Mathématiques Appliquées Bordeaux

Licence d' Ingénierie Mathématiques

Année 2 et 3

L4MIngMa3: modèles et méthodes d'optimisation

Leçon 5: Formuler un problème combinatoire (un art) et Introduction aux algorithmes combinatoires

1. Planification de la production pour un système dynamique
   - 2 formulations: bonne et moins bonne
   - propriété des solutions optimales extrêmes
   - Algorithme exact direct
2. Arbre de recouvrement de poids minimum
   - Algorithme exacte (glouton)
3. Problème de localisation des dépôts
   - 2 formulations: bonne et moins bonne
   - Algorithme heuristique (glouton)
4. Ordonancement
   - Algorithme exacte (échange)
5. Voyageur de commerce
   - Algorithmes heuristiques (glouton et échange)

François Vanderbeck, MAB, Bur.: A33:257, Email: fv@math.u-bordeaux1.fr

Planification de la production pour un système dynamique

semaine	1	2	3	4	5	6	7	8	9	10
demandes prévues				10	12	15	14	11	10	8
Stocks de sécurité				2	2	3	3	2	2	2
niveaux requis				12	12	16	14	10	10	8
stocks disponibles				15						
demandes nettes				0	9	16	14	10	10	8
demandes déphasées	0	9	16	14	10	10	8			
production **lot-pour-lot**	0	9	16	14	10	10	8			
stocks en fin de sem	0	0	0	0	0	0	0			
productions déphasées				0	9	16	14	10	10	8
stocks réels				5	2	3	3	2	2	2
lots **EOQ** ($= \sqrt{\frac{2\,K\,D}{h}}$)	0	37	0	37	0	0	0			
stocks en fin de sem	0	28	12	35	25	15	7			
productions déphasées				0	37	0	37	0	0	0
stocks réels				5	30	15	38	27	17	9
production **Optimale**	0	25	0	42	0	0	0			
stocks en fin de sem	0	16	0	28	18	8	0			
productions déphasées				0	25	0	42	0	0	0
stocks réels				5	18	3	31	20	10	2

Modèle de base: un produit, sans limite de capacité

- **Problème:** déterminer un planning de production
  sur un horizon de temps discret: $t = 1, \ldots T$,
  qui satisfasse aux demandes $d_1, d_2, \ldots, d_T$,
  et en minimisant les coûts qui incluent
  des coûts de production unitaire, $p_1, p_2, \ldots, p_T$,
  des coûts de production fixes si on produit: $f_1, f_2, \ldots, f_T$,
  des coûts unitaires de stockage, $h_1, h_2, \ldots, h_T$.

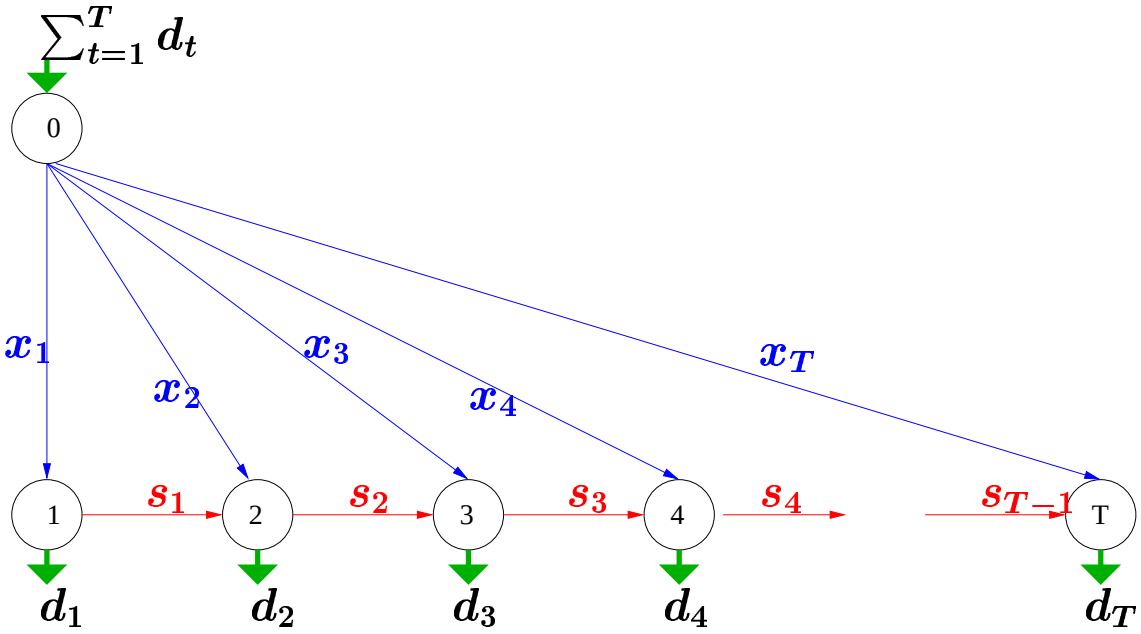
- **Variables:**
  Soit x_t la quantité produite en période t
  Soit $y_t = 1$ si on produit en période t, 0 sinon
  Soit s_t le niveau de stock disponible en fin de période t qui est reporté en période $t + 1$.

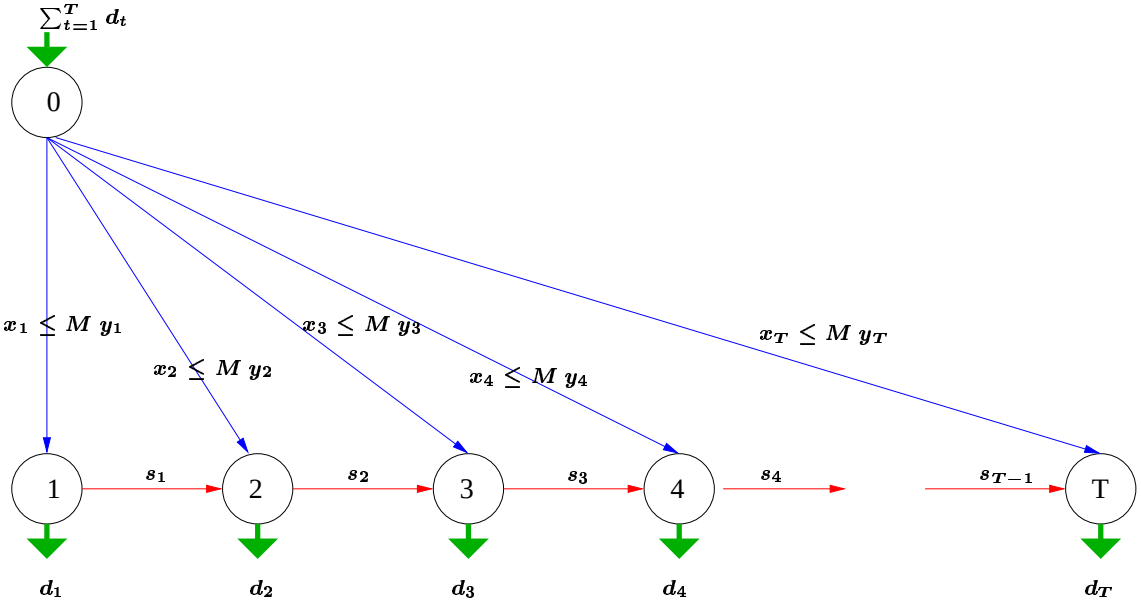
- **Formulation 1:**

$$\begin{aligned}
\min \quad & \sum_{t=1}^{T} (p_t\, x_t + f_t\, y_t + h_t\, s_t) \\
s_{t-1} + x_t \;&=\; d_t + s_t \qquad \text{pour } t = 1, \ldots, T \\
0 \le x_t \;&\le\; M\, y_t \qquad \text{pour } t = 1, \ldots, T \\
x_t,\ s_t \;&\ge\; 0 \qquad \text{pour } t = 1, \ldots, T \\
y_t \;&\in\; \{0, 1\} \qquad \text{pour } t = 1, \ldots, T
\end{aligned}$$

où p.e. $M = d_{tT} = \sum_{\tau=t}^{T} d_\tau$ dans l'équation t.

Modèle de base: un produit, sans limite de capacité
Illustration





- **Nouvelles variables:**
  $w_{i,t}$ la fraction de la demande t produite en pér. $i \leq t$
  $y_t = 1$ si on produit en période t, 0 sinon.

- **Formulation 2: (cas $S_T = 0$)**

$$\min \quad \sum_{i,t=1}^{T} c_{i,t}\, w_{i,t} + \sum_{t=1}^{T} f_t\, y_t$$

$$\begin{aligned}
\sum_{i=1}^{T} w_{i,t} &= 1 \qquad \text{pour } t = 1, \ldots, T \\
0 \leq w_{i,t} &\leq y_t \qquad \text{pour } i, t = 1, \ldots, T \\
y_t &\in \{0, 1\} \qquad \text{pour } t = 1, \ldots, T
\end{aligned}$$

  où $c_{i,t} = (p_i + h_i + h_{i+1} + \ldots, h_{t-1})\, d_t$.

- **Équivalence:**

$$\begin{aligned}
x_i &= \sum_{t=1}^{T} d_t\, w_{i,t} \qquad \text{pour } i = 1, \ldots, T \\
s_i &= \sum_{t=1}^{i} x_t - \sum_{t=1}^{i} d_t \qquad \text{pour } i = 1, \ldots, T
\end{aligned}$$

- **Comparaison:** la formulation 2 donne l'enveloppe convexe des solutions entières (est idéale).

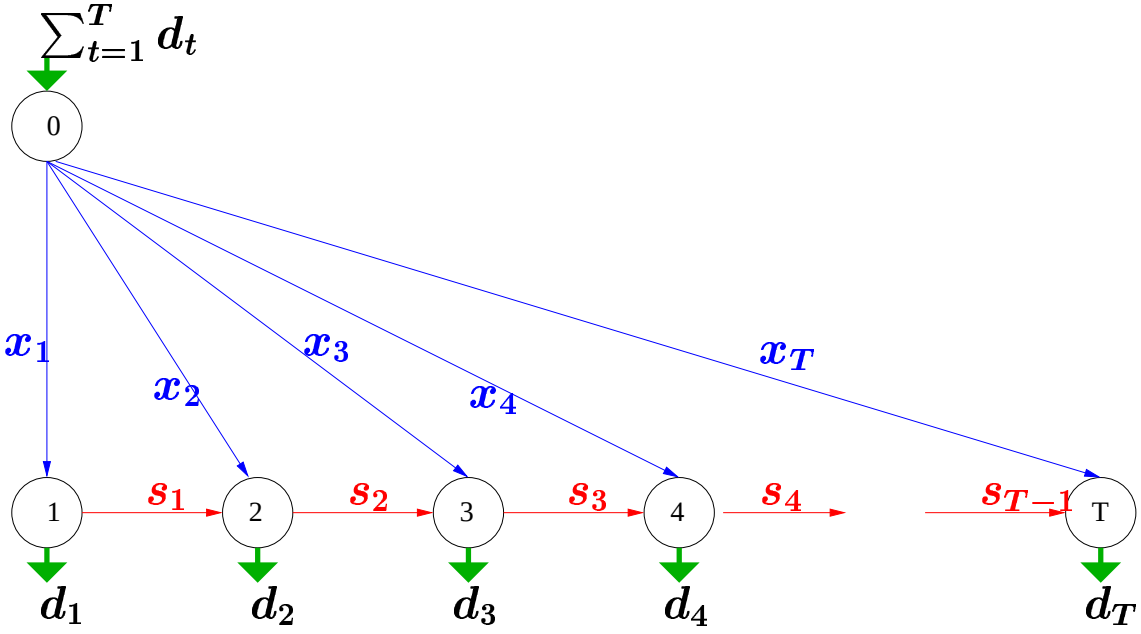
PROBLÈME DE PRODUCTION PAR LOT:

Propriétés des solutions optimales extrêmes:

(i) $s_{t-1}\, x_t = 0$

(ii) $x_t > 0 \;>\; x_t = d_{t\,l}$ pour $l \geq t$

(iii) $s_{t-1} > 0 \;>\; s_{t-1} = d_{t\,l}$ pour $l \geq t$



Programme Dynamique (WAGNER-WITHIN, 1959): $O(T^2)$

C^t represente le coût minimum d'un plan de production couvrant les demandes 1 à t.

$$C^0 = 0$$

$$C^t = \min_{1 \leq k \leq t} \{C^{k-1} + f_k + p_k\, d_{k,t} + h_k\, d_{k+1,t} + \ldots + h_{t-1}\, d_t\}$$

$$\text{pour } t = 1, \ldots, T$$

La solution optimale est donnée par C^T.

Problème d'arbre de recouvrement de poids minimum

Applications:

- construire un réseau ferroviaire de coût minimum;
- installer un "pipeline" pour connecter des stations de forage et raffineries;
- dessiner un réseau télécom élémentaire;
- établir les connections électriques sur un panneau.

Problème: Etant donné un graphe avec des coûts sur les arêtes, trouver un arbre de recouvrement de coût total minimum.

Algorithme de Kruskal (GLOUTON)

- trier les arêtes par ordre non-décroîssant des coûts:

$$c_{e_1} \leq c_{e_2} \leq \ldots \leq c_{e_m} .$$

- Considérer les arêtes dans cet ordre et les ajouter à l'arbre en construction si elles ne créent pas de cycle
- stopper quand tous les noeuds sont connectés.

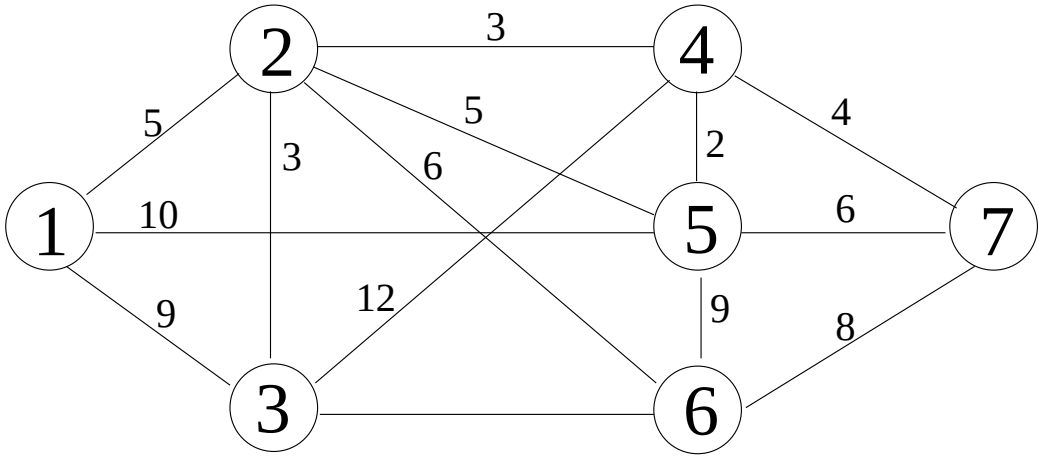
Algorithme de Prim (GLOUTON)

- Initialiser l'arbre en choisissant un noeud.
- Ajouter, de manière itérative, l'arête de coût minimum incidente à l'arbre courant.
- stopper quand tous les noeuds sont connectés.

ALGORITHME de Kruskal

pas 0: $T = \emptyset$, $k = 1$, classer $c_{e_1} \leq c_{e_2} \leq \ldots \leq c_{e_m}$.
pas 1: si $T \cup \{e_k\} \not\supseteq$ cycle, $T = T \cup \{e_k\}$.
pas 2: si $|T| = n - 1$, STOP avec $T^* = T$ optimal.
pas 3: si $k = m$, STOP sans solution réalisable.
pas 4: $k = k + 1$, retourner au Pas 1.

Exemple:



Preuve qu'on obtient un T optimal: Supposons par l'absurde

$$\exists\ T^o = (f_1, f_2, \ldots, f_{n-1}) \neq T^* = (g_1, g_2, \ldots, g_{n-1})$$

tel que $c(T^o) < c(T^*)$. Soit $f_i = g_i$ pour $i = 1, \ldots, l$ mais $f_{l+1} \neq g_{l+1}$. Alors, $c_{g_{l+1}} < c_{f_{l+1}}$ et $g_{l+1} \notin T^o$ puisque $c_{f_{l+1}} < c_{f_{l+2}} < \ldots < c_{f_{n-1}}$. Donc, $T^o \cup \{g_{l+1}\} \ni$ cycle avec cycle $\subseteq (g_{l+1}, f_{l+1}, \ldots, f_{n-1})$. Soit $f_c \neq g_{l+1}$ une arete du cycle, l'enlever redonne un arbre dont le coût est $< c(T^o)$. Itérer ainsi redonne T^* tel que $c(T^*) < c(T^o)$. Contradiction.

Problème de localisation des dépôts

Choisir des sites pour localiser des installations (entrepôts, magasins, boîtes postales, bouches d'incendie) et décider de l'affectation des clients aux sites (clients desservis par chaque site).

Données:

$N = \{1, \ldots n\}$	ensemble de sites potentiels;
$I = \{1, \ldots m\}$	ensemble de clients;
c_j	coût de l'ouverture d'une installation au site j;
$d_{i\,j}$	coût pour servir le client j à partir de j.

Variables:

$$x_j = \begin{cases} 1 & \text{Si on fait une installation au site } j; \\ 0 & \text{sinon;} \end{cases}$$

$y_{i\,j}$ = fraction de la demande du client i déservie à partir du site j;

Formulation 1:

$$\min \sum_{j \in N} c_j\, x_j + \sum_{i,j} d_{i\,j}\, y_{i\,j} \qquad (1)$$

$$\sum_{j \in N} y_{i\,j} = 1 \quad \text{pour tout } i$$

$$\sum_{i} y_{i\,j} \leq m\, x_j \quad \text{pour tout } j$$

$$y_{i\,j} \geq 0 \quad \text{pour tout } i,\ j$$

$$x_j \in \{0, 1\} \quad \text{pour tout } j$$

Formulation 2:

$$
\begin{aligned}
\textbf{min} \quad & \sum_{j \in N} c_j \; x_j \; + \; \sum_{i,j} d_{i\,j} \; y_{i\,j} && (2)\\
& \sum_{j \in N} y_{i\,j} \; = \; 1 \quad \text{pour tout } i \\
& y_{i\,j} \; \leq \; x_j \quad \text{pour tout } i,\; j \\
& y_{i\,j} \; \geq \; 0 \quad \text{pour tout } i,\; j \\
& x_j \; \in \; \{0, 1\} \quad \text{pour tout } j
\end{aligned}
$$

Comparaison des formulations 1 et 2:

Heuristique glouton pour le problème de localisation de dépôts

$$\min \sum_{j=1}^{n} f_j \, x_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{i\,j} \, y_{i\,j} \qquad (3)$$

$$\sum_{j=1}^{n} y_{i\,j} = 1 \quad \text{pour tout } i$$

$$y_{i\,j} \leq x_j \quad \text{pour tout } i,\ j$$

$$y_{i\,j} \geq 0 \quad \text{pour tout } i,\ j$$

$$x_j \in \{0, 1\} \quad \text{pour tout } j$$

où

$$x_j = \begin{cases} 1 & \text{Si on fait une installation au site } j; \\ 0 & \text{sinon;} \end{cases}$$

$y_{i\,j}$ = fraction de la demande i servie par le site j

<u>Glouton:</u>

S = ensemble de dépôts choisis.

$c(S) = \sum_{j \in S} f_j + \sum_i \min_{j \in S} d_{i,j}$.

Initialement, $S = \emptyset$.

Itérativement ajouter à S le j qui minimise l'augmentation du coût tant que S non réalisable ou que le coût diminue.

S est réalisable si $|S| \geq 1$.

Problèmes d'ordonnancement

Exemple: L'artisan gérant son carnet de commande

Sur le carnet de commande d'un artisan, figurent 5 tâches dont les durées exprimées en jours de travail sont données ci-dessous:

tâches	A	B	C	D	E
durées	$\mathbf{1}$	$\mathbf{5}$	$\mathbf{3}$	$\mathbf{8}$	$\mathbf{9}$
dates de livraison prévue	$\mathbf{2}$	$\mathbf{7}$	$\mathbf{8}$	$\mathbf{11}$	$\mathbf{13}$

Sachant que l'artisan ne peut travailler qu'à la réalisation d'une tâche à la fois et qu'il doit terminer la tâche en cours avant de pouvoir commencer la suivante, comment doit-il programmer ces tâches si son but est

1. de **finir au plus vite**;
2. de **minimiser le temps d'attente** de ses clients (en moyenne);
3. de **commencer** ces projets **au plus vite** (en moyenne);
4. de **terminer** les projets en moyenne le plus tôt possible **par rapport à leur date de livraison**
5. de **minimiser le retard maximum**
6. de **minimiser le nombre de projets en retard**
7. de **minimiser la perte d'intérêt bancaire** si le client paye à la date = max { date de réception, date prévue pour la livraison}.

EXEMPLE minimiser le temps de complétion: $1//\sum_j C_j$

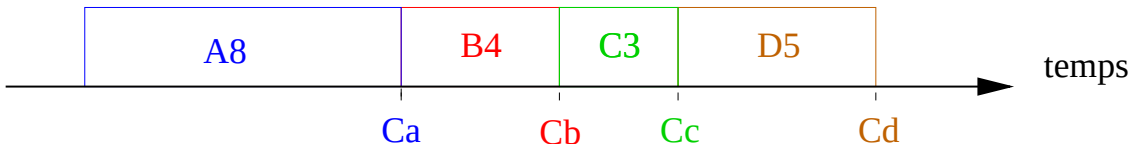
- **Ressource rare** = machine unique
- **Activités** = n jobs à réaliser sur cette machine.
  Temps d'exécution: p_j pour $j = 1, \ldots, n$
- **But** = minimiser les temps d'attente temps de completion moyen $= \overline{C} = \frac{1}{n}\sum_{j=1}^{n} C_j$

  mesure { du temps d'attente moyen
  du nombre moyen de jobs dans le système
  du niveau de stock moyen }

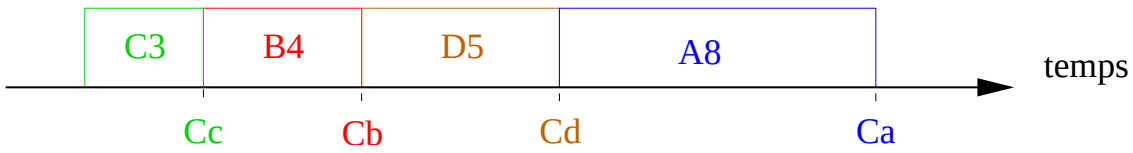
- **Exemple numérique:** 4 jobs:

job	A	B	C	D
durée p_j	8	4	3	5

1. Séquence $< A, B, C, D >$



2. Séquence $< C, B, D, A >$

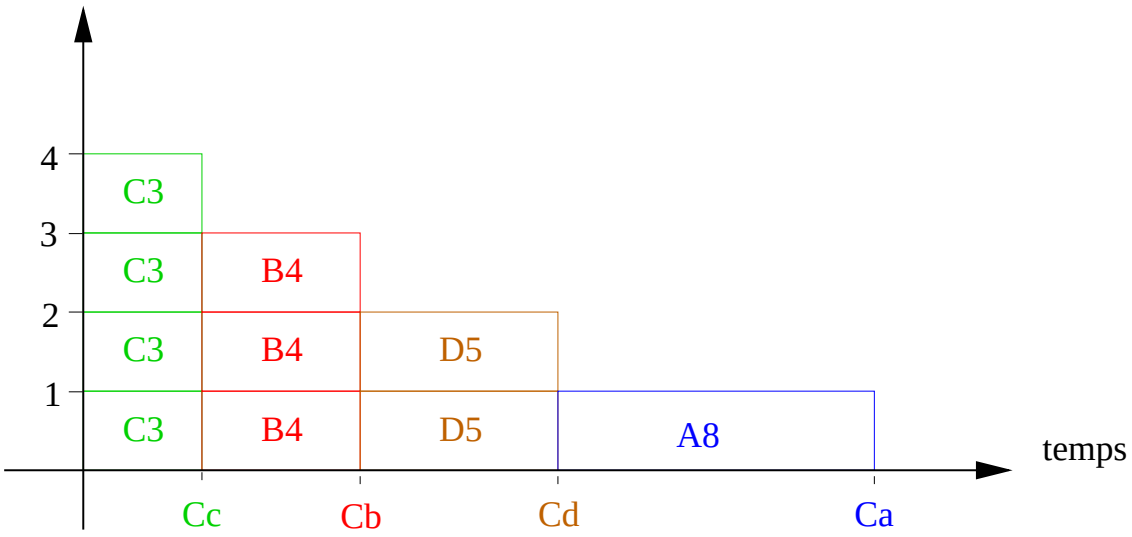


- **Séquence optimale:** effectuer les tâches dans l'ordre non croissant des durées (**Shortest Processing Time** rule).

Équivalence entre minimiser C, I, W, et L sur 1 machine

- **temps de complétion total:**

$$C = \sum_j C_j = A$$



- **Stock (inventory) total:**

$$I = \{np_{[1]} + (n-1)p_{[2]} + \cdots + p_{[n]}\} = A$$

- **Temps d'attente total:**

$$W = A - \sum_j p_j$$

- **Avance/retard total sur les dates de livraison:**

$$L = \sum_j L_j = \sum_j (C_j - d_j) = (\sum_j C_j) - (\sum_j d_j) = A - \sum_j d_j$$

Minimisé par une séquence SPT (Shortest Processing Time)

L'argument d'échange 2 à 2

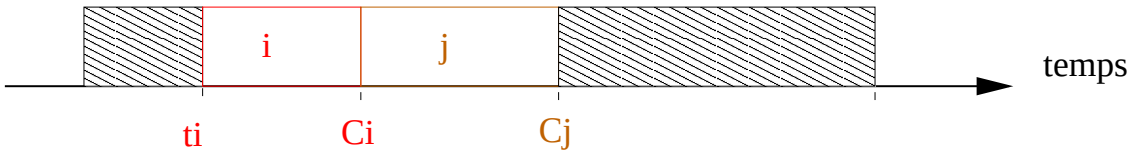
Théorème:

Dans un problème à une seule machine, la somme des temps de complétion ($\sum_j C_j$) est minimisée par le séquencement SPT *(Weighted Shortest Processing Time):*

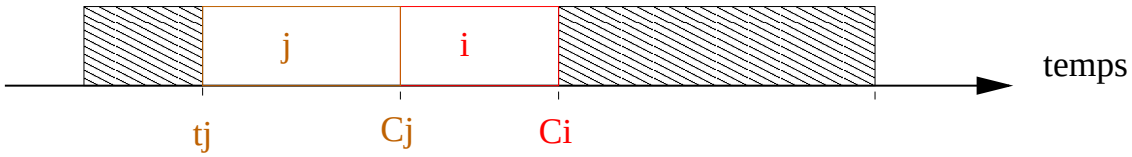
$$p_{[1]} \leq p_{[2]} \leq \ldots \leq p_{[n]}$$

Preuve: (par contradiction)

Considérons une séquence S qui ne satisfasse pas l'ordre SPT, alors $\exists$ 2 tâches adjacentes i et j telles que i précède j bien que $p_i > p_j$



Cette séquence ne peut être optimale car la séquence S' obtenue en échangeant la position de i et j donne un coût plus petit:



$$Z = \sum_{k \neq i,j} C_k + (t + p_i) + (t + p_i + p_j)$$

$$Z' = \sum_{k \neq i,j} C_k + (t + p_j) + (t + p_i + p_j)$$

$$Z - Z' = p_i - p_j > 0$$

Heuristiques glouton pour le problème de voyageur de commerce

1. **Nearest Neighbor:** itérativement aller vers la ville nonencore visitée qui est la plus proche:

2. **Cheapest arc:** choisir itérativement l'arc le moins cher qui ne crée pas de cycle

3. **Cheapest insertion:** insertion la moins cher dans le sous-tour construit:

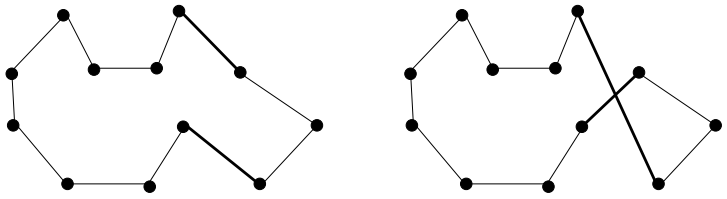
$$k = \text{argmin}_{(i,j)\in\text{tour}} \{c_{i\,k} + c_{k\,j} - c_{i\,j}\} .$$

4. ...

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} - & 30 & 26 & 50 & 50 \\ & - & 24 & 40 & 40 \\ & & - & 24 & 26 \\ & & & - & 30 \\ & & & & - \end{pmatrix} \end{array}$$

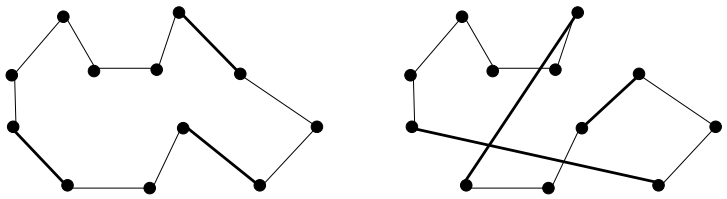
HEURISTIQUE D'ÉCHANGE pour le TSP

- **2-opt:**
  - $N(x) = \{$ tours qu'on peut obtenir en échangeant 2 arêtes $\}$.
  - Quand on enlève 2 arêtes d'un tour, il n'y a qu'une manière de reconstruire un tour.



  - Complexité: $\binom{2}{n} = O(n^2)$ voisins et $O(1)$ opérations pour calculer $c\,x'$

- **3-opt:**
  - $N(x) = \{$ tours qu'on peut obtenir en échangeant 3 arêtes $\}$.
  - Quand on enlève 3 arêtes d'un tour, il y a plusieurs manières de reconstruire un tour.



  - Complexité: $\binom{3}{n} = O(n^3)$ voisins

- **k-opt: . . .**

Inconvénient de la méthode de recherche locale

rester coincer dans un optimum local