamique peuvent alors éventuellement permettre de résoudre le problème avec une complexité polynomiale ou pseudo-polynomiale

Méthodes de résolution

Principales (classes de) méthodes de résolution:

- Théorie des graphes
- Processus stochastiques
  - Les processus stochastiques concernent tous les problèmes aléatoires, en particulier des problèmes de fiabilité (de systèmes, de composants électroniques…) et des phénomènes d'attente
- Simulation informatique

Méthodes de résolution

Principales (classes de) méthodes de résolution:

- Programmation linéaire
- Programmation non-linéaire
- Méthodes arborescentes
  - Les méthodes de type « A* » ou « branch and bound » sont couramment utilisées pour trouver la solution exacte d'un problème de RO. Pour une résolution efficace, un soin particulier est apporté au calcul de bornes supérieures ou inférieures pour la valeur de la solution

Méthodes de résolution

Principales (classes de) méthodes de résolution:

- Heuristiques et métaheuristiques
  - Lorsque la solution optimale ne peut être obtenue en un temps raisonnable, on a souvent recours à des méthodes approchées de type heuristique ou métaheuristique (Recuit simulé, Recherche locale, Algorithmes génétique, Colonie de fourmis,…)

Programmation linéaire

Programmation linéaire

Modélisation:

- En Recherche Opérationnelle (RO), modéliser un problème consiste à identifier:
  - Les variables intrinsèques (inconnues)
  - Les différentes contraintes auxquelles sont soumises ces variables
  - L'objectif visé (optimisation)
- Dans un problème de programmation linéaire (**PL**) les contraintes et l'objectif sont des fonctions **linéaires** des variables. On parle aussi de programme linéaire

Programmation linéaire

Modélisation:

- Exemple d'un problème de production: Une usine fabrique 2 produits P1 et P2 nécessitant des ressources d'équipement, de main d'œuvre et de matières premières disponibles en quantité limitée

	P1	P2	Disponibilité
Équipement	3	9	81
Main d'œuvre	4	5	55
Matière première	2	1	20

P1 et P2 rapportent à la vente 6 euros et 4 euros par unité.

Programmation linéaire

Modélisation:

- Quelles quantités (non entières) de produits P1 et P2 doit produire l'usine pour maximiser le bénéfice total venant de la vente des 2 produits???

Programmation linéaire

Modélisation:

- Variables: x1 et x2 sont les quantités des produits P1 et P2 fabriqués ($x_1, x_2 \in \mathbb{R}$)

- Fonction objectif à maximiser: La fonction objectif F correspond au bénéfice total: $F(x_1, x_2) = 6x_1 + 4x_2$. On cherche donc

$$\max_{(x_1,x_2)} [F(x_1, x_2) = 6x_1 + 4x_2]$$

Programmation linéaire

Modélisation:

- Contraintes:
  - Disponibilité de chacune des ressources:

$$3x_1 + 9x_2 \leq 81$$
$$4x_1 + 5x_2 \leq 55$$
$$2x_1 + x_2 \leq 20$$

  - Positivité des variables: $x_1, x_2 \geq 0$.

Programmation linéaire

Modélisation:

- En résumé, le problème de production se modélise sous la forme d'un programme linéaire:

$$\max_{(x_1,x_2)} [F(x_1, x_2) = 6x_1 + 4x_2]$$

$$\begin{cases} 3x_1 + 9x_2 \leq 81 \\ 4x_1 + 5x_2 \leq 55 \\ 2x_1 + x_2 \leq 20 \end{cases}$$

$$x_1, x_2 \geq 0.$$

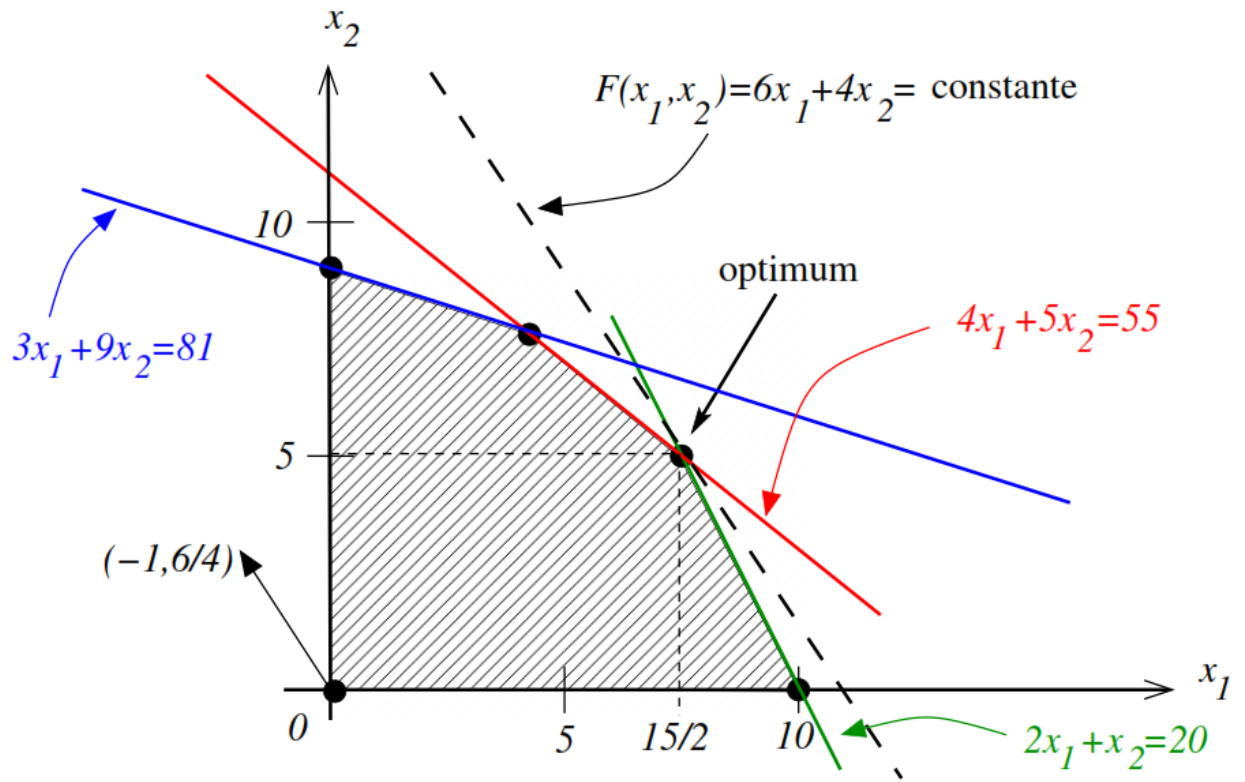
Programmation linéaire

Résolution graphique: (PL à 2 variables)

- Les contraintes où apparaissent des inégalités correspondent géométriquement à des demi-plans.
- Intersection de ces demi-plans = ensemble des variables satisfaisant à toutes les contraintes.
- L'ensemble des contraintes est un polygone convexe.

Programmation linéaire

Résolution graphique: (PL à 2 variables)



Programmation linéaire

Détermination du maximum de F:

- Fonction objectif $F(x_1, x_2) = 6x_1 + 4x_2 \Rightarrow$ droite de coefficient directeur $(-1, 6/4)$.
- Pour déterminer max F, on fait « glisser » la droite (translation parallèle à la direction de la droite) du haut vers le bas jusqu'à rencontrer l'ensemble des variables satisfaisant les contraintes $\Rightarrow$ solution optimale $(x_1, x_2) = (15/2\,, 5)$ avec $\max(F) = 65$.

Programmation linéaire

Détermination du maximum de F:

- **Remarque:** On remarque que le maximum de F est atteint en un sommet du polygone convexe des contraintes.

Programmation linéaire

Forme canonique mixte:

$$\max_{(x_1,\dots,x_n)} \left[F(x_1,\dots,x_n) = c_1x_1 + \cdots + c_nx_n = \sum_{j=1}^{n} c_jx_j \right]$$

$$\begin{cases} contraintes\ inégalités:\ \forall i \in I_1, \sum_{j=1}^{n} a_{ij}x_j = a_{i1}x_1 + \cdots + a_{in}x_n \leq b_i \\ contraintes\ égalités:\ \forall i \in I_2, \sum_{j=1}^{n} a_{ij}x_j = b_i \\ contraintes\ de\ signes: \forall i \in J_1, x_i \geq 0 \\ \forall j \in J_2,\ \ x_j\ de\ signe\ quelconque \end{cases}$$

Programmation linéaire

Forme canonique mixte:

- $I = I_1 \cup I_2$: Ensemble des indices de contraintes $card(I) = m$
- $J = J_1 \cup J_2$: Ensemble des indices des variables $card(J) = n$

Programmation linéaire

Forme canonique mixte:

Notations:

- Vecteurs:

$$\boldsymbol{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$$
$$\mathbf{c} = (c_1, \dots, c_n)^T \in \mathbb{R}^n$$
$$\boldsymbol{b} = (b_1, \dots, b_n)^T \in \mathbb{R}^m$$

- Matrice A de taille $m \times n$:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

Programmation linéaire

Forme canonique pure:

- Sous cette forme, pas de contraintes d'égalité $I_2 = \emptyset \; et J_2 = \emptyset$
- Un programme linéaire (PL) est dit sous forme canonique pure s'il s'écrit:

$$\max_x [F(x) = c \quad x = c_1 x_1 + \cdots + c_n x_n]$$

$$\begin{cases} Ax \leq b \\ x \geq 0 \end{cases}$$

Programmation linéaire

Forme standard:

- Sous cette forme, $I_1 = \emptyset \; et \; J_2 = \emptyset$
- Un programme linéaire (PL) est dit sous forme standard s'il s'écrit:

$$\max_x [F(x) = c^T x]$$
$$\begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

Programmation linéaire

Forme standard:

- On dit de plus que le PL est sous forme standard simpliciale si A de taille $m \times n$ avec $m \leq n$, se décompose en:

$$A = (I_m | H)$$

  - I_m matrice identité de taille $m \times m$
  - H matrice de taille $m \times (n - m)$

Programmation linéaire

Variables d'écarts:

Proposition:

- Tout PL sous forme standard s'écrit de façon équivalente en un PL sous forme canonique pure et inversement.

Démonstration:

i. Soit PL sous forme canonique pure. On a

$$Ax \leq b \quad \Leftrightarrow \quad Ax + e = b, \qquad e \geq 0$$

où $e = (e_1, \dots, e_m)^T$ sont appelées variables d'écart.

Programmation linéaire

Variables d'écarts:

Proposition:

- Tout PL sous forme standard s'écrit de façon équivalente en un PL sous forme canonique pure et inversement.

Démonstration:

$$\text{Ainsi } \begin{cases} Ax \leq b \\ x \geq 0 \end{cases} \Leftrightarrow \begin{cases} \left(A \middle| I_m\right) \begin{pmatrix} x \\ e \end{pmatrix} = b \\ \begin{pmatrix} x \\ e \end{pmatrix} \geq 0 \end{cases} \Leftrightarrow \begin{cases} \tilde{A}\tilde{x} = b \\ \tilde{x} \geq 0 \end{cases}$$

$$\text{avec } \tilde{A} = \left(A \middle| I_m\right) \text{ matrice de taille } m \times (n+m).$$

Programmation linéaire

Variables d'écarts:

Proposition:

- Tout PL sous forme standard s'écrit de façon équivalente en un PL sous forme canonique pure et inversement.

Démonstration:

ii. (Réciproque) Soit PL sous forme standard. On a

$$Ax = b \quad \Leftrightarrow \begin{cases} Ax \leq b \\ Ax \geq b \end{cases} \Leftrightarrow \begin{cases} Ax \leq b \\ -Ax \leq -b \end{cases} \Leftrightarrow \begin{pmatrix} A \\ -A \end{pmatrix} x \leq \begin{pmatrix} b \\ -b \end{pmatrix} \Leftrightarrow \tilde{\tilde{A}}x \leq \tilde{b}$$

où $\tilde{\tilde{A}}$ est une matrice de taille $2m \times n$ et $\tilde{b} \in \mathbb{R}^{2m}$.

Programmation linéaire

Variables d'écarts:

- Exemple: Problème de production de l'introduction.

  PL sous forme standard. On introduit 3 variables d'écarts e_1, e_2, e_3.

$$\max_{(x_1,x_2)} [F(x_1, x_2) = 6x_1 + 4x_2]$$

$$\begin{cases} 3x_1 + 9x_2 + e_1 = 81 \\ 4x_1 + 5x_2 + e_2 = 55 \\ 2x_1 + x_2 + e_3 = 20 \\ x_1, x_2 \geq 0 \\ e_1, e_2, e_3 \geq 0 \end{cases}$$

  Les inconnues sont désormais x_1, x_2, e_1, e_2, e_3.

Programmation linéaire

Solutions de base réalisables:

- PL sous forme standard $(Ax = b)$.
- **Hypothèse de rang plein:** On suppose que la matrice A est de taille $m \times n$ avec $\boxed{rang(A) = m \leq n}$.

- Rappel: $rang(A)$ = nombre maximal de lignes de A linéairement indépendantes (= nombre max. de colonnes linéairement indépendantes).

Programmation linéaire

Solutions de base réalisables:

- Remarques: Sous l'hypothèse de rang plein:
  - Le système $Ax = b$ admet toujours des solutions.
  - Si $m < n$, le système $Ax = b$ admet une infinité de solution.
  - Si $m = n$, la solution est unique et vaut $x = A^{-1}b$, dans ce cas, il n'y a rien à maximiser….

Quelques définitions:

- **Définition:** On appelle solution réalisable tout vecteur x qui satisfait les contraintes du PL i.e tel que $Ax = b$ et $x \geq 0$.

Programmation linéaire

Solutions de base réalisables:

- **Définition:** Soit $B \subset \{1, \dots, n\}$ un ensemble d'indices avec $card(B) = m$ tel que les colonnes $A^i, j \in B$, est inversible. On dit que l'ensemble B des indices est une base.
  - Les variables $x_B = (x_j, j \in B)$ sont appelées variables de base.
  - Les variables $x_H = (x_j, j \notin B)$ sont appelées variables hors-base.
- Remarques:
  - Sous l'hypothèse de rang plein, il existe toujours une base non vide.

Programmation linéaire

Solutions de base réalisables:

- Quitte à renuméroter les indices, on peut toujours écrire les décompositions par blocs:

  $A = (A_B | A_H)$ où A_H est la matrice formée des colonnes A^j, $j \notin B$

  $x = \begin{pmatrix} x_B \\ x_H \end{pmatrix}$.

  Le système $Ax = b$ est équivalent à

  $$A_B x_B + A_H x_H = b.$$

  $\Longrightarrow$ on peut fixer les variables hors-base et les variables de base sont alors complétement déterminées (la matrice A_B est inversible)

Programmation linéaire

Solutions de base réalisables:

- **Définition:** On dit que $x = \begin{pmatrix} x_B \\ x_H \end{pmatrix}$ est une solution de base associée à la base B si $x_H = 0$.
- **Propriétés des solutions de base réalisables:** Si $x = \begin{pmatrix} x_B \\ x_H \end{pmatrix}$ est une solution de base réalisable alors $x_H = 0$ et $x_B = A_B^{-1} b$.
- Remarque: Il y a au plus C_n^m solutions de base (toutes ne sont pas réalisables).

Programmation linéaire

Solutions de base réalisables:

- Exemple: Problème de production de l'introduction.

  PL sous forme standard.

$$\max_{(x_1,x_2)} [F(x_1, x_2) = 6x_1 + 4x_2]$$

$$\begin{cases} 3x_1 + 9x_2 + e_1 = 81 \\ 4x_1 + 5x_2 + e_2 = 55 \\ 2x_1 + x_2 + e_3 = 20 \\ x_1, x_2 \geq 0 \\ e_1, e_2, e_3 \geq 0 \end{cases}$$

Programmation linéaire

Solutions de base réalisable:

On a $m = 3, n = 5, rang(A) = m = 3.$

Une base est donnée par $B = \{3,4,5\}$ avec $A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

La solution de base réalisable correspondante est:

$x = (x_1, x_2, e_1, e_2, e_3)^T = (\underbrace{0,0}_{x_H}, \underbrace{81,55,20}_{x_B = A_B^{-1} b})^T.$

Programmation linéaire

Propriétés géométriques des solutions de base réalisables:

- On note

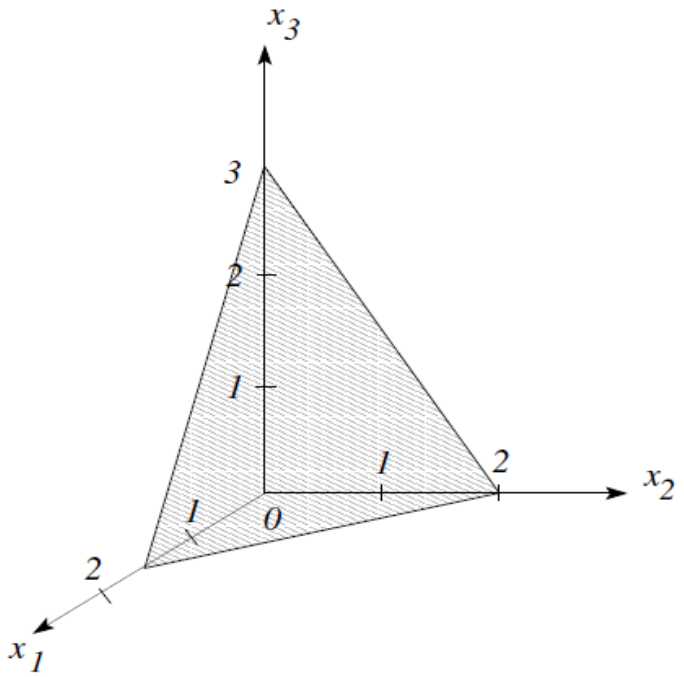
$$D_R = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$$

  L'ensemble des solutions réalisables d'un PL sous forme standard.
- **Définitions (rappels):**
  - Un polyèdre Q de $\mathbb{R}^n$ est défini par $Q = \{x \in \mathbb{R}^n \mid Ax = b\}$ où A est une matrice $m \times n$.
  - Un ensemble E est dit convexe si $\forall x, y \in E, \lambda x + (1 - \lambda)y \in E$ pour tout $0 \leq \lambda \leq 1$.
- **Proposition:** L'ensemble D_R est un polyèdre convexe, fermé.

Programmation linéaire

Propriétés géométriques des solutions de base réalisables:

- Exemple: $D_R = \left\{ x \in \mathbb{R}^3 \mid 2x_1 + \frac{3}{2}x_2 + x_3 = 3,\ x_1, x_2, x_3 \geq 0 \right\}$



Programmation linéaire

Propriétés géométriques des solutions de base réalisables:

Caractérisation de l'optimum:

- **Définitions:** Un point $x \in D_R$ est un sommet (ou point extrême) si et seulement s'il n'existe pas $y, z \in D_R$, $y \neq z$ tels que $x = \lambda y + (1 - \lambda)$z avec $0 < \lambda < 1$.
- **Théorème:**
  - x est une solution de base réalisable si et seulement si x est un sommet de D_R.
  - L'optimum de la fonction objectif F sur D_R, s'il existe, est atteint en au moins un sommet de D_R.

Programmation linéaire

Propriétés géométriques des solutions de base réalisables:

- Tout se passe donc avec les solutions de base: pour résoudre un PL sous forme standard, il suffit de se restreindre aux solutions de base réalisables (les sommets de D_R)
- **3 situations possibles:**
  - $D_R = \emptyset$: le PL n'a pas de solution.
  - $D_R \neq \emptyset$ mais la fonction objectif F n'est pas majorée sur D_R: le maximum de F vaut $+\infty$ (cas exclu si D_R est borné)
  - $D_R \neq \emptyset$ et F est majorée sur D_R: le PL admet une solution optimale (non nécessairement unique)

Programmation linéaire

Propriétés géométriques des solutions de base réalisables:

- Remarques: Au plus C_n^m solutions de base réalisables. Pour déterminer une solution de base, on doit résoudre $A_B x_B = b$. Par une méthode directe de type Gauss/LU requière de l'ordre de $\mathcal{O}(m^3)$ opérations.

  $\Longrightarrow$ Exploration exhaustive de toutes les solutions de base (comparaison des coûts correspondants): $\mathcal{O}(m^3 C_n^m)$ opérations.

  Ce nombre est vite très grand avec n et m. par exemple, avec $n = 20$ et $m = 10$, on a 3×10^8 opérations.

Programmation linéaire

Propriétés géométriques des solutions de base réalisables:

- **Méthode du simplexe:** On explore seulement les sommets qui permettent d'augmenter la fonction objectif $\Longrightarrow$ on réduit le nombre de solution de base à explorer.

Algorithme du simplexe

- On a vu que pour résoudre un PL, il suffit de se restreindre aux solutions de bases réalisables.
- Méthode du simplexe due à Dantzig (1947).

- **Deux phases:**
  - Phase 1 – Initialisation: Trouver une solution de base réalisable (ou bien détecter l'impossibilité).
  - Phase 2 – Progression: On passe d'un sommet à un sommet voisin pour augmenter la fonction objectif.

Algorithme du simplexe

- PL sous forme standard:

$$\max_{x \in \mathbb{R}^n} [F(x) = c^T x]$$
$$\begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

- On dispose d'une base B et d'une solution de base réalisable $\underline{\boldsymbol{x}}$ avec (à une permutation près des colonnes de A)

$$A = (A_B \mid A_H) \quad \text{et} \quad \underline{x} = \begin{pmatrix} \underline{x}_B \\ \underline{x}_H \end{pmatrix}$$

- Où A_B matrice $m \times n$, inversible (variables de base)

  A_H matrice $m \times (n - m)$ (variables hors base)

Algorithme du simplexe

- **But:** On veut trouver une autre base B^* et une solution de base réalisable $\underline{\boldsymbol{x}}^*$ telles que $\underline{\boldsymbol{x}}^*$ est meilleure que $\underline{\boldsymbol{x}}$ c'est-à-dire:

$$F(\underline{\boldsymbol{x}}^*) > F(\underline{\boldsymbol{x}})$$

- **Principe de la méthode du simplexe:** Faire rentrer une variable hors-base dans la nouvelle base (variable entrante) et faire sortir à la place une variable de base (variable sortante).

Algorithme du simplexe

Variable entrante - calcul des coûts réduits:

- Fonction objectif F exprimée en fonction des variables hors-base.
- Ensemble des solutions réalisables $D_R = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$.

- Proposition (Coûts réduits):
  - Pour tout $x \in D_R$, on a

$$F(x) = F(\underline{x}) + L_H^T x_H$$

où $L_H^T = c_H^T - c_B^T A_B^{-1} A_H$ est le vecteur des coûts réduits.

Algorithme du simplexe

Démonstration:

- On a $b = Ax = A_B x_B + A_H x_H$ avec A_B inversible, donc $x_B = A_B^{-1}(b - A_H x_H)$. On obtient donc:

$$F(x) = c^T x = c_B^T x_B + c_H^T x_H \text{ avec } c = \begin{pmatrix} c_B \\ c_H \end{pmatrix}$$
$$= c_B^T A_B^{-1}(b - A_H x_H) + c_H^T x_H$$
$$= c_B^T A_B^{-1} b + (c_H^T - c_B^T A_B^{-1} A_H) x_H$$

Or $\underline{x}_B = A_B^{-1} b$ (car $\underline{x}_H = 0$) et $c_B^T A_B^{-1} b = c^T x = F(\underline{x})$ donc

$$F(x) = F(\underline{x}) + (c_H^T - c_B^T A_B^{-1} A_H) x_H$$

Algorithme du simplexe

Variable entrante:

- Si les coûts réduits sont tous négatifs i.e. $L_H \leq 0$, il n'est alors pas possible d'augmenter la fonction objectif F: l'algorithme se termine normalement c'est-à-dire qu'on a trouvé une solution de base réalisable $\underline{x}$ optimale.
- Dans ce cas contraire (i.e. $\exists (L_H)_i \geq 0$), on a intérêt à faire entrer dans la base, la variable hors-base qui a le coût réduit positif le plus grand possible.
- On note $e \notin B$ l'indice de la variable entrante. On choisit e tel que:

$$(L_H)_e = \max_j \{(L_H)_j, (L_H)_j > 0\}$$

ce qu'on note par $\boxed{e = argmax_j \{(L_H)_j, (L_H)_j > 0\}}$

Algorithme du simplexe

Remarque:

- Si on traite d'un problème de minimisation c'est-à-dire avec

$$\min F(x)$$

alors la variable entrante x_e est déterminée par l'indice

$$e = argmin_j\{(L_H)_j, (L_H)_j < 0\}$$

Algorithme du simplexe

Variable sortante:

- Une fois l'indice e choisi, il faut déterminer la variable qui doit quitter la base. En maintenant la relation $Ax = b$ avec $x \geq 0$, on augmente la variable entrante x_e jusqu'à annuler une des variables de base. Cette variable sera alors la variable sortante.

$Ax = b \Leftrightarrow A_B x_B + A^e x_e = b$ où A^e désigne la e-i^{ème} colonne de A

$\Leftrightarrow x_B = A_B^{-1}(b - A^e x_e)$

$\Leftrightarrow x_B = \underline{x}_B - A_B^{-1} A^e x_e$

$\Leftrightarrow x_B = \underline{x}_B - z x_e$ avec $\boxed{z = A_B^{-1} A^e \in \mathbb{R}^m}$

Algorithme du simplexe

- On doit avoir $x_B = \underline{x}_B - zx_e \geq 0$
  - Si $z \leq 0$, on peut augmenter x_e autant qu'on veut, on aura toujours la positivité de la variable de base x_B. La fonction objectif n'est pas majorée sur $D_R(\max F = +\infty) \implies$ arrêt de l'algorithme.
  - Sinon (i.e. il existe $z_i > 0$), pour avoir la positivité $(\underline{x}_B)_i - z_i x_e \geq 0$ pour tout i, on choisit la variable sortante x_s pour laquelle le rapport $(\underline{x}_B)_i / z_i$ pour $i = 1, \dots, m$ avec $z_i > 0$, est **le plus petit possible**:

Algorithme du simplexe

Variable sortante (indice):

$$s = \underset{i}{\operatorname{argmin}} \left\{ \frac{(\underline{x}_B)_i}{z_i} \, , z_i > 0 \right\}$$

- On a, dans ce cas, $x_s = 0$ et $x_B \geq 0$

Remarque:

- La valeur de la variable entrante est donnée par

$$x_e = \min_i \left\{ \frac{(\underline{x}_B)_i}{z_i} \, , z_i > 0 \right\}$$

Algorithme du simplexe

Méthode du simplexe – Phase 2 – Progression:

1) - Calcul des variables de base réalisables:

Etant donné $A = (A_B|A_H)$, on calcule $\underline{x_B} = A_B^{-1}b \geq 0$.

- Calcul des coût réduits:

$L_H^T = c_H^T - c_B^T A_B^{-1} A_H \qquad \left(F(x) = F(\underline{x}) + L_H^T x_H\right)$

- **Si** $L_H \leq 0$ **alors** $\underline{x_B}$ est une solution optimale (→arrêt de l'algorithme).

2) variable entrante: $e = argmax_j\{(L_H)_j, (L_H)_j > 0\}$

3) variable sortante: - Calcul de $z = A_B^{-1}A^e$ puis

$$- s = \underset{i}{\operatorname{argmin}}\left\{\frac{(\underline{x_B})_i}{z_i}\ , z_i > 0\right\}$$

4) On obtient une nouvelle base $\tilde{B}$ et une nouvelle matrice $A_{\tilde{B}}$ dans laquelle la colonne A^e remplace la colonne A^s. Calcul de $A_{\tilde{B}}^{-1}$ et retour en 1.

Algorithme du simplexe

- PL sous forme standard: $\max_{x \in \mathbb{R}^n} [F(x) = c^T x]$

$$\begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

- Principe: On exprime les variables de base x_B ainsi que F en fonction des variables hors-base x_H. On obtient un système linéaire qu'on appelle **dictionnaire**.
- Exemple du problème de production:

$$\max_{(x_1, x_2)} [F(x_1, x_2) = 6x_1 + 4x_2]$$

$$\begin{cases} 3x_1 + 9x_2 + e_1 = 81 \\ 4x_1 + 5x_2 + e_2 = 55 \\ 2x_1 + x_2 + e_3 = 20 \\ x_1, x_2 \geq 0, e_1, e_2, e_3 \geq 0 \end{cases}$$

Algorithme du simplexe

Etape 1:

- Solution de base réalisable initiale: $x_1 = 0,\ x_2 = 0,\ e_1 = 81,\ e_2 = 55,\ e_3 = 20$ avec $F = 0$.
- Dictionnaire: On exprime les variables de base e_1, e_2, e_3 en fonction des variables hors-base x_1, x_2.

$$\begin{aligned} e_1 &= 81 - 3x_1 - 9x_2 \\ e_2 &= 55 - 4x_1 - 5x_2 \\ e_3 &= 20 - 2x_1 - x_2 \\ \hline F &= 6x_1 + 4x_2 \end{aligned}$$

Algorithme du simplexe

- Variable entrante x_e: $\max_{>0}\{6,4\} = 6 \Rightarrow \boxed{x_e = x_1}$.
- Variable sortante x_s: On maintient $e_1 \geq 0, e_2 \geq 0, e_3 \geq 0$

$$\min_{>0}\left\{\frac{81}{3}, \frac{55}{4}, \frac{20}{2}\right\} = 10 \Rightarrow \boxed{x_s = e_3}$$

- Nouvelle solution de base réalisable: $x_1 = 10,\ x_2 = 0,\ e_1 = 51,\ e_2 = 15,\ e_3 = 0$ avec $F = 60$

Algorithme du simplexe

Etape 2:

- Dictionnaire: On exprime la nouvelle variable de base x_1 en fonction de x_2 et e_3 (nouvelle variable hors-base). On utilise la 3ème équation du dictionnaire de l'étape 1 et on substitue x_1 dans les autres relations.

$$
\begin{aligned}
x_1 &= 10 - \tfrac{1}{2}x_2 - \tfrac{1}{2}e_3 \\
e_1 &= 81 - 3(10 - \tfrac{1}{2}x_2 - \tfrac{1}{2}e_3) - 9x_2 \\
e_2 &= 55 - 4(10 - \tfrac{1}{2}x_2 - \tfrac{1}{2}e_3) - 5x_2 \\
\hline
F &= 6(10 - \tfrac{1}{2}x_2 - \tfrac{1}{2}e_3) + 4x_2
\end{aligned}
$$

Algorithme du simplexe

- On obtient ainsi le dictionnaire (étape 2)

$$
\begin{aligned}
x_1 &= 10 - \tfrac{1}{2}x_2 - \tfrac{1}{2}e_3 \\
e_1 &= 51 - \tfrac{15}{2}x_2 + \tfrac{3}{2}e_3 \\
e_2 &= 15 - 3x_2 + 2e_3 \\
\hline
F &= 60 + x_2 - 3e_3
\end{aligned}
$$

Algorithme du simplexe

- Variable entrante x_e: $\max_{>0}\{1, -3\} = 1 \Rightarrow \boxed{x_e = x_2}$.
- Variable sortante x_s: On maintient $x_1 \geq 0, e_1 \geq 0, e_2 \geq 0$

$$\Rightarrow x_2 = \min_{>0}\left\{\frac{10}{1/2}, \frac{51}{15/2}, \frac{15}{3}\right\} = 5 \Rightarrow \boxed{x_s = e_2}$$

- Nouvelle solution de base réalisable (étape 2): $x_1 = \frac{15}{2}$, $x_2 = 5$, $e_1 = \frac{27}{2}$, $e_2 = 0$, $e_3 = 0$ avec $F = 65$.

Algorithme du simplexe

Etape 3:

- Dictionnaire: On exprime la nouvelle variable de base x_2 en fonction des variables hors-base e_2 et e_3. On utilise la 3ème équation du dictionnaire de l'étape 2 et on substitue x_2 dans les autres relations.

$$\begin{aligned} x_2 &= 5 - \tfrac{1}{3}e_2 + \tfrac{2}{3}e_3 \\ x_1 &= \tfrac{15}{2} + \tfrac{1}{6}e_2 - \tfrac{5}{6}e_3 \\ e_1 &= \tfrac{27}{2} + \tfrac{5}{2}e_2 - \tfrac{7}{2}e_3 \end{aligned}$$

$$F = 65 - \tfrac{1}{3}e_2 - \tfrac{7}{3}e_3$$

Algorithme du simplexe

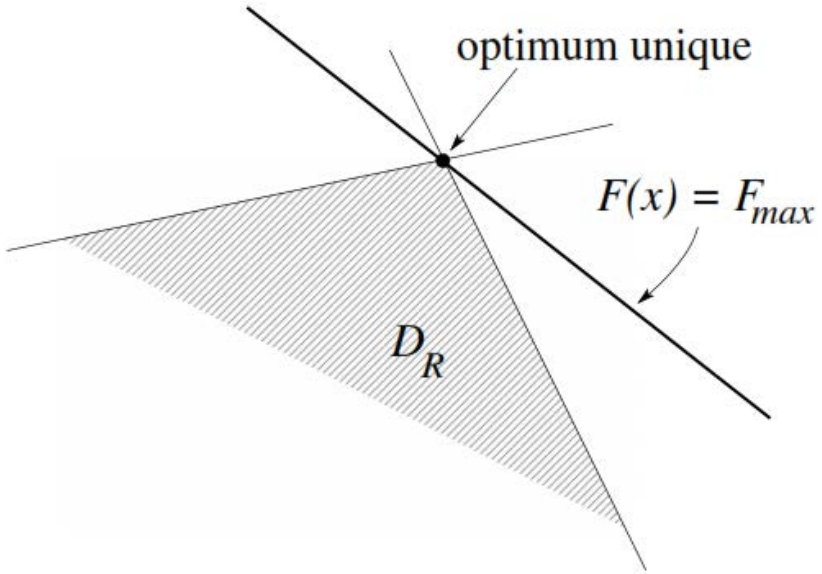
- Tous les coûts réduits sont ≤ 0 donc on ne peut plus augmenter F: l'optimum est atteint et la solution optimale est

$$x_1^* = \frac{15}{2},\ x_2^* = 5,\ e_1^* = \frac{27}{2},\ e_2^* = 0,\ e_3^* = 0 \text{ avec } F = 65$$

Algorithme du simplexe

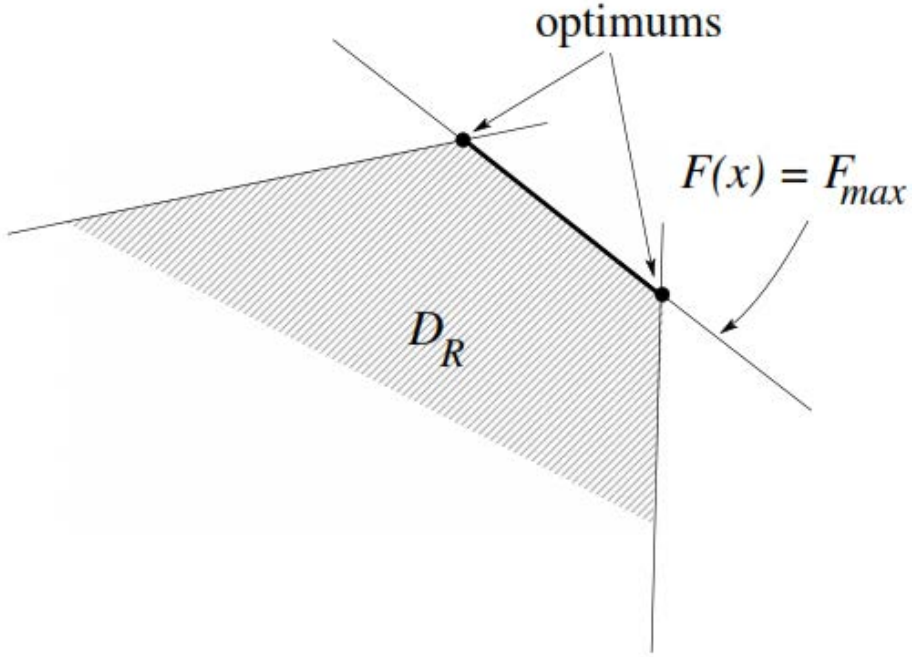
- À chaque étape de l'algorithme du simplexe (en phase 2), il y a des cas remarquables qui conduisent tous à l'arrêt de l'algorithme.

1. Si les coûts réduits $L_H < 0$, alors la solution de base réalisable courante est l'**unique optimum**.



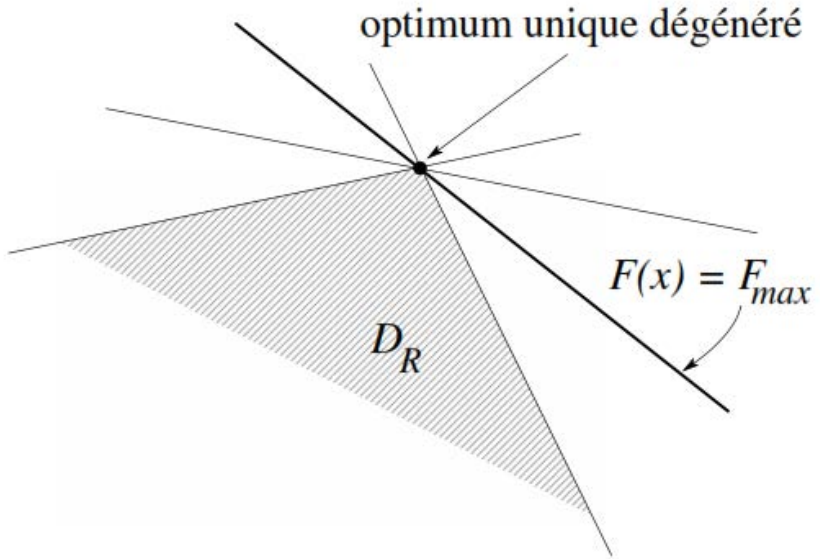
Algorithme du simplexe

2. Si les coûts réduits $L_H \leq 0$, alors il y a deux cas remarquables:
   i. Si $(L_H)_e = 0$ et $x_e > 0$, alors l'**optimum n'est pas unique**.



Algorithme du simplexe

ii. Si $(L_H)_e = 0$ et $x_e = 0$, alors l'**optimum est unique** (à priori). Dans ce cas, la base est dite dégénérée c'est-à-dire qu'il existe une variable de base nulle.



3. Si $(L_H)_e > 0$ et x_e est non borné alors la fonction F **n'est pas majorée**.

Algorithme du simplexe

- Une solution de base réalisable est dite **dégénérée** si au moins une des variables de base est nulle.
- Théorème: Si au cours de l'algorithme du simplexe, aucune base rencontrée n'est dégénérée, alors l'algorithme se termine en un nombre fini d'itérations.
- Par conséquent, on ne rencontre jamais une base déjà rencontrée à une itération précédente. Le nombre de solution de base réalisable étant fini ($\leq C_n^m$), l'algorithme s'arrête nécessairement en un nombre fini d'itérations.

Algorithme du simplexe

- Remarque: S'il existe une base dégénérée, alors on peut rencontrer un éventuel cyclage de l'algorithme: on retrouve une base déjà rencontrée et on boucle indéfiniment. Pour traiter les cas de dégénérescence, on peut appliquer la **règle de Bland** (1977) qui assure l'arrêt de l'algorithme en un nombre fini d'itérations.

- Règle de Bland: Lorsque plusieurs variables sont susceptibles d'entrer ou de sortir de la base, on choisit toujours celle qui a l'indice le plus petit.

Algorithme du simplexe

Introduction:

- PL sous forme canonique pure avec les contraintes: $Ax \leq b,\ \ x \geq 0$
- On peut déterminer facilement une solution de base réalisable dans le cas où $b \geq 0$. En effet, sous forme standard les contraintes deviennent $Ax + e = b$, avec $x, e \geq 0$ où e sont les variables d'écarts.
- Une solution de base réalisable évidente dans ce cas, est $x = 0, e = b \geq 0$.
- Mais pour un PL sous forme standard, il n'y a pas toujours de solution de base réalisable évidente.

Algorithme du simplexe

Variables auxiliaires:

- PL sous forme standard: $\max_{x \in \mathbb{R}^n} [F(x) = c^T x]$

$$\begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

- On ne suppose pas que la matrice $A \in \mathcal{M}_{m \times n}$ est de rang plein, ni qu'il existe bien des solutions réalisables.
- Pour obtenir une solution de base réalisable ou bien pour détecter l'impossibilité, on introduit un problème de programmation linéaire **auxiliaire** pour des variables supplémentaires appelées **variables artificielles**.

Algorithme du simplexe

Programme auxiliaire:

- Le programme auxiliaire associé à (PL) s'écrit:

$$(PLA) \begin{cases} \min\limits_{(x,a)} \sum\limits_{i=1}^{m} a_i \\ Ax + a = b \\ x \geq 0 \\ a \geq 0 \end{cases}$$

où $a = (a_1, \dots, a_m)$ sont appelées **variables artificielles**.

Algorithme du simplexe

- Proposition (évidente): Un (PL) admet une solution réalisable si et seulement si le problème auxiliaire (PLA) admet une solution de base optimale avec $a = 0$.
- Détermination d'une solution de base réalisable via le problème auxiliaire: On applique l'algorithme du simplexe au problème auxiliaire (PLA). À la fin du simplexe, le coût minimal est **nul** sinon on a détecté l'impossibilité pour (PL) (i.e. $D_R = \emptyset$). Si tout s'est déroulé normalement (coût nul), on cherche à éliminer de la base toutes les variables artificielles.
- Deux cas possibles:
  - On a réussi à faire sortir toutes les variables artificielles. On passe à la phase 2.
  - S'il reste des variables artificielles dans la base (base dégénérée) alors les lignes associées à ces variables sont des contraintes redondantes qu'**on élimine**.

Algorithme du simplexe

Phase d'initialisation du simplexe (phase 1):

- On note F_{aux} la valeur de la fonction objectif du problème auxiliaire (PLA) à la fin du simplexe, c'est-à-dire $F_{aux} = \min_{(x,a)} \sum_{i=1}^{m} a_i$.

1. Si $F_{aux} = 0$ et $\nexists a_j \in X_B$ où X_B désigne l'ensemble des variables de base pour (PLA), alors fin normale de la phase 1. On passe à la phase 2 du simplexe.
2. Si $F_{aux} = 0$ et $\exists a_j \in X_B$ avec $a_j = 0$, alors on supprime les lignes et colonnes associées aux a_j et on passe à la phase 2.
3. Si $F_{aux} > 0$ alors pas de solution réalisable $(D_R = \emptyset)$.

Algorithme du simplexe

Complexité du simplexe:

- Complexité = nombre d'itération dans le simplexe (phase 2).
  - On peut construire des exemples avec une complexité exponentielle en $\mathcal{O}(2^n)$ itérations (Klee-Minty, 1972).
  - Mais dans la pratique la complexité du simplexe croît peu avec le nombre n de variables. En pratique, le nombre d'itérations est proportionnel au nombre m de contraintes (de m à $3m$ itérations).
  - Si on tient compte de la résolution des systèmes linéaires avec une formule de mise à jour de l'inverse (Shermann-Morrison), on a $\mathcal{O}(m^2)$ opérations pour l'inverse.