Logistique de distribution et du transport

Abdellah El Fallahi

2020-2025

Contenu: modèles et méthodes pour la logistique de distribution

- Introduction
- Problèmes stratégiques de localisation
- Problèmes d'acheminement (chemins et flots)
- Problèmes de tournées de livraison et de collecte
- Gestion des moyens (véhicules, conducteurs)
- Problème de remplissage
- Bin packing problem

Chapitre 1 Introduction générale: gestion de production et logistique, généralités sur le transport

GP et logistique et transport

- Rappel rôle de la GP
  - Réunir les éléments nécessaires en entrée (amont de la production)
  - Définir un plan de production
  - Bonne utilisation des ressources de l'entreprise
  - Pour satisfaire au mieux une demande (produit ou service)
  - Contrôler: stocks, qualité, coût, délais
  - Veiller à l'entretien et l'amélioration continue de l'unité de production

GP et logistique et transport

- Sens historique: soutien des opération militaires
- Par analogie: soutiens des systèmes de production (SP)
- Synonyme: gestion des flux (physique)
  - En entrée: processus d'achats, approvisionnement, gestion des stocks de matières premières (MP) ou des composants
  - En sortie: gestion des stocks en aval (produits finis (PF)) et de distribution (transport en aval)
  - En interne, stocks intermédiaires, transitique, …

GP et logistique et transport

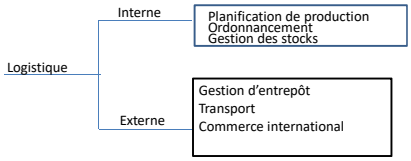
- Évolution de la logistique
  - d'abord gestion des flux physiques
  - puis des flux d'informations, avec l'informatique
  - extensions en amont et en aval : supply chain (logistique globale)
- Définition de la logistique par l'ASLOG :
  - art de mettre à disposition la bonne ressource,
  - au bon endroit, au bon moment,
  - en quantité appropriée, et au moindre coût !

rôle-clé du logisticien : vue globale du SP.

GP et logistique et transport

- Synchronisation de la production et du transport
- Etape de supply chain :
  - flux physiques entre différents sites.
  - de plus en plus sous-traité (80% des cas).
  - Prestation de services : gestion d'entrepôts, stocks avancés, emballage…
- Problèmes spécifiques :
  - regroupements, éclatements, gestion des véhicules, ...
- Existe sans la production : transport de passagers, poste, transport express de petits colis (FedEx, UPS).
- Le Coût de transport est généralement très important:
  - Et donc la nécessité d'une logistique de transport

GP et logistique et transport



Problème à résoudre : synchronisation du transport et de la production.

- Avantages:
  - Transport en amont de la production
  - Transport en aval de la production

Généralités sur le transport

Maillon élémentaire de la chaîne logistique

Composants d'un maillon de transport (ou bipoint):

- une origine,
- une destination,
- une cargaison,
- et un mode de transport.

Chaînes de maillons séparés par des ruptures de charges

- éclatement (ou : dégroupage) d'une cargaison
- regroupement (ou : groupage) de cargaisons
- transbordement (changement de mode)
- stockage (attente prolongée de la cargaison)

Généralités sur le transport

- Transport en compte propre
  - L'entreprise a sa propre flotte et ses chauffeurs.
- Transport en compte d'autrui (84% des cas) :
  - Le transport est sous-traité.

– On distingue alors:

- Le chargeur (shipper, entreprise à l'origine)
- Les transporteurs (carriers) qui assurent le transport.

Généralités sur le transport

- Modes de transport et quelques chiffres (France, 2002)

Transport national de marchandises.
Cumul sur les bipoints avec 2 extrémités en France.

Mode	*% en tonnes*	*% en t.km*
Rail	8	28
Route	90	70
Voie fluviale	2	2

Remarques:

- 80% des flots sont locaux (internes à une région)
- rail considéré intéressant à partir de 400 km

Généralités sur le transport

- Remarques (suite) :
  - moyenne 45 km (en compte propre) et 128 km (compte d'autrui) →longs trajets sous-traités
  - part négligeable de l'avion (< 1%)
  - importations/exportations): flux 10 fois plus petits !

Autres modes de transport :

- transport maritime
- transport aérien
- oléoducs et gazoducs (produits pétroliers)

Au niveau mondial, le bateau ne représente que 10% du tonnage, l'avion 2% (mais 5% en valeur)…

Généralités sur le transport

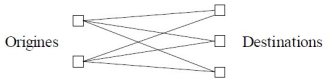
- Réseaux de transport

Un réseau de transport est formé :
  - de nœuds (points) d'enlèvement
  - de nœuds de livraison
  - de nœuds de routage (de transit) intermédiaires
  - des liaisons entre nœuds (route, rail, air etc.).

  Nœuds de transit (= avec ruptures de charges) :
  - entrepôts ou dépôts où la cargaison est stockée
  - plate-forme logistique (hub), séjour de qq heures
- Hub intermodal : avec plusieurs modes de transport, exemple inter-connexion rail-autoroute-avion.

Généralités sur le transport

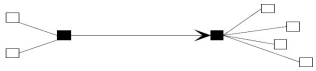
- Réseau avec liaisons directes :



- On a un véhicule par liaison.
- En anglais: truckload transportation (TL).
- Exemple :
  - livraison de carburant à des dépôts par des raffineries.

Généralités sur le transport

- Réseau arborescent (out-tree), ici avec liaisons directes :



- Appelé aussi réseau en trompette ou de distribution
  - Une origine (ou très peu)
  - Grand nombre de destinations
  - Entrepôts ou hubs intermédiaires (en noir)
  - Stockages ou éclatement près de destinations
- Exemple: approvisionnement des supermarchés

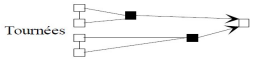
Généralité sur le transport

- Réseau arborescent avec tournées :
- Si chaque demande client << capacité d'un camion, un véhicule peut traiter plusieurs clients : tournée (trip).

- Moins de véhicules par rapport aux livraisons directes.
- En anglais, transport less than truckload (LTL).

Généralités sur le transport

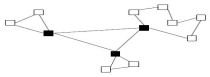
- Réseau en anti-arborescence (in-tree)



- Ou en entonnoir:
  - Plusieurs origines
  - Livraisons directes aux hubs (TL)
  - ou tournées de collecte (LTL)
  - groupage aux hubs
- Exemple: groupage des déchets industriels ou ménagers

Généralités sur le transport

- Réseau complexe ou en diabolo ou de messagerie



- Nombreux nœuds origines/destination ou les deux
- Quantité petite par bipoint, pas de stockage long
- Tournées autour des hubs (livraison ou collecte)
- Exemple:

Généralités sur le transport

- Niveaux de décision en transport
  - Stratégique ou long terme (un an ou plus),
  - Tactique ou moyen terme (quelques moins),
  - Opérationnel ou cour terme (quelques jours),
  - Temps réel ou très court (heure ou minute).
- Coûts et niveau de responsabilité décroissant.
- On va voir en 1ère lieu le niveau stratégique : problèmes de localisation (choix d'emplacement) d'usines etc.

Chapitre 2

Graphes complexité, programmation mathématique

Complexité

- Notation « grand O »
- F et g deux fonction de R^+ dans R^+, f est d'ordre g (f = O(g)) si lim(f/g) à l'infini = constante.
- Exemple $f(x) = 2x^2 + 5x$ est en $O(X^2)$
- Soit A un algorithme travaille sur des données, la complexité de A est une fonction qui exprime:
  - Le nombre d'instructions exécutées, à l'ordre près
  - Dans le pire de cas,
  - En fonction de la taille n des données

Complexité

- Exemple, tri de n nombres :
  - tri par recherche des minima successifs : $O(n^2)$,
  - algorithmes plus rapides : heapsort en $O(n.\log_2 n)$.
- Utilité de la notion de complexité :
  - évaluer / comparer des algorithmes sur le papier,
  - remplace temps de calcul (dépendant de l'ordinateur) par une mesure de croissance asymptotique,
- mesure peu critique, à l'ordre près (exemple, comptage des permutations d'entiers pour un tri).

Complexité

- Deux grands groupes d'algorithmes :
  - polynomiaux ou complexité de l'ordre d'un polynôme : $O(n^2)$, $O(n.\log n)$, $O(n^{2.5})$,
  - les autres, dits exponentiels : 2^n, $n!$, k^n, n^n, $n^{\log n}$.
- Enumération complète des solutions : exponentiel !
  - affecter n personnes à n postes n! affectations.
  - n coups aux échecs, k choix par coup → k^n séquences
- Les algorithmes exponentiels sont inexploitables pour les grands problèmes

Complexité

- Problème d'optimisation : optimiser une fonction f (fonction-objectif) sur un ensemble S. Si S fini, on a un pb d'optimisation combinatoire (POC).
- Algorithme exact ou optimal : trouve l'optimum s'il existe. Exemple : plus court chemin dans un réseau routier de n villes, calculable en $O(n^2)$.
- Beaucoup de POC sont sans algorithme exact polynomial, ils sont NP-difficiles ou NP-complets. L'énumération des solutions est possible, mais prend trop de temps.

Complexité

- Exemple de POC NP-difficile, le problème d'empilement :
- une seule bobine de 6970 m de câble, n commandes de 2463, 2597, 315, 1089, 759, 599, 1283, 1625, 2292 et 211 m
- but = maximiser la longueur de câble livrée.
- Solution :

6970 = 2597 + 315 + 759 + 1283 + 1625 + 211.

Ici, 2^{10} = 1024 sous-ensembles possibles.

Si n = 1000 et un milliard de sous-ensembles/s :

$402*10^9$ siècles !

Complexité

Comment résoudre un problème NP-difficile ?

- énumération complète (si petit problème),
- énumération « intelligente » (pb de taille moyenne)
  - Programmation dynamique (résolution de sous-problèmes emboités),
  - Méthodes arborescentes (séparation en sous-cas dont beaucoup sont éliminés par des tests).
- méthodes approchées (heuristiques) si grand pb : bonnes solutions mais sans garantie d'optimalité.

- Pbs de transport NP-complets (tournées de véhicules) : seules des heuristiques peuvent traiter les grandes dimmensions.

Exemples de calcul de la complexité 1/2

- Insert-sort(A)
  - For j:= 1 to n-1 do
    - Do k:=A[j]
      - i:=j-1
      - While i>=0 and A[i]>k
        - » Do A[i+1]:=A[i]
          - i=i-1
        - » A[i+1]:=k

Exemples de calcul de la complexité 2/2

- Merge_sort(A,p,r)
  - If p<r
  - Then q:[(p+r)/2]
    - Merge_sort(A,p,q)
    - Merge_sort(A,q+1,r)
    - Merge (A,P,q,r)

Chapitre 2

Problèmes de localisation

Objectifs pédagogiques

- Connaitre l'Importance de la localisation
- Les différents problème de localisation
- Modélisation des problèmes de localisation
- Méthodes de résolutions des problèmes de localisation
- Notions de couverture

Importance de la localisation 1/2

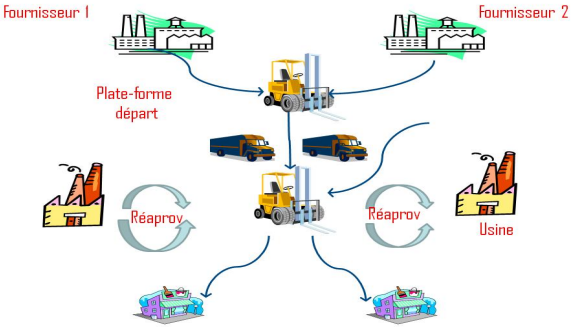
- Un réseau de transport bien conçu, impliquera une économie importante des ressources. Tout transporteur est amené à bien prendre ces décisions de localisation à cause de:
  - les marchés sont plus étendus, mais souvent avec une clientèle dispersée,
  - les livraisons à distance sont plus faciles, grâce à l'ouverture des frontières,
  - les sites possibles sont donc plus nombreux
- La localisation est un problème complexe, crucial à cause des enjeux financiers, et avec des compromis à trouver

Importance de la localisation 2/2

- Exemple de choix
  - Installation dans un pays à bas coût de main d'œuvre (Low cost contries LCC):
    - augmentation des coûts d'expédition si on est loin des zones de consommation
    - Manque de main d'œuvre qualifiée
    - …
  - Installation près des consommateurs :
    - Augmentation des coûts d'approvisionnement de la matière première exotiques (bois, minerais).
    - Main d'œuvre chère
    - …

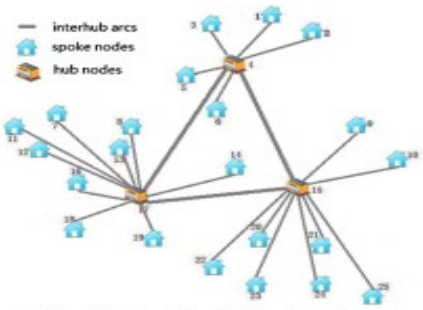
Réseaux de Transport

LE SHÉMA MULTI PLATE-FORME



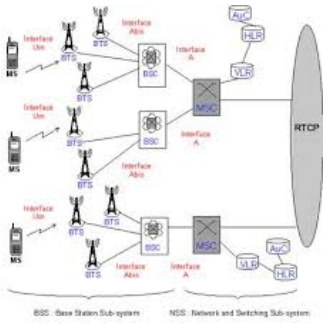
Source: Gestion des entrepôts et plates- formes. Fabrice MOCELLIN, 2ème Edition DUNOD Paris 2006

P-hubs



Optimal hubs locations in 25 nodes network

Réseaux de Télécommunication



Classification et exemples 1/8

- Objectif : trouver *l'emplacement optimal d'installations* (usines etc.) sur un ensemble de *sites possibles, pour :*
  - minimiser le nombre d'installations pour couvrir les demandes,
  - ou maximiser la demande couverte avec un nombre fixe d'installations,
  - ou minimiser les coûts (construction, fonctionnement, charges, coûts de livraison aux clients, etc).

Classification et exemples 2/8

- Problèmes à résoudre:
  - combien d'installations faut-il placer ?
  - où doit-on placer chaque installation ?
  - comment affecter les demandes aux sites créés ?
- Problèmes associés
  - Transports (placement d'entrepôts, de hubs)
  - Placement d'unités de production,
  - Logistique de secours (interventions, ambulances),
  - Télécommunications (relais TV, BTS), etc.
  - Le centres hospitaliers
  - Les écoles
  - Les centres de polices

Classification et exemples 3/8

- Critères de classification en localisation
- Localisation planaire, discrète, et sur un réseau
- *Planaire ou continue : les sites peuvent être partout* dans le plan (relais de TV).
- *Discrète : nombre fini de sites, et on connaît la matrice* des "distances" au sens large entre sites ou *distancier:* en km, en temps, en coût, etc.
- *Sur un réseau : localisation discrète sur les nœuds* d'un réseau.

Classification et exemples 4/8

- *Type de graphe : problèmes un peu plus faciles sur* certains types de réseau. Exemple: stations de pompage sur un *arbre de pipe-lines.*
- *Nombre de sites à localiser. Peut être une variable du* de décision ou une donnée.
  - Le cas particulier *d'un seul site* est facile (méthodes du barycentre).
- *Problèmes statiques ou dynamiques.*
  - *Pour les problèmes* dynamiques:
    - les paramètres et les variables évoluent dans le temps.

Classification et exemples 5/8

- *Problèmes déterministes ou probabilistes.*
  - *Paramètres* connus (demande connues, …)
  - Paramètres variables dans le cas probabiliste (modèles probabilistes, plus complexes).
- *Problèmes mono ou multi-produits. Quand la nature* des produits n'est pas importante (transport en vrac), on se ramène à un produit fictif pour simplifier: palettes, m^3 de liquide…
- *Secteur privé ou public.*
  - *Le privé maximise le profit ou* minimise le coût, et la même entité paie les coûts d'installation puis de fonctionnement.
  - Secteur public: facilité pour l'acquisition du terrain

Classification et exemples 6/8

- Public : notion de service plus importante. L'entité ne paie souvent que l'installation ou le fonctionnement.
- *Objectifs simples ou multiples. Si objectifs antagonistes,* il faut souvent une approche *d'optimisation multicritère.*
- *Capacité finie ou infinie. La capacité d'un site est le plus* souvent *limitée. Parfois elle est « infinie » (= suffisante):* un relais de TV couvre tous les habitants de sa zone.

Classification et exemples 7/8

- *Affectation des demandes. Chaque demande est traitée* par un site (*assignment) ou par plusieurs (allocation).* Second cas fréquent si sites de capacité finie
- *Problèmes à un ou plusieurs niveaux.*
  - *Exemple : réseaux* de distribution à 1 étage (usines → clients) ou 2 (usines→ entrepôts (hub)→ clients).
- *Installations indésirables. Placement des installations* près des clients, sauf les "indésirables« .
  - Décharge loin d'une ville → coût de transport élevé
    - d'où choix entre le coût de transport et nb de personnes affectées par la décharge
    - La recherche d'un compromis entre plusieurs objectifs
      - Optimisation multi-objective

Classification et exemples 8/8

- *Interactions entre sites*
  - Deux sites intéressants pour un hypermarché peuvent desservir des zones en commun:
    - ouvrir les 2 sites en même temps n'est pas souhaitable.
    - Problème du choix entre les deux sites
  - Les interactions entre sites voisins compliquent beaucoup les problèmes.
    - BTS (problème d'interférence)

Localisation d'une installation 1/3

- Scores charge-distance (load-distance scores) Données
  - *m sites possibles pour une nouvelle installation,*
  - *n entités (clients, fournisseurs, etc.) à desservir,*
  - mesure simple de « *charge » Lj pour chaque entité j.* A définir :
    - tonnage entrant ou sortant,
    - nombre de déplacements/semaine,
    - patients venant consulter, etc.
  - distances d_{ij} *entre tout site i et toute entité j.*
- Objectif: choisir un site *i* minimisant un score estimant* le travail de déplacement des charges.

Localisation d'une installation 2/3

- Méthode 1:
  - calcul pour tout site *i d'un score charge-distance ld(i),*
  - puis choix du site de score minimal.

$$ld(i) = \sum_{j=1,n} L_j . d_{ij}$$

  - *Exemple : Localisation d'un dispensaire dans une ville.*
  - entités : quartiers.
  - charges: nombre d'habitants (patients potentiels).
  - distances : du centre des secteurs aux sites possibles.

Localisation d'une installation 3/3

- Méthode 2: le Barycentre

Pour la localisation *continue (= non discrète).*

- on connaît les coordonnées x_j *et* y_j *de l'entité j.*
- on cherche les coordonnées *(x*,y*) de l'installation.*
- choix évident : *centre de gravité des charges*

$$x^* = \frac{\sum_{j=1,n} L_j . x_j}{\sum_{j=1,n} L_j}, \quad y^* = \frac{\sum_{j=1,n} L_j . y_j}{\sum_{j=1,n} L_j}$$

* Le lieu obtenu doit souvent être ajusté.

* La localisation planaire de plusieurs installations est plus difficile (PNL).

Exemple

- Une ville souhaite installer un nouveau dispensaire pour desservir 7 quartiers dont les informations sont présentées par le tableau suivant: avec des coordonnées en km et le nombre d'habitants en mil habitants.
- Le quadrillage des rues des villes US impose l'utilisation de la distance "Manhattan

Quartier	Coordonnées (x,y)	Population
A	(2.5,4.5)	2
B	(2.5,2.5)	5
C	(5.5,4.5)	10
D	(5,2)	7
E	(8,5)	10
F	(7,2)	20
G	(9,2.5)	14

- Deux terrains sont disponibles au centre des quartiers C et F. Déterminer le meilleur site en utilisant des scores charge × distance en 10^3 habitants × km et la norme L1. Comparer ensuite avec une localisation "continue" basée sur le centre de gravité (faire dès le début une représentation graphique).

Solution

Quartier j	Coordonnées (x_j, y_j)	Population L_j	Site en C = (5.5,4.5)		Site en F = (7,2)		$L_j.x_j$	$L_j.y_j$
			Distance d_{ij}	Score $l_i.d_{ij}$	Distance d_{ij}	Score $l_i.d_{ij}$		
A	(2.5,4.5)	2	3	6	7	14	5	9
B	(2.5,2.5)	5	5	25	5	25	12.5	12.5
C	(5.5,4.5)	10	0	0	4	40	55	45
D	(5,2)	7	3	21	2	14	35	14
E	(8,5)	10	3	30	4	40	80	50
F	(7,2)	20	4	80	0	0	140	40
G	(9,2.5)	14	5.5	77	2.5	35	126	35
		68		239		168	453.5	205.5

Problèmes de couverture

Couverture Totale

Couverture Maximale

Problèmes de couverture 1/21

- Problème de couverture totale
- Données :
  - *m points de demande ou "clients" (indexés par i)*
  - *n sites potentiels (indexés par j)*
  - *D_c distance de couverture (distance max de service)*
  - booléens *a_{ij} =1 ssi i est couvert par le site j (distance ≤ D_c).*
  - *c_j coût d'ouverture du site j.*

  Objectif: déterminer les sites à ouvrir pour couvrir *toutes les demandes, avec un coût total minimal.*

Problèmes de couverture 2/21

- NP-difficile. Formulable par un PL en 0-1 :

$$(1)\ Min \sum_{j=1,n} c_j x_j$$

$$(2)\ \forall i = 1...m : \sum_{j=1,n} a_{ij} x_j \geq 1$$

$$(3)\ \forall j = 1...n : x_j \in \{0,1\}$$

$$(4)\ ...$$

  - (3) X_j les variables de décisions (0-1, 1 si le site *j* est ouvert)
  - (1) la fonction-objectif à minimiser (coût des sites ouverts)
  - (2) contraintes, tout client *i* couvert par au moins un site

Problèmes de couverture 3/21

- Ecriture matricielle:
  - *A : matrice binaire m×n des a_{ij}*
  - 1I : vecteur formé de *m "1"*.
    - Min *c.x*
    - *A.x ≥ 1I*
    - *x ∈ {0,1}*.
- Explication : une colonne *j de A est un m-vecteur binaire* codant le sous-ensemble des clients couverts par le site *j.*
  - Il faut choisir un sous-ensemble de colonnes *j (x_j = 1)* dans lequel chaque client apparaît au moins une fois.

Problèmes de couverture 4/21

- Techniques de réduction de taille :
  - soit *L_i la ligne i de A et C_j la colonne j.*
  - par convention, *$C_k \geq C_j \Leftrightarrow a_{ik} \geq a_{ij}$ pour tout i.*
  - *un site k domine un site j si $c_k \leq c_j$ et $C_k \geq C_j$*
  - *k couvre tous les clients de j sans coûter plus cher!*
  - donc on peut éliminer la colonne *j et forcer x_j à prendre la valeur 0.*

A	1	...	*k*	...	*j*	...	*n*
1			0		0		
2			1		1		
3			1		0		
...			0		0		
m			1		1		

Problèmes de couverture 5/21

- Techniques de réduction (suite) :

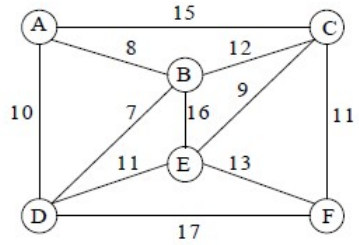
Moins évident : *un client k domine un client i ssi $L_k \leq L_i$.* Tout site *j qui couvre k (a_{kj} = 1) couvre aussi i.* Si la contrainte (2) pour *k est vérifiée, celle pour i aussi.* On peut donc supprimer la ligne (contrainte) *i de A.*

A	1	2	3		...		*n*
1							
k	0	0	1	0	0	1	1
i	1	0	1	1	0	1	1
...							
m							

Problèmes de couverture 6/21

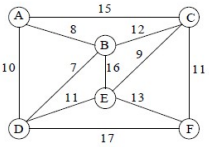
- **Techniques de réduction (suite)**
- Ouverture évidente
- Si une ligne *i de A a un seul 1, en a_{ij}, alors seul le site j* couvre le client *i. On peut forcer x_j à prendre la valeur 1 et supprimer toute* autre contrainte *k satisfaite (telle que a_{kj} = 1).*

- **Utilisation des réductions**
  - teste des lignes et colonnes en ordre quelconque
  - si on trouve des réductions, on doit tout retester
  - en effet, une réduction peut en créer d'autres
  - stop quand on ne trouve plus de réductions.



Problèmes de couverture 7/21

- Exemple avec *m = n = 6 et* $D_c = 12$ *:*



$$\begin{array}{lll} \text{Min } X_A + X_B + X_C + X_D + X_E + X_F & & \\ (1)\ X_A + X_B \quad + X_D & & \geq 1 \\ (2)\ X_A + X_B + X_C + X_D & & \geq 1 \\ (3)\ \quad X_B + X_C \quad + X_E + X_F & & \geq 1 \\ (4)\ X_A + X_B \quad + X_D + X_E & & \geq 1 \\ (5)\ \quad X_C + X_D + X_E & & \geq 1 \\ (6)\ \quad X_C \quad + X_F & & \geq 1 \\ X_A, X_B, X_C, X_D, X_E, X_F \in \{0,1\} & & \end{array}$$

Problèmes de couverture 8/21

- Réductions :
  - Site A dominé par B et F par C : on force $X_A = X_F = 0$.
  - Seul C couvre F : $X_C = 1$ et suppression des contraintes satisfaites (2),(3),(5),(6).
  - (1) et (4), si A couvert alors D aussi : supprimer (4).
- Après réduction, le PL devient :

$$\begin{array}{l} \text{Min } X_B + X_D + 1 \\ X_B + X_D \geq 1 \\ X_B, X_D \in \{0,1\} \end{array}$$

- 2 optima à deux sites ouverts: $X_b = X_c = 1$ ou $X_c = X_d = 1$

Couverture Maximale

Problèmes de couverture 9/21

- Problème de couverture maximale
- Pb de couverture totale parfois inadapté :
  - trop de sites à ouvrir
  - clients de même poids, même si demandes faibles.
- Couverture maximale : maximiser la demande totale couverte, avec un nombre limité de sites ouverts.
- NP-difficile*. Paramètres et variables additionnels :
  - *p nombre max. de sites à ouvrir (donné),*
  - *d_i demande du point i (donnée)*
  - *z_i variable 0-1 indiquant si le client i est couvert ou non.*

Problèmes de couverture 10/21

- On a encore une formulation sous forme de PL en 0-1:

$$(1)\quad Max \sum_{i=1,m} d_i z_i$$

$$(2)\quad \forall i = 1 \ldots m : z_i \leq \sum_{j=1,n} a_{ij} x_j$$

$$(3)\quad \sum_{j=1,n} x_j \leq p$$

$$(4)\quad \forall j = 1 \ldots n : x_j \in \{0,1\}$$

$$(5)\quad \forall i = 1 \ldots m : z_i \in \{0,1\}$$

Problèmes de couverture 11/21

- équation (1): demande totale satisfaite, à maximiser.
- Contraintes (2) *: comment régler les z_i ?*
- Par définition, *z_i = 1 ⇔ client i couvert :*

$$z_i = 1 \Leftrightarrow \sum_{j=1,n} a_{ij} x_j > 0$$

- *Contrainte (3): ouverture d'au plus p sites, on peut avoir des clients non couverts*
- Si moins de *p sites suffisent, la fonction-objectif sera* égale à la demande totale.

Problèmes de couverture 12/21

- Résolution des problèmes de couverture
- Avec un logiciel de programmation linéaire
  - PL en 0-1 donc en nombres entiers
  - algorithme du simplexe inutilisable.
  - méthodes arborescentes (*xj = 0 ou xj = 1)+réductions*
  - exemple de problème gérable : 100 clients, 20 sites.
- Note : les problèmes de partitionnement (*A.x = 1) sont* encore plus durs car pas toujours faisables. Exemple : découpage électoral, secteurs commerciaux.

Problèmes de couverture 13/21

- Heuristiques gloutonnes

PL trop longs à résoudre pour les grands problèmes. Des heuristiques sont alors nécessaires.

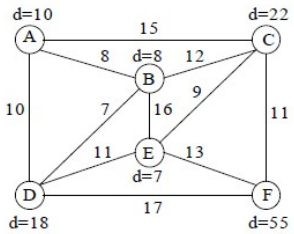
- Heuristiques gloutonnes (les plus simples)
  - suite de décisions définitives (sans retours en arrière)
  - choix le plus avantageux à chaque étape
  - selon un certain critère (à définir)
  - exemple, Plus Proche Voisin pour le TSP.

Problèmes de couverture 14/21

- Heuristique gloutonne de Chvàtal (couverture totale)
- Ouvrir le site de + faible coût par nouveau client non couvert
- *Cout := 0 // coût total des sites ouverts*
- *Ouverts := Ø // ensemble des sites ouverts*
- *Repeat*
- *j := site libre de coût moyen minimal par nouveaux points couvert : c(j) / nb de nouveaux points couverts.*
- *Ajouter j à Ouverts*
- *Cout := Cout + c(j)*
- *enlever du pb le site j et les nouveaux points couverts*
- *Until (couverture complète).*

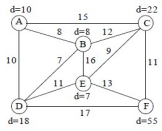
Problèmes de couverture 15/21

- Heuristique analogue pour la *couverture maximale :*
- site augmentant le plus la demande totale couverte.
- *Ouverts := Ø // ensemble des sites ouverts*
- *NbOuverts := 0 // nombre de sites ouverts*
- *QteCouverte := 0 // quantité totale couverte*
- *Repeat*
- *j := site libre augmentant QteCouverte au maximum*
- *NbOuverts := NbOuverts + 1*
- *Ajouter j à Ouverts*
- *QteCouverte := QteCouverte +somme des nouvelles demandes couvertes*
- *Until (NbOuverts = p) ou (toutes les demandes sont satisfaites).*
- *p insuffisant : NbOuverts = p et QteCouverte < demande totale.*
- *p suffisant : NbOuverts < p, QteCouverte = demande totale.*



Problèmes de couverture 16/21

- Exemple d'application



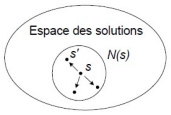
Site choisi	Clients couverts	Qté couverte si ouvert en 1er	Couverte si ouvert après C
A	A, B, D	36	36
B	A, B, D	36	36
C	C, E, F	84	déjà ouvert
D	A, B, D, E	43	36
E	C,D,E	47	18
F	C,F	77	0

Problèmes de couverture 17/21

- Recherches locales (*local search)*
- Principes :
  - heuristiques pour l'optimisation combinatoire
  - amélioration progressive d'une solution initiale *S,* par exemple donnée par une heuristique gloutonne
  - basées sur un "petit" sous-ensemble de solutions *V(S),* appelé voisinage de *S (neighborhood, N(S)).*
  - *V(S) est défini implicitement par une* transformation simple de *S.*

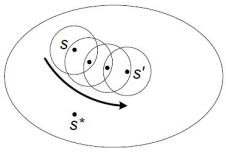
Problèmes de couverture 18/21

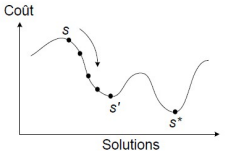
- Chercher une solution initiale S
- **Repeat**
  - Construire un voisinage N(S)
  - chercher une meilleure solution *S'* dans *N(S)*
  - si *S' trouvée alors S := S'*
- **Until *S' non trouvée***
- Deux versions sont possibles pour le choix de la meilleure solution. A chaque itération on prend :
  - soit la 1ère solution *S' trouvée qui améliore S*
  - soit la meilleure solution *S' du voisinage.*
- *S optimum local pour le type de voisinage choisi*



Problèmes de couverture 19/21

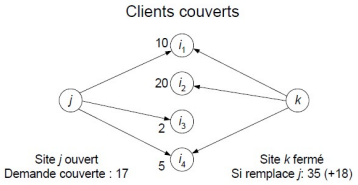
- 2 vues imagées, solution *S initiale, S' finale, S* optimale*





Problèmes de couverture 20/21

- Exemple de voisinage pour la couverture maximale :
- tester toutes les façons de remplacer un site ouvert par un autre actuellement fermé, *p.(n-p) solutions à tester.*



Exercice

- On considère le problème de couverture totale défini par une matrice binaire quelconque *A*, *m*×*n*. m points de demandes (clients), n sites potentiels.
- a) Rappeler très brièvement en termes d'ensembles la signification des lignes et colonnes de *A*. A quelle condition existe-t-il des solutions?
- b) En supposant le problème réalisable, que signifie le fait que la somme d'une ligne *i* soit égale à un entier $k \leq n$? Que la somme d'une colonne *j* soit égale à un entier $k \leq m$? Que peut-on conclure si une colonne *j* ne contient pas de zéros? Si toute paire de colonnes n'a aucune ligne avec deux 1?
- c) Simplifier au maximum la matrice suivante en précisant les éliminations (quelle ligne ou colonne est dominée par quelle autre). Puis résoudre optimalement le problème en précisant les sites ouverts et les sites qui couvrent chaque client.

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

Exercice 3

- Pendant la conception de la ville de Tamasna, les responsables ont prévu qu'elle aura M quartiers. Le nombre d'enfants estimé pour chaque quartier i est donné par hi, et on a prévu l'ouverture de n écoles au maximum dans cette ville pour scolariser ses enfants (n<=m). La distance entre deux quartiers i et j est donnée par dij. Une école ouverte ne peut couvrir qu'au maximum k quartiers dont le quartier qui l'abrite. Les élèves d'un quartier doivent être affectés à la même école. Pour cette première version on considère qu'on n'a pas de contrainte concernant la distance de couverture.
- **Questions :**
  - Modéliser le problème de localisation des écoles sous la forme d'un programme linéaire
  - On suppose maintenant que M=6, n=6 et k=3. Les autres données du problème sont présentées dans le tableau 1. Modéliser ce problème sous la forme d'un problème de couverture totale. Résoudre ce problème en utilisant la méthode de chvatal, la distance de couverture est de 12.

Exercice 3

	Q1	Q2	Q 3	Q4	Q5	Q6
Q1	0	8	15	10	24	27
Q2		0	12	7	16	33
Q3			0	19	9	11
Q4				0	11	17
Q5					0	13
Q6						0
Nb élèves	120	135	95	80	105	115
Coût d'installation	8	7	8.5	6	9	5.5

P-centres

P-centre

- L'objectif de ce problème linéaire est de minimiser la distance maximale entre un client et le centre auquel il est rattaché, et non plus la somme des distances. Les contraintes restent cependant les mêmes. Dans la formulation de ce problème, il n'y a alors que la fonction à minimiser qui change. La distance maximale à minimiser est

$$z \geq \sum d_{ij} x_{ij} \qquad \forall i$$

P-centre

- On considère que tous les clients sont accessibles (pas de distance de couverture) mais on tient compte des distances.
- Problème des p-centres (NP-difficile)
- Fréquents en logistique de secours. Données :
  - *m clients à servir (indice i), sans distance de couverture,*
  - *n sites (indice j) mais on peut ouvrir p sites au maximum,*
  - distances d_{ij} *au sens large entre nœuds et sites.*
- Objectif
- Placer les sites en minimisant la distance maximale aux clients.
- Exemple : placer 4 casernes de pompiers dans une ville pour minimiser le temps d'intervention sur un incendie.

P-centres

(1) Min z

(2) $\forall i = 1 \ldots m : \sum_{j=1,n} y_{ij} = 1$

(3) $\sum_{j=1,n} x_j = p$

(4) $\forall i = 1 \ldots m, \forall j = 1 \ldots n : y_{ij} \leq x_j$

(5) $\forall i = 1 \ldots m : z \geq \sum_{j=1,n} d_{ij} . y_{ij}$

(6) $\forall j = 1 \ldots n : x_j \in \{0,1\}$

(7) $\forall i = 1 \ldots m, \forall j = 1 \ldots n : y_{ij} \in \{0,1\}$

(8) $z \geq 0$

- PL dit "mixte" : variables entières et réelles.
- Variables de décision :
  - *xj =1 si site j choisi,*
  - *yij =1 si i affecté au site j.*
- Variable réelle *z :*
  - distance maxi d'intervention.

p-médians 3/7

- PL plus compliqué que les précédents !
- Facile : (1) objectif à minimiser, (2) chaque nœud est affecté à un site, (3) *p sites ouverts, (6)(7)(8) définition des variables.*
- Chaque contrainte (4) traduit l'implication : si le site *j fermé, aucun client ne peut lui être affecté.*
- Contraintes (5) minimise la distance maximale d'intervention :
  - calcul du maximum des *dij tels que yij = 1*.*
  - on *linéarise en bornant les distances d'intervention par z.*
  - cette borne *z va être compressée par la minimisation.*
  - à l'optimum, z sera égal à la distance d'intervention maximale.

p-médians 4/7

- Le problème p-médian a pour objectif de minimiser la somme totale des coûts de distribution entre les clients et leur site de rattachement (le coût totale du transport). C'est alors un problème linéaire et pour formuler ce problème, il faut tout d'abord expliciter les contraintes sous forme d'équations ou inéquations. Ici sont explicitées des cas généraux.
  - On veut que chaque client i soit affecté à un seul site j
  - Tout client i affecté ne peut l'être que si le site j existe et est ouvert $x_{ij} <= y_j$
  - Ensuite, on verifie l'ouverture exacte de p sites
  - En fin, (contraintes d'intégrité) x et y doivent être des entiers

p-médians

(1) $Min \sum_{i=1,m} \sum_{j=1,n} q_i . d_{ij} . y_{ij}$

(2) $\forall i = 1 \ldots m : \sum_{j=1,n} y_{ij} = 1$

(3) $\sum_{j=1,n} x_j = p$

(4) $\forall i = 1 \ldots m, \forall j = 1 \ldots n : y_{ij} \leq x_j$

(5) $\forall j = 1 \ldots n : x_j \in \{0,1\}$

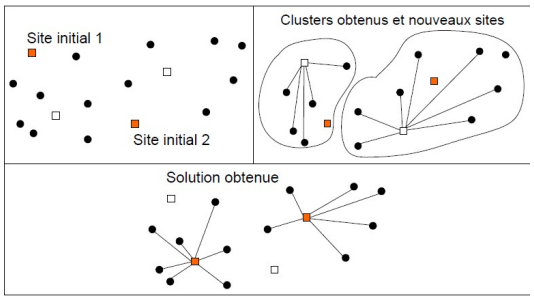
(6) $\forall i = 1 \ldots m, \forall j = 1 \ldots n : y_{ij} \in \{0,1\}$

- **PL en 0-1 plus simple :**
  - mêmes variables binaires
  - (1) distance moyenne
  - (2) clients affectés à un site
  - (3) nombre de sites à ouvrir
  - (4) pas de clients pour un site fermé
  - qi: la quantité demandée par le client i
  - dij: le côut de transport unitaire entre le centre j et le client i
- **Note :**
- La méthode des scores charge distance traite le cas *p = 1.*

P-centres et p-médians 6/7

- Une heuristique très efficace valable aussi dans le cas continu :
  - *ouvrir p sites au hasard*
  - *REPEAT*
  - *affecter chaque client au plus proche site ouvert*
  - *on obtient des clusters (un site + ses clients)*
  - *calculer le meilleur site dans chaque cluster*
  - *UNTIL l'ensemble des sites ne change pas.*
- Le calcul du meilleur site dans chaque cluster se fait :
  - avec la méthode des scores, si la liste de sites est finie
  - en calculant un centre de gravité, si localisation continue.

P-centres et p-médians 7/7



Pb de localisation d'entrepôts 1/4

- En anglais *location-allocation problems :*
  - déterminer quels entrepôts ouvrir (*locate).*
  - allouer les clients aux entrepôts (*allocate).*
  - un client peut être livré par plusieurs entrepôts !
  - minimiser coût des entrepôts + coûts de transport.
- Cas avec capacités d'entrepôts "infinies" et "un seul" produit ou *single-commodity, uncapacitated facility location problem :*
- *m sites (indice i) et n clients (indice j)*
  - f_i *coût fixe d'ouverture du site i*
  - d_j *demande du client j,* c_{ij} *coût unitaire de transport de i à j*
  - y_i *variable binaire d'ouverture*
  - x_{ij} *quantité livrée par le site i au client j.*

Pb de localisation d'entrepôts 2/4

$$(1)\quad \text{Min} \sum_{i=1,m} f_i . y_i + \sum_{i=1,m} \sum_{j=1,n} c_{ij} . x_{ij}$$

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

$$(3)\quad \forall i = 1 \ldots m : \sum_{j=1,n} x_{ij} \leq M \cdot y_i$$

$$(4)\quad \forall i = 1 \ldots m : y_i \in \{0,1\}$$

$$(5)\quad \forall i = 1 \ldots m, \forall j = 1 \ldots n : x_{ij} \geq 0$$

- *PL mixte :*
- (1) coût total, à minimiser
- (2) demandes satisfaites
- (3) : *M grande constante > 0.*
- Un dépôt fermé ne livre rien.
- Un dépôt ouvert peut ne rien livrer mais l'optimisation va le fermer!

Pb de localisation d'entrepôts 3/4

- Problème stratégique (long terme) et macroscopique (agrégé) :
  - Un "client" peut être une ville avec une demande annuelle agrégée et estimée.
  - Mélange de coûts ponctuels d'ouverture et coûts de transport quotidiens? En fait *fi peut être un coût d'exploitation quotidien.*
  - Sites de capacité infinie? Les sites peuvent être des terrains libres : on construira les sites ouverts en fonction des quantités livrées dans la solution. Borner la capacité des sites*.
  - Un seul produit? En fait, on veut dire que les produits sont *indiscernables pour le transport (palettes, containers).*

Pb de localisation d'entrepôts 4/4

- Grâce aux capacités infinies, chaque client peut toujours être *affecté à un seul site (le moins coûteux).*
- On peut étendre le modèle à des *capacités limitées :* certains clients doivent alors être livrés à partir de *plusieurs sites.*
- On peut aussi généraliser à *plusieurs produits.*

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

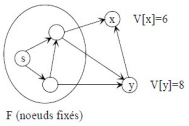
- Algorithme à fixation de labels de Dijkstra

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.*

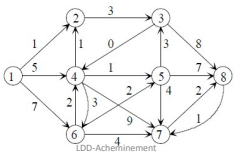
Chemins optimaux 10/25

- 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∈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

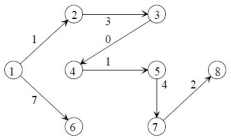
- *Complexité. Le 1er for a n itérations. Il contient un for de n itérations et un for d'au plus n itérations (car x a au* plus *n successeurs) : l'algo est en* $O(n^2)$.
- Il existe une version en *O(m.log2 n), basée sur des* structures de données complexes :
  - si *G est complet (m = n2), cette version 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 :*

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

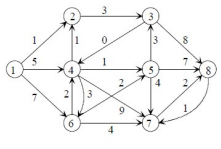
Chemins optimaux 21/25

- 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

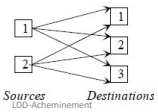
- A comparer avec *O(n^2) pour Dijkstra.*
- En pratique, l'algo est rapide : par exemple *m ≈ 4n pour* un graphe routier. Les labels sont souvent stabilisés en *k < n-1 itérations (k = 1 si réseau en étoile !).*
- Pour détecter un éventuel *circuit absorbant, il faut* remplacer le *until par until (q = 0) or (k = n). A la fin, si k = n mais q ≠ 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.

Algorithme de Balas et Hammer

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

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

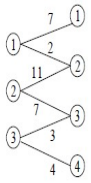
	D1	D2	D3	D4	D5	Offre
O1	7	12	1	5	9	12
O2	15	3	12	6	14	11
03	8	16	10	12	7	14
O4	18	8	17	11	16	8
Demande	10	11	15	5	4	

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

Algorithme du Stepping-stone

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

- 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

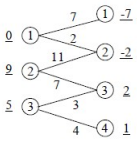
- Dans l'arbre précédent, si x_{21} augmente de 0 à 1 :
  - pour respecter disponibilités et demandes, le flux doit diminuer sur (1,1) et (2,2) et augmenter sur (1,2).
  - la variation de coût est appelée coût marginal de (2,1), elle vaut D_{21} = 6-7+2-11 = -10.
  - l'augmentation de x_{21} est rentable 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:

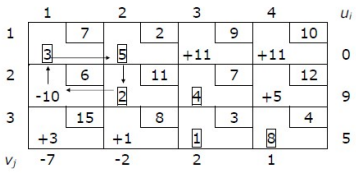


- Coût marginal si on utilise (i,j) : $D_{ij} = v_j - u_i + c_{ij}$. Exemple, on retrouve $D_{21} = -10 = v_1 - u_2 + c_{21} = -7-9+6$. Sans potentiels, combien faut-il d'opérations pour D_{31} ?

Le problème de transport 11/16

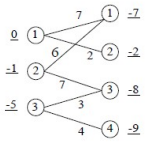
- Calculs du stepping-stone dans un tableau m xn.
- Détail d'une itération :
  - rappeler des coûts en haut et à droite de chaque case
  - écrire les flux ≠0 de l'arbre actuel (encadrés)
  - calculer les potentiels u_i et v_j
  - calculer les coûts marginaux dans les cases à flux nuls
  - si aucun n'est 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



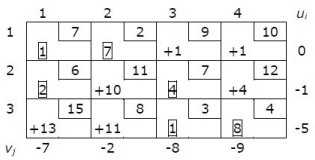
- Le seul arc avec gain est (2,1), coût marginal -10. Le flux augmente sur (2,1) et (1,2), et diminue sur (1,1) et (2,2). x_{21} peut passer de 0 à 2 et x_{22} s'annule.

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



- 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, O_3, O_4 et 5 destinations D_1, D_2, D_3, D_4, D_5. Chaque

	D1	D2	D3	D4	D5	Offre
O1	7	12	1	5	9	12
O2	15	3	12	6	14	11
O3	8	16	10	12	7	14
O4	18	8	17	11	16	8
Demande	10	11	15	5	4	

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

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

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

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

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

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

Min-cost Max flow

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

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

- conservant le flot en chaque nœud (Kirchhoff)

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

- 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).

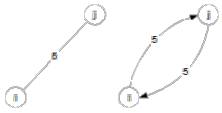
Problème du flot maximum 4/23

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

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

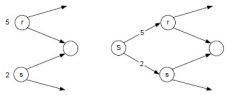
Problème du flot maximum 5/23

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



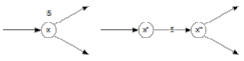
Problème du flot maximum 6/23

- 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 +∞ 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 :

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

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

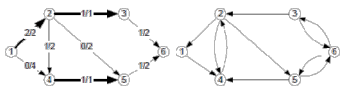
$$\forall (S,T),\ \sum_{\substack{(i,j)\in U \\ i\in S, j\in T}} \Phi_{ij} = F + \sum_{\substack{(j,i)\in U \\ j\in T, i\in S-\{s\}}} \Phi_{ji} \quad \text{ou}$$

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

- 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.

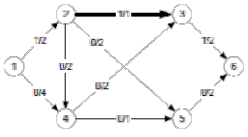
Problème du flot maximum 19/23

- 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

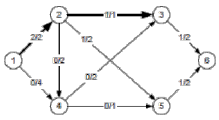
d) Un exemple

- 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 :



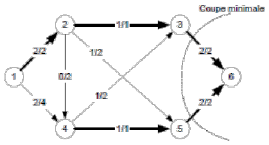
Problème du flot maximum 21/23

- 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.



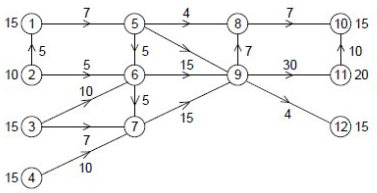
Problème du flot maximum 23/23

- 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.

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

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.

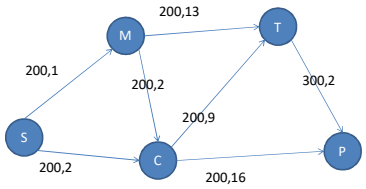
Problèmes de flots de coût minimal

b) Complexité

Soit K le maximum des capacités.

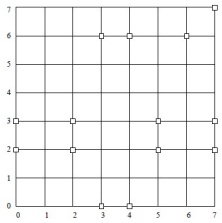
- L'algorithme de Busacker et Gowen est en $O(m.n^2.K)$, mais il est rapide en moyenne.

Exemple



Fin du chapitre

exercice

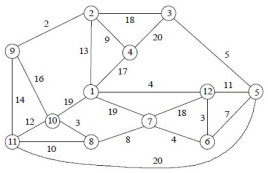


- Résoudre avec l'algorithme de PPV
- Algorithme de fletcher
- Algorithme de shamos

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



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).*

Chapitre 4
Problèmes de tournées

1. Introduction 1/3

- Données d'un problème de base en tournées
  - un réseau G = (X,U,C), C_{ij} est souvent un temps
  - un nœud dépôt s dans X
  - une flotte de v véhicules de capacité W
  - un sous-ensemble A de X de n nœuds-clients
  - demande connue q_i pour chaque client i

1. Introduction 2/3

b) Objectif
- construire des tournées de coût total minimal.
- tournée = trajet d'un véhicule qui part du dépôt, traite certains clients et revient au dépôt.
- hypothèse : tout client est traité par une seule tournée et en une seule visite dans cette tournée.

c) Contraction des problèmes

En pratique, on contracte ces problèmes :
- ensemble S de n+1 nœuds : dépôt (index 0) et clients indexés de 1 à n
- distancier $D_{(n+1)x(n+1)}$ entre ces nœuds.

1. Introduction 3/3

- Revient à définir un graphe complet H = (S, SxS, D) sur ces nœuds : on peut aller de tout nœud i (client ou dépôt) à tout nœud j avec un coût D_{ij} correspondant au plus court chemin de i à j.

Avantages de la contraction :

- réduction de taille : exemple, réseau à 5000 nœuds mais 500 clients connus et 100 à livrer un jour donné.
- gain de temps : chemins optimaux précalculés.

2. Le Pb du Voyageur de Commerce

- Le PVC (ou TSP, Traveling Salesman Problem) est le pb de tournées le plus simple : la somme des demandes n'excède pas W, on a une seule tournée.
- Toute tournée est donc une suite T = (T_1, T_2, ..., T_n) de clients, de coût total :

$$C(T) = D(o, T_1) + \sum_{i=1}^{n-1} D(T_i, T_{i+1}) + D(T_n, 0)$$

- Il y a O(n!) ordres possibles.
- Le PVC est NP-difficile, même si S est un ensemble de points du plan et D la distance euclidienne (PVC euclidien).

2. Le PVC

2.1 Méthodes optimales

Elles peuvent résoudre de petits problèmes :

- méthodes arborescentes (Little, Carpanetto)

2.2 Bornes inférieures (BI) du coût optimal

- Pour évaluer les heuristiques.
- Parfois optimales.

2. Le PVC

a) PVC "orienté" (asymmétrique)

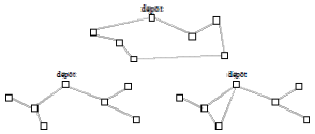
- La solution du PL d'affectation donne une BI.
- Si elle n'a aucun sous-cycle, c'est une solution optimale du PVC.

b) PVC "non orienté" (symétrique)

- Un arbre est un graphe connexe sans cycle.
- Un arbre recouvrant d'un graphe H est formé d'arêtes de H et relie tous les nœuds de H.

2. Le PVC

- BI de l'arbre recouvrant de coût minimal (ARCM) A*:



Tournée optimale(haut),
ARCM A* (gauche) et 1-arbre optimal B* (droite)

2. Le PVC

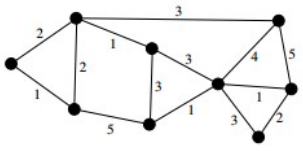
En effet, ôtons une arête à une tournée
optimale T*, on obtient un arbre recouvrant A particulier (non ramifié), et C(A*) <= C(A) <= C(T*).
Amélioration de la borne de l'arbre : les 1-arbres

- Un 1-arbre est un arbre recouvrant auquel on ajoute une arête incidente au dépôt, ce qui crée un seul cycle.
- Un 1-arbre B* de coût minimal donne une borne inférieure, car toute tournée est un 1-arbre particulier, avec tous les nœuds sur l'unique cycle.

2. Le PVC

- L'algorithme de Prim calcule un arbre recouvrant A* de coût minimal.
- On obtient un 1-arbre optimal B* en ajoutant à A* la plus petite arête incidente au dépôt.
- **Algorithme de Prim**
  - partir d'un arbre A réduit au dépôt
  - cout := 0
  - for count := 1 to nc do
    - chercher la plus petite arête (i,j) telle que i€A et j€A
    - ajouter (i,j) à A
    - cout := cout + D(i,j)
  - endfor

Exercice d'application de l'algorithme de Prim



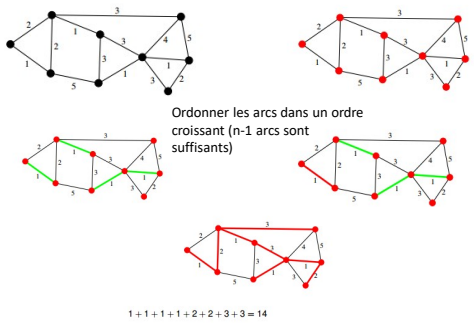
La méthode de Kruskal

- On part d'une forêt d'arbres constitués de chacun des sommets isolés du graphe.
- À chaque itération, on ajoute à cette forêt l'arête de poids le plus faible ne créant pas de cycle avec les arêtes déjà choisies.
- On stoppe quand on a examiné toutes les arêtes

Algorithme de Kruskal

- Initialiser T avec
  - sommets : tous les sommets de G
  - arêtes : aucune
- Traiter les arêtes de G l'une après l'autre par poids croissant :
  - Si une arête permet de connecter deux composantes connexes de T,
    - alors l'ajouter à T
    - sinon ne rien faire
  - Passer à l'arête suivante
- S'arrêter quand il n'y a plus d'arêtes
- Retourner T

La méthode de Kruskal



2. Le PVC

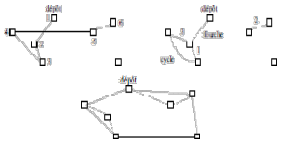
2.3 Heuristiques constructives
Construisent une solution par des décisions successives.

a) Exemples
- heuristique Plus Proche Voisin (PPV)
- heuristique de Fletcher
- méthode de l'élastique (Shamos, PVC euclidien)

PPV part du dépôt puis relie à chaque étape le client libre le plus proche. Fletcher part de nœuds isolés et prend à chaque étape la plus petite arête ne faisant ni fourche, ni cycle avec les arêtes déjà prises.

2. Le PVC

- Shamos construit l'enveloppe convexe, puis l'allonge le moins possible pour relier les nœuds restants.



PPV (gauche), Fletcher (droite) et Shamos (centre)
Les numéros désignent les décisions successives

2. Le PVC

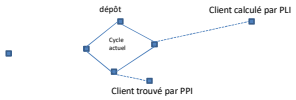
b) Méthodes d'insertion

- A partir d'une boucle sur le dépôt, elles insèrent un client à chaque étape.
- 3 versions :
  - Plus Proche Insertion (PPI),
  - Plus Lointaine Insertion (PLI),
  - et Meilleure Insertion (MI).

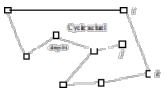
Itération de base pour PPI et PLI :

- Chercher le client libre j le plus près (PPI) ou le plus loin (PLI) du cycle en cours de construction.
- insérer j entre deux nœuds i et k du cycle pour minimiser la variation de coût $D_{ij} + D_{jk} - D_{ik}$.

2. Le PVC



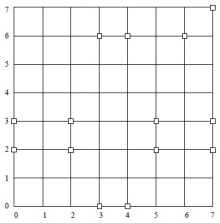
La meilleure insertion n'est pas forcément avant où après
le nœud du cycle le plus proche. Il faut donc tester toutes les insertions possibles pour j :



2. Le PVC

- Itération de base pour MI :
- tester chaque client libre j et chaque couple de nœuds consécutifs (i, k) sur le cycle.
- insérer le nœud donnant la plus faible variation de coût, à la position correspondante.
- Les méthodes d'insertion donnent de bons résultats en moyenne. De manière anti-intuitive, PLI donne les meilleurs résultats moyens dans le cas euclidien!

exercice



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

2. Le PVC

c) Méthode de l'arbre de Christofides*

- Calcul d'un ARCM (algo de Prim). Puis "tour de l'arbre" et élimination des nœuds déjà visités.

- Résultat médiocre en moyenne, mais on peut prouver qu'elle n'est jamais à plus de deux fois l'optimum : elle calcule une tournée T telle que C(T) / C(T*) <=2.

2. Le PVC

- Ce type de résultat est appelé garantie de performance (worst case performance ratio). Il n'est connu que pour certaines heuristiques car il est souvent difficile à établir.
- Meilleure heuristique à garantie de performance pour le PVC : C(T) / C(T*) <= 1.5. Décevant, mais beaucoup d'heuristiques ont une performance moyenne meilleure!

d) Boîtes noires d'heuristiques

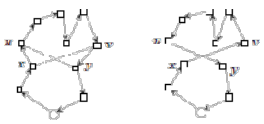
- Les heuristiques atteignent souvent leur pire cas sur des données différentes : on a intérêt à les exécuter toutes et à sortir le meilleur résultat. Beaucoup de logiciels de tournées utilisent de telles "boîtes noires" d'heuristiques.

2. Le PVC

2.4 Recherches locales

- Améliorent progressivement une solution de départ, calculée par exemple par une heuristique simple. Elles construisent une suite de solutions de coûts décroissants.

a) Transformation ou "mouvement" 2-OPT



2. Le PVC

Elle consiste à "croiser" deux arêtes :

- variation D = D(x,v) + D(u,y) – D(x,u) – D(v,y)
- section u à v inversée, pas toujours faisable

- De même, k-OPT consiste à enlever k arêtes et à reconnecter les k chaînes obtenues avec k autres arêtes.
- Tester toutes les transformations k-OPT pour une solution donnée est faisable en $O(n^k)$. En pratique, on utilise 2-OPT et 3-OPT, mais pas k-OPT avec k > 3.*

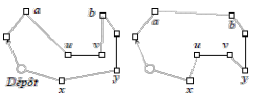
2. Le PVC

c) Exemple de mise en œuvre de 2-OPT
calculer une solution initiale S

- **repeat**
- $\Delta_{min} := \infty$
- **for each** couple d'arêtes ((u,x),(v,y)) non contigües
  - Δ= D(x,v)+D(u,y)–D(x,u)–D(v,y)
  - **if** $\Delta < \Delta_{min}$ then*
    - $\Delta_{min} := \Delta$; p := ((u,x),(v,y))
  - **endif**
- endfor
- **if** $\Delta_{min} < 0$ then
  - croiser réellement les arêtes de la paire p dans S
- **endif**
- **until** $\Delta_{min} = \infty$

2. Le PVC

b) Transformation Or-OPT

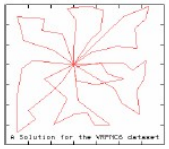


Déplacement d'une chaîne de nœuds consécutifs* :

- Δ = D(a,b)+D(y,v)+D(u,x)-D(a,u)-D(v,b)-D(y,x)
- sens de circulation conservé, sauf sur (u,v)
- en pratique, on se limite à des chaînes de 1 à 3 nœuds

3. Le Pb de Tournées de Véhicules

- Le PTV (VRP, Vehicle Routing Problem) étend le PVC au cas de plusieurs tournées. Il est encore plus difficile : on ne connaît pas l'optimum pour certains pbs à 75 clients.



Exercice: VRP

- Une société de distribution cherche à optimiser la livraison de ses clients. Les données des clients ainsi que celle des tournées est données par le tableau suivant. L'entrepôt est situé au point de cordonnées (0,0). Les véhicules utilisés par l'entreprise sont homogènes avec une capacité de 30 palettes
- QUESTIONS :
  - Modéliser le problème sous la forme d'un programme linéaire
  - Résoudre le problème sous la forme de m-tsp
  - Résoudre le problème en utilisant la méthode de clarke-wright

Exercice

client	X	Y	demande
1	3	3	7
2	7	2	5
3	5	4	10
4	3	6	6
5	-1	5	16
6	-1	1	5
7	-2	2	3
8	-4	1	5
9	-1	-2	10
10	-4	-1	5
11	-3	-3	5
12	-2	-5	10
13	2	-5	3
14	3	-3	5
15	5	-2	10
16	6	-3	7

3. Le Pb de Tournées de Véhicules

3.1 Heuristiques constructives inspirées du PVC

- Toute heuristique du PVC est utilisable pour construire les tournées du PTV une par une. On crée une tournée quand la capacité du véhicule est épuisée. C'est le mode séquentiel.
- Par exemple, on peut appliquer PPV en mode séquentiel. Le nombre de véhicules utilisés (= de tournées) nv fait partie des résultats.

3. Le Pb de Tournées de Véhicules

Si nv est imposé, on utilise le mode parallèle :

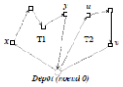
- on construit nv tournées T_1, T_2, ..., T_{nv} en parallèle.
- chaque itération de l'heuristique détermine les meilleures décisions pour T_1, T_2, ..., T_{nv}.
- exemple avec PPV : on détermine le prochain client pour T_1, puis T_2 etc jusqu'à T_{nv} et on recommence.
- Ce mode parallèle donne des tournées mieux équilibrées.

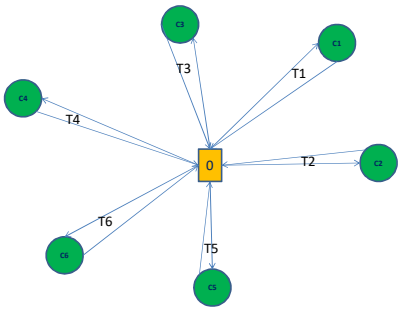
3. Le Pb de Tournées de Véhicules

3.2 Heuristique de Clarke &Wright

Ou méthode de la marguerite. Principe :

- initialiser la marguerite (n tours avec un client chacun) fusions de 2 tours tant que cela procure un gain à chaque étape, fusion de gain maximal





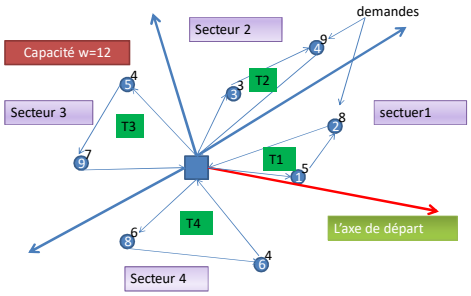
3. Le Pb de Tournées de Véhicules

- Si le véhicule de la tournée T_1 peut faire T_2 sans revenir au dépôt (demande totale <= W), le gain de la fusion est : e(y,u) = D(y,s) + D(s,u) – D(y,u).
- NB: 4 fusions possibles si réseau non orienté.
- Une itération de l'algo teste toutes les paires de tournées et les 4 cas par paire. Elle réalise la fusion de gain > 0 maximal, si elle existe. Sinon, c'est la fin de l'algo.
- Chaque itération diminuant le coût et économisant aussi une tournée, la méthode est réputée pour minimiser à la fois le coût et le nombre de véhicules utilisés.

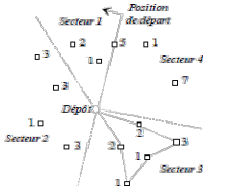
3. Le Pb de Tournées de Véhicules

3.3 Heuristique de Gillett et Miller

- Méthode de type cluster first - route second : on construit des groupes de clients puis on calcule une tournée dans chaque groupe. Principe :
  - trier les clients par angle polaire croissant / dépôt
  - les balayer à partir d'un client de départ quelconque
  - ceci donne des secteurs compatibles avec la capacité
  - calculer une tournée par secteur avec PPV
  - améliorer chaque tournée avec 2-OPT
  - bons résultats en campagne ou à grande échelle et avec un dépôt central.



3. Le Pb de Tournées de Véhicules



Principe de la sweep heuristic de Gillett et Miller

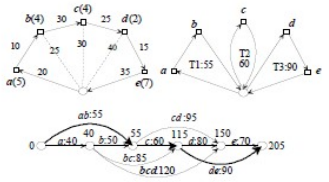
3. Le Pb de Tournées de Véhicules

- 3.4 Heuristique de Beasley
  - Méthode de type route first - cluster second : on calcule
  - un tour géant visitant tous les clients, puis on découpe ce
  - tour en tournées réalisables.
  - Le découpage peut se faire optimalement, cf. exemple
  - suivant avec un tour géant S=(a,b,c,d,e) à nc=5 clients.
  - Les demandes sont entre parenthèses. Les allers-retours
  - possibles au dépôt sont en pointillés. La capacité des
  - véhicules est W = 10.

3. Le Pb de Tournées de Véhicules

- On construit un graphe auxiliaire H avec nc+1 nœuds indexés de 0 à nc. Si la sous-suite de clients S_i ... S_j peut constituer une tournée réalisable, elle est modélisée dans H par un arc (i-1,j), valué par le coût de la tournée.
- Le découpage optimal en tournées correspond à un chemin optimal du 1er au dernier nœud dans H (en trait gras). On obtient 3 tournées, pour un coût total de 205.

3. Le Pb de Tournées de Véhicules



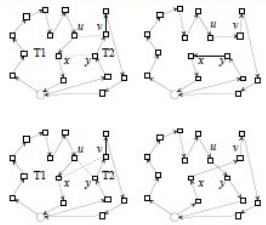
Exemple pour l'heuristique de Beasley

3. Le Pb de Tournées de Véhicules

- 3.5 Recherches locales
- Les recherches locales vues pour le PVC (2-OPT et Or-OPT) peuvent être appliquées tournée par tournée. Mais on peut aussi les appliquer à deux tournées.

3. Le Pb de Tournées de Véhicules

Exemple de 2-OPT sur deux tournées (2 cas) :



3. Le Pb de Tournées de Véhicules

- 4. Quelques complications
- 4.1 Dépôts multiples
- Si on n'a pas de contraintes de capacité des dépôts, on affecte chaque client au dépôt le plus proche et on résout un VRP par dépôt. Sinon, on affecte les clients par PL, pour minimiser la somme des distances aux dépôts tout en respectant les capacités des dépôts (formuler ce PL en exercice).

3. Le Pb de Tournées de Véhicules

4.2 Flotte hétérogène

On a plusieurs types de véhicules, de capacités différentes. On connaît le nombre de véhicules de chaque type. La pratique montre qu'on a toujours intérêt à utiliser en priorité les plus gros véhicules.

4.3 Contraintes d'accès

Si certains arcs du réseau sont interdits à un type de véhicule (poids ou hauteur limités), il faut construire pour ce type un distancier spécifique, en calculant des PCC n'empruntant pas d'arcs interdits. Cela peut se faire en enlevant temporairement ces arcs du réseau.

Exercice application de l'algorithme de Clarke &Wright

Matrice de distance

From \ To	0	1	2	3	4	5
0	-	28	31	20	25	34
1		-	21	29	26	20
2			-	38	20	32
3				-	30	27
4					-	25
5						-

Customer	Quantity
1	37
2	35
3	30
4	25
5	32

Exercice 1

On considère le problème du voyageur de commerce euclidien avec les 10 nœuds suivants, définis par leurs coordonnées. Les mailles ont 1 km de côté. Le nœud-dépôt est le numéro 1. Dans les calculs suivants, choisir toujours le nœud de plus petit numéro en cas d'ex-aequo.

N° nœud	x	y	N° nœud	x	y
1	2	3	6	5	4
2	3	3	7	5	0
3	3	5	8	0	4
4	0	1	9	4	2
5	4	4	10	3	0

a) Calculez un arbre recouvrant de coût minimal puis un 1-arbre de coût minimal (partir du dépôt).
b) Calculez une solution par l'heuristique Plus Proche Voisin (Nearest Neighbour). Quelle est sa distance maximale à l'optimum en % ?
c) Calculez une solution par la méthode de l'arbre de Christofides. Faire le « tour de l'arbre » en suivant des numéros de nœuds croissants. Distance maximale et à l'optimum en % ?
d) Peut-on améliorer la meilleure solution heuristique ?

Exercice 2

Une minoterie doit livrer de la farine à 18 boulangers répartis sur un réseau routier. Pour simplifier, on suppose que la distance entre deux nœuds est égale à la distance euclidienne. Le tableau indique pour chaque nœud ses coordonnées X et Y dans un repère orthonormé gradué en km et sa demande en tonnes de farine. Le dépôt a les coordonnées (3,3). La minoterie a 4 camions de capacité 10 tonnes.

Nœud	X	Y	Demande	Nœud	X	Y	Demande
1	1	0	2	10	0	3	1
2	2	0	2	11	6	3	2
3	6	0	1	12	2	4	3
4	0	1	2	13	5	4	2
5	2	1	1	14	6	4	1
6	5	1	1	15	4	5	2
7	1	2	3	16	0	6	2
8	3	2	2	17	3	6	1
9	5	2	1	18	5	6	1

a) Appliquez l'heuristique Plus Proche Voisin pour construire les tournées une par une. Dans le cas du VRP, on doit prendre à chaque étape le plus proche nœud non encore traité, mais compatible avec la charge du camion. Représentez graphiquement les tournées et précisez bien la longueur de chaque tournée, la longueur totale et le nombre de camions nécessaires.

b) Appliquez la Sweep Heuristic de Gillett et Miller pour former des secteurs, en commençant par le client n° 18 et en tournant dans le sens inverse des aiguilles d'une montre. Résolvez ensuite le problème de voyageur de commerce dans chaque secteur avec l'heuristique de Shamos.

Problème :

- Une société de distribution est chargée de livrer 9 clients à partir d'un seul dépôt, les tableaux suivants donnent les informations necessaires pour établir le plan de distribution. Le premier tableau illustre les distances entre les différents points de distribution inclus le dépôt (nœud 1), le deuxième tableau résume les demandes des neufs points clients en tonnes. L'entreprise dispose d'un ensemble de véhicules de même type ayant une capacité de 16 tonnes.
- **Questions :**
  - Modéliser le problème de tournées de véhicule (problème général)
  - Résoudre le problème de la société de distribution en utilisant la méthode de Clarke and Wright
  - On considère la combinaison suivante Ch=(2,6,5,3,4,7,8,10,9) donner la décomposition de cette combinaison sous forme de tournées en utilisant la méthode de Beasley.

	1	2	3	4	5	6	7	8	9	10
1	0									
2	25	0								
3	43	29	0							
4	57	34	52	0						
5	43	43	72	45	0					
6	61	68	96	71	27	0				
7	29	49	72	71	36	40	0			
8	41	66	81	95	65	66	31	0		
9	48	72	89	99	65	62	31	11	0	
10	71	91	114	108	65	46	43	46	36	0

Nœud	2	3	4	5	6	7	8	9	10
quantité	4	6	10	4	7	11	5	4	8

Chapitre 5

Problèmes de remplissage

Problèmes de Remplissage 1/10

- **Problème de sac à dos** (Knapsack Problem KP)

Modélisez une situation analogue au remplissage d'un sac à dos, ne pouvant dépasser la capacité maximale du sac **W**, avec un ensemble donné de n produits ayant chacun un poids **wi** et une valeur **bi.** Maximisez la valeur totale des objets mis dans le sac à dos.

1) Modélisez le problème sous forme d'un problème linéaire.
2) Résoudre le problème.

Problèmes de Remplissage 2/10

- **Modélisation du problème** (Knapsack 0-1)

Variable de décision

$xj = \begin{cases} 1 \text{ si le produit est transporté} \\ 0 \text{ sinon} \end{cases}$

Fonction objectif

Maximiser le bénéfice : $\sum_{i=1}^{n} xj\ bi$ avec $i \in \{0,...,n\}$

Contraintes :

$\sum_{i=1}^{n} xj\ wi \leq W$ // ne pas dépasser la capacité du sac à dos .

Problèmes de Remplissage 3/10

Résolution avec programmation dynamique

Principes :

- ✓ L'objectif est de diviser le sac de capacité k en k sous-sacs dont la capacité allant de zero à k.
- ✓ La méthode commence par la construction d'un tableau V donnant les bénéfices possible pour chaque sous-sac
- ✓ Ordonner les produits par volume croissant
- ✓ Chaque case du tableau représente le bénéfice maximum possible **V(i,w)** pour les i premiers objets avec une capacité w.

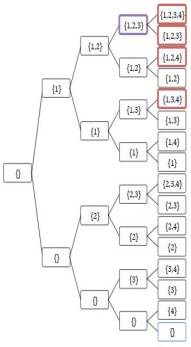
Méthode approché

Objets	1	2	3	4
pi	13	12	8	10
Vi	7	4	3	3

- calculer le rapport (*vi / pi) pour chaque objet i ;*
- trier tous les objets par ordre décroissant de cette valeur ;
- sélectionner les objets un à un dans l'ordre du tri et ajouter l'objet sélectionné dans le sac si le poids maximal reste respecté.

W=30

Méthode Exacte: B&B



241

Programmation Dynamique

242

Problèmes de Remplissage 4/10

- **Résolution avec programmation dynamique**

Les sous-sacs à dos ordonnés selon leur capacité

	1	2	…	…	…	W
1						
…						
n						

L'ensemble des produits que le sac peut contenir allant de 1 à n. les produits sont ordonnés dans

243

Problèmes de Remplissage 5/10

- **Algorithme programmation dynamique**

```
// Remplir le tableau de bénéfice
    For w=0 to W     // initialisation
        V [0,w]=0
    For i=0 to n
        V [i,0]=0
    For  i= 1 to n
        For w=1 to W
            if (pi <= w)
                if (vi + V [i-1,w-pi] > V[i-1,w])
                    V[i,w]=V[i-1,w-pi] +vi
                Else  V[i,w]=V[i-1,w]
            Else   V[i,w]=V[i-1,w]
        End For
    End For
```

244

Problèmes de Remplissage 6/10

- **Algorithme programmation dynamique**

// Pour déterminer les produits qui sont dans le sac à dos on procède de la manière suivante

Soient i=n et k=W

if (V [i,k] ≠ V[i-1,k]) // Entre l'itération i-1 et i , le produit a été ajouté

i = i-1 , k= k-wi // la capacité restante après l'ajout du produit

Else i =i-1

Problèmes de Remplissage 7/10

Exercice d'application :

Cherchez à à résoudre le problème suivant par programmation dynamique :

- Ensemble d'élément n=4
- i(wi, bi) ➔ 1(2,3)
  2(3,4)
  3(4,5)
  4(5,6)
- Capacité totale W=5

Problèmes de Remplissage 8/10

Solution :

i\ wi	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

➔ Le bénéfice maximale est de **V(4,5) = 7**

Problèmes de Remplissage 9/10

- **Calcul du bénéfice** (pour i=1)

Pour w=1

wi =2 ➔ w-wi = -1

V (1,1) = V(0,1) =0

Pour w =2

wi =2 ➔ w-wi =0

V (0,0)+3 = 3> V(0,2)=0

Donc V (1,2)= 3+V(1,1) =3

Pour w=3

V (1,3) = 3

Problèmes de Remplissage 10/10

- **Déterminons les produits qui sont dans le sac :**

On a V (4,5) = V(3, 5) ➔ { 4 } ∉ S

V (3,5) = V(2,5) ➔ { 3 } ∉ S

V (2,5) ≠ V(1,5) ➔ { 2 } ∈ S

Donc itération actuelle → nouvelle itération

i = 2 → i = 2-1 =1

k = 5 → k = 2

Finalement **S = {1 ,2}**

249

Bin Packing Problem

https://www.geeksforgeeks.org/program-first-fit-algorithm-memory-management/

Description

- The **bin packing problem** is an optimization problem, in which items of different sizes must be packed into a finite number of bins or containers, each of a fixed given capacity, in a way that minimizes the number of bins used.
- The problem has many applications, such as filling up containers, loading trucks with weight capacity constraints, creating file backups in media, and technology mapping in FPGA semiconductor chip design.

Mathematical model

minimize $K = \sum_{j=1}^{n} y_j$

subject to $K \geq 1,$

$$\sum_{i \in I} s(i) x_{ij} \leq B y_j, \quad \forall j \in \{1, \ldots, n\}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i \in I$$

$$y_j \in \{0,1\}, \qquad \forall j \in \{1, \ldots, n\}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in I \, \forall j \in \{1, \ldots, n\}$$

where $y_j = 1$ if bin j is used and $x_{ij} = 1$ if item i is put into bin j.

Solving Algorithms for the BPP

- **Next Fit (NF)** always keeps a single open bin. When the new item does not fit into it, it closes the current bin and opens a new bin. Its advantage is that it is a bounded-space algorithm since it only needs to keep a single open bin in memory
- **Next-k-Fit (NkF)** is a variant of Next-Fit, but instead of keeping only one bin open, the algorithm keeps the last *k* bins open and chooses the first bin in which the item fits. Therefore, it is called a *k-bounded space* algorithm

Solving Algorithms for the BPP

- **First-Fit (FF)** keeps all bins open, in the order in which they were opened. It attempts to place each new item into the *first* bin in which it fits
- **Best-Fit (BF)**, too, keeps all bins open, but attempts to place each new item into the bin with the *maximum* load in which it fits

Algorithms for the BPP

- **Worst-Fit (WF)** attempts to place each new item into the bin with the *minimum* load
- **Almost Worst-Fit (AWF)** attempts to place each new item inside the *second most empty* open bin (or emptiest bin if there are two such bins). If it does not fit, it tries the most empty one

Solving methods

- **First-fit-decreasing (FFD)** orders the items by descending size, then calls First-Fit
- **Next-fit-decreasing (NFD)** orders the items by descending size, then calls Next-Fit.
- **Modified first-fit-decreasing (MFFD)**, improves on FFD for items larger than half a bin by classifying items by size into four size classes large, medium, small, and tiny, corresponding to items with size > 1/2 bin, > 1/3 bin, > 1/6 bin, and smaller items respectively

Exemple

- A(8), B(7), D(9), E(6), F(9), G(5), H(15), I(6), J(7), K(8)
- Bin withsize =20

Exemple 2

- A(0.5), B(0.7), C(0.5), D(0.2), E(0.4), F(0.2), G(0.5), H(0.1), I(0.6)
- Bin size = 1

```
C implementation of First- Fit algorithm
#include<stdio.h>

// Function to allocate memory to
// blocks as per First fit algorithm
void firstFit(int blockSize[], int m, int processSize[], int n)
{
    int i, j;
    // Stores block id of the
    // block allocated to a process
    int allocation[n];

    // Initially no block is assigned to any process
    for(i = 0; i < n; i++)
    {
        allocation[i] = -1;
    }

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (i = 0; i < n; i++)       //here, n -> number of processes
    {
        for (j = 0; j < m; j++)      //here, m -> number of blocks
        {
            if (blockSize[j] >= processSize[i])
            {
                // allocating block j to the ith process
                allocation[i] = j;

                // Reduce available memory in this block.
                blockSize[j] -= processSize[i];
    break;   //go to the next process in the queue
            }
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < n; i++)
    {
        printf(" %i\t\t\t", i+1);
        printf("%i\t\t\t", processSize[i]);
        if (allocation[i] != -1)
            printf("%i", allocation[i] + 1);
```

```
// Driver code
int main()
{
    int m;    //number of blocks in the memory
      int n;  //number of processes in the input queue
      int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    m = sizeof(blockSize) / sizeof(blockSize[0]);
    n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

```
# Python3 implementation of First-Fit algorithm

# Function to allocate memory to
# blocks as per First fit algorithm
def firstFit(blockSize, m, processSize, n):

        # Stores block id of the
        # block allocated to a process
        allocation = [-1] * n

        # Initially no block is assigned to any process

        # pick each process and find suitable blocks
        # according to its size ad assign to it
        for i in range(n):
                for j in range(m):
                        if blockSize[j] >= processSize[i]:

                                # allocate block j to p[i] process
                                allocation[i] = j

                                # Reduce available memory in this block.
                                blockSize[j] -= processSize[i]

                                break

        print(" Process No. Process Size      Block no.")
        for i in range(n):
                print(" ", i + 1, "                ", processSize[i],
                      "                ", end = " ")
                if allocation[i] != -1:
                        print(allocation[i] + 1)
                else:
                        print("Not Allocated")

# Driver code
if __name__ == '__main__':
        blockSize = [100, 500, 200, 300, 600]
        processSize = [212, 417, 112, 426]
        m = len(blockSize)
        n = len(processSize)

        firstFit(blockSize, m, processSize, n)
```

https://www.geeksforgeeks.org/program-first-fit-algorithm-memory-management/