Introduction à la recherche opérationnelle et à l'optimisation combinatoire

Cours RO202

Zacharie ALES
(zacharie.ales@ensta.fr)

Adapté de cours de Marie-Christine Costa, Alain Faye et Sourour Elloumi

Créé le 21/01/2018
Modifié le 21/11/2022 (v6)

ENSTA IP PARIS

Sommaire

Recherche opérationnelle

Définition 1 [Wikipedia]

Ensemble des méthodes et techniques rationnelles orientées vers la recherche du **meilleur choix**

Définition 2

Mettre au point des méthodes, les implémenter au sein d'outils (logiciels) pour trouver des résultats ensuite confrontés à la réalité

Et repris jusqu'à satisfaction du demandeur

Discipline au carrefour entre

- Mathématiques
- Économie
- Informatique

- Par nature en prise directe avec l'industrie

Problème d'optimisation combinatoire

Caractéristiques

- 1 problème $\longrightarrow$ grand nombre de solutions
- 1 solution $\longrightarrow$ 1 valeur   Mais pas infini

Définition - Problème d'optimisation combinatoire

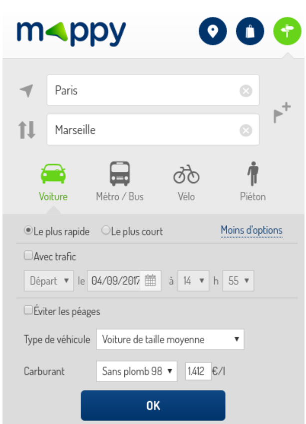
Maximiser ou Minimiser une fonction objectif tout en respectant un ensemble de contraintes

Problème discret

Recherche d'une solution optimale entière
Les variables sont généralement dans $\{0, 1\}$, $\mathbb{N}$ ou $\mathbb{Z}$

Sommaire

Premier exemple : cheminer



Critères

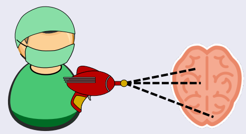
1. ....
2. ....
3. ....

Calcul du chemin minimisant un critère

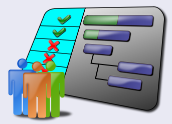
Solution trouvée facilement par un algorithme de graphes

Autres exemples

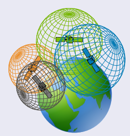
Curiethérapie



Ordonnancement



Positionnement de capteurs



Gestion de ressources

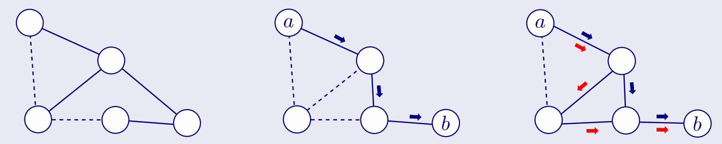
- Gestion des stocks
- Transport et logistique
- Router, relier
- ...

Entreprises très concernées par la RO

Google, Bouygues Telecom, Huawei, SNCF, Artelys, EDF, Air Liquide, Total, LocalSolver, Air France, Microsoft, Eurodecision, Orange, Engie, IBM

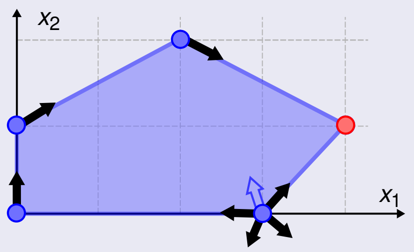
Programme

Optimisation dans les graphes
Chapitre 1

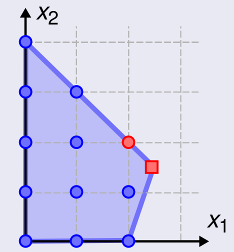


1.1 - Arbre couvrant   1.2 - Chemin   1.3 - Flot

Programmation linéaire (PL)
Chapitre 2



PL en nombres entiers
Chapitre 3



Sommaire

Sommaire

Qu'est-ce qu'un graphe ?

"Des points et des traits ou des flèches"

Point de vue mathématique

Une relation binaire

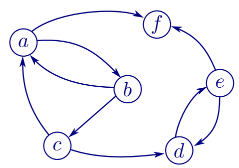
Point de vue pratique

Représentation abstraite d'un réseau

Ex : réseau de télécommunication

Permet de

- Visualiser des échanges
- Modéliser des systèmes réels
- Jouer
  Voir cours Jeux, Graphes et RO (RO203)
- ...



Domaines variés

- Économie
- Informatique
- Industrie
- Chimie
- Sociologie
- ...

Graphes orientés

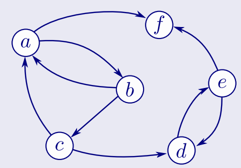
Notation - Graphe orienté

$G = (V, A)$

Ensemble de sommets — Ensemble d'arcs $\subseteq V \times V$

Exemple

- $V = \{a, b, c, d, e, f\}$
- $A = \{(ab), (ba), (bc), (ca), (cd), (af), ...\}$
  Aussi noté : (a, b)



Vocabulaire

Extrémité initiale

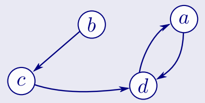
Soit $h = (ab) \in A$

Extrémité finale

- a et b sont adjacents ou voisins
- a est prédécesseur de b
- b est successeur de a

Définition - Graphe simple

Graphe ne possédant pas deux arcs ayant les même extrémités initiales et terminales



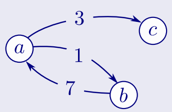
Définition - Multigraphe

Graphe non simple

Définition - Graphe valué

Graphe dont les arcs portent une valuation

Distance, coût, gain, ...



Prédécesseurs et successeurs

Définition - Successeur d'un sommet

$\Gamma(v) = \{\text{successeurs du sommet v}\}$

$V \mapsto P(V)$ (aussi noté Γ^+)

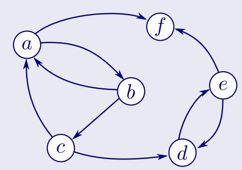
Définition - Prédécesseur d'un sommet

$\Gamma^{-1}(v) = \{\text{prédécesseurs du sommet v}\}$

Aussi noté Γ^-

Exemple

- $\Gamma(b) = \; ......$
- $\Gamma(f) = \; .$
- $\Gamma^{-1}(b) = \; ...$
- $\Gamma^{-1}(d) = \; ......$



Chemin et circuit

Définition - Chemin

Suite d'arcs telle que l'extrémité terminale d'un arc coïncide avec l'extrémité initiale de l'arc suivant

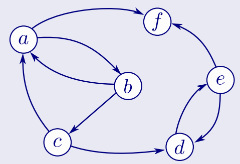
Définition - Circuit

Chemin dont les deux extrémités coïncident

- chemin **simple** : pas deux fois le même arc
- chemin **élémentaire** : pas deux fois le même sommet

Exemple

- Chemin :
  - ....
  - ....
- Circuit :
  - ....
  - ....



Racine, degrés

Définition - Racine

Sommet r tel qu' ....

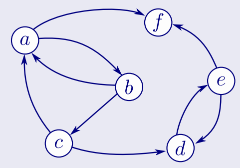
Définition - Degré intérieur (resp. extérieur) d'un sommet x

Nombre d'arcs dont x est l'extrémité terminale (noté $d^-(x)$)

resp. initiale resp. $d^+(x)$

Exemples

- Racine : . .
- $d^-(a) =$., $d^-(f) =$.
- $d^+(a) =$., $d^+(b) =$., $d^+(f) =$.



Graphes non orientés

Définition - Arête

Arc "sans orientation"

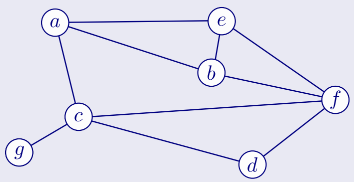
Notation - Graphe non orienté

$$G = (V, E)$$

Ensemble de sommets Ensemble d'arêtes

Exemple

- $V =$ ....
- $E =$ ....

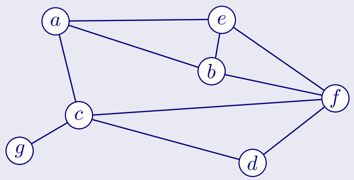


Définition - Chaîne

Séquence d'arêtes $\{e_1, e_2, ..., e_n\}$ telle qu'il existe une séquence de sommets $\{v_1, v_2, ..., v_{n+1}\}$ telle que $e_i = [v_i v_{i+1}]$

Exemple

- Chaîne : ....



Définition - Voisinage

Les sommets x et y sont dits voisins si $[xy] \in E$

Notation - $N(x)$

$N(x) = \{\text{voisins de } x\}$

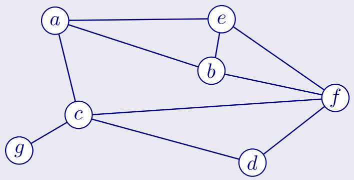
Définition - Degré

$d(x) = |N(x)|$

Nombre d'arêtes adjacentes à x

Exemple

- b est voisin de ......
- $N(c) =$ ....
- $d(b) =$.
- $d(c) =$.



Cycle élémentaire

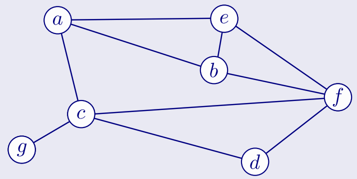
Définition - Cycle (élémentaire)

Chaîne dont les deux extrémités coïncident

(et qui ne passe pas 2 fois par le même sommet)

Exemple

- Cycle : ....



Définition - Cycle Hamiltonien

Cycle élémentaire passant par tous les sommets

Hypothèses pour la suite

- Les graphes sont simples

  Une seule arête ou un seul arc entre deux sommets
- Les cycles sont élémentaires
- Les graphes sont sans boucle

  Pas d'arête ou d'arc (x,x)

Définition - Relation de connexité $\mathcal{R}$

Soit x et y deux sommets d'un graphe $G = (V, A)$

- $x\mathcal{R}y \Leftrightarrow x$ et y sont reliés par une chaîne

Définition - Composante connexe

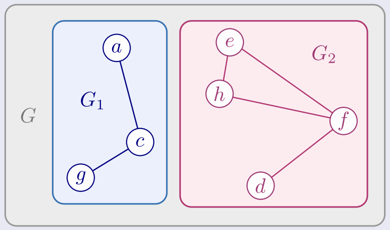
$\mathcal{R}$ est une relation d'équivalence dont les classes d'équivalences sont appelées **composantes connexes**

Définition - Graphe connexe $G = (V, A)$

G ne possède qu'une unique composante connexe

Exemple

- $a\mathcal{R}g$
- G_1 et G_2 ....
- G ....



Arbre

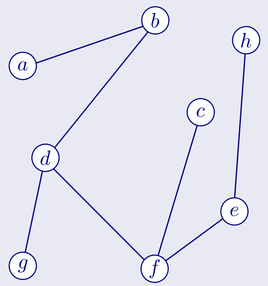
Définition - Arbre

Graphe .... et ....

Définition - Forêt

Graphe ....

Exemple



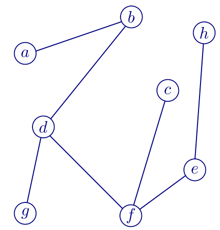
Théorème

Soit $G = (V, E)$ un graphe

$n \geq 2$

Il y a équivalence entre les propriétés suivantes

1. G est connexe sans cycle (*i.e.*, G est un arbre)
2. G est connexe minimal (*i.e.*, retirer une arête rend G non connexe)
3. G ne contient aucun circuit et possède $n - 1$ arêtes
4. G est sans cycle maximal (*i.e.*, ajouter une arête forme un cycle)
5. G est sans cycle et possède $n - 1$ arcs
6. Tous couples de sommets de G est relié par un unique chemin



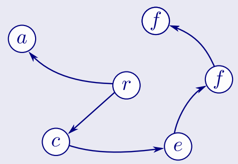
Graphes orientés - Arborescence

Définition - Arborescence

$G = (V, A)$ arbre possédant une racine r telle que

- r est reliée à tout $v \in V$ par un chemin unique

Exemple



Propriété

- $d^-(r) =$.
- $d^-(x) =$. pour tout $v \neq r$

arborescence = "arbre enraciné" = "arbre" en informatique

- arbre généalogique, tournois, arbre des espèces animales,...

Sommaire

Problème

Comment relier des objets en minimisant la longueur totale des liens ?

Donnée - Graphe non orienté valué

Objets à relier — Liens possibles

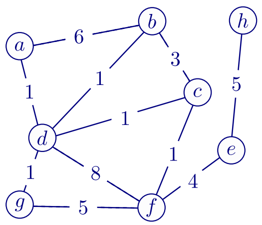
$G = (V, E, p)$

Longueur du lien

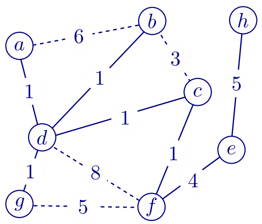
Formulation du problème

Sélectionner des arêtes d'un graphe orienté valué $G = (V, E, p)$ afin de former un arbre :

- couvrant chaque sommet et
- dont la somme des poids des arêtes est minimale



Graphe initial

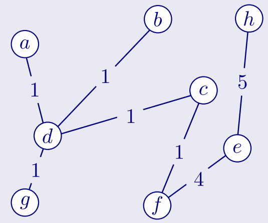


Une solution

Solution optimale - **Arbre couvrant de poids minimal**

- Arbre$\rightarrow$ graphe sans cycle et connexe
- **Couvrant** $\rightarrow$ passant par tous les sommets
- Minimal $\rightarrow$ de longueur totale min

Exemple



Arbre couvrant de poids minimal

Comment obtenir un arbre couvrant de poids minimal ?

Algorithme de Kruskal

Données

poids $p : E \mapsto \mathbb{R}$

- $G = (V, E, p)$: graphe non orienté valué

Résultat

$E_2 \subseteq E$

- $H = (V, E_2)$: arbre couvrant de poids minimal de G

Algorithme de Kruskal

Données : $G = (V, E, p)$

Résultat : Arbre couvrant de poids minimal de G

$k \leftarrow 0$

$E_2 \leftarrow \emptyset$

$L \leftarrow$ Liste des arêtes de E triées par ordre de poids croissant

Nombre de sommets du graphe

pour k allant de 1 à $n - 1$ **faire**

  $w \leftarrow 1^{\text{ère}}$ arête de L ne formant pas de cycle avec E_2

  $E_2 \leftarrow E_2 \cup \{w\}$

retourner $H = (V, E_2)$

Complexité de l'algorithme

Complexité du tri $\mathcal{O}(m \log m)$

Nombre d'arêtes

La détection de cycle peut se faire efficacement en associant un représentant à chaque composante connexe (structure de données Union-find)

Quelques notions de complexité

Complexité $\mathcal{O}(n)$ d'un algorithme $\mathcal{A}$

Dans le pire des cas, $\mathcal{A}$ s'exécute en un nombre d'étapes proportionnel à $n \in \mathbb{N}$

Problème "facile" P (ou problème polynomial)

Polynomial par rapport à la taille des données d'entrée

On connaît un algorithme résolvant P de complexité polynomiale

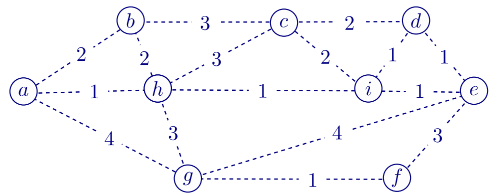
Ex : $\mathcal{O}(log\ n)$, $\mathcal{O}(n^2)$, $\mathcal{O}(n^{10} + 3n^2)$, ...

Problème "difficile" P

On ne connaît aucun algorithme permettant de résoudre P en un nombre polynomial d'étapes

Ex : problème dont les seuls algorithmes connus sont de complexité $\mathcal{O}(e^n)$, $\mathcal{O}(n!)$

Algorithme de Kruskal - Exemple



Arête	ah	de	di	ei	hi	fg	ab	bh	ci	cd	bc	ch	ef	gh	ag	ge
Poids	1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	4
k																

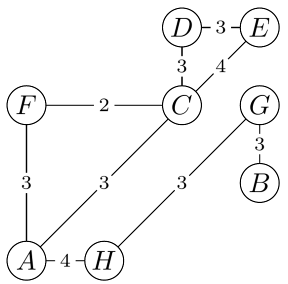
- $n = 9 \rightarrow$ stop après 8 sélections
- $p(H) = (1 + 1 + 1 + 1 + 1 + 2 + 2 + 3) = 12$

Quiz !

Question 1

Voici la liste des arêtes de ce graphe ordonnées par poids croissant : (F, C), (A, C), (A, F), (B, G), (C, D), (D, E), (H, G), (A, H), (C, E)

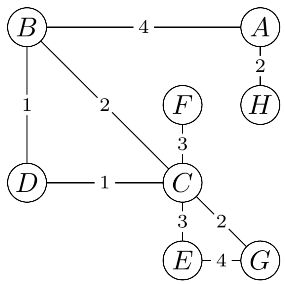
Indiquer les 4 premières arêtes ajoutées à l'arbre lorsqu'on applique l'algorithme de Kruskal.



Question 2

Voici la liste des arêtes de ce graphe ordonnées par poids croissant : (D, B), (D, C), (C, B), (G, C), (H, A), (C, F), (E, C), (B, A), (E, G)

Indiquer les 4 premières arêtes ajoutées à l'arbre lorsqu'on applique l'algorithme de Kruskal.



Preuve d'optimalité - Algorithme de Kruskal

Notations

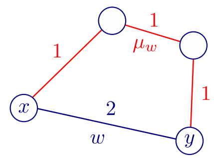
- $G = (V, E, p)$: graphe initial
- $H = (V, E_2)$: arbre couvrant obtenu par l'algorithme de Kruskal

Propriété 1

Soient

- $w = [xy] \in E \backslash E_2$
- μ_w : chaîne de x à y dans H

alors, $p(w) \geq max_{u \in \mu_w} p(u)$



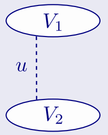
Preuve d'optimalite - Algorithme de Kruskal

Notations

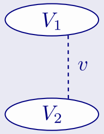
- H : arbre obtenu par l'algorithme de Kruskal de poids $p(H)$
- $H^{(1)}$: arbre optimal de poids $p(H^{(1)})$
- $u \in H^{(1)} \backslash H$ reliant V_1 et V_2
  Avec $V_1 \cup V_2 = V$
- $v \in H \backslash H^{(1)}$ reliant V_1 et V_2

Montrons que
$p(H) = p(H^{(1)})$

Optimal - $H^{(1)}$



Algorithme - H



- .... (propriété 1)
- .... ($H^{(1)}$ est optimal)

Soit $H^{(2)} = H^{(1)} \cup \{v\} \backslash \{u\}$
On a donc ....

On répète le processus...

Considérons $w \in H^{(2)} \backslash H$
On a donc $p(H^{(3)}) = p(H^{(1)})$

On répète jusqu'à ce que $H^{(\dots)} = H$

Algorithme de Kruskal

Remarque

L'algorithme de Kruskal est un **algorithme glouton**

Définition - Algorithme glouton

A chaque étape, faire le choix le plus intéressant à cet instant et ne plus le remettre en question

Caractéristiques des algorithmes gloutons

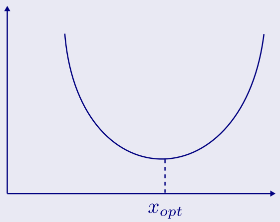
- Facile
- Rapide — Algorithme dit **heuristique**
- Rarement optimale
  L'abre couvrant de poids minimal est une exception

Algorithmes gloutons

Choix glouton = Choix **localement** optimal

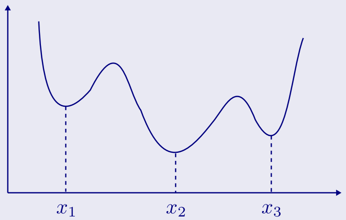
Optimum local $\neq$ Optimum global

Fonction concave ou convexe



Optimum local = Optimum global
($\leq$ optimum global entier)

Fonction quelconque



- x_1, x_2, x_3 : optimums locaux
- x_2 : optimum global

Arbre couvrant de poids maximal

Maximisation

Même algorithme en triant les arêtes par ordre de poids décroissant

Difficulté de l'implémentation

Détection des cycles
— Fonction fournie dans le TP

Sommaire

Le voyageur de commerce

Problème du voyageur de commerce

Comment passer une fois par chaque ville tout en minimisant la longueur totale parcourue ?

Graphe valué associé

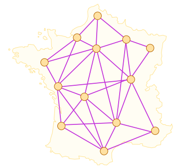
Villes — Routes possibles

$$G = (V, E, p)$$

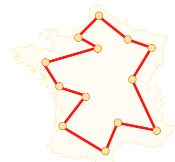
Longueur des routes

On cherche un cycle **hamiltonien** de valeur minimale

Passant par tous les sommets



Graphe initial



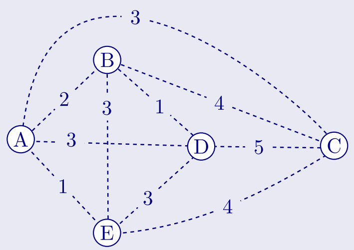
Solution

Source : Wikipedia

Algorithme glouton pour le voyageur de commerce

Données : $G = (V, E, p)$: graphe initial
Résultat : $H(V, E_2)$: cycle hamiltonien
$k \leftarrow 0$
$E_2 \leftarrow \emptyset$
$L \leftarrow$ Liste des arêtes de G triées par ordre de longueur croissante
pour k allant de 1 à n **faire**
  $w \leftarrow$ 1ère arête de L ne formant pas de sous-cycle* avec E_2 et telle que les degrés des sommets restent ≤ 2
  $E_2 \leftarrow E_2 \cup \{w\}$
retourner $H = (V, E_2)$

Exemple



$L = (AE, BD, AB, BE, AD, DE, AC, BC, CE, CD)$

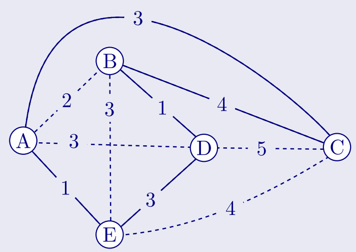
Solution heuristique de valeur ..

* : cycle ne contenant pas l'ensemble des sommets du graphe

Algorithme glouton pour le voyageur de commerce

L'algorithme ne donne pas la solution optimale

- solution gloutonne : longueur 13
- solution optimale : longueur 12



Le problème du voyageur de commerce est un problème « difficile »

Sommaire

Problèmes de cheminement

Problème 1

Trouver un chemin d'un sommet à un autre de longueur minimale

Problème 2

Trouver les plus courts chemins d'un sommet à tous les autres

Problème 3

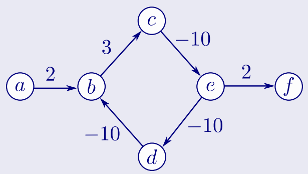
Trouver un plus court chemin pour toutes paires de sommets

Applications du routage

- Réseaux de télécommunications
- GPS routier
- Distribution d'eau, de gaz
- ...

Définition - Circuit absorbant

Circuit . . . .



Théorème

- Il existe un chemin de longueur minimale finie de r à tous les sommets du graphe

si et seulement si

- r est une racine du graphe et le graphe ne contient pas de circuit absorbant

Cas où l'on est sûr de l'absence de circuit absorbant

- Toutes les longueurs sont positives ou nulles
- Le graphe est sans circuit

Sommaire

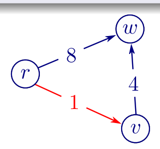
Algorithme de Dijkstra
Problème 1 et 2
Cas des valuations positives

Principe de l'algorithme

Construire une arborescence $H(V, A_2)$
- dont r est la racine et
- correspondant au plus court chemin entre r et les autres sommets

Idée de l'algorithme

- Le plus court chemin entre r et son sommet le plus proche v est $p(r, v)$
- Même raisonnement pour le sommet le plus proche de r ou v
- On répète cette idée jusqu'à ce que
  - Problème 1 : le sommet cible soit atteint
  - Problème 2 : tous les sommets soient atteints



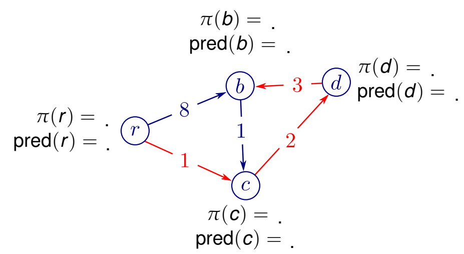
Algorithme de Dijkstra
Problème 1 et 2
Cas des valuations positives

Notation

Prédécesseur de x sur le meilleur chemin connu de r à x

Soient les applications $\text{pred}(x)$ et $\pi(x)$

Longueur du meilleur chemin connu entre r et x



Algorithme de Dijkstra
Problème 1 et 2
Cas des valuations positives

Données :
$G = (V, A, p)$: graphe de poids positifs
$r \in V$: sommet origine
Résultat : $H = (V, A_2)$ arborescence des plus courts chemins

Sommets déjà considérés comme pivot
Origine des arcs
Arêtes de l'arborescence

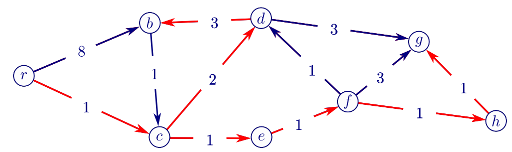
$(\text{pivot}, V_2, \pi(r), A_2) \leftarrow (r, r, 0, \emptyset)$
pour $v \in V \backslash \{r\}$ **faire** ⟵ Initialisation de π (aucun sommet de $V \backslash \{r\}$ n'est pour l'instant atteint)
  $\pi(v) \leftarrow +\infty$

pour j allant de 1 à $n-1$ **faire** ⟵ Pour tout pivot
  pour $y \in V \backslash V_2$ tel que $(\text{pivot}, y) \in A$ **faire**
    si $\pi(\text{pivot}) + p(\text{pivot}, y) < \pi(y)$ **alors** ⟵ Si utiliser $(\textit{pivot}, y)$ fournit un meilleur chemin vers y
      $\pi(y) \leftarrow \pi(\text{pivot}) + p(\text{pivot}, y)$
      $\text{pred}(y) \leftarrow \text{pivot}$
  $\text{pivot} \leftarrow \textit{argmin}_{z \notin V_2}\, \pi(z)$
  $V_2 \leftarrow V_2 \cup \{\text{pivot}\}$

pour tout $x \in V \backslash \{r\}$ **faire** ⟵ Construire A_2 à partir de pred
  $A_2 \leftarrow A_2 \cup \{(\text{pred}(x), x)\}$

retourner $H(V, A_2)$

Algorithme de Dijkstra - Exemple



Données : $G = (V, A, p)$: graphe de poids positifs
Résultat : $H = (V, A_2)$ arborescence des plus courts chemins
$(V_2, \text{pivot}, \pi(r), A_2) \leftarrow (r, r, 0, \emptyset)$
pour $v \in V \backslash \{r\}$ **faire**
  $\pi(v) \leftarrow +\infty$

pour j allant de 1 à $n-1$ **faire**
  pour tout sommet $y \notin V_2$ tel que $y \in \Gamma^+(\text{pivot})$ **faire**
    si $\pi(\text{pivot}) + p(\text{pivot}, y) < \pi(y)$ **alors**
      $\pi(y) \leftarrow \pi(\text{pivot}) + p(\text{pivot}, y)$
      $\text{pred}(y) \leftarrow \text{pivot}$
  $\text{pivot} \leftarrow \textit{argmin}_{z \notin V_2}\, \pi(z)$
  $V_2 \leftarrow V_2 \cup \{\text{pivot}\}$

pour tout $x \in V \backslash \{r\}$ **faire**
  $A_2 \leftarrow A_2 \cup \{(\text{pred}(x), x)\}$

retourner $H(V, A_2)$

Valeurs de π

		π						
j	**pivot**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
-	-	∞	∞	∞	∞	∞	∞	∞
1	r	8	1	|	|	|	|	|
2	c	|	|	3	2	|	|	|
3	e	|	|	|	|	3	|	|
4	d	6	|	|	|	|	6	|
5	f	|	|	|	|	|	|	4
6	h	|	|	|	|	|	5	|
7	g	|	|	|	|	|	|	|

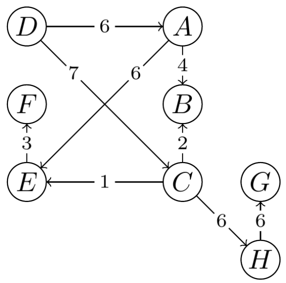
Valeurs de pred

		pred						
j	**pivot**	**b**	**c**	**d**	**e**	**f**	**g**	**h**
1	r	r	r					
2	c	|	|	c	c			
3	e	|	|	|	|	e		
4	d	d	|	|	|	|	d	
5	f	|	|	|	|	|	|	f
6	h	|	|	|	|	|	h	|
7	g	|	|	|	|	|	|	|

Quiz !

Question 3

Déterminer la plus courte distance pour atteindre chaque sommet à partir du sommet D en utilisant l'algorithme de Dijkstra.



Algorithme de Dijkstra

Problème 1 et 2

Cas des valuations positives

Preuve

Récurrence sur j

Complexité de l'algorithme

1. Actualisation de π
   - à une itération : $\mathcal{O}(d^+(pivot))$
   - nombre total d'opérations : $\mathcal{O}(\sum_{v \in V} d^+(v)) = \mathcal{O}(|A|)$
2. Détermination du pivot
   - recherche du plus petit élément parmi q (q allant de n-1 à 1)
   - nombre total d'opérations : $\mathcal{O}(\sum_{q=1}^{n-1} q) = \mathcal{O}(\frac{n(n-1)}{2}) = \mathcal{O}(n^2)$

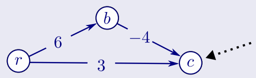
$m := |A| \leq n^2$ donc complexité globale : $\mathcal{O}(n^2)$

En pratique $\mathcal{O}(n + m\, ln(n))$ avec implémentation adéquate

Utilisation de tas de Fibonacci pour calculer l'*argmin*

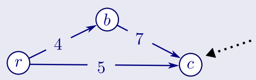
Cas où l'algorithme de Dijkstra ne fonctionne pas

Minimisation avec valeurs négatives



c sera pivot avant b
donc bc ne sera pas considéré

Maximisation



c sera pivot avant b
donc bc ne sera pas considéré

Sommaire

Algorithme de Bellman

Problème 1 et 2, Graphes sans circuits

Définition - Tri topologique des sommets d'un graphe $G = (V, A)$

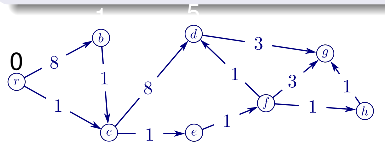
Ordre total sur V tel que i précède j pour tout $ij \in A$

Propriété

On peut toujours trier topologiquement les sommets d'un graphe sans circuit

Algorithme - Tri topologique des sommets d'un graphe orienté

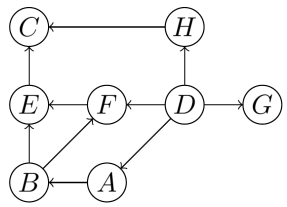
- Le sommet de départ a pour valeur 0 ; les autres ne sont pas valués
- À chaque itération : on value un sommet non valué dont tous les prédécesseurs sont valués



Quiz !

Question 4

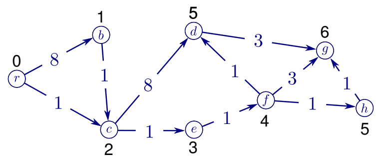
Déterminer l'ordre topologique des sommets de ce graphe.



Algorithme de Bellman

Problème 1 et 2

Graphes sans circuits



Données : $G = (V, A, p)$: graphe sans circuit

$T \leftarrow$ Sommets de V ordonnés selon le tri topologique

$\pi(r) \leftarrow 0$

pour j allant de 1 à n **faire**

$\pi(T[j]) \leftarrow \min_{v \in \Gamma^-(T[j])} (\pi(v) + p(v, T[j]))$

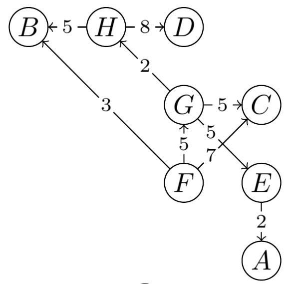
	r	b	c	e	f	d	h	g
Ordre	0	1	2	3	4	5	5	6
b	0	8	∞	∞	∞	∞	∞	∞
c	|	|	1	|	|	|	|	|
e	|	|	|	2	|	|	|	|
f	|	|	|	|	3	|	|	|
d	|	|	|	|	|	4	|	|
h	|	|	|	|	|	|	4	|
g	|	|	|	|	|	|	|	5

Quiz !

Question 5

Déterminer la plus courte distance pour atteindre chaque sommet à partir du sommet F en utilisant l'algorithme de Bellman.

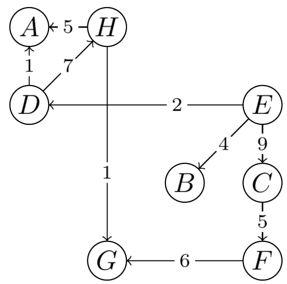
L'ordre topologique des sommets est le suivant : F1 - G2 - C3 - E3 - H3 - A4 - B4 - D4



Question 6

Déterminer la plus courte distance pour atteindre chaque sommet à partir du sommet E en utilisant l'algorithme de Bellman.

L'ordre topologique des sommets est le suivant : E1 - B2 - C2 - D2 - F3 - H3 - A4 - G4



Question 5 et 6 bellman

Algorithme de Bellman

Problème 1 et 2

Graphes sans circuits

Remarques

- Pour maximiser : remplacer min par max
- Gère les longueurs négatives
  Contrairement à l'algorithme de Dijkstra
- Très bonne complexité : $\mathcal{O}(m)$

Sommaire

Algorithme de Roy-Warshall-Floyd

Problèmes 1, 2 et 3

Objectif

Trouver le cheminement minimal entre toute paire de sommets
Pas de contraintes sur le graphe

Principe

- $M = \{m(x,y)\}_{x,y \in V}$
  Longueur du plus court chemin actuellement connu entre x et y
- Initialement $m(x,y) = p(x,y)$
  Ou ∞ si $(xy) \notin A$
- À chaque étape on considère $z \in V$ et, pour tout : $(xy) \in A$
  - si "passer par z" améliore le chemin actuel de x à y, $m(x,y)$ est mis à jour
- A la fin de l'algorithme :
  - $m(x,y) =$ plus court chemin de x à y

Variable de l'algorithme

pré(x,y) = prédécesseur de y sur le chemin minimum de x à y

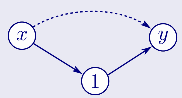
- préd : tableau de taille $|V| \times |V|$
  Initialement : préd(x, y) = x si $(xy) \in A$ et $\emptyset$ sinon

Algorithme de Roy-Warshall-Floyd

Problèmes 1, 2 et 3

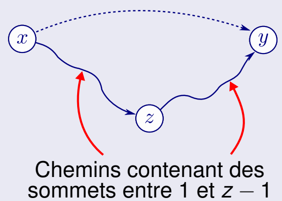
Étape 1 ($z = 1$)

L'arc (x,y) est ajouté s'il n'existait pas



Étape z

- Au début de l'étape z, chaque arc représente un chemin d'au plus z arcs



- Si passer par z améliore le chemin de x à y, on modifie m et préd
  préd(x,y)$\leftarrow$ préd(z,y)

Algorithme de Roy-Warshall-Floyd
Problèmes 1, 2 et 3

Données : $G = (V, A, p)$: graphe quelconque
Résultat : $M = m(x, y)$: valeur d'un plus court chemin de x à y
pour $(x, y) \in A$ **faire**
  $m(x, y) \leftarrow p(x, y)$
  préd$(x, y) \leftarrow x$

pour tout $(x, y) \notin A$ **faire**
  $m(x, y) \leftarrow \infty$
  préd$(x, y) \leftarrow \emptyset$

pour tout $z \in V$ **faire**
  pour tout $x \in V$ **faire**
    pour tout $y \in V$ **faire**
      si $m(x, y) > m(x, z) + m(z, y)$ **alors**
        $m(x, y) \leftarrow m(x, z) + m(z, y)$
        préd$(x, y) \leftarrow$préd(z, y)
    si $m(x, x) < 0$ **alors**
      STOP (il y a un circuit absorbant)

Algorithme de Roy-Warshall-FLoyd
Problèmes 1, 2 et 3

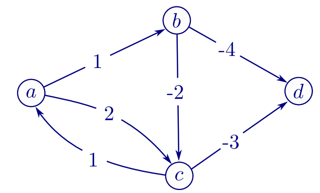


Tableau initial m, prèd

	a	b	c	d
a		1, a	2, a	
b			-2, b	-4, b
c	1, c			-3, c
d				

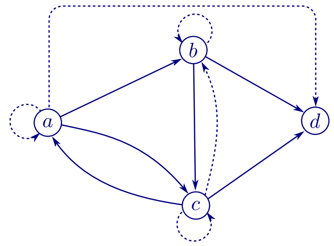


Tableau final m, prèd

	a	b	c	d
a	3, c	4, a	2, b	-1, c
b	-1, c	3, c	3, c	-5, c
c	1, c	5, a	3, b	-3, c
d				

Algorithme de Roy-Warshall-FLoyd
Problèmes 1, 2 et 3

Caractéristiques

- Détecte les circuits négatifs
  Valeur négative sur des termes de la diagonale
- $\mathcal{O}(n^3)$
  3 boucles imbriquées
- En cas de maximisation
  - remplacer ∞ par $-\infty$
  - échanger $<$ et $>$

Preuve d'optimalité par récurrence

- Au début de l'étape z on a les plus courts chemins passant par les z sommets déjà considérés
  De longueur au plus z
- A la fin dernière étape on a donc considéré tous les chemins

Quel algorithme pour trouver le chemin de valeur minimale ?

Caractéristique	Dijkstra	Bellman	Roy-Warshall-Floyd
Entre 2 sommets	x	x	x
Entre 1 sommets et tous les autres	x	x	x
Entre tous les couples de sommets			x
Gère les chemins maximaux		x	x
Poids négatifs		x	x
Graphe avec circuits	x		x
Gère les circuits absorbants			x
Complexité	$\mathcal{O}(n + m\ln(n))$	$\mathcal{O}(m)$	$\mathcal{O}(n^3)$

Remarque

Il existe d'autres algorithmes

Problèmes difficiles

- Trouver un chemin élémentaire de longueur minimale en présence de circuits absorbants — Passant par tous les sommets
- Trouver un chemin de longueur maximale dans un graphe avec valuations positives et circuits

En résumé

Notions abordées

- Plusieurs problèmes d'optimisation dans les graphes
  Arbres couvrants de poids minimal
  Plus courts chemins
  Voyageur de commerce
- Classes d'algorithmes
  Algorithmes gloutons
  Programmation dynamique
- Classe de problèmes
  - Problèmes faciles (ou polynomiaux)
    Une solution optimale peut être obtenue par un algorithme de complexité polynomiale
  - Problèmes "difficiles"

Sommaire

Classe Graph

graph.py

```
import numpy as np

class Graph:

    n = 0 # Nombre de sommets
    nodes = np.array([]) # Noms des sommets
    adjacency = np.empty(0) # Liens du graphe

    # Cree un graphe a partir d'un tableau de noms de sommets
    def __init__(self, sNames):
        self.nodes = np.copy(sNames)
        self.n = len(self.nodes)
        self.adjacency = np.zeros((self.n, self.n))

    # Permet d'ajouter une arete au graphe
    def addEdge(self, name1, name2, weight):
        id1 = np.where(self.nodes == name1)[0][0]
        id2 = np.where(self.nodes == name2)[0][0]
        self.adjacency[id1, id2] = weight
        self.adjacency[id2, id1] = weight

    ...
```


Classe Graph

Exemple d'utilisation

```
import numpy as np
import graph

def main():

    g = graph.Graph(np.array(["a", "b", "c", "d", "e", "f", "g"]))

    # Add the edges
    g.addEdge("a", "b",  1.0)
    g.addEdge("a", "c",  3.0)

    print(g.adjacency[0][1]); // Affiche 1.0
```

Sommaire

Matroïdes

Notations

- $E = \{e_1, e_2, .., e_n\}$: ensemble d'éléments fini et non vides
  $E \neq \emptyset$
- $I \subset \mathcal{P}(E)$

Définition - **Matroïde**

Couple $M = (E, I)$ tel que

- $I \neq \emptyset$ $(F \in I$ et $F' \subset F) \Rightarrow F' \in I$
- I famille de sous-ensembles **indépendants**
- Soient $F \in I$ et $H \in I$ tels que $card(F) < card(H)$, $\exists x \in H \backslash F$ tel que $F \cup \{x\} \in I$
  Propriété d'échange

Matroïdes

Définition - **Base**

Ensemble indépendant maximal pour l'inclusion

Propriété

Toutes les bases d'un matroïde ont le même cardinal

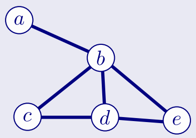
Exemple - Matroïde matriciel (H. Whitney)

- A : matrice donnée
- E : ensemble de lignes de A
- $H \in I$: ensemble de lignes de A linéairement indépendantes

Couple $M = (E, I)$ ne définissant pas un matroïde

- E : ensemble des sommets d'un graphe
- I : ensemble des ensembles stables de ce graphe
  Stable : ensemble de sommets deux à deux non adjacents

Exemple



- Tout ensemble inclus dans un stable est stable

mais...

- $\{a, c, e\}$, $\{a, d\}$ et $\{b\}$ sont des stables maximaux qui n'ont pas le même cardinal

Exemple de matroïde

Soient :

- $G = (V_G, E_G)$: graphe connexe
- $I_G = \{F \subset E_G \text{ tel que } G' = (V_G, F) \text{ sans cycle}\}$
- $M_G = (E_G, IG)$: matroïde graphique

Matroïdes pondérés

$M = (E, I, w)$ tel que

- $w_e > 0$: poids de $e \in E$

Poids de $F \subset E$

$w(F) = \sum_{e \in F} w(e)$

Problème

Trouver $F \subset I$ tel que $w(F)$ est maximal (ou minimal)

Algorithme glouton - Matroïde pondérés

Données : $M = (E, I, w)$: matroïde
Résultat : $F \in I$
$F \leftarrow \emptyset$
$L \leftarrow$ éléments de E ordonnés par poids décroissant
pour $i = 1$ à n **faire**
  si $F \cup \{e_i\} \subset I$ **alors**
    $F \leftarrow F \cup \{e_i\}$
retourner F

Théorème

L'algorithme glouton donne toujours l'optimum pour le problème du matroïde pondéré

Complexité

$$\underbrace{\mathcal{O}(n \log n)}_{\text{Complexité du tri}} + \underbrace{\mathcal{O}(n \times f(n))}_{\text{Complexité du test}}$$

- $\mathcal{O}(n \times f(n))$: complexité de la boucle "pour"

Remarque

Si on connaît la taille K d 'une base on peut remplacer la boucle par "tant que card F < K"

Conséquence

L'algorithme de Kruskal pour la recherche d'un arbre couvrant est optimal

Q1 : CF-AC-BG-CD
Q2 : BD-CD-CG-AH
Q3 : D0-A6-C7-E8-B9-F11-H13-G19
Q4 : A2-B3-C6-D1-E5-F4-G2-H2
Q5 : F0-G5-C7-E10-H7-A12-B3-D15
Q6 : E0-B4-C9-D2-F14-H9-A3-G10