UN PROBLÈME D'ORDO. À UNE MACHINE NP-COMPLET

Intro

On a vu que : $1 \mid \_ \mid L_{max} \in P$ via EDD

$1 \mid r_i, empt \mid L_{max} \in P$ via EDD

une seule machine — date de début au plus tôt — mode préemptif — max L_i où $L_i = C_i - d_i$ où d_i est une date de fin souhaitée fixée

On a deux problèmes "faciles".
Dans le premier cas c'est facile car on n'a pas de contraintes, il suffit de traiter en priorité le plus urgent.
Dans le second cas les contraintes apportent a priori de la difficulté, mais le mode préemptif, qui autorise le découpage "gratuit" des tâches, contourne cette difficulté : on fait le plus urgent parmi les tâches disponibles, sachant que si une tâche + urgente arrive on pourra s'interrompre pour la traiter.

On va voir ici que sans le mode préemptif les contraintes de dates de début au plus tôt rendent le problème NP-Complet.

idée : + contraintes ⇒ + dur
+ préemptif ⇒ + facile

Pté

$1 \mid r_i \mid L_{max}$ est NP-complet.

Si on veut être exact c'est le problème de décision associé à ce problème d'optimisation qui est NP complet, on va le noter (P)

P

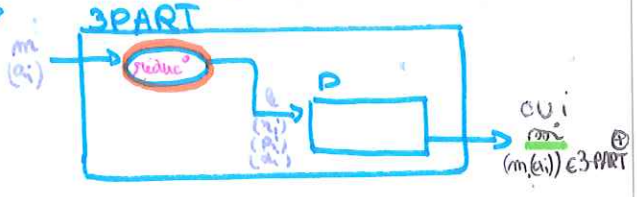
Entrée
- $(r_i)_{i \in [1..n]}$
- $(p_i)_{i \in [1..n]}$
- $(d_i)_{i \in [1..n]}$
- $k \in \mathbb{N}$

Sortie : oui si le L_{max} du pb d'ordo. défini par les durées (p_i), les dates de début au + tôt (r_i), les dates de fin souhaitées (d_i) est inférieur à k.
non sinon

① MQ (P) est NP-dur

Pour MQ un problème est NP-dur on montre qu'il est plus dur qu'un problème dont on sait déjà qu'il est NP-dur : on fait une réduction. Ici on choisit le problème de 3-partition.

Par réduction de 3 PART à P



3 PARTITION

Entrée :
- $m \in \mathbb{N}$
- $(a_i)_{i \in [1..n]}$ où $n = 3m$

Sortie :
- Oui s'il existe $(S_k)_{k \in [1..m]}$ une partition de $[1..n]$ telle que $\forall k \in [1..m] \quad \sum_{i \in S_k} a_i = \sum_{i \in S_l} a_i$
- Non sinon

En fait on travaillera plutôt avec un ss-problème de 3-PARTITION qui est encore NP-complet : 3-PART

3-PART

Entrée :
- $m \in \mathbb{N}$
- $(a_i)_{i \in [1..n]} \in \mathbb{N}^n$ où $n = 3m$ vérifiant
  - → il existe $B \in \mathbb{N}$ tq $\sum_{i=1}^{n} a_i = B \times m$
  - → $\forall i \in [1..n], \ a_i \in]B/4, B/2[$

Sortie :
- oui s'il existe $(S_k)_{k \in [1..m]}$ une partition de $[1..n]$ telle que $\forall k \in [1..m], \ \sum_{i \in S_k} a_i = B$
- non sinon

On considère $m, (a_i)_{i \in [1..n]}$ une instance de 3-PART
On sait donc qu'il existe $B \in \mathbb{N}$ tq $\begin{cases} \sum_{i=1}^{n} a_i = B \times m \\ \forall i \in [1..n], \ a_i \in]B/4, B/2[\end{cases}$

Il nous faut ici construire une instance de (P) qui sera positive ssi $(m, (a_i))$ est une instance positive de 3-PART.

Assez naturellement on fixe k à 0 ce qui revient à décider si les tâches peuvent être toutes réalisées à l'heure, ce qui transforme en quelques sortes la contrainte molle des d_i (due date) en contrainte dure (dead-line).

↳ On pose $k = 0$.

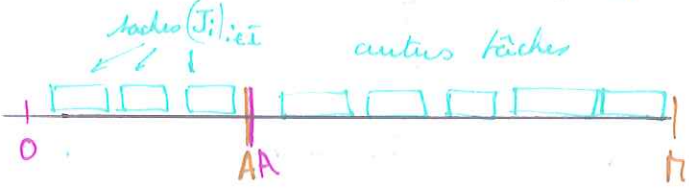
On voit alors la tâche i comme un bloc, de taille p_i qu'on peut placer entre r_i et d_i.

En fixant $p_i = a_i$ cela permet d'exprimer des contraintes sur la somme des a_i, en fixant par exemple $\forall i, r_i = 0$, $\forall i, d_i = A$

Pour $i \in [1..n]$ on pose $p_i = a_i$

on impose $\sum_{i=1}^{n} a_i \leq A$.

Si on considère $I \subset [1..n]$ on peut aussi mettre une contrainte sur la somme des a_i pour $i \in I$, en fixant par exemple $\forall i \in I \begin{cases} r_i = 0 \\ d_i = A \end{cases}$ $\forall i \notin I \begin{cases} r_i = A \\ d_i = M \gg A \end{cases}$ ce qui impose $\sum_{i \in I} a_i \leq A$

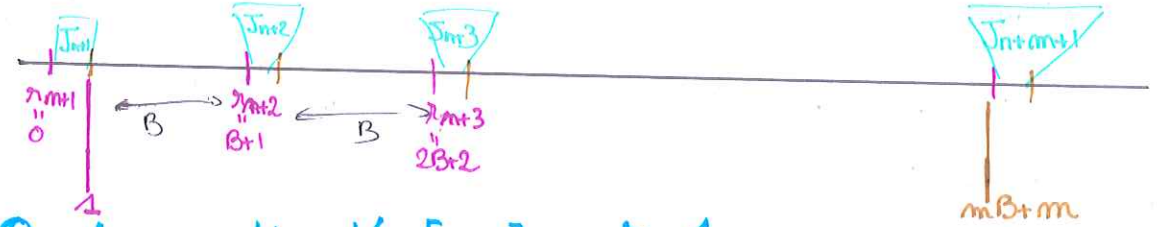


⚠ Cependant il ne **FAUT PAS** considérer une partition et chercher à forcer les $(a_i)_{i \in S_k}$ à rentrer dans un même intervalle $[r^k, d^k]$ de taille B.
En effet la construction de l'instance de P à partir de celle de 3-PART doit se faire pour n'importe quelle instance ($\oplus$ ou $\ominus$) et en temps polynomial. On peut donc faire une disjonction de cas mais à condition que le test soit polynomial, en particulier cela exclut de tester si on a une instance positive ou négative d'un problème NP-complet !!!

Pour coller à l'idée de partition on veut pouvoir séparer les tâches, pour cela on va introduire des tâches qu'on va contraindre à être à une certaine place, on a ainsi des blocs fixes, des blocs séparateurs.

On considère $(m+1)$ tâches unitaires fixées tous les B temps

ie $\forall k \in [1..m+1]$ $p_{n+k} = 1$
$r_{n+k} = (k-1)B + k - 1$
$d_{n+k} = (k-1)B + k$



On fixe enfin $\forall i \in [1..n]$ $r_i = 1$, $d_i = mB + m$

- NQ $\left(\begin{matrix} m \\ (a_i) \end{matrix}\right) \in 3\text{PART}^+ \Rightarrow \left(\begin{matrix} (r_i)_{i \in [1..n+m+1]} \\ (p_i) \\ (d_i) \\ k \end{matrix}\right) \in P^+$ ← ensemble des instances positives du problème P c'à-d pour lesquelles la réponse est OUI

Supposons donc avoir une partition $(S_k)_{k \in [1..m]}$ de $[1..n]$ tel que $\forall k \in [1..m]$ $\sum_{i \in S_k} a_i = B$.

On peut alors placer, peu importe l'ordre, les tâches $(J_i)_{i \in S_k}$ entre les tâches J_{n+k} et J_{n+k+1} qui sont espacées de B, et ce pour tout $k \in [1..m]$. Ainsi on a un ordonnancement dans lequel toutes les tâches sont réalisées à temps donc $L_{max} \leq 0$ (mais puisque les tâches séparatrices sont réalisées juste à temps $L_{max} = 0$)

Ainsi $\begin{pmatrix} (r_i) \\ (p_i) \\ (d_i) \\ k \end{pmatrix}$ est une instance positive de (P). □

- MQ $\left(\begin{pmatrix} (r_i)_{i \in [1..n+m+1]} \\ (p_i) \\ (d_i) \\ k \end{pmatrix} \in P^+ \right) \Rightarrow \left(\begin{pmatrix} (a_i)_{i \in [1..n]} \\ m \end{pmatrix} \in 3\text{-PART}^+ \right)$

Supposons donc avoir un ordonnancement qui satisfait les contraintes de fenêtre de temps $[r_i, d_i]$.
On considère alors, pour $k \in [1..m]$, $S_k = \{ i \in [1..n] \mid J_i$ est effectuée entre J_{n+k} et $J_{n+k+1} \}$
Clairement $(S_k)_{k \in [1..m]}$ est une partition de $[1..n]$
puisque les contraintes imposent que les tâches $(J_i)_{i \in [1..n]}$ sont toutes réalisées entre deux tâches séparatrices.
De plus on a aussi $\forall k \in [1..m]$, $\sum_{i \in S_k} a_i = \sum_{i \in S_k} p_i \leq r_{n+k+1} - d_{n+k} = B$ (B)
□□□

donc $MB = \sum_{i \in [1..n]} a_i = \sum_{k=1}^{m} \sum_{i \in S_k} a_i \leq mB$.

Puisque tous les a_i sont > 0, cela implique nécessairement $\forall k \in [1..m]$ $\sum_{i \in S_k} a_i = B$.
Donc $((a_i)_{i \in [1..n]}, m)$ est une instance positive de 3-PART. □

En conclusion 3-PART se réduit à (P) donc (P) est NP-dur

② MQ $(P) \in NP$

Pour MQ ce problème est NP il suffit de pouvoir vérifier en temps polynomial qu'une solution qu'on nous propose est admissible.

Supposons qu'on nous propose une partition de $[1..n]$ sous la forme d'un tableau indexé par $[1..n]$ et rempli (aléatoirement ou par un oracle) de valeur entre 1 et m pour signifier l'appartenance à une classe (ie $T[i] = k$ signifie $i \in S_k$)
Il suffit de parcourir ce tableau muni de m compteurs, $(C_k)_{k \in [1..m]}$
et d'ajouter a_i à C_k dès qu'il y a k dans la case d'indice i.
Ainsi en temps linéaire on obtient $C_k = \sum_{i \in S_k} a_i$, reste alors à tester les m^2 égalités entre ces $(C_k)_{k \in [1..m]}$. La vérification est donc polynomiale $\mathcal{O}(n+m^2)$
↳ En conclusion $P \in NP$.