ALGORITHME "POUR G_{max}"

Cadre

On considère une famille de n tâches $(J_i)_{i\in[\![1..n]\!]}$ de durées respectives $(p_i)_{i\in[\![1..n]\!]}$.

On utilisera à nouveau la typologie α,β,γ.

De plus comme notre problème sera régulier (ie le critère à minimiser sera régulier) et nous suffira ici aussi de parler de permutation, un ordonnancement s'en déduisant par calage à gauche.

le problème

$1\,|\,-\,|\,g_{max}$

- 1 : une seule machine
- $-$: tâches non préemptives, sans contraintes
- g_{max} : $g_{max} = \max_{i\in[\![1..n]\!]} g_i(C_i)$ où les $(g_i)_{i\in[\![1..n]\!]}$ sont des fonctions croissantes fixées, et les (C_i) les dates de fin des tâches.

Pté

Si $k\in[\![1..n]\!]$ est tel que $g_k(P) = \min_{i\in[\![1..n]\!]} g_i(P)$ où $P = \sum_{i=1}^{n} p_i$

alors il existe un ordonnancement optimal dans lequel J_k est exécuté en dernier

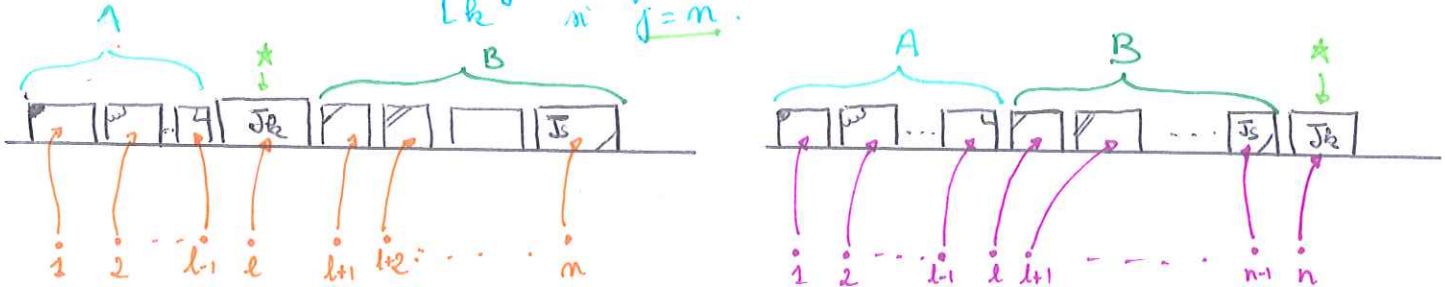
Preuve On considère σ une permutation correspondant à un ordonnancement optimal.

On suppose que $\sigma(n) \neq k$, c'est-à-dire que la tâche J_k n'est pas exécutée en dernier dans cet ordo. sans quoi on aurait fini.

Il existe donc $\begin{cases} s\in[\![1..n]\!] \text{ tq } \sigma(n) = s, & \text{ie } J_s \text{ est la dernière exécutée.} \\ l\in[\![1..n]\!] \text{ tq } \sigma(l) = k, & \text{ie } J_k \text{ est exécutée en } l\text{-ième.} \end{cases}$

On considère l'ordonnancement dans lequel on rejette J_k à la fin, c-à-d celui associé à la permutation

$$\sigma' = j \mapsto \begin{cases} \sigma(j) & \text{si } j\in[\![1..l-1]\!] \\ \sigma(j+1) & \text{si } j\in[\![l..n-1]\!] \\ k & \text{si } j = n. \end{cases}$$



Comparons les dates de fin dans les deux ordonnancements :

- $\forall i \in \sigma'([1..l-1])\ \ C_i' = C_i$ (cas où $J_i \in A$)

  En effet si $i = \sigma'(j)$ où $j \in [1..l-1]$
  on a $C_i' = C'_{\sigma'(j)} = \sum_{a \le j} p_{\sigma'(a)} = \sum_{a \le j} p_{\sigma(a)} = C_{\sigma(j)} = C_i$

- $\forall i \in \sigma'([l..n-1])\ \ C_i' \le C_i$ (Cas où $J_i \in B$)

  En effet si $i = \sigma'(j)$ avec $j \in [l..n-1]$.
  on a $C_i' = \sum_{a \le j} p_{\sigma'(a)} = \sum_{a \le l-1} \underbrace{p_{\sigma'(a)}}_{p_{\sigma(a)}} + \sum_{a=l}^{j} \underbrace{p_{\sigma'(a)}}_{p_{\sigma(a+1)}} = \sum_{a \le j+1} p_{\sigma(a)} - p_{\sigma(l)} \le \sum_{a \le j+1} p_{\sigma(a)} = C_{\sigma(j+1)} = C_i$

- $C_k' = P$

  En effet $C_k' = C'_{\sigma'(n)} = \sum_{a \le n} p_{\sigma'(a)} = \sum_{a \le n} p_a = P$.

Comparons alors les g_{max} :

$g'_{max} = \max_{i \in [1..n]} g_i(C_i)$) par associativité du max

$= \max \Big[\max_{i \in \underbrace{\sigma'[1..l-1]}_{\sigma[1..l-1]}} g_i(\underbrace{C_i'}_{C_i}) \ ; \ \max_{i \in \underbrace{\sigma'[l..n-1]}_{\sigma[l+1..n]}} g_i(\underbrace{C_i'}_{\le C_i}) \ ; \ g_k(\underbrace{C_k'}_{P}) \Big]$

) par croissance des g_i

$\le \max \Big[\max_{i \in \sigma[1..l-1]} g_i(C_i) \ ; \ \max_{i \in \sigma[l+1..n]} g_i(C_i) \ ; \ g_k(P) \Big]$

$= \max \Big[\max_{i \in \sigma([1..n] \setminus \{l,n\})} g_i(C_i) \ ; \ g_s(C_s) \ ; \ g_k(P) \Big]$) car $k = \sigma(l)$, $s = \sigma(n)$

$= \max \Big[\underline{\qquad\qquad} \ ; \ g_s(C_s) \ ; \ g_k(C_k) \Big]$) car $g_k(C_k) \le g_k(P) \le g_s(P) = g_s(C_s)$

$= \max g_i(C_i)$

$= g_{max}$

Ainsi σ' est lui aussi optimal, et présente J_k à la fin.
On a donc montré la dominance de l'ensemble $\{\sigma \mid \sigma(n) \in \operatorname{argmin}(g_i(P))\} =: E$
(et même des $E_k = \{\sigma \mid \sigma(n) = k\}$ pour tout $k \in \operatorname{argmin}(g_i(P))$)

Correction

L'algorithme se termine puisqu'il n'y a que n boucles. La question ici est de savoir s'il est correct c-à-d s'il fournit un ordonnancement optimal.

Pour montrer que c'est bien le cas on s'appuye sur deux invariants de boucle : (mais à la sortie de la j-ème boucle)

H_j : $\begin{cases} Res_j \subseteq I_{n-j} \ [1..n] \\ \# Res_j = j \\ P_j = \sum_{j \in I} p_i \end{cases}$

K_j En notant $Res_j = \begin{array}{|c|}\hline k_j \\ \vdots \\ k_1 \\ \hline \end{array}$ il existe un ordonnancem^t opt. des tâches $(J_i)_{i \in [1..n]}$ terminant par $J_{k_j}, J_{k_{j-1}} \dots J_{k_1}$.

- H_0 vraie car $Res = \emptyset$ et $P_0 = \sum_{i=1}^{\hat{n}} p_i$. ($I_{n-0} = I_n = [1..n]$)
- K_0 vraie car $Res = \emptyset$ donc on impose rien.
- H_j reste vraie à chaque étape par construction.
- K_1 est exactement la propriété qu'on a démontré précédemment puisque au rang 1, $Res = \{k_1\}$ et $k_1 \in \operatorname{argmin}_{i \in [1..n]} g_i(C_i)$.
- Montrons l'hérédité en général : supposons que K_j est vraie pour un certain $j \leq n-1$ et montrons qu'alors K_{j+1} est vraie aussi.

  Par K_j il existe σ un ordonnancem^t global optimal, terminant par $J_{k_j}, J_{k_{j-1}} \dots J_{k_1}$, de valeur $g_{max}(\sigma)$.

  Par la propriété précédente et par définition de k_{j+1}, il existe un ordonnancement des tâches de I_j, disons $\gamma \in B_j([1..n-j], I)$ optimal et se terminant par $J_{k_{j+1}}$.

  On peut alors remplacer les $n-j$ premières tâches ordonnées selon σ par celle ordonnées selon γ sans perdre l'optimalité.

  On considère en effet $\sigma' \equiv \gamma$ sur $[1..n-j]$ $\quad \left(\text{ie } \forall i \in [1..n-j],\ \sigma'(i) = \gamma(i)\right.$
  $\equiv \sigma$ sur $[n-j+1..n]$ $\quad \left.\forall i \in [1..j],\ \sigma'(n+1-i) = k_i\right)$

  alors $g_{max}(\sigma') = \max_{i \in [1..n]} g_i(C_i(\sigma'))$

  $= \max\left[\underbrace{\max_{i \in [1..n-j]} g_i(C_i(\gamma))}_{\leq \max_{i \in [1..n-j]} g_i(C_i(\sigma)) \text{ par optimalité de } \gamma}, \max_{i \in [n-j+1..n]} g_i\left(\underbrace{C_i(\sigma')}_{C_i''(\sigma)}\right)\right]$

  $\leq \max_{i \in [1..n]} g_i(C_i(\sigma))$ qui est déjà minimale (par optimalité de σ)

  Donc on a bien un ordonnancement global optimal qui se termine par $J_{k_{j+1}}, J_{k_j}, J_{k_{j-1}} \dots J_{k_1}$, autrement dit K_{j+1} est vraie

- En particulier la propriété K_{n-1} dit qu'il existe un ordonnancem^t opt. finissant par $J_{k_{n-1}}, J_{k_{n-2}} \dots J_{k_1}$, or le seul de cette forme est ici $J_{k_n}, J_{k_{n-1}} \dots J_{k_1}$ et c'est ce que contient la file Res après la n-ième boucle. CQFD.

algo

La propriété précédente affirme qu'on peut décider de mettre J_k à la fin sans craindre de louper l'optimum pourvu que $g_k(P)$ soit minimal.
Il nous reste alors à ordonnancer les tâches $(J_i)_{i \in [\![1..n]\!] \setminus \{k\}}$ dont la somme des durées est non plus P mais $P - p_k$.
Pour trouver cet ordonnancement on recommence le même procédé. Cela donne l'algorithme suivant.

ORDO-GMAX.($(p_i), (g_i)$)

$P \leftarrow \sum_{i=1}^{n} p_i$

$I \leftarrow [\![1..n]\!]$

Res $\leftarrow$ PILE_VIDE.

Pour j allant de 1 à n

  trouver $k \in I$ tq $g_k(P)$ soit minimal

  $P \leftarrow P - p_k$

  $I \leftarrow I \setminus \{k\}$

  Res.ajouter(k)

Retourner Res.

$1\,|\,1\,|\,g_{max} \in P$

$O(n^2)$

complexité

Cet algorithme est en $O(n^2)$ car la recherche du k se fait en $O(n)$, en imaginant qu'évaluer g_i est en $O(1)$ et qu'on gère l'ensemble I comme un tableau de n booléens.

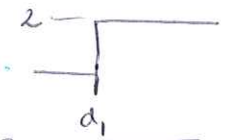
Rq La structure de tas n'apparaît rien ici puisqu'à chaque étape il faut reconstruire tout le tas puisque P change.

ex

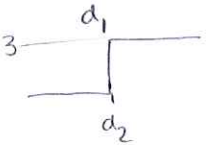
	1	2	3	4	5
p_i	5	3	2	4	3
d_i	10	10	6	13	4

P	17	12 (-5)	8 (-4)	5 (-3)	3 (-2)
$g_1(P)$	2	/	/	/	/
$g_2(P)$	3	3	0	/	/
$g_3(P)$	11	6	2	0	/
$g_4(P)$	5	0	/	/	/
$g_5(P)$	39	24	12	2	0
k	1	4	2	3	5
res	1	4 1	2 4 1		

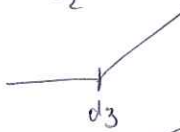
$g_1 = 2\,\mathbb{1}_{[d_1, +\infty[}$ =



$g_2 = 3\,\mathbb{1}_{[d_2, +\infty[}$ =

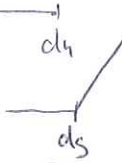


$g_3 = x \mapsto \begin{cases} x - d_3 & \text{si } x > d_3 \\ 0 & \text{sinon} \end{cases}$ =



$g_4 = x \mapsto \begin{cases} 3 + \frac{1}{2}(x - d_4) & \text{si } x \geq d_4 \\ 0 & \text{sinon} \end{cases}$ =

$g_5 = x \mapsto \begin{cases} 3(x - d_5) & \text{si } x > d_5 \\ 0 & \text{sinon} \end{cases}$ =



→ l'ordonnancement optimal est 5 3 2 4 1
il est de $g_{max} = 2 = \max(2, 0, 0, 0, 0)$

cas particulier

$1 \mid\mid L_{max}$

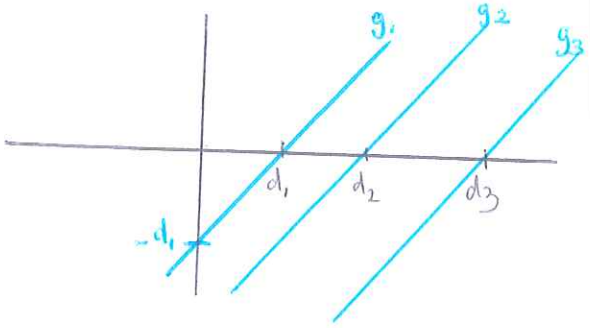
une machine

$\forall i \in [1..n] \quad L_i = C_i - d_i$

$L_{max} = \max_{i \in [1..n]} L_i$

où les (d_i) sont des dates de fin souhaitées fixées.

En posant $\forall i \in [1..n]$, $g_i = x \mapsto x - d_i$
ce problème se réécrit $1 \mid\mid g_{max}$
et grâce à l'algorithme de Lawler on sait résoudre ce problème en temps quadratique. Donc $1 \mid\mid L_{max} \in P$



Mais l'algo. de Lawler, très général, n'exploite pas la structure particulière de ce problème.
En effet ici toutes les fonctions sont linéaires et de même pente, il n'y a donc pas besoin de recalculer tous les $g_i(P)$ à chaque étape pour trouver le minimiseur.
Plus précisément on a $\forall i,j \in [1..n] \quad g_i(0) \leq g_j(0) \Rightarrow \forall x \; g_i(x) \leq g_j(x)$
soit $-d_i \leq -d_j \Rightarrow$
$d_j \leq d_i \Rightarrow \forall P \quad g_i(P) \leq g_j(P)$

En notant σ la permutation classant les tâches par $d_i \nearrow$
(par date de fin au plus tôt = earliest due date = EDD)
règle de Jackson
on a $\forall i,j \in [1..n] \quad i \leq j \Rightarrow d_{\sigma(i)} \leq d_{\sigma(j)} \Rightarrow \forall P \; -g_{\sigma(j)}(P) \leq g_{\sigma(i)}(P)$
$\Rightarrow$ l'algo. de Lawler placera $J_{\sigma(j)}$ après $J_{\sigma(i)}$.
($J_{\sigma(j)}$ sera sélectionné d'abord pour être placé à la fin).

Ainsi $(J_{\sigma(i)})_{i \in [1..n]}$ est la permutation que fournira l'algo. précédent, mais on peut fournir cette permutation en $O(n \log(n))$ par simple tri.

On pourrait montrer la validité de la règle EDD par argument d'échange, mais on s'appuye ici sur la correction de l'algo précédent.

On retiendra $1 \mid\mid L_{max} \in P$ via EDD en $O(n \log(n))$