CALAGE À GAUCHE ET CRITÈRE RÉGULIER

Cadre On considère n tâches notées $(J_i)_{i\in[1..n]}$ de durées $(p_i)_{i\in[1..n]}$.
On va s'intéresser au mode préemptif donc un ordonnancement ne correspond plus de manière bijective à une permutation de $[1..n]$.

Pour un ordonnancement donné on note encore C_i la date de fin de la tâche J_i. Notons qu'ici la tâche J_i peut avoir été exécutée avant $C_i - p_i$, interrompue, puis reprise plus tard, et ce plusieurs fois.

Déf On parlera de critère régulier lorsqu'on cherche un ordo. minimisant $F(C_1, C_2, \dots, C_n)$ où F est une fonction croissante en chacune de ses variables (les autres étant fixées).

ex → $\sum_{i=1}^{n} C_i$ est un critère régulier

car $F = (x_i)_{i\in[1..n]} \longmapsto \sum_{i=1}^{n} x_i$ est croissante en chacune de ses var.

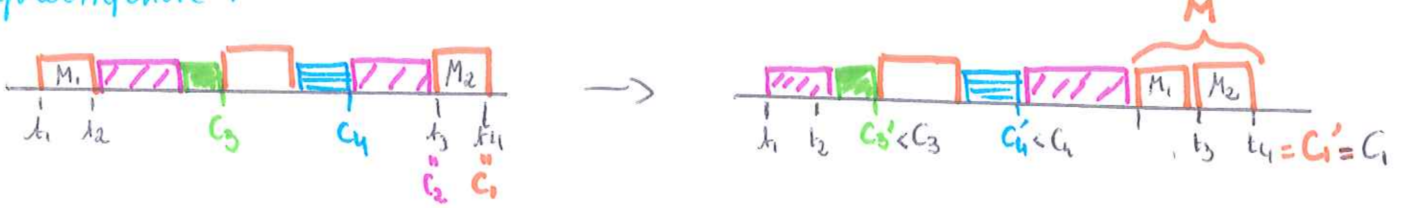
en effet $F(x_1, x_2, \dots x_{i-1}, \bullet, x_{i+1}, \dots, x_n)$ est $\nearrow$, et ce $\forall x \in \mathbb{R}^n$.

→ $\max_{i\in[1..n]} g_i(C_i)$ où $\forall i \in [1..n]$, $g_i \nearrow$. est aussi un critère rég.

Pté Un problème à une machine, sans date de début au plus tôt et dont le critère est régulier, admet toujours un ordonnancement optimal sans temps mort et sans préemption.

Preuve Si on a un ordonnancem^t optimal qui présente un temps mort entre t_1 et t_2, c-à-d que la machine n'est pas utilisée dans cet intervalle de temps, on peut décaler toute la partie de l'ordonnancem^t après t_2 de $t_2 - t_1$; ainsi les C_i concernés seront moindre, donc F va diminuer (car $F \nearrow$). On obtient ainsi un ordo. encore optimal.
En itérant on obtient donc un ordo. optimal sans temps mort.

Si on a un ordo optimal qui présente une tâche J_i morcelée = préemptée, elle présente au moins un morceau final M_2 exécuté entre t_3 et t_4, ainsi $C_i = t_4$ et un autre morceau, exécuté auparavant M_1 entre t_1 et t_2.
Ici implicitement $t_1 < t_2 < t_3 < t_4$.
En décalant ce qui se passe entre t_2 et t_3 entre t_1 et $t_3 - (t_2 - t_1)$, et M_1 entre $t_3 - (t_2 - t_1)$ et t_3 on a
→ diminué (au sens large) les C_j pour $j \neq i$
→ laissé inchangé $C_i = t_4$
→ rassemblé les morceaux M_1 et M_2 en un seul.
En itérant on peut rassembler chaque tâche en un seul morceau sans augmenter F (car les $C_j \searrow$) et donc obtenir un ordo. optimal sans préemption.

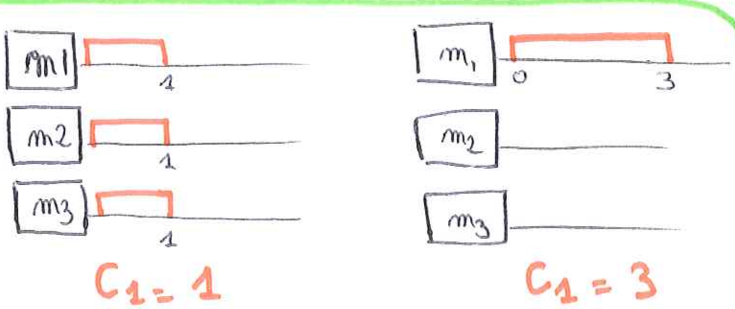


⚠ Dans le cas où il y a plusieurs machines ou dans le cas où il y a des dates de début au plus tôt (les r_i non tous égaux) ce n'est plus vrai

A ex

$m = 3$ machines.
$n = 1$ $p_1 = 3$

$F = \max_{i \in [1,n]} C_i$
$= C_1$

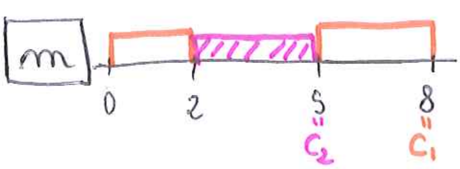


Ici le mode non préemptif est strictement moins bon que le mode préemptif

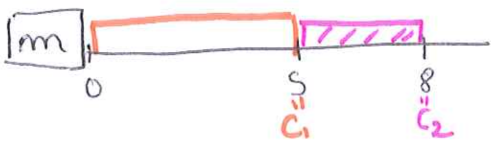
B ex

$m = 1$ machine
$n = 2$ $p_1 = 5$ $d_1 = 7$ $r_1 = 0$
$p_2 = 3$ $d_2 = 5$ $r_2 = 2$

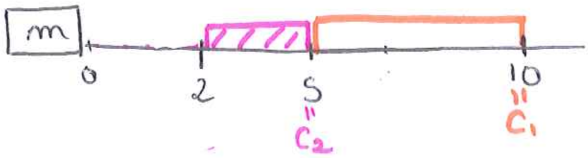
$F = \max_{i \in [1,n]} (C_i - d_i)$



$F = \max(8-7, 5-5)$
$= 1$



$F = \max(5-7 ; 8-5) = 3$



$F = \max(10-7, 5-5) = 3$

Ici l'optimum n'est atteint qu'en mode préemptif

Pté

Un ordonnancement sans temps mort et sans préemption correspond de manière unique à une séquence c-à-d un ordre ou une permutation des tâches.

On obtient l'ordonnancement à partir de la séquence par calage à gauche (comme le fait l'algo ci-dessous)

algo 1

Ordo (n, P, L)

- n = nombre de tâches
- P contient les $(p_i)_{i\in[1..n]}$
- L est la liste des tâches dans l'ordre.

```
Ordo (n, P, L)
  IND, D, J tableaux indexés par [1..n]
  F         tableau indexé par [0..n]
  F[0] <- 0
  Pour i de 1 à n
     IND[L.TÊTE()] = i
     L <- L.SUCC()
  Pour k de 1 à n
     D[k] <- F[k-1]
     J[k] <- IND[k]
     F[k] <- D[k] + P[IND[k]]
  retourner D, F[1..n], J.
```

CCl 1

Pour un problème à une machine, sans date de début au plutôt et dont le critère est régulier on pourra toujours se contenter de donner une séquence pour décrire l'ordonnancement optimal

⚠ En mode non préemptif, avec des dates de début au plus tôt, des temps morts peuvent apparaître, même si des tâches sont disponibles

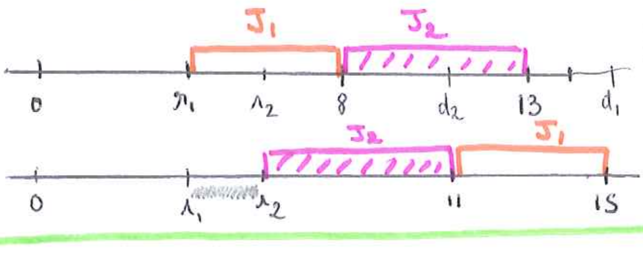
ex

$n = 2$ $\quad r_1 = 4$ $\quad p_1 = 4$ $\quad d_1 = 15$ $\quad$ on cherche à minimiser L_{max}.

$\quad\quad\ \ r_2 = 6$ $\quad p_2 = 6$ $\quad d_2 = 11$

sans temps morts non forcés $\quad L_{max} = L_2 = 2$

avec temps mort non forcé $\quad L_{max} = 0$



Malgré ces temps d'inactivité non forcés par les r_i, c-à-d alors qu'il y a des tâches disponibles, on peut quand même définir un ordonnancement à partir d'une séquence par calage à gauche avec date de début. C'est ce que fournit l'algo ci-dessous.

algo 2

Ordo (n, P, R, L)

- n = nombre de tache
- P contient les $(p_i)_{i \in [1..n]}$
- R ——— $(r_i)_{i \in [1..n]}$
- L définit l'ordre des tâches, la séquence.

```
IND, D, J  tableaux indexés par [1..n]
F  tableau indexé par [0..n]
F[0] ← 0

Pour i de 1 à n
    IND[L.TÊTE()] ← i
    L ← L.SUCC()

Pour k de 1 à n
    D[k] ← max(F[k-1], R[IND[k]])
    J[k] ← IND[k]
    F[k] ← D[k] + P[IND[k]]

retourner D, J, F[1..n]
```

Pté Pour un problème à une machine, en mode non préemptif, avec dates de début et dont le critère est régulier, il existe un ordonnancement optimal obtenu par calage à gauche d'une séquence.

Preuve On considère O un ordonnancem^t optimal pr un tel problème.
On peut alors considérer L la listes des tâches dans l'ordre où elles sont exécutées dans O. On note D et F les tableaux obtenus par l'algo préc, c'est-à-dire les dates de début et de fin des tâches dans l'ordo obtenu par calage à gauche à partir de L.
On suppose que O n'est pas cet ordonnancem^t.
On considère alors k minimal tq la k-ième tâche de O ne s'effectue pas entre D[k] et F[k]. Disons qu'elle commence plutôt en $d \neq D[k]$.
Néc $d > D[k]$ puisque la k-ième tache ne peut commencer avant la fin de la préc (ie $d \geq F[k-1]$) ni avant sa date de début au plus tôt (ie $d \geq R[IND[k]]$)
En avançant toutes les tâches à partir de la kième de $d - D[k]$ dans O on obtient un ordonnancem^t encore optimal par régularité du critère (on fait ↘ les C_i donc ↘ le critère à minimiser)
En itérant ainsi on obtient bien un ordo optimal obtenu par calage à gauche.

CCl 2 En mode non préemptif, pour un problème à une machine avec date de début et un critère régulier, on pourra toujours se contenter de donner une séquence pour décrire l'ordonnancement optimal.

⚠ Lorsqu'on a des dates de début, autoriser la préemption peut permettre d'avoir un ordonnancement optimal strictement meilleur (Cf exemple B)

↳ En mode préemptif avec dates de début on ne pourra pas se contenter de considérer les ordonnancements obtenus par calage à gauche, puisque ceux-ci ne profitent jamais de la préemption.
En fait ils introduisent des temps d'inactivité alors qu'il y a des tâches disponibles, seulement parce qu'elles sont moins prioritaires.
Au contraire ici on va profiter de la préemption et toujours exécuter une tâche s'il y en a de disponible, quitte à l'interrompre quand une tâche plus prioritaire devient disponible, et à la reprendre plus tard.
Un tel ordonnancement est appelé ordonnancement de liste, la liste étant, comme la séquence pour le calage à gauche, ce qui définit l'ordre sur les tâches, qu'on a vu ici comme un ordre de priorité.

C'est que fournit l'algo ci-dessous.

algo 3

```
Ordo (n, P, R, L)
```

n = nombre total de tâches.
P contient les $(p_i)_{i \in [1..n]}$
R ——— $(r_i)_{i \in [1..n]}$
L définit l'ordre sur les tâches.

```
F, D, J tableaux indexés par [0..n] dynamiques
IND tableau indexé par [1..n]
F[0] ← 0

Pour i allant de 1 à n
    IND[L.TETE()] ← i
    L ← L.SUCC(L)
```

A j, L associe facilement L[j] la jème tâche la + prio.
A l'inverse, IND associe à une tâche son indice de priorité, ou plutôt à i le j tq J_i est la j-ème tâche ds L.

```
U, T ← TAS_MIN_VIDE().
Pour i allant de 1 à n
    U.AJOUTER(i, R[i])
```

T sera le tas des tâches disponibles, avec pour clé leur indice de priorité. On le remplira au fur et à mesure.
U est le tas des tâches restant à considérer, avec pour clé l'instant à partir duquel elles sont disponibles.

```
k ← 1 ; r ← R[U.MIN()]
```

r = la prochaine (ici la 1ère) date à laquelle des tâches deviennent disponibles

```
Tant que U.EST_NON_VIDE()
    Tant que U.EST_NON_VIDE() et R[U.MIN()] = r
        j ← U.EXTRAIRE_MIN()
        T.AJOUTER(j, IND[j])
```

on considère toutes les tâches qui deviennent disponibles au temps r (ie de clé r dans U), et on les ajoute au tas T des tâches disponibles selon leur indice de priorité.

```
    Si U.EST_VIDE()
    alors r +∞
    sinon r ← R[U.MIN()]
```

On met à jour r la prochaine date à laquelle des nouvelles tâches deviennent disponibles. Si toutes le sont déjà, U est vide et on considère r = +∞.

```
    Tant que T.EST_NON_VIDE() et F[k-1] < r
        i ← T.MIN()
```

→ la tâche qu'on va exécuter est J_i : la + prioritaire des disponibles

```
        D[k] ← max(R[i] ; F[k-1])
        J[k] ← i
        F[k] ← min(D[k] + P[i], r)
```

→ On "crée" un bloc qui correspond à l'exécution de la tâche $J[k] = J_i$ entre les temps D[k] et F[k].

```
        P[i] ← P[i] - (F[k] - D[k])
        Si P[i] = 0
        alors T.EXTRAIRE_MIN()
        k ← k+1
retourner D[1..k-1] ; J[1..k-1], F[1..k-1]
```

On met à jour P[i], la durée restant à effectuer sur la tâche i, et s'il reste rien on supprime la tâche J_i de T.
} On met à jour k l'indice du bloc.

ex $n = 5$

i	1	2	3	4	5
p_i	12	8	10	8	6
r_i	15	19	1	19	40

IND 4 2 5 3 1

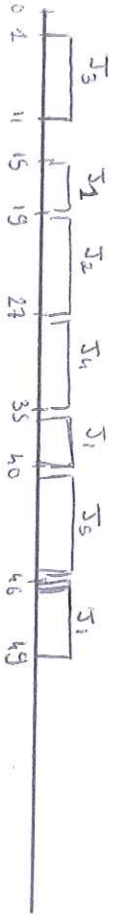
$L = 5\ 2\ 4\ 1\ 3$

$U =$ 3^1 ; 1^{15}, 2^{19} ; 4^{19}, 5^{40}

exécution du l'algo

résultat de l'algo

	D	J	F
1	1	3	11
2	15	1	19
3	19	2	27
4	27	4	35
5	35	1	40
6	40	5	46
7	46	1	49



$k = 1$

$r = R[3] = 1$

$T = 3^5$ $U =$ 1^{15} ; 4^{19}, 2^{19} ; 5^{40}

$r = R[1] = 15$

$i = 3$ $D[1] = \max(1, F[0])$ $= 1$ ($F[0] = 0$) $J[1] = 3$ $F[1] = \min(1 + 10, 15) = 11$

$P[3] = 0$ $T = \emptyset$ $k = 2$

$T = 1^4$ $U =$ 4^{19} ; 5^{40}, 2^{19}

$r = R[5] = 19$

$i = 1$ $D[2] = \max(15, 11) = 15$ $J[2] = 1$ $F[2] = \min(15 + 12, 19) = 19$

$P[1] = 12 - (19 - 15) = 8$ T reste 1^4 $k = 3$

$T =$ 4^3 ; 1^4 $U =$ 2^{19} ; 5^{40}

$T =$ 2^2 ; 1^4, 4^3 $U = 5^{40}$

$r = R[5] = 40$

$i = 2$ $D[3] = \max(19, 19) = 19$ $J[3] = 2$ $F[3] = \min(19 + 8, 40) = 27$

$P[2] = 0$ $T =$ 4^3 ; 1^4 $k = 4$

$i = 4$ $D[4] = \max(19, 27) = 27$ $J[4] = 4$ $F[4] = \min(27 + 8, 40) = 35$

$P[4] = 0$ $T = 1^4$ $k = 5$

$i = 1$ $D[5] = \max(15, 35) = 35$ $J[5] = 1$ $F[5] = \min(35 + 8, 40) = 40$

$P[1] = 8 - (40 - 35) = 3$ T reste 1^4 $k = 6$

$T =$ 5^1 ; 1^4 $U = \emptyset$

$r = +\infty$

$i = 5$ $D[6] = \max(40, 40) = 40$ $J[6] = 5$ $F[6] = \min(40 + 6, +\infty) = 46$ $\begin{cases} P[5] = 0 \\ T = 1^4 \\ k = 7 \end{cases}$

$i = 1$ $D[7] = \max(15, 46) = 46$ $J[7] = 1$ $F[7] = \min(46 + 3, +\infty) = 49$ $P[1] = 0$ $T = \emptyset$. $k = 8$.