Chapitre 3

Problèmes d’acheminement

Introduction 1/2

- Transport *truckload, entre des sources, avec des disponibilités* données, et des *destinations, avec des demandes* données. Les arcs du réseau ont des *coûts et éventuellement* des *capacités (débit maximum).*
- problèmes de chemins optimaux
- problèmes de distribution sans capacités
  - "problème de transport" (réseau à 2 couches)
  - "problème d'affectation" (cas avec quantités = 1)
  - "pb de transbordement" (réseau quelconque)

Introduction 2/2

- pbs de distribution avec capacités
  - pb du flot maximal
  - pb du flot de coût minimum
  - pb de multiflots (flots non miscibles)

Contrairement aux problèmes de localisation et de tournées, ces problèmes (sauf le dernier) disposent d'algorithmes rapides (polynomiaux).

On peut aussi les résoudre par programmation linéaire.

Chemins optimaux 1/25

- Le problème
- Graphe orienté valué *G = (X,U,C)*
- *X ensemble de n nœuds, U ensemble de m arcs*
- *Cij est le "poids" ou "coût" de l'arc (i,j).*
- Le "coût" peut être un vrai coût, une distance, une durée, etc. Si *G non orienté, on remplace chaque arête [i,j] par* deux arcs *(i,j) et (j,i) de même coût. Le coût d'un chemin* est défini en général comme le coût total de ses arcs.

Chemins optimaux 2/25

- On veut calculer un *chemin de coût minimal (ou plus court chemin ou PCC) entre 2 nœuds donnés de G.*
- Le problème a un sens si *G ne contient pas de circuit de* coût négatif (appelé *circuit absorbant).*
- On peut alors se limiter aux chemins *élémentaires (sans* nœuds répétés), avec au plus *n - 1 arcs. Cette propriété* est exploitée par tous les algorithmes.

Chemins optimaux 3/25

- Exemples

a) PCC dans les réseaux de transport

- Nœuds = villes ou carrefours, arcs = routes ou rues, coûts = distances ou temps.
- Ces données sont vendues sur CD par l'IGN, Michelin et TeleAtlas* par exemple.
- Exemple. France entière avec villes et carrefours principaux : 40000 nœuds et 140000 arcs environ.*

Chemins optimaux 4/25

b) Chemin de fiabilité maximale

- En temps de guerre, un espion doit aller d'une ville *s à* une ville *t. Les routes forment un graphe G = (X,U,P) où p(i,j) est la probabilité de passer l'arc (i,j) sans être pris.*
- L'objectif est de trouver un chemin qui maximise la probabilité d'arriver en *t (chemin de fiabilité maximale).*
- Ici, le "coût" d'un chemin est inhabituel (*produit des* valeurs des arcs), et on est en *maximisation.*

Chemins optimaux 5/25

c) Chemin de coût moyen optimal

- Un caboteur doit choisir un cycle sur *n ports. On connaît* pour chaque couple de ports (*x,y) le profit prévu b(x,y)* et la durée *d(x,y).*
- L'objectif est de trouver un cycle *μ avec un profit* maximum *C(μ) par unité de temps :*

$$C(\mu) = \sum_{(x,y)\in\mu} b(x,y) \Big/ \sum_{(x,y)\in\mu} d(x,y)$$

- C'est un *problème de cycle de coût moyen maximal dans* un graphe bivalué *G = (X,U,B,D).*

Chemins optimaux 6/25

- Les algorithmes de chemins optimaux
- On considère le cas où le coût d'un chemin est la *somme* des coûts des arcs. Il existe des algorithmes pour 3 types de problèmes :
- **Pb A** : trouver un PCC entre deux nœuds *s et t.*
- **Pb B** : trouver un PCC d'un nœud *s vers tout autre* nœud, donc une arborescence de *n - 1 chemins.*
- **Pb C :** trouver un PCC entre tout couple de nœuds, sous forme d'une matrice *D n × n appelée distancier.*

Chemins optimaux 7/25

- Un algo pour un problème peut être utilisé pour résoudre les deux autres, parfois en l'exécutant plusieurs fois.
- Algorithmes vus : Dijkstra et Bellman, pour le pb B. Ils calculent pour tout nœud *x un label V(x), coût des PCC* entre le nœud de départ *s et le nœud x.*
- L'algo de Dijkstra est du type à *fixation de labels (label setting algorithm) : à chaque itération, il calcule la valeur* définitive d'un label *V(x).*
- Celui de Bellman est du type à *ajustement de labels* (*label correcting algorithm) : il peut changer jusqu'à la* dernière itération le label de chaque noeud.

Chemins optimaux 8/25

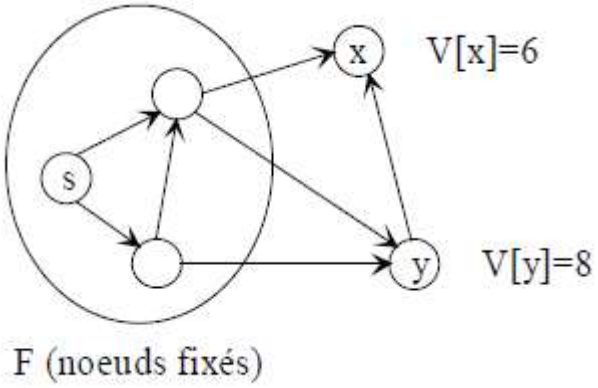
- Algorithme à fixation de labels de Dijkstra

a) Principe

- Attention : l'algo suppose des coûts *non-négatifs.*
- Au début, le nœud de départ *s a un label nul, les autres* un label +∞. L'ensemble *F des nœuds fixés (de label* définitif) est vide.
- A chaque itération, on cherche le nœud non fixé *x de* label minimal et on le fixe.

Chemins optimaux 9/25

- En effet, *V(x) ne peut plus diminuer si les coûts sont* non-négatifs. Exemple :



- Le nœud *x, de label minimal, peut aller dans F. En effet,* si *V(x) n'était pas définitif, il y aurait un chemin négatif* de *y à x, contradiction.*

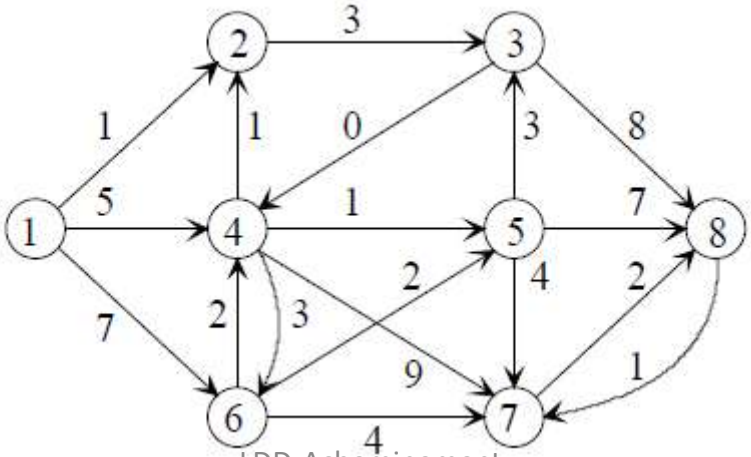
Chemins optimaux 10/25

- Ensuite, pour tout successeur *y de x, on met à jour V(y) si le chemin passant par x améliore V(y), c'est-à dire* si *V(x) + C(x,y) < V(y).*
- On note alors qu'on arrive à *y en venant de x, grâce à un* tableau *P de prédécesseurs : P(y) := x.*
- Si on veut un PCC de *s vers un seul nœud t, on stoppe* quand *t est fixé. Sinon, on doit faire n itérations pour* fixer tous les nœuds.
- A la fin, *V et P définissent alors un PCC de s vers tout* autre nœud.

Chemins optimaux 11/25

b) Exemple d'exécution (départ *s = 1)*

- Le tableau donne les labels au début des itérations et en fin d'algorithme. Le nœud fixé à chaque itération a un label encadré. Les tirets rappellent les nœuds déjà fixés.



Chemins optimaux 12/25

V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]	V[8]	Nœud fixé
[0]	∞	∞	∞	∞	∞	∞	∞	1
-0-	[1]	∞	5	∞	7	∞	∞	2
-0-	-1-	[4]	5	∞	7	∞	∞	3
-0-	-1-	-4-	[4]	∞	7	∞	12	4
-0-	-1-	-4-	-4-	[5]	7	13	12	5
-0-	-1-	-4-	-4-	-5-	[7]	9	12	6
-0-	-1-	-4-	-4-	-5-	-7-	[9]	12	7
-0-	-1-	-4-	-4-	-5-	-7-	-9-	[11]	8
-0-	-1-	-4-	-4-	-5-	-7-	-9-	-11-	FIN

Chemins optimaux 13/25

- Remarques :
  - Un label peut décroître *plusieurs fois.*
  - Pour les petits graphes, au lieu du tableau, on peut calculer les labels directement sur le dessin.
  - On obtient le chemin en regardant d'où vient le label. Nœud 8 : V(8) = 11 = V(7) + 2 : on vient du nœud 7.
  - Informatiquement, avec le tableau *P : P[8] = 7*.*
  - Parfois, il existe plusieurs chemins optimaux

Chemins optimaux 14/25

c) Algorithme de Dijkstra en style informatique

```
Mettre V à +∞, V[s]=P=0, F=∅        // initialisation
for k := 1 to n do                  // boucle principale (k est un compteur)
  Vmin := +∞                        // cherche x non fixé de label min
  for y := 1 to n with y∉F and V[y] < Vmin  do
    x       := y
    Vmin    := V[y]
  endfor
  ajouter x à F                     // on fixe x
  for each y successeur de x do     // mise à jour des successeurs y
    if Vmin + C[x,y] < V[y] then    // si le label de y est améliorable
      V[y]  := Vmin + C[x,y]
      P[y]  := x                    //mémorise d'où on vient
    endif
  endfor
endfor
```

Chemins optimaux 15/25

- Pour calculer un chemin entre deux nœuds *s et t,* remplacer la boucle principale par : *repeat ... until t ∈ F.*
- Pour avoir un *distancier D, n × n, appeler l'algorithme* pour *s variant de 1 à n et copier V dans la ligne s de D.*

d) Analyse de l'algorithme

  - Pour un algorithme d'optimisation combinatoire, c'est :
  - prouver la *convergence (l'algo s'arrête-t-il?),*
  - prouver l'*optimalité (l'algo est-il optimal?),*
  - évaluer la *complexité (est-il efficace?).*

Chemins optimaux 16/25

- Notre algo *converge : il fixe un nœud par itération.*
- Il est *optimal : les propriétés suivantes sont vraies à* chaque itération (preuve par récurrence).
- si un nœud *z est fixé ($z \in F$), alors V(z) est optimal.*
- si *z non fixé, V(z) est le coût des PCC de s à z dont* tous les nœuds sont fixés sauf le dernier (*z).*

Chemins optimaux 17/25

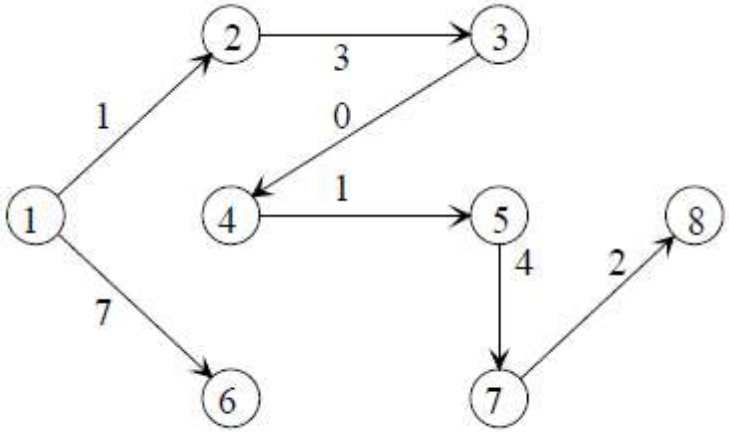
- *Complexité. Le 1er for a n itérations. Il contient un for de n itérations et un for d'au plus n itérations (car x a au* plus *n successeurs) : l'algo est en* $O(n^2)$.
- Il existe une version en *O(m.log2 n), basée sur des* structures de données complexes :
  - si *G est complet (m = n2), cette version est plus lente.*
  - Si *G est de type routier (m ≈ 4n), elle est plus rapide:* quelques secondes pour un distancier 1000×1000.

  Exercice : comparer les 2 complexités pour *n = 1000.*

Chemins optimaux 18/25

e) Stockage et récupération des chemins

- *P[x] : prédécesseur de x sur le chemin optimal de s à x.*
- Le tableau *P décrit (à l'envers) l'arborescence des PCC :*



Chemins optimaux 19/25

- Pour récupérer le chemin de *s à x (à l'envers) on* remonte vers *s avec un algorithme très simple :*
- //chemin de s à x (à l'envers)
- **repeat**
  - écrire x
  - x := P[x]
- **until x = 0**

Chemins optimaux 20/25

2.5 Algorithme de Bellman (ajuste des labels)

a) Principe

- Accepte les coûts < 0. Méthode de *programmation dynamique, définie par les relations de récurrence :*

$$\begin{cases} V_0(s) = 0 \\ \forall y \neq s : V_0(y) = +\infty \\ \forall k > 0, \forall y : V_k(y) = \underset{x \text{ préd. de } y}{Min} \{V_{k-1}(y), V_{k-1}(x) + C(x,y)\} \end{cases}$$

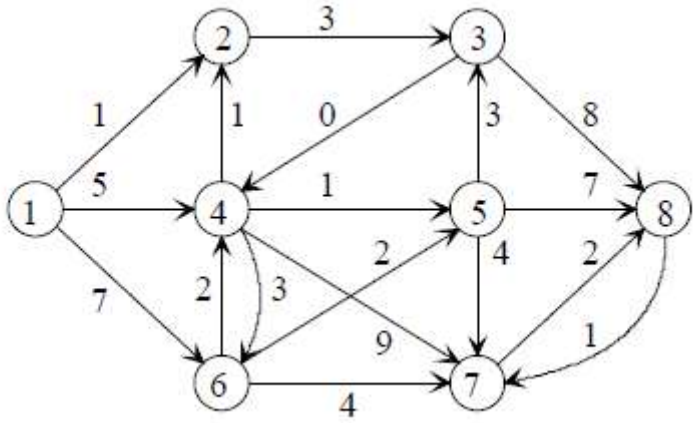
Chemins optimaux 21/25

- Principe :
  - calcul des PCC *d'au plus k arcs, k = 1,2,3...*
  - *V0 tableau initial de labels, V_k labels à l'étape k.*
  - 3ème relation : un PCC de *k arcs de s à y prolonge un* PCC de *k-1 arcs de s vers un prédécesseur x de y.*
  - En pratique, on calcule les tableaux *V_k successifs avec* la 3ème relation.
  - On stoppe quand les labels ne varient plus.

Note : pour un *graphe sans circuit (gestion de projet), il* suffit d'appliquer la 3ème relation niveau par niveau.

Chemins optimaux 22/25

b) Exécution sur le graphe-exemple (*s = 1)*



k	Labels dans V_k							
	1	2	3	4	5	6	7	8
0	0	+∞	+∞	+∞	+∞	+∞	+∞	+∞
1	0	1	+∞	5	+∞	7	+∞	+∞
2	0	1	4	5	6	7	11	+∞
3	0	1	4	4	6	7	10	12
4	0	1	4	4	5	7	10	12
5	0	1	4	4	5	7	9	12
6	0	1	4	4	5	7	9	11
7	0	1	4	4	5	7	9	11

- A cause du PCC de 6 arcs (1,2,3,4,5,7,8), l'algo va jusqu'à *V6. En calculant V7, les labels ne changent plus.*

Chemins optimaux 23/25

c) Algorithme détaillé avec 2 tableaux *V (Vk-1) et W (Vk)*

```
initialiser V à +∞, V[s] et P à 0        // initialisation
k := 0
repeat                                   // boucle principale
  k := k + 1                             // calcule W (Vk) à partir de V (Vk-1)
  initialiser W à + ∞ et q à 0           // q nombre de labels modifiés
  for y := 1 to n do                     // calcul de W(y) = Vk(y)
    for each x prédécesseur de y
      if V[x] + C(x,y) < W[y] then
        W[y]  := V[x] + C(x,y)
        P[y]  := x;  q := q + 1
      endif
    endfor
  endfor
  V := W                                 // W devient V pour l'itération suivante
until  q = 0.                            // on stoppe si aucun label n'est modifié
```

Chemins optimaux 24/25

d) Analyse de l'algorithme

- ***Convergence***. *L'algo boucle s'il y a un circuit absorbant.* Sinon, les labels sont stables au plus tard pour *k = n-1* (nb max d'arcs d'un PCC élémentaire*). L'algo fait une itération de plus constatant que *q = 0.*
- ***Optimalité***. *Par récurrence, on montre que V définit les* PCC d'au plus *k arcs en fin d'itération k.*
- ***Complexité***. *Les deux for traduisent la 3ème relation* consultent tous les arcs *(x,y) de G : une itération coûte O(m). Comme il y a au plus n itérations, l'algo est en O(mn), soit O(n^3) si G est complet.*

Chemins optimaux 25/25

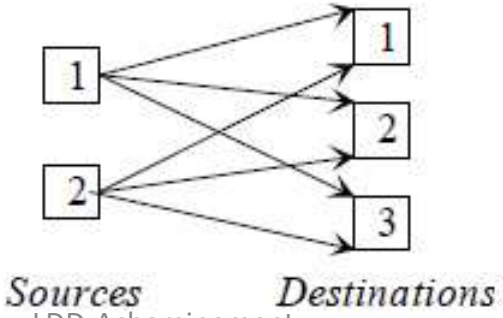
- A comparer avec *$O(n^2)$ pour Dijkstra.*
- En pratique, l'algo est rapide : par exemple *$m \approx 4n$ pour* un graphe routier. Les labels sont souvent stabilisés en *$k < n-1$ itérations ($k = 1$ si réseau en étoile !).*
- Pour détecter un éventuel *circuit absorbant, il faut* remplacer le *until par until ($q = 0$) or ($k = n$). A la fin, si $k = n$ mais $q \neq 0$, c'est que les labels ne sont pas* stabilisés : *G contient un circuit absorbant.*
- Si on veut calculer des PCC d'au plus *e étapes (e arcs), il* suffit de stopper quand *$k = e$.*

Le problème de transport 1/16

- Description
- Problème classique (transportation problem en anglais). étudié dès 1930 : Hitchcock (USA), Kantorovitch (URSS).
- Données :
  - X : m origines, disponibilités ai (A = somme des ai)
  - Y : n destinations, demandes bj (B = somme des bj)
  - coûts unitaires de transport cij entre X et Y.
  - Objectif : calculer un plan de transport (flux xij) pour satisfaire les demandes en minimisant le coût total.

Le problème de transport 2/16

- Applications
- Transport entre des origines et des destinations, dans un réseau de capacité non limitée. Exemple, transports maritimes : X ports européens, Y ports africains, c_{ij} coût de transport/t du port i au port j → graphe biparti.



Le problème de transport 3/16

- Autre exemple : distribution dans un réseau routier à deux couches. Un arc (i,j) modélise un PCC de i à j dans le réseau routier réel. Son coût cij a été calculé avec l'algorithme de Dijkstra, par exemple.
- Formulation par PL
  - On suppose le problème équilibré (A = B), sinon :
  - si B > A, source fictive de disponibilité B-A.
  - si B < A, destination fictive n+1 de demande A-B.
  - les arcs incidents aux nœuds fictifs ont un coût nul.
  - dans la solution, il faut ignorer les flux sur ces arcs.

Le problème de transport 4/16

(1) $Min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} . x_{ij}$ Fonction-objectif.

(2) $\forall i = 1 \ldots m : \sum_{j=1}^{n} x_{ij} = a_i$ Respect des disponibilités.

(3) $\forall j = 1 \ldots n : \sum_{i=1}^{m} x_{ij} = b_j$ Satisfaction des demandes

(4) $\forall i = 1 \ldots m, \forall j = 1 \ldots n : x_{ij} \geq 0$ Flux non-négatifs.

exemple

C	1	2	3	4	a_i
1	7	2	9	10	8
2	6	11	7	12	6
3	15	8	3	4	9
b_j	3	7	5	8	

Le problème de transport 5/16

- Solutions réalisables et arbres
- Heuristique du "coin nord-ouest" (CNO).

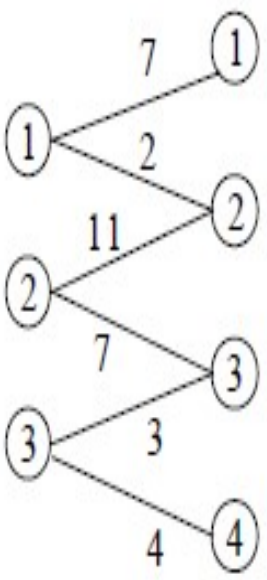
C	1	2	3	4	a_i
1	7	2	9	10	8
2	6	11	7	12	6
3	15	8	3	4	9
b_j	3	7	5	8	

X	1	2	3	4
1	3	5	0	0
2	0	2	4	0
3	0	0	1	8

- Départ en haut à gauche (le CNO). Si on peut épuiser l source, on change de source. Si on peut satisfaire la destination, on passe à la suivante. Coût trouvé : 116.

Le problème de transport 6/16

- La solution forme un arbre (graphe connexe sans cycle) de m+n-1 arêtes. La théorie de la PL montre que :
- il existe toujours un optimum dont les arcs à flux xij non nuls forment un arbre sur G.
- si les demandes et dispos sont des entiers, il existe un optimum à flux entiers (donc pas besoin de définir des variables entières dans le PL).



Le problème de transport 7/16

- Algorithme du stepping-stone
- On peut utiliser la PL pour résoudre le problème, mais il existe un algo plus rapide, le stepping-stone (marelle).
- Grâce aux propriétés précédentes, il peut considérer uniquement les solutions entières en forme d'arbre.
- Partant d'un arbre initial (comme celui de CNO), il construit des arbres successifs de coût décroissant.

Le problème de transport 8/16

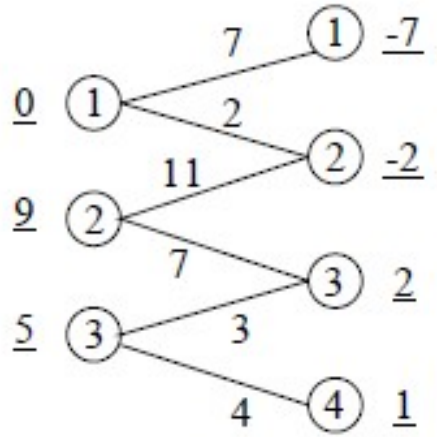
- Dans l'arbre précédent, si x_{21} augmente de 0 à 1 :
  - pour respecter disponibilités et demandes, le flux doit diminuer sur (1,1) et (2,2) et augmenter sur (1,2).
  - la variation de coût est appelée coût marginal de (2,1), elle vaut D_{21} = 6-7+2-11 = -10.
  - l'augmentation de x_{21} est rentable mais limitée à 2, car x_{12} s'annule.
  - on obtient un nouvel arbre moins coûteux, avec l'arête (2,1) en plus et (1,2) en moins.

Le problème de transport 9/16

- Le stepping-stone calcule à chaque itération un nouvel arbre moins coûteux, en cherchant un nouvel arc de coût marginal négatif D_{ij} le plus petit possible. Il s'arrête quand tous les coûts marginaux sont >=0. On peut prouver que la solution est alors optimale.
- Pour calculer vite les D_{ij}, on munit tout dépôt i d'un potentiel u_i, et toute destination j d'un potentiel v_j. La source 1 a un potentiel 0. Puis, pour un client j voisin d'une source à u_i calculé, on pose $v_j = u_i - c_{ij}$ (le flux descend des dépôts) et, pour toute source i voisine d'un client à v_j calculé, on pose $u_i = v_j + c_{ij}$.

Le problème de transport 10/16

- Exemple de potentiels sur l'arbre CNO:

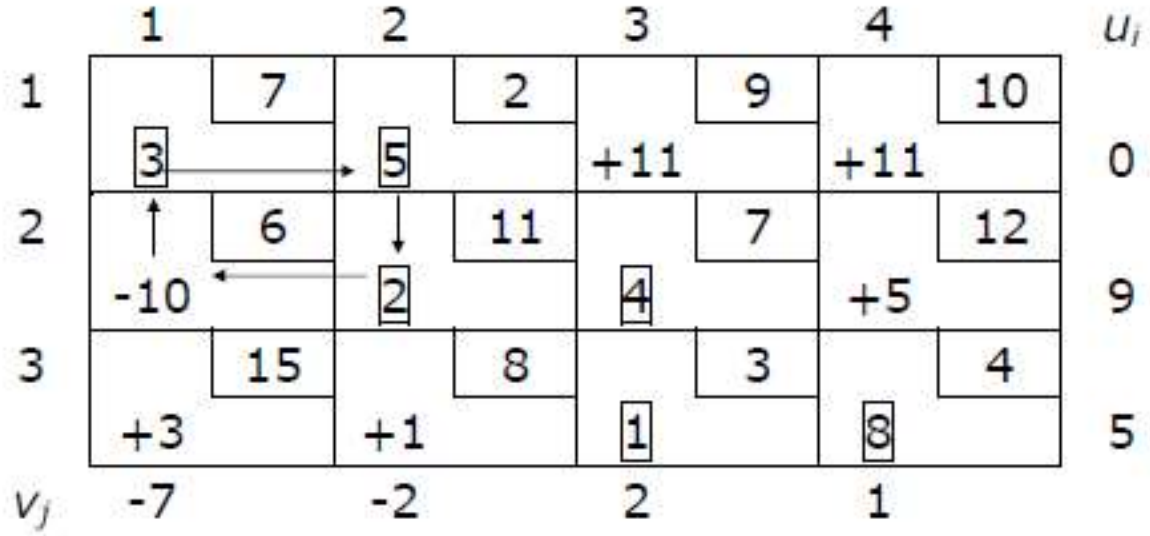


- Coût marginal si on utilise (i,j) : $D_{ij} = v_j - u_i + c_{ij}$.
  Exemple, on retrouve $D_{21} = -10 = v_1 - u_2 + c_{21} = -7-9+6$.
  Sans potentiels, combien faut-il d'opérations pour D_{31} ?

Le problème de transport 11/16

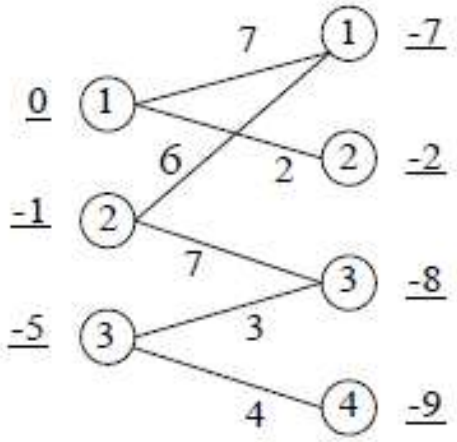
- Calculs du stepping-stone dans un tableau m xn.
- Détail d'une itération :
  - rappeler des coûts en haut et à droite de chaque case
  - écrire les flux ≠0 de l'arbre actuel (encadrés)
  - calculer les potentiels u_i et v_j
  - calculer les coûts marginaux dans les cases à flux nuls
  - si aucun n'est négatif, fin
  - calculer l'arête [i,j] de coût réduit négatif minimum
  - identifier le cycle créé par [i,j] avec des flèches
  - mise à jour les flots sur le cycle (changement d'arbre).

Le problème de transport 12/16



- Le seul arc avec gain est (2,1), coût marginal -10. Le flux augmente sur (2,1) et (1,2), et diminue sur (1,1) et (2,2). x_{21} peut passer de 0 à 2 et x_{22} s'annule.

Le problème de transport 13/16



- Nouvelle solution avec potentiels de l'itération suivante.
- On a bien un arbre, de coût 116 + (-10) *2= 96.
- C'est l'ancien coût + $D_{21}.x_{21}$.

Le problème de transport 14/16

	1	2	3	4	u_i
1	7 [1]	2 [7]	9 +1	10 +1	0
2	6 [2]	11 +10	7 [4]	12 +4	-1
3	15 +13	8 +11	3 [1]	4 [8]	-5
v_j	-7	-2	-8	-9	

- Nouveau tableau. Coûts marginaux >=0 : optimum! A la fin, il vaut mieux vérifier le respect des disponibilités et des demandes, ainsi que le coût de la solution.

Le problème de transport 15/16

- Dégénérescence
- Une solution dégénérée a moins de m+n-1 flux ≠0 : l'arbre est fragmenté en 2 sous-arbres (ou plus).

X	1	2	3	4	a_i :
1	1	3			4
2		3			3
3			3	4	7
b_j :	1	6	3	4	

- en PL, une solution dégénérée a des variables de base à 0. Raison ici : les sources 1 et 2 ont une dispo totale égale à l'offre totale des destinations 4 et 3.

Le problème de transport 16/16

- Pour ne pas rater l'optimum, il faut absolument reconnecter les sous-arbres en incluant des arcs de flux infiniment petit e dans la solution : on peut ajouter (2,3) ou (3,2) dans l'exemple. Ces arcs sont traités ensuite exactement comme les autres.
- L'algo peut faire des changements d'arbre à gain nul (en fait e) si un arc fictif se trouve sur le cycle créé par le nouvel arc. Il faut continuer quand même les itérations et stopper seulement quand les D_{ij} sont >=0. La seule précaution à prendre est de ne pas revenir sur une solution antérieure.

Exercice

- 4 origines O_1, O_2, 0_3, O_4 et 5 destinations D_1, D_2, D_3, D_4, D_5. Chaque origine a une offre à respecter et chaque destination a une demande à satisfaire. Le tableau suivant présente les informations nécessaires

	D1	D2	D3	D4	D5	Offre
O1	7	12	1	5	9	12
O2	15	3	12	6	14	11
03	8	16	10	12	7	14
O4	18	8	17	11	16	8
Demande	10	11	15	5	4	

- Quelles quantités de marchandises à envoyer des origines vers les destinations en respectant l'offre et en satisfaisant la demande au moindre coût?

Solution

	D1	D2	D3	D4	D5	Offre
O1			12			12
O2		11				11
03	10				4	14
O4			3	5		8
Demande	10	11	15	5	4	

Coût de la solution = 259

Méthode de Balas et hammer

- Cette règle est basée sur le calcul des regrets. Le regret associé à une ligne ou à une colonne est la différence entre le coût minimum et le coût immédiatement supérieur dans cette ligne ou dans cette colonne. C'est une mesure de la priorité à accorder aux transports de cette ligne ou de cette colonne, car un regret important correspond a une pénalisation importante si on n'utilise pas la route de coût minimum. On remet constamment à jour ces regrets sur le tableau restant lorsqu'on a saturé une ligne ou une colonne en envoyant la quantité maximum sur la route de coût le plus faible dans la ligne ou la colonne de regret maximum

Problème d'affectation 1/2

- Dans un atelier à n postes et n personnes, on a mesuré pour chaque personne *i* et chaque poste *j* le nombre d'erreurs par heure c_{ij} quand on la met sur ce poste. Le but est d'affecter chaque personne à un poste et chaque poste à une personne pour minimiser le nombre total d'erreurs.

Problème d'affectation 2/2

$$(1)\quad Min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}.x_{ij}$$

$$(2)\quad \forall i = 1\ldots n : \sum_{j=1}^{n} x_{ij} = 1$$

$$(3)\quad \forall j = 1\ldots n : \sum_{i=1}^{n} x_{ij} = 1$$

$$(4)\quad \forall i = 1\ldots n, \forall j = 1\ldots n : x_{ij} \geq 0$$

Problème du flot maximum 1/23

6.1 Définition et modélisations

- Définition
  - Soit un réseau avec des capacités (débit maximal permis) sur les arcs.
  - Le problème du flot maximal consiste à faire circuler un flot de débit maximal entre deux nœuds donnés s et t, en respectant les capacités.
- Modèle de graphe orienté value G = (X,U,C,s,t) :
  - X ensemble de *n* nœuds, U ensemble de *m* arcs
  - C_{ij} capacité maximale (entière) de l'arc (*i*,*j*)
  - nœud source s et nœud puits t.

Problème du flot maximum 2/23

- Un flot est une application Φ de U dans IN:
- respectant les capacités des arcs

$$\forall (i,j) \in U,\ 0 \leq \Phi_{ij} \leq C_{ij}$$

- conservant le flot en chaque nœud (Kirchhoff)

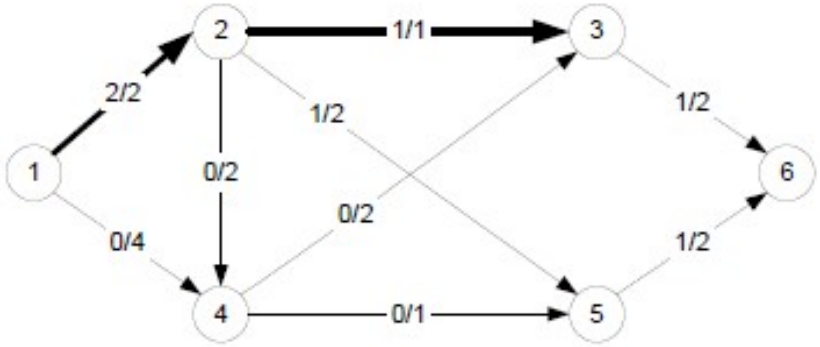
$$\forall i \neq s,t,\ \sum_{j \text{ succ de } i} \Phi_{ij} = \sum_{j \text{ préd de } i} \Phi_{ji}$$

- de débit F égal au flot sortant de s ou entrant en t

$$F = \sum_{j \text{ succ de } s} \Phi_{sj} = \sum_{j \text{ préd de } t} \Phi_{jt}$$

Problème du flot maximum 3/23

- Le problème du flot maximal consiste à trouver un flot maximisant F
- Exemple:
  - s = 1, t = 6, flot de débit 2 (flux/capacité).



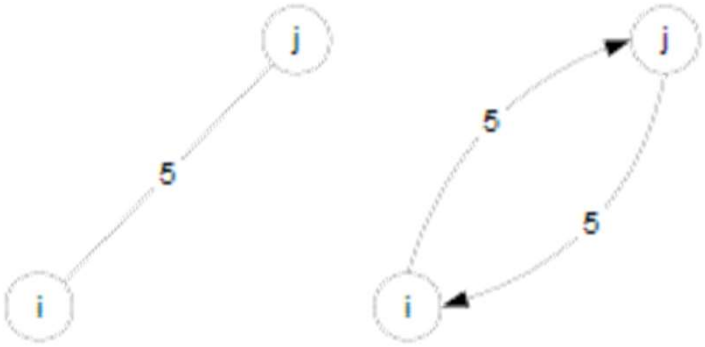
Problème du flot maximum 4/23

- Modèle de programmation linéaire
- C et Φ considérés comme des matrices :

$$\begin{cases} Max\ F \\ \sum_{j\ \text{succ de}\ i} \Phi_{ij} - \sum_{j\ \text{préd de}\ i} \Phi_{ji} = \begin{cases} F \text{ si } i = s \\ 0 \quad \forall i \neq s,t \\ -F \text{ si } i = t \end{cases} \\ 0 \leq \Phi_{ij} \leq C_{ij}, \quad \forall (i,j) \in U \end{cases}$$

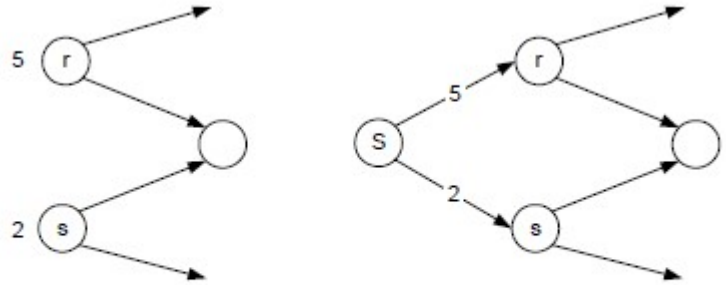
Problème du flot maximum 5/23

- Cas particuliers
- G non orienté. On remplace toute arête [i,j] par 2 arcs (i,j) et (j,i) de même capacité.



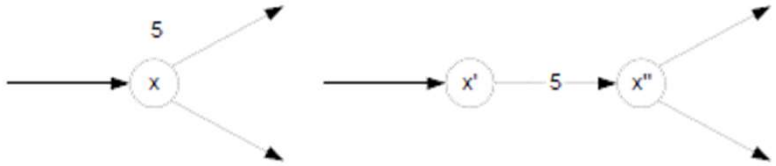
Problème du flot maximum 6/23

- Sources multiples et disponibilités limitées. On les connecte à une super-source.



Problème du flot maximum 7/23

- Capacité sur un nœud x. On remplace x par x’ et x’’, avec un arc (x’, x’’) de même capa que x.



- Nœud *j* avec un seul prédécesseur *i* et un seul successeur *k*, on peut simplifier en remplaçant (*i*,*j*) et (*j*,*k*) par (*i*,*k*), avec $C_{ik} = \min (C_{ij}, C_{jk})$.

Problème du flot maximum 8/23

Exemples d'applications

a) Transports de marchandises

- Les arcs sont des trajets avec des moyens de transport de capacités connues (tonnes / jour par exemple).

b) Transports de fluides

- Les flots sont très utilisés pour les problèmes d'adduction d'eau et pour modéliser des réseaux de canalisations (oléoducs, gazoducs).

Problème du flot maximum 9/23

c) Fiabilité dans les réseaux de télécoms

- Si un réseau a k chemins disjoints (sans nœuds communs) entre 2 nœuds s et t, il peut résister à k-1 coupures de chemins.
- On peut calculer k grâce à une méthode de flot : tout arc reçoit une capacité $+\infty$ et chaque nœud sauf s et t une capacité 1. Un flot de débit k va tracer k chemins disjoints grâce aux capacités unitaires.
- En maximisant le flot, on obtient le nb maximal de chemins disjoints de s à t.

Problème du flot maximum 10/23

- Coupes et autres concepts utiles
  - Une coupe (cut) de G est une partition des nœuds Z = (S,T) avec s dans S et t dans T. Un arc sortant de Z va de S à T, un arc entrant va de T à S. La capacité de la coupe est la somme des capacités de ses arcs sortants :

$$C(S,T) = \sum_{\substack{(i,j)\in U \\ i\in S, j\in T}} C_{ij}$$

- Propriété 1 : Le débit de tout flot est inférieur ou égal à la capacité de toute coupe. Un flot est donc maximal si on trouve une coupe de capacité égale au débit.

Problème du flot maximum 11/23

- En effet, sommons les lois des nœuds sur S :

$$\forall (S,T),\ \sum_{i\in S}\ \sum_{j \text{ succ de } i}\Phi_{ij} = F + \sum_{\substack{i\in S\\ i\neq s}}\ \sum_{j \text{ préd de } i}\Phi_{ji}$$

- Puis soustrayons les flux des arcs à 2 extrémités dans S :

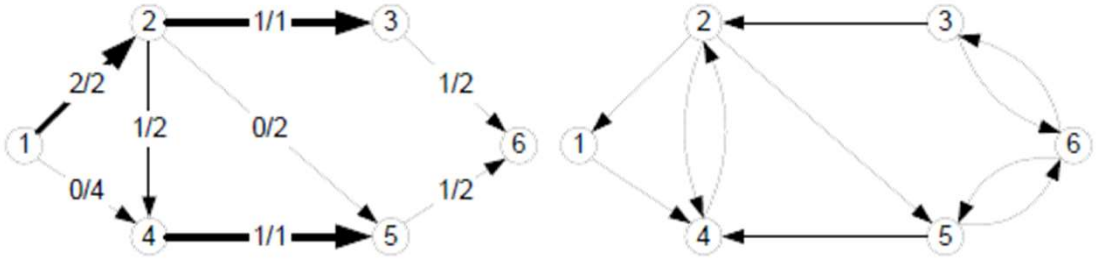
$$\forall (S,T),\ \sum_{\substack{(i,j)\in U\\ i\in S, j\in T}}\Phi_{ij} = F + \sum_{\substack{(j,i)\in U\\ j\in T, i\in S-\{s\}}}\Phi_{ji} \quad \text{ou}$$

$$\forall (S,T),\ F = \sum_{\substack{(i,j)\in U\\ i\in S, j\in T}}\Phi_{ij} - \sum_{\substack{(j,i)\in U\\ j\in T, i\in S-\{s\}}}\Phi_{ji}$$

- On a bien F <= C(S,T) : la 1ère somme est bornée par C(S,T), et la 2ème somme est positive ou nulle.

Problème du flot maximum 12/23

- Un chemin améliorant va de s à t avec des arcs (i, j) insaturés : $F_{ij} < C_{ij}$. Un flot est complet s'il n'y a plus de chemins améliorants. Mais il n'est pas forcément optimal.
- Exemple : le flot peut augmenter sur la chaîne améliorante [1,4,2,5,6] qui emprunte l'arc (2,4) à l'envers.



Problème du flot maximum 13/23

- Pour décrire les augmentations possibles, on peut construire un graphe d'écart Ge(Φ). Il contient les nœuds de G, les arcs non saturés, et un arc (j, i) pour tout arc (i, j) de flot non nul dans G. Une chaîne améliorante de G correspond à un chemin de s à t dans Ge(Φ).
- La plupart des algorithmes de flot ne génèrent pas de graphe d'écart. Avec les listes de successeurs et de prédécesseurs pour tout nœud *i*, ils recherchent des chaînes améliorantes en empruntant les arcs (i, j) non saturés et les arcs (j, i) de flot non nul.

Problème du flot maximum 14/23

- Algorithme de Ford et Fulkerson

a) Principe

- On part du flot nul. Chaque itération cherche une chaîne améliorante μ de s à t dans G. On stoppe si μ n'existe pas. Sinon, on calcule l'augmentation de flot δ. Soit μ+ l'ensemble des arcs avant de μ (leur flux va augmenter) et μ- les arcs arrière (leur flux va baisser). Alors :

$$\delta = \min\left(\min_{(i,j)\in \mu^+} \left(C_{ij} - \Phi_{ij}\right), \min_{(i,j)\in \mu^-} \left(\Phi_{ij}\right) \right)$$

- Le flux va augmenter de δ sur μ+ et diminuer de δ sur μ-.

Problème du flot maximum 15/23

```
F := 0
initialiser le flot Φ à 0
repeat
  chercher une chaîne améliorante μ
  if μ existe then
    calculer l'augmentation de débit δ
    ajouter δ aux flux des arcs de μ+
    diminuer de δ les flux des arcs de μ-
    F := F + δ
  endif
until μ n'existe pas
```

Problème du flot maximum 16/23

- On cherche μ avec une procédure qui marque des nœuds de proche en proche à partir de s :
- on marque s avec un « + ».
- on fait les marquages suivants, en ordre quelconque, jusqu'à ce qu'on n'en trouve plus.
- si (i, j) insaturé, i déjà marqué et j non marqué, on marque j avec la marque « +i ».
- si (i, j) de flux non nul, j déjà marqué et i non marqué, on marque i avec la marque « -j »
- on a trouvé une chaîne si t est marqué.

Problème du flot maximum 17/23

b) Complexité de l'algorithme

- On peut l'implémenter en $O(n.m^2)$. Il existe des algorithmes plus rapides mais plus compliqués :
- celui de Karzanov en $O(n^3)$
- celui de Ahuja en O(m.n.log Cmax)

Problème du flot maximum 18/23

c) Optimalité de l'algorithme de Ford-Fulkerson

- A la fin, les nœuds marqués et non marqués forment une coupe (S,T). Soit la relation

$$F = \sum_{\substack{(i,j)\in U \\ i\in S, j\in T}} \Phi_{ij} - \sum_{\substack{(j,i)\in U \\ j\in T, i\in S-\{s\}}} \Phi_{ji}$$

- $\Phi_{ij} = C_{ij}$ pour (i,j) sortant de S, et $\Phi_{ji} = 0$ pour tout arc (j,i) entrant dans S, sinon on aurait pu marquer j. La 1ère somme est égale à la capacité de la coupe, la seconde est nulle. On a un flot de débit égal à la capacité de la coupe, ce flot est donc maximal d'après la propriété 1.

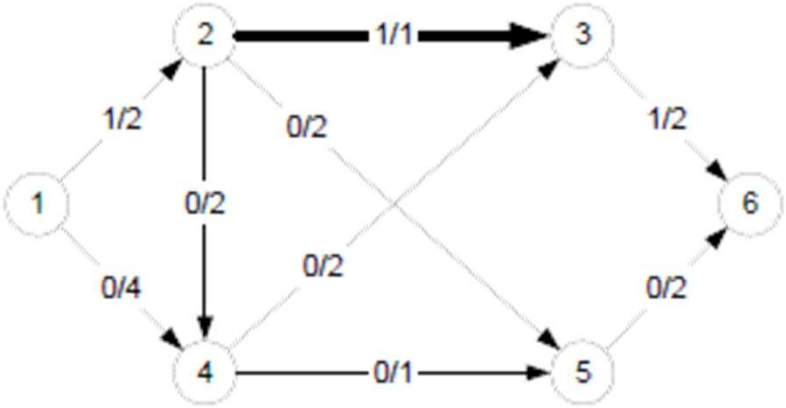
Problème du flot maximum 19/23

- D'où les propriétés 2 et 3, la 2 est un théorème célèbre appelé **max-flow**, **min-cut** :
- Propriété 2 : Le débit maximal d'un flot de G est égal à la capacité minimale des coupes de G.
- Propriété 3 : L'algorithme de Ford et Fulkerson trouve un flot maximal de s à t.

Problème du flot maximum 20/23

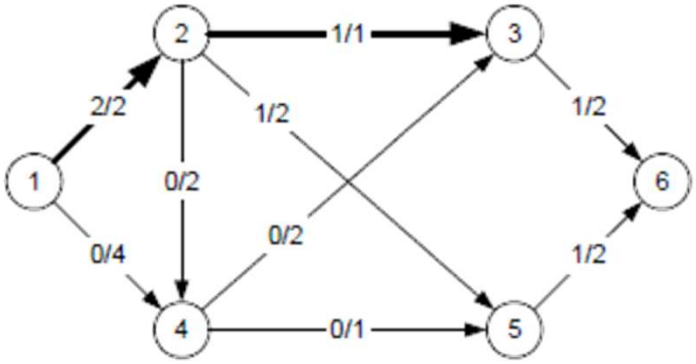
d) Un exemple

- Reprenons le graphe précédent. Par convention, testons les successeurs des nœuds en ordre croissant. Partant du flot nul, on trouve la chaîne (1,2,3,6), avec d = 1 (arc (2,3)). On obtient le flot suivant, de débit F = 1 :



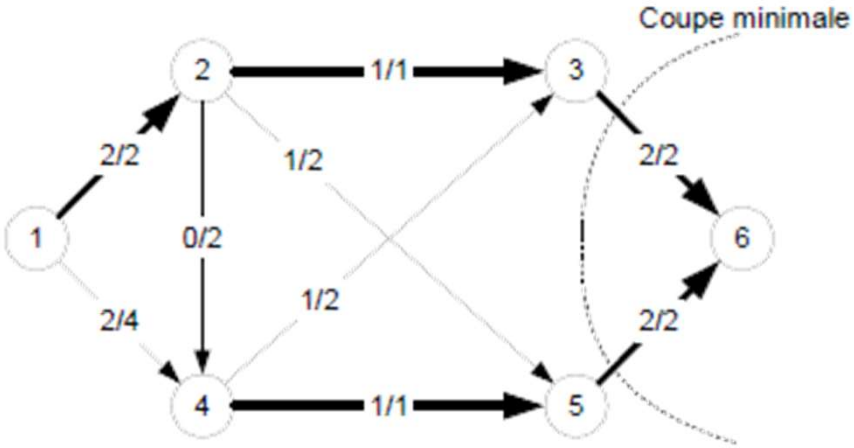
Problème du flot maximum 21/23

- On trouve ensuite (1,2,5,6) avec $\delta = 1$, dû à l'arc (1,2). L'augmentation produit un débit F = 2 et sature l'arc (1,2), ce qui donne le flot suivant :



Problème du flot maximum 22/23

- Puis on trouve (1,4,3,6) et (1,4,5,6) avec $\delta = 1$. On obtient le flot suivant. On marque 1, 4, 3, puis 2 en prenant (2,3) à l'envers, puis 5 en prenant l'arc avant (2,5). 6 n'est pas atteint : l'algorithme est terminé.



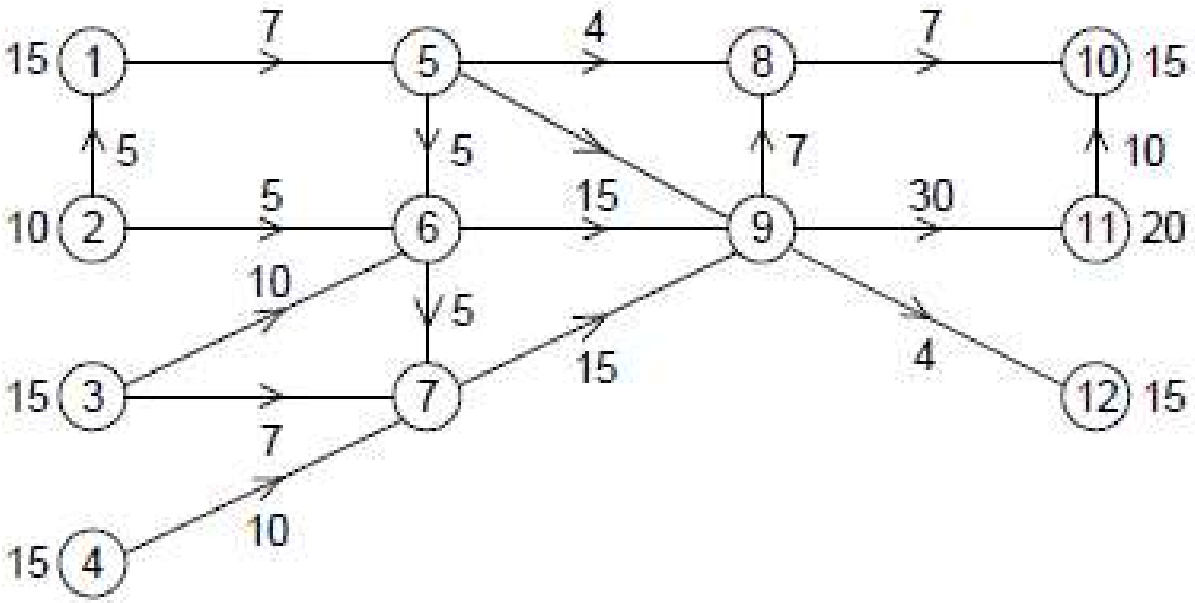
Problème du flot maximum 23/23

- Le flot max a donc un débit de 4.
- Vérifions la propriété 2 en traçant la coupe minimale :
  - les arcs sortant de la coupe à savoir (3,6) et (5,6) sont saturés.
  - ici, la coupe n'a pas d'arcs arrière, sinon ils auraient un flot nul.
  - la capacité de la coupe vaut bien 4.

Problème d'adduction d'eau 1/3

- Le graphe ci-dessous représente un réseau régional d'adduction d'eau. Les nœuds 1 à 4 sont des sources ou réservoirs pouvant fournir respectivement 15, 10, 15 et 15 milliers de m3 par jour. Les nœuds 10, 11 et 12 sont 3 villes dont les demandes journalières prévues dans dix ans sont respectivement de 15, 20 et 15 milliers de m^3. Les disponibilités et demandes sont données près des nœuds. Les arcs sont des canalisations, aqueducs etc. avec des capacités en milliers de m^3/j.

Problème d'adduction d'eau 2/3



Problème d'adduction d'eau 2/3

a) Déterminer le flot maximal permis par ce réseau et donner la coupe minimale. Le réseau suffira-t-il à satisfaire les 3 villes dans 10 ans ?

b) Le Conseil Régional décide de refaire en priorité (en les élargissant) les deux canalisations les plus vétustes, entre les nœuds 1 et 5, et 9 et 12. Si aucune autre canalisation n'est refaite, jusqu'à quelle capacités peut-on augmenter utilement ces deux canalisations ? Déterminer le nouveau flot maximal après ces réparations.

c) Devant le coût des travaux, le Conseil souhaite finalement refaire une seule des deux canalisations. Laquelle recommandez-vous et pourquoi (ne pas recalculer le flot).

Problèmes de flots maximum à coût minimal

Problèmes de flots à coût minimal

- Définition et modélisation
  - Définition*
    - Soit un réseau G = (X,U,C,W,s,t) avec :
    - X ensemble de n nœuds, U ensemble de m arcs,
    - C_{ij} capacité (entière) de l'arc (i,j),
    - W_{ij} coût de passage d'une unité de flux sur (i,j),
    - une source s et un puits t.
- Le problème du flot de coût minimal consiste à faire circuler un flot de s à t, de débit donné F, en minimisant le coût total.

Problèmes de flots de coût minimal

- On peut adapter le PL vu pour le flot maximum, le débit F devient une constante.

$$\begin{cases} Min \sum_{(i,j)\in U} W_{ij}\Phi_{ij} \\ \sum_{j \text{ succ de } i} \Phi_{ij} - \sum_{j \text{ préd de } i} \Phi_{ji} = \begin{cases} F \text{ si } i = s \\ 0 \quad \forall i \neq s,t \\ -F \text{ si } i = t \end{cases} \\ 0 \le \Phi_{ij} \le C_{ij}, \quad \forall (i,j) \in U \end{cases}$$

Problèmes de flots de coût minimal

b) Exemple de flots de coût minimum

- On a F tonnes de matériel à transporter entre deux usines. Les arcs du réseau représentent des liaisons avec différents moyens de transport (route, rail, avion), chacune ayant une capacité et un coût. Le problème consiste à acheminer le matériel à coût minimal.

c) Graphe d'écart

- Comme pour le flot max, il existe des algorithmes de graphes plus rapides que la PL, mais cette fois ils utilisent le graphe d'écart. Pour un flot Φ, le graphe d'écart Ge(Φ) doit tenir compte des coûts :

Problèmes de flots de coût minimal

- un arc non saturé (i, j) de G est conservé dans le graphe d'écart avec son coût ;
- tout arc (i, j) de flot non nul de G donne lieu à un arc inversé (j, i) de coût -Wij dans Ge.
- Comme pour le flot maximal, un chemin π de s à t dans le graphe d'écart correspond à une chaîne améliorante μ de G avec des arcs avant et arrière.
- Une augmentation de flot le long de μ donnera un gain ou une perte selon le coût total du chemin π.

Problèmes de flots de coût minimal

- Algorithme de Busacker et Gowen
- Principe
  - On part du flot nul : débit Q = 0 et coût total CT = 0. On calcule un flot de débit F imposé, et de coût minimal.
  - Différence avec Ford-Fulkerson : les augmentations se font le long de *chaînes améliorantes de coût minimal.*
  - A chaque itération, on cherche donc un chemin π de coût minimal z, de s à t dans le graphe d'écart.

Problèmes de flots de coût minimal

initialiser Q, CT et les flux des arcs à 0*
repeat
    construire le graphe d'écart H
    calculer dans H un chemin de coût min π de s à t, soit z son coût
    if π existe **then**
        soit μ la chaîne de G correspondant à π
        calculer l'augmentation de flot δ
        $\delta := \text{Min}(\delta, F-Q)$ //Note : F limité à Q*
        ajouter δ aux flux des arcs de μ^+
        soustraire δ aux flux des arcs de μ^-
        $Q := Q + \delta$; $CT := CT + \delta.z$
    endif
until (μ non trouvée) or ($Q = F$).

Problèmes de flots de coût minimal

- Il faut utiliser l'algo de Bellman pour calculer π, à cause des coûts négatifs possibles dans H.
- Si F=constante, l'algo se termine quand Q=F grâce au plafonnement de δ. Si F=+∞, il stoppe quand il ne trouve plus de chaîne améliorante : on a alors un flot maximal et de coût minimal.
- Dans les 2 cas, le coût CT est minimal. L'optimalité de l'algo repose sur la propriété suivante, qu'on admettra :
- Propriété 4 : à chaque itération, le flot obtenu est de coût minimal parmi tous les flots de débit Q.

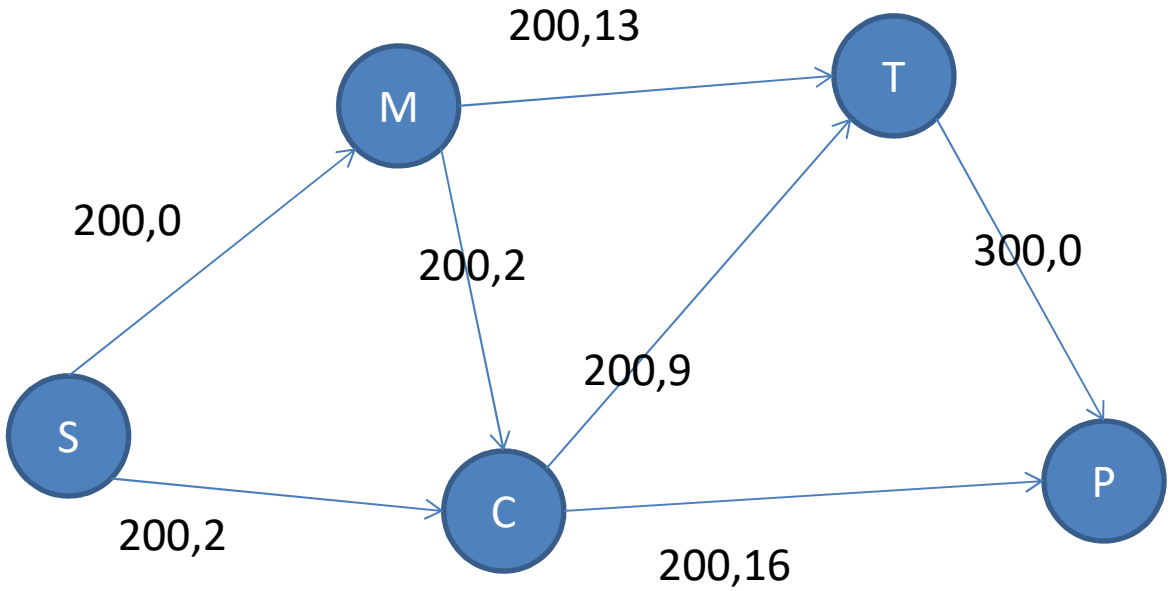
Problèmes de flots de coût minimal

b) Complexité

Soit K le maximum des capacités.

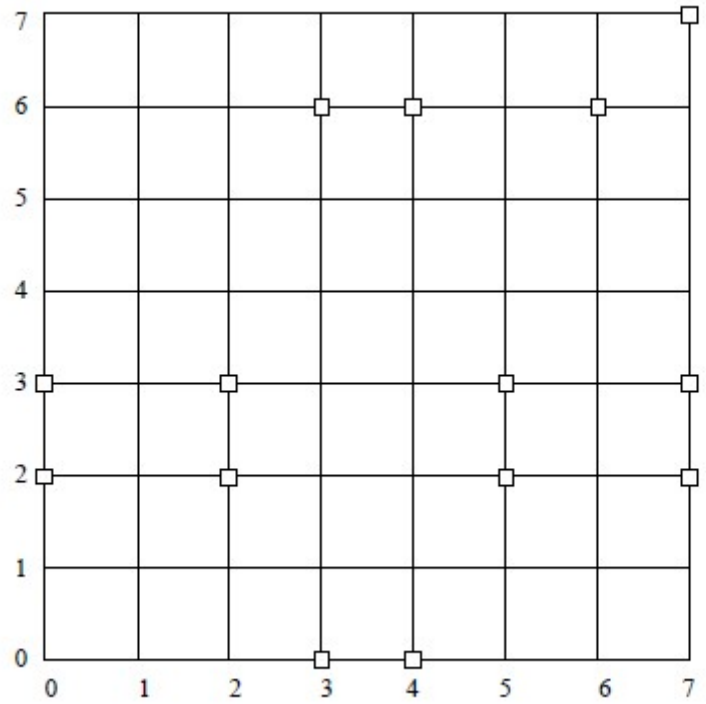
- L'algorithme de Busacker et Gowen est en $O(m.n^2.K)$, mais il est rapide en moyenne.

Exemple



Fin du chapitre

exercice

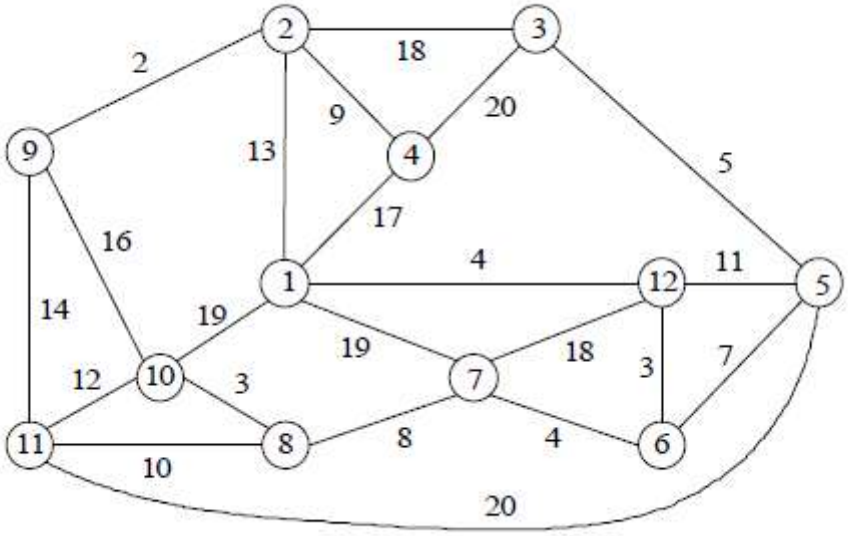


- Résoudre avec l'algorithme de PPV
- Algorithme de fletcher
- Algorithme de shamos

exercice

- L'objectif de l'exercice est de prendre conscience de la difficulté des problèmes d'optimisation combinatoire. La figure représente un réseau routier à saler en cas de gel, avec des distances en km. Chaque route peut être traitée en un seul passage et dans n'importe quel sens. L'inégalité triangulaire n'est pas vérifiée *car certaines routes ont de nombreux virages, non représentés.* Pour avoir des données très simples, on suppose que chaque route a besoin d'une tonne de sel et qu'on dispose au noeud-dépôt 1 de camions identiques de capacité 5 tonnes, en nombre non limité.

exercice


2
3
9
4
1
12
5
7
10
11
8
6
2
18
9
20
13
5
17
16
4
11
14
19
19
18
3
7
12
3
8
4
10
20

exercice

- Le but est de calculer des tournées de longueur totale minimale salant toutes les routes. Une tournée part du dépôt, traite une suite de routes dont la demande totale n'excède pas la capacité du camion, et revient au dépôt. Les routes successives traitées par un camion peuvent être non adjacentes. Une route est traitée par un seul camion et en un seul passage (pas de salage partiel). Elle peut être traversée sans saler par plusieurs camions et plusieurs fois par un même camion. En binôme, essayez de trouver la meilleure solution possible. Ce problème où on traite des demandes sur les arcs ou arêtes d'un réseau avec des véhicules de capacité limitée est appelé *CARP (Capacitated Arc Routing Problem).*

Problèmes de remplissage

- Problème du sac à dos
- Programmation en nombre entiers
  - Méthode de branch & bound
  - Méthode des coupes de gomory
  - Résolution en utilisant Cplex et lingo