FALQ Anne-Elisabeth Avril-Août 2017

Rapport de stage

Approche polyédrale pour les problèmes d'ordonnancement juste-à-temps à une machine

Encadré par Safia Kedad-Sidhoum et Pierre Fouilhoux

Préface

Ce document tend à présenter en une vingtaine de pages le stage de 5 mois que j'ai effectué au LIP6 (Laboratoire d'informatique de Paris 6), sous la direction de Safia Kedad-Sidhoum et Pierre Fouilhoux. Il s'agit d'un stage de master 2, qui entre dans le cadre du master mathématiques de la modélisation de l'UPMC que j'ai suivi cette année.

Le but de ce stage était d'étudier les résultats d'ordonnancement sur les problèmes juste-à-temps d'une part, et les résultats sur les polyèdres d'ordonnancement d'autre part, afin de pouvoir étudier puis résoudre les problèmes juste-à-temps par une approche polyédrale. Dans l'espoir d'obtenir une caractérisation complète du polyèdre des solutions pour un problème polynomial, on s'est intéressé en particulier aux problèmes juste-à-temps à une machine, dont plusieurs sont réputés polynomiaux. On s'est finalement focalisé sur le problème à une machine avec date d'échéance commune non restrictive, dont on sait qu'il est polynomial ou NP-difficile selon les hypothèses que l'on met sur les coûts d'avance et de retard.

Sommaire

État de l'art sur l'approche polyédrale du problème de minimisation de la somme pondérée des encours

Étude polyédrale d'un problème d'ordonnancement une machine avec coûts d'avance et de retard

Conclusion

ANNEXES

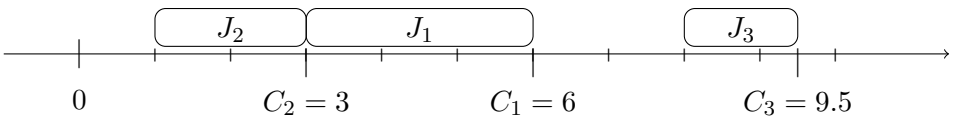
1 Introduction aux problèmes d'ordonnancement à une machine avec coûts d'avance et de retard

Il s'agit d'un problème d'ordonnancement à **une machine**, c'est-à-dire qu'on veut exécuter un ensemble de tâches J sur une machine qui ne peut réaliser qu'une seule tâche à la fois. De plus les tâches ne peuvent être interrompues en cours exécution, il faut les faire d'une traite : on dit qu'on est en mode **non préemptif**. Le temps d'exécution dépend de la tâche, mais pas de l'instant où elle est exécutée, ni de l'ordre dans lequel sont effectuées les tâches. Autrement dit notre machine est infatigable, imperturbable, une vraie machine quoi ! On note $\boldsymbol{p_j}$ cette **durée d'exécution** pour chaque tâche $j \in J$. On supposera cette durée positive, et même strictement positive car une tâche instantanée pourrait être intercalée n'importe où.

Le but est donc de caler ces tâches dans le temps et formellement un **ordonnancement** est la donnée des périodes d'exécution de chacune des tâches. Puisqu'on connaît les durées des tâches, on peut se contenter de donner les **dates de fin** de chaque tâche, qu'on notera usuellement $(\boldsymbol{C_j})_{j\in J}$ (C pour "completion time" en anglais). Ce sera notre premier codage, notre première façon de décrire un ordonnancement réel de manière non ambiguë, mais on utilisera aussi d'autres codages par la suite.

Exemple :

i	p_i
1	3
2	2
3	1.5



NB : une hypothèse implicite mais importante est que l'on ne peut remonter dans le temps avant 0. Si je fais un planning maintenant je ne peux dire que le mieux c'est d'avoir déjà fait mes devoirs hier !

Plus précisément le problème est de trouver un ordonnancement qui minimise une certaine fonction coût, comme par exemple $\sum_{j\in J} C_j$. Ce problème, qui a notamment été étudié avec une approche polyédrale par Queyranne [12] (Cf section 5), a le bon goût d'être **régulier**, c'est-à-dire que les solutions où les tâches sont faites au plus tôt sont toujours meilleures (au sens large).

On va s'intéresser quant à nous à des **coûts d'avance-retard**, c'est-à-dire qu'on doit finir chaque tâche $j \in J$ à une date $\boldsymbol{d_j}$ dite date d'échéance ou **due-date**. Ce n'est pas une deadline : on peut néanmoins finir la tâche j après d_j, mais il faudra alors payer une **pénalité de retard**, proportionnelle au retard. Là où c'est un peu plus surprenant, c'est que si on finit avant la due-date on devra aussi payer une pénalité, proportionnelle à l'avance. D'un point de vue pratique cela peut modéliser les coûts de stockage qu'engendrent les produits finis avant leur date de livraison. On parle alors de problème **juste-à-temps**, car en dehors du cas où on finit pile à l'heure, on a une pénalité. La pénalisation de l'avance est une hypothèse importante car elle **casse la régularité** du problème, on perd la dominance des ordonnancements tassés à gauche c'est-à-dire qu'on ne peut plus caler les tâches le plus tôt possible au risque de fournir une solution non optimale.

On commence par l'état de l'art que l'on a séparé en deux parties tant les points de vue diffèrent. En effet, dans la première partie on fait un tour d'horizon des problèmes d'ordonnancement juste-à-temps à une machine, puis on se concentre sur deux articles où la due-date est commune et non restrictive, tandis que dans la deuxième partie on détaille l'article de Queyranne [12] qui traite un problème régulier par une approche polyédrale. Dans la troisième partie on présente notre travail qui fait en quelque sorte la synthèse puisqu'on a étudié le problème juste-à-temps avec due-date commune non-restrictive par une approche polyédrale similaire à celle de Queyranne. On joint en annexes quelques définitions utiles, un catalogue de contre-exemples ainsi qu'un résumé des notations utilisées.

État de l'art des problèmes d'ordonnancement juste-à-temps à une machine

2 Cartographie des problèmes

2.1 Expression du coût et codage des ordonnancements

On introduit considère chaque tâche $j \in J$, $\boldsymbol{\alpha_j}$ et $\boldsymbol{\beta_j}$, les **coûts unitaires d'avance et de retard**. Ainsi pour le codage par les C_i, le coût global de l'ordonnancement s'écrit :

$$\boxed{\sum_{j \in J} \alpha_j [d_j - C_j]^+ + \beta_j [C_j - d_j]^+} \quad \text{où } [t]^+ \text{ désigne la partie positive d'un réel } t \text{ i.e } [t]^+ = \max(0, t)$$

La quantité $[d_j - C_j]^+$ représente l'**avance**, elle est toujours positive et nulle si la tâche est en retard (ou à l'heure) : on la note $\boldsymbol{e_j}$ (e pour "earliness"). Symétriquement $[C_j - d_j]^+$ représente le **retard**, elle est aussi toujours positive, mais nulle si la tâche est en avance (ou à l'heure) : on la note $\boldsymbol{t_j}$ (t pour "tardiness").

La donnée de ces deux quantités pour chaque tâche caractérise un ordonnancement, et constitue un codage des ordonnancements plus adapté aux problèmes juste-à-temps. En effet le coût s'exprime alors plus facilement, et surtout de manière linéaire : $\boxed{\sum_{j \in J} \alpha_j e_j + \beta_j t_j}$

2.2 Différentes hypothèses

Le problème d'ordonnancement juste-à-temps qu'on a décrit dans sa version la plus générale, se décline en une multitude de sous-problèmes lorsqu'on ajoute des hypothèses restrictives sur les instances considérées.

On peut ajouter des **hypothèses** sur la due date
- unicité de la due date i.e. $\forall j \in J,\ d_j = d$
- restrictivité/non restrictivité de la due date [1] i.e. $d < p(J)$/ $d \geqslant p(J)$ [2]

sur les poids
- symétrie i.e. $\alpha_j = \beta_j$ (qu'on note alors w_j)
- indépendance i.e. $\forall j \in J,\ \alpha_j = \alpha$ et $\beta_j = \beta$

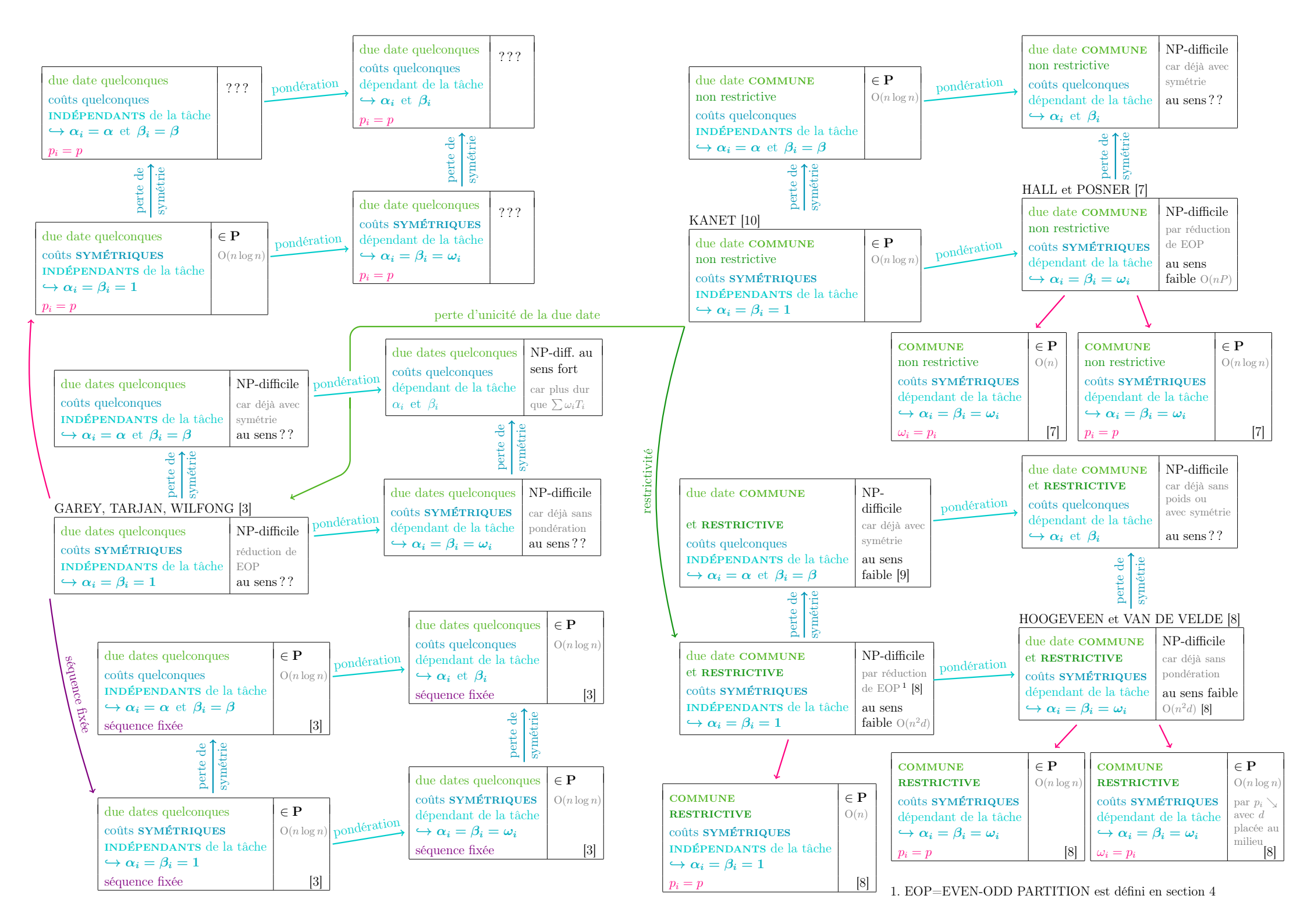
sur les durées
- tâches de même durée i.e. $\forall j \in J,\ p_j = p$
- $p_j = w_j$

: On peut avoir l'impression aux titres des articles qu'ils traitent parfois le cas général d'une due-date quelconque, mais les articles cités ici traitent soit le cas restrictif, soit le cas non restrictif. Le cas général est en fait aussi dur que le cas restrictif.

On peut aussi changer de problème en ajoutant des **contraintes** de précédence, pour retrouver d'autres cas polynomiaux, comme c'est le cas avec la contrainte séquence fixée

Le schéma ci-après résume les principaux problèmes d'ordonnancement juste-à-temps. Conçu comme un genre de cartographie, on part du problème classique de Kanet [10] (due date non restrictive, poids tous égaux, donc supposés égaux à 1) et on se déplace de problème en problème en suivant les flèches. La première démarche de déplacement est d'enlever une hypothèse forte (flèches vertes), ce qui nous ramène à un problème plus général et NP-difficile. On cherche ensuite les sous-problèmes qui peuvent êtres polynomiaux en ajoutant d'autres hypothèses (flèches roses) ou contraintes (flèches violettes). On présente les quatre variations de chaque problème obtenues par changement d'hypothèses sur les coûts (flèches turquoises).

1. La restrictivité n'a vraiment de sens que lorsque la due-date est unique (i.e. commune)
2. cf. notations page 32

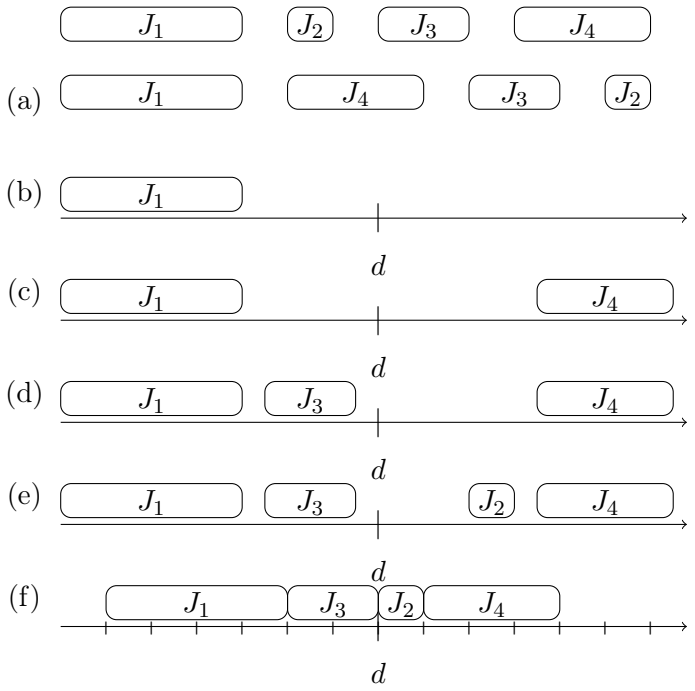


1. EOP=EVEN-ODD PARTITION est défini en section 4

3 Résultats fondateurs de Kanet [10] et dominances

3.1 Idée de l'algorithme de Kanet

Kanet [10] a étudié le problème d'ordonnancement juste-à-temps autour d'une **due-date commune** et **non restrictive**, pour des pénalités d'avance et de retards toutes égales (on prend alors $\forall j \in J$, $\boldsymbol{\alpha_j = \beta_j = 1}$). Il a montré que ce problème est polynomial et se résout en **O(n log(n))** grâce à un tri des tâches par durées décroissantes (a), puis à une distribution des tâches de part et d'autre de la due date, de l'extérieur vers l'intérieur. Dans l'idée, on place la tâche la plus longue en première position (b), puis la deuxième plus longue en dernière position (c), puis la troisième plus longue en deuxième position (d) et ainsi de suite (e), puis on resserre tout autour de la due date (f).



3.2 Idée de la preuve de validité de l'algorithme

Beaucoup des propriétés qui nous intéressent, tant dans cette partie que par la suite, donnent des conditions nécessaires ou des conditions suffisantes pour qu'un ordonnancement soit optimal. Afin de bien distinguer les deux,d'être aussi précis que possible et d'écrire sous un même formalisme les résultats des différents articles, on utilisera les concepts de dominance stricte et dominance large :

Si **tous** les ordonnancements optimaux vérifient une propriété `p`,
on dit qu'on a **dominance stricte** des ordonnancements vérifiant `p`

optimalité ⇒ `p`

Si **au moins un** ordonnancement optimal vérifie la propriété `p`
on dit alors qu'on a **dominance large** des ordonnancements vérifiant `p`

optimalité ⇏ `p`

Dans les deux cas on peut se contenter de chercher l'optimum parmi les ordonnancements vérifiant `p`, et c'est tout l'intérêt des dominances. Ici en particulier, la preuve de validité de l'algorithme de Kanet repose sur deux propriétés de dominance qui nous permettent de nous ramener à un simple problème d'affectation. Pour les énoncer on introduit deux nouvelles notations.

Notations 3.1

L'ordonnancement $\mathcal{S}$ vérifie ▭ ⇔ *il est sans trou*

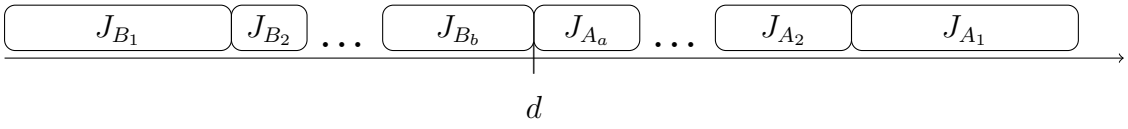
🕐 ⇔ *il admet une tâche qui finit à l'heure*

Propriété 3.2

Pour le problème de Kanet, on a dominance stricte des ordonnancements ▭
dominance large des ordonnancements 🕐

Idée de preuve : Si un ordonnancement présente un trou on peut, selon que l'on est à droite ou à gauche de la due-date, avancer ou retarder (strictement) une tâche de manière à la rapprocher (strictement) de d et ainsi réduire (strictement) les pénalités qu'elle génère. D'où la première dominance (stricte).
Si un ordonnancement optimal présente une tâche t à cheval sur d, il constitue nécessairement un bloc (d'après le premier point) et par optimalité le poids (donc ici le nombre) des tâches précédant t est le même que celui des tâches suivant t (t comprise). On peut donc décider d'avancer (ou de retarder) tout le bloc pour faire finir (ou commencer) t à la date d, et ce sans changer le coût de l'ordonnancement. On se ramène ainsi à un ordonnancement optimal et 🕐. □

Grâce à ces dominances on décide de chercher un optimum ▭▭ et ⏲, qui est donc de la forme suivante :



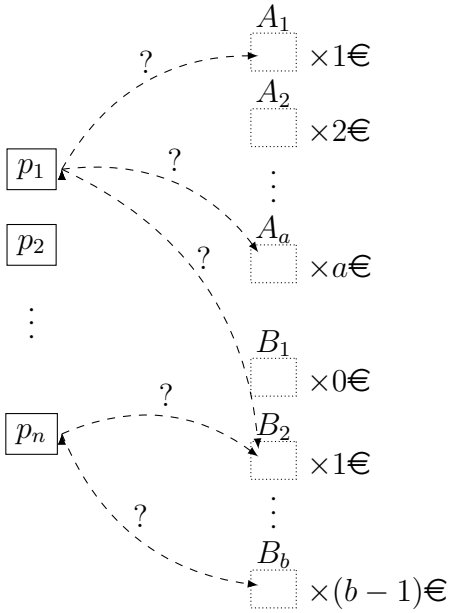
En notant comme ci-dessus $B_1, B_2, \dots B_b$ les indices des tâches en avance (B pour "before"), et $A_1, A_2, \dots A_a$ ceux des tâches en retard (A pour "after"), le coût s'écrit $\sum_{k=1}^{a} k\ p_{A_k} + \sum_{k=1}^{b} (k-1)\ p_{B_k}$

Quand a et b sont fixés, c'est-à-dire quand on impose le nombre de tâche en avance (resp. en retard), choisir un ordonnancement revient à choisir qui sont les $(A_k)_{k\in[1..a]}$ et les $(B_k)_{k\in[1..b]}$ parmi $[1..n]$, et ce de manière bijective. Cela qui revient à distribuer les durées des tâches devant les différents coefficients correspondant. Il est alors naturel (et optimal !) d'affecter toujours la plus grande durée restante au plus petit coefficient encore disponible. Formellement, en notant σ un tri [3] des tâches par durées décroissantes on considère l'ordonnancement où :

$$\begin{cases} \forall k \in [1..b],\ \ B_k = \sigma(2k-1) \\ \forall k \in [1..a],\ \ A_k = \sigma(2k) \end{cases}$$

qui est de coût :

$$\begin{aligned} f(a,b) = \ & 0\times p_{\sigma(1)} + 1\times p_{\sigma(3)} + \dots + (b-1)\times p_{\sigma(2b-1)} \\ & + 1\times p_{\sigma(2)} + \dots + \quad \dots \quad + a\times p_{\sigma(2a)} \end{aligned}$$



On voit facilement que cette fonction de a et b est minimale si a et b sont égaux quand n est pair, et si $b = a+1$ quand n est impair, sans quoi on se prive de petits coefficients [4]. En résumé :

$$\begin{aligned} \min_{\mathcal{S}\text{ ordo}} \text{coût}(\mathcal{S}) &= \min_{\substack{a,b \\ a+b=n}} \ \min_{\substack{\mathcal{S}\text{ ordo} \\ \#E(\mathcal{S})=a \\ \#T(\mathcal{S})=b}} \text{coût}(\mathcal{S}) \\ &= \min_{\substack{a,b \\ a+b=n}} \ \begin{aligned} & 0\times p_{\sigma(1)} + 1\times p_{\sigma(3)} + \dots + (b-1)\times p_{\sigma(2b-1)} \\ & + 1\times p_{\sigma(2)} + \dots + \quad \dots \quad + a\times p_{\sigma(2a)} \end{aligned} \\ &= \begin{cases} \begin{aligned} & 0\times p_{\sigma(1)} + 1\times p_{\sigma(3)} + \dots + (k-1)\times p_{\sigma(2k-1)} \\ & + 1\times p_{\sigma(2)} + \dots + \quad \dots \quad + k\times p_{\sigma(2k)} \end{aligned} & \text{si } n \text{ est pair, disons } n=2k \\ \begin{aligned} & 0\times p_{\sigma(1)} + 1\times p_{\sigma(3)} + \dots + k\times p_{\sigma(2k+1)} \\ & + 1\times p_{\sigma(2)} + \dots + k\times p_{\sigma(2k)} \end{aligned} & \text{si } n \text{ est impair, disons } n=2k+1 \end{cases} \end{aligned}$$

Comme l'ordonnancement obtenu par l'algorithme de Kanet correspond à ce coût dans les deux cas, il est bien optimal.

Dans la partie suivante on complique le problème en permettant aux pénalités de dépendre de la tâche, c'est le problème étudié par Hall et Posner [7].

3. On entend par là que σ est une permutation de $[1..n]$ qui trie les tâches par durées décroissantes i.e. $p_{\sigma(1)} \geqslant p_{\sigma(2)} \geqslant \dots \geqslant p_{\sigma(n)}$
4. Si vous n'êtes pas convaincus imaginez que $n = 5$, mais $a = 3$ et $b = 2$. On a une place à $0€(B_1)$, deux places à $1€(A_1$ et $B_2)$, puis une place à $2€$et une à $3€(A_2$ et $A_3)$, alors qu'avec $a = 2$ et $b = 3$ on aurait deux places à $2€(A_2$ et $B_3)$, ce qui est plus avantageux.

4 Résultats de Hall et Posner [7]

Dans l'article [7] on s'intéresse au problème juste-à-temps autour d'une due-date **commune** et **non restrictive**, avec des coûts **symétriques** mais **dépendant** de la tâche, qu'on note $(\omega_j)_{j\in J}$.
On suppose dans cette partie que les tâches sont indexées par **ratios $r_j := \omega_j/p_j$ décroissants**.

4.1 Principaux résultats

Pour énoncer les résultats de dominance la première partie, on introduit de nouvelles notations.

Notations 4.1

L'ordonnancement $\mathcal{S}$ vérifie $\nearrow\searrow^r \Leftrightarrow \begin{cases} \textit{les tâches en avance sont classées par ratios croissants} \\ \textit{les tâches en retard sont classées par ratios décroissants} \end{cases}$

$\searrow\nearrow_i \Leftrightarrow \begin{cases} \textit{les tâches en avance sont classées par indices décroissants} \\ \textit{les tâches en retard sont classées par indices croissants} \end{cases}$

Propriété 4.2

Pour le problème de Hall et Posner, on a dominance large des ordonnancements $\square\!\!\square$
dominance large des ordonnancements 🕐
dominance stricte des ordonnancements $\nearrow\searrow^r$
dominance large des ordonnancements $\searrow\nearrow_i$

Remarques : Si les coûts sont non nuls (i.e.$\forall j \in J,\ \omega_j > 0$), alors on a dominance stricte des ordo. $\square\!\!\square$.
Si les ratios sont tous distincts (i.e $\forall (i,j) \in J^<, r_i > r_j$), alors il y a unicité du tri des tâches par ratio décroissant, donc $\nearrow\searrow^r$ équivaut à $\searrow\nearrow_i$, on a alors dominance stricte des ordonnancements $\searrow\nearrow_i$.
Comme dans la propriété 3.2, on peut être plus précis quant à la dominance des ordonnancements 🕐 : en fait on peut toujours se ramener d'un ordonnancements optimal à un ordonnancements optimal 🕐 sans changer l'ordre des tâches. Cela assure la dominance large des ordonnancements $\square\!\!\square$, 🕐 et $\nearrow\searrow^r$ ainsi que des ordonnancements $\square\!\!\square$, 🕐 et $\searrow\nearrow_i$.

Propriété 4.3

Soit $\mathcal{S}$ ordonnancement $\square\!\!\square$ *et* 🕐 .
On note s l'indice de sa première tâche et t celui de la tâche qui finit à l'heure.
***Si** $\mathcal{S}$ est optimal, **alors** on a :*
$$\begin{array}{rcccl} \omega\big(E(\mathcal{S})\big) - \omega_t & \leqslant & \omega\big(T(\mathcal{S})\big) & \leqslant & \omega\big(E(\mathcal{S})\big) + \omega_t \\ p\big(E'(\mathcal{S})\big) - 2p_t & \leqslant & p\big(T(\mathcal{S})\big) & \leqslant & p\big(E'(\mathcal{S})\big) \end{array}$$

Définition 4.4

On définit WEIGTED EARLINESS TARDINESS le problème de décision associé à notre problème d'optimisation :

WET — Entrée : $\boldsymbol{n}$, $(\boldsymbol{p_j})_{j\in[1..n]} \in (\mathbb{N}^*)^n$, $(\boldsymbol{w_j})_{j\in[1..n]} \in (\mathbb{N})^n$, $\boldsymbol{d} \in \mathbb{N}$ tq $d \geqslant p([1..n])$, modélisant respectivement le nombre, les durées, les coûts et la due-date commune de n tâches
$\boldsymbol{y} \in \mathbb{N}$
Sortie : oui s'il existe un ordonnancement de coût $\leqslant y$
non sinon

et le problème de décision EVEN-ODD PARTITION :

EOP — Entrée : $(\boldsymbol{a_i})_{i\in[1..2n]} \in (\mathbb{N}^*)^{2n}$ strictement croissante
Sortie : oui s'il existe une bipartition I/J de $[1..2n]$ tq : $\begin{cases} \sum_{i\in I} a_i = \sum_{j\in J} a_j \\ \forall k \in [1..n], \{2k\text{-}1, 2k\} \cap I \neq \emptyset \\ \forall k \in [1..n], \{2k\text{-}1, 2k\} \cap J \neq \emptyset \end{cases}$
non sinon

Propriété 4.5

Le problème WET est NP-difficile au sens faible.

Idée de preuve : Pour montrer que WET∈NP il suffit de montrer qu'on peut, en temps polynomial, vérifier qu'un ordonnancement est valide, calculer son coût et le comparer à y. La vrai difficulté est de montrer que WET est NP-difficile. Hall et Posner [7] proposent une preuve par réduction de EOP, qui est NP-difficile comme l'ont montré Garey, Tarjan et Wilfong [3].
Le fait que ce soit au sens faible découle de l'existence d'un algorithme de programmation dynamique qui résout ce problème avec une complexité pseudo-polynomiale. Il fait l'objet de la partie 4 de l'article de Hall et Posner [7], mais on ne le développera pas ici.

Hall et Posner [7] mentionnent aussi quatre sous-problèmes polynomiaux. Les deux derniers étant très particuliers (puisqu'ils nécessitent qu'une tâche soit vraiment très longue par rapport aux autres), on n'a fait apparaître que les deux premiers dans la cartographie page 5.
Bien que peu réaliste, le cas $w_j = p_j$ est un cas classique, et c'est notamment lui qui permet, dans un cadre régulier, de créer des contraintes traduisant le non-chevauchement (cf. section 5.2).
Le cas $p_j = 1$ est intéressant quant à lui parce que c'est le pendant de l'algorithme de Kanet, si on inverse les ω et les p.

État de l'art sur l'approche polyédrale du problème de minimisation de la somme pondérée des encours

5 Résultats de Queyranne [12]

Dans l'article [12], on s'intéresse à un problème d'ordonnancement **régulier** très simple : minimiser la somme pondérée des encours ou "Completion time" en anglais (i.e. $\min \sum_{j\in J} \omega_j C_j$).
Puisqu'il s'agit d'un problème régulier, on a la dominance des ordonnancements **calés à gauche** (qu'on notera $\vdash\!\square$) et dès 1956, Smith affirme [16] qu'il faut et qu'il suffit de classer les tâches par **ratio** $\boldsymbol{r_j := \frac{w_j}{p_j}}$ **décroissant** (ce qu'on notera $\searrow^r$). On appelle cette dominance stricte très classique la règle de Smith. Elle fournit ici un algorithme en $\mathrm{O}(n \log n)$ qui consiste essentiellement en un tri.

Mais dans cet article, on change de point de vue, on ne parle pas vraiment de dominance parce que, plus que trouver un ordonnancement optimal, on veut d'abord comprendre la structure polyédrale de l'enveloppe convexe des vecteurs C codant les ordonnancements réalisables pour une famille de tâches donnée. Les durées $(p_j)_{j\in J}$ sont donc fixée, en revanche on ne fixe pas de poids particulier.

5.1 Structure polyédrale

Définition 5.1

On notera Q l'ensemble des vecteurs codant des ordonnancements réalisables, i.e.

$$\boldsymbol{Q} := \left\{ C \in \mathbb{R}^J \;\middle|\; \begin{array}{ll} \forall j \in J,\ C_j \geqslant p_j & \textit{positivité} \\ \forall (j,k) \in J^2,\ C_k \geqslant C_j + p_j \text{ ou } C_j \geqslant C_k + p_j & \textit{non-chevauchement} \end{array} \right\}$$

Notre problème s'écrit alors $\min\limits_{C\in Q} \sum\limits_{j\in J} \omega_j C_j$ ou encore $\boxed{\min\limits_{C\in Q} \langle \omega | C \rangle}$

Propriété 5.2 (*règle de Smith*)

Soit $\omega \in \mathbb{R}^J = \mathbb{R}^n$
***Si** ω admet une composante strictement négative **alors** $\langle \omega | . \rangle$ n'est pas minorée sur Q.*
***Sinon** les $C \in Q$ minimisant $\langle \omega | . \rangle$ sont exactement ceux codant des ordo. $\vdash\!\square$ et $\searrow^r$*

Remarque : D'un point de vue polyédral, cette propriété assure que les points extrêmes de $\mathrm{Conv}(Q)$ sont des vecteurs codant des ordonnancements $\vdash\!\square$ et $\searrow^r$ et justifie qu'on s'y intéresse plus particulièrement.

Notations 5.3

Pour chaque "ordre" $\sigma \in \mathfrak{S}_n$ on définit :

- $\boldsymbol{C^\sigma} := \left(\sum\limits_{k=1}^{\sigma^{-1}(j)} p_{\sigma(k)} \right)_{j\in J}$ *le vecteur codant l'ordonnancement $\vdash\!\square$ et $\searrow^r$ dans lequel les tâches sont dans l'ordre σ c'est-à-dire $\sigma(1)$ puis $\sigma(2)$ jusqu'à $\sigma(n)$*
- $\forall k \in [1..n],\ \boldsymbol{u^{k,\sigma}} := \mathbb{1}_{\sigma([k..n])}$ *l'ajout de ce vecteur à un vecteur codant un ordo. dans l'ordre σ revient à insérer une unité de temps avant la i-ème tâche*
- $\boldsymbol{K^\sigma} := C^\sigma + \mathrm{CC}^\circ\left(\{u^{i,\sigma} \mid i \in [1..n]\}\right)$ [5] *le cône des vecteurs codant les ordonnancements où les tâches apparaissent dans l'ordre σ*

5. Pour $E \subset \mathbb{R}^n$, $\mathrm{CC}^\circ(E)$ désigne le cône convexe contenant 0 engendré par E

Lemme 5.4

$$Q = \bigcup_{\sigma\in\mathfrak{S}} \underbrace{\mathrm{Conv}(\{C^\sigma\}) + \mathrm{CC}^\circ\left(\{u^{i,\sigma} \mid i \in [1..n]\}\right)}_{=K^\sigma}$$

Corollaire 5.5

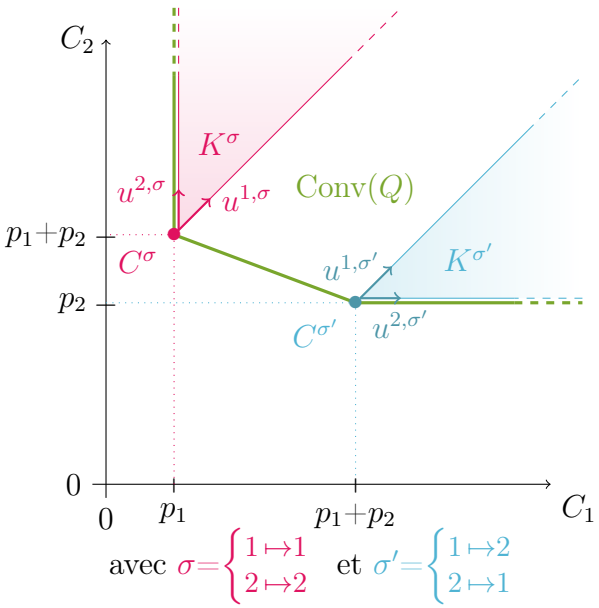
$$\overline{\mathrm{Conv}(Q)} = \underbrace{\mathrm{Conv}\{C^\sigma \mid \sigma \in \mathfrak{S}_n\}}_{:=A} + \underbrace{\mathrm{CC}^\circ\left\{u^{i,\sigma} \mid {}^{i\in[1..n]}_{\sigma\in\mathfrak{S}_n}\right\}}_{:=B}$$

Preuve : Par un résultat d'analyse convexe, cf. 19.6 [13] □

Corollaire 5.6

$$\mathrm{Conv}(Q) = \mathrm{Conv}\left(\{C^\sigma \mid \sigma \in \mathfrak{S}_n\}\right) + (\mathbb{R}^+)^J$$

Preuve : On voit facilement que $B \subset (\mathbb{R}^+)^J$, et comme pour tout $i \in [1..n]$ on a $\mathbb{1}_i = u^{n,\tau}$ pour τ la transposition (i, n) par exemple, on en déduit que $B = (\mathbb{R}^+)^J$. D'autre part on peut montrer que $A + B \subset \mathrm{Conv}(Q)$, ce qui assure que $\overline{\mathrm{Conv}(Q)} = \mathrm{Conv}(Q)$ et permet de conclure. □



5.2 Description de Conv(Q) par des inégalités

Notre objectif est maintenant d'obtenir une description de Conv(Q), non plus géométrique mais par des inégalités linéaires. Cependant la description précédente permet à Queyranne une jolie preuve que le système d'équations qu'il propose caractérise bien le polyèdre des solutions.

On sait qu'une inégalité définissant une facette de Conv(Q) est forcément de la forme $\langle \omega \mid C\rangle \geqslant c$ où $\omega \in (\mathbb{R}^+)^J$ et où la constante c vaut $\min_{C\in Q}\langle \omega \mid C\rangle$ [6].

Queyranne s'intéresse aux inégalités associées aux vecteurs $\boldsymbol{\omega^S} := \sum_{j\in S} p_j \mathbb{1}_j$ pour $S \subset J$, [7] et au polyèdre défini par ces seules inégalités, le but étant de montrer que ce polyèdre est Conv(Q).

Notations 5.7

On définit le polyèdre $\boldsymbol{P^Q} := \left\{C \in \mathbb{R}^J \;\middle|\; \forall S \in \mathcal{P}^*(J),\ \ p * C(S) \geqslant g(S)\right\}$

où $\boldsymbol{g} := \begin{pmatrix} \mathcal{P}(J) \to \mathbb{R} \\ S \longmapsto \min\limits_{C\in Q} p * C(S) \end{pmatrix}$

et où pour $C \in \mathbb{R}^J$, *et* $S \subset J$ *on note* $\boldsymbol{p} * \boldsymbol{C}(S) := \sum_{j\in S} p_j C_j = \langle \omega^S \mid C\rangle$.

Propriété 5.8

On a l'inclusion $\mathrm{Conv}(Q) \subset P^Q$ *et l'égalité des cônes de récession* $0^+(P^Q) = (\mathbb{R}^+)^J = 0^+(\mathrm{Conv}(Q))$.

Preuve : Par construction P^Q contient Conv(Q) puisqu'il est définit par des inégalités qui sont toutes valides pour Conv(Q), d'où le premier point. Puisqu'on n'a que des bornes inf. sur des fonctions à coefficients positifs, ajouter un vecteur de $(\mathbb{R}^+)^J$ à un point de P^Q ne peut nous en faire sortir, d'où le second. □

Remarques : Grâce à cette égalité des cônes de récession il nous suffit de montrer $\mathrm{Extr}(P^Q) \subset \mathrm{Conv}(Q)$. Par ailleurs le fait que le cône de récession de P^Q soit $(\mathbb{R}^+)^J$, nous permet d'affirmer que chacun de ses points extrêmes s'écrit comme unique minimum d'une fonction linéaire à coefficients positifs.

6. C'est aussi $\min_{x\in\mathrm{Conv}(Q)} \langle \omega \mid x\rangle$

7. Ce qui revient à considérer le coût dans le cas $\omega_j = p_j$ en oubliant les tâches dont l'indice est hors de S.

Lemme 5.9

Pour tout $S \subset J$, on a $g(S) = \frac{1}{2}\big(p(S)^2 + p * p(S)\big)$

Preuve : En utilisant la règle de Smith, on sait que tous les ordonnancements commençant par les tâches de S (celles de ratio 1) tassées à gauche minimisent la fonction coût associée à ω^S, peu importe comment sont placées les autres tâches après, puisqu'elles n'interviennent pas dans le coût [8]. En calculant $p * C(S)$ pour C le vecteur d'un tel ordonnancement, on obtient le résultat. □

Propriété 5.10

Pour tout $(A, B) \in \mathcal{P}(J)^2$, on a : $g(A \cup B) = g(A) + g(B) - g(A \cap B) + p(A_{\setminus B})\, p(B_{\setminus A})$.
En particulier on en déduit que g est une fonction ***super-modulaire*** [9]

On cherche maintenant à se ramener de P^Q à un polyèdre P' sous-modulaire, car on sait bien optimiser sur ce genre de polyèdre ce qui va nous permettre de bien caractériser les points extrêmes. Pour cela on introduit un changement de variable θ, pas tant pour passer de super-modulaire à sous-modulaire, mais aussi pour "normaliser" les contraintes, passer du type $\langle \omega^S | x \rangle \leqslant g'(S)$ au type $x(S) \leqslant g'(S)$.

Notations 5.11

On introduit le changement de variable $\boldsymbol{\theta} := \begin{pmatrix} \mathbb{R}^J \to \mathbb{R}^J \\ C \mapsto \big(p_j[p(J) - C_j]\big)_{j \in J} \end{pmatrix}$

On définit $\boldsymbol{P} := \big\{ x \in \mathbb{R}^J \;\big|\; \forall S \in \mathcal{P}^*(J),\, x(S) \leqslant g'(S) \big\}$ *où* $\boldsymbol{g'} := \begin{pmatrix} \mathcal{P}(J) \to \mathbb{R} \\ S \longmapsto p(S)p(J) - g(S) \end{pmatrix}$

Propriété 5.12

La fonction θ établit une bijection entre P^Q et P'.
Avec le changement de variable $x = \theta(C)$, pour tout $\omega \in \mathbb{R}^J$ on a $\langle \frac{\omega}{p} | x \rangle = p(J)\omega(J) - \langle \omega | C \rangle$

Propriété 5.13

La fonction g' est positive, croissante et ***sous-modulaire****.*
En particulier, le polyèdre P' est donc un polyèdre sous-modulaire.

Rappel 5.14 (*Algo. de maximisation en O(nlogn) sur un polyèdre sous-modulaire* [10])

Si *P' est un polyèdre sous-modulaire associé à la fonction g' et si $\omega \in (\mathbb{R}^+)^J$*
alors *en notant σ la permutation de J qui trie les composantes de ω par ordre décroissant et en posant pour tout $j \in J$, $x^*_{\sigma(j)} := g'(\sigma[1..j]) - x^*(\sigma[1..j\text{-}1])$, on a $x^* \in \underset{x \in P'}{\operatorname{argmax}} \langle \omega | x \rangle$*

Propriété 5.15

Sous les notations du rappel 5.14, le vecteur $C^* := \theta^{-1}(x^*)$ vérifie $C^* = C^\sigma$

Propriété 5.16

On a l'inclusion $\mathrm{Extr}(P^Q) \subset \mathrm{Extr}\big(\mathrm{Conv}(Q)\big) = \{C^\sigma | \sigma \in \mathfrak{S}_n\}$

Preuve : Si $C^* \in \mathrm{Extr}(P)$, d'après 5.8 il existe $\omega \in (\mathbb{R}^+)^J$ tel que $\{C^*\} = \underset{C \in P^Q}{\operatorname{argmin}} \langle \omega | C \rangle$. D'après 5.12 on a alors $\{\theta(C^*)\} = \underset{x \in P'}{\operatorname{argmax}} \langle \frac{\omega}{p} | x \rangle$. Or le vecteur x^* donné par l'algorithme glouton rappelé en 5.14 est un tel maximiseur, donc nécessairement $\theta(C^*) = x^*$, et alors d'après 5.15 on a $C^* = C^\sigma$ où σ est la permutation qui trie $\frac{\omega}{p}$ de manière décroissante. Autrement dit notre point extrême code un ordonnancement et $\searrow^r$, c'est celui que donne la règle de Smith ! [11] □

8. On peut noter au passage que dans le cas $\omega_j = p_j$, l'ordre des tâches n'importe pas pour les ordo.
9. cf. définition 9.7
10. cf. Section 44.2 de l'encyclopédie de Schrijver [14] pour une preuve
11. Attention on ne pouvait pas a priori utiliser la règle de Smith ici, on minimisait sur P^Q (via P') or elle permet de minimiser sur $\mathrm{Conv}(Q)$ et on n'avait pas encore montré l'égalité entre $\mathrm{Conv}(Q)$ et P^Q.

Corollaire 5.17

On a l'égalité $P^Q = \text{Conv}(Q)$.
De plus on peut monter que toutes les inégalités définissant P^Q *en 5.7 sont essentielles.*

Queyranne a donc fournit une description complète et minimale de Conv(Q), mais on retiendra surtout qu'il a introduit des inégalités qui semblent **traduire le non-chevauchement** [12] des tâches.

Attention, cela ne veut pas dire qu'un vecteur qui vérifie ces inégalités code un ordonnancement réalisable, c'est-à-dire sans chevauchement. En effet pensez au vecteur $C^m=(p_1 + \frac{p_2}{2}, p_2 + \frac{p_1}{2})$ milieu de $C^1 = (p_1, p_1 + p_2)$ et $C^2 = (p_1 + p_2, p_2)$. Il vérifie bien les inégalités et pourtant ne code pas un ordonnancement réalisable. En fait on ne peut espérer avoir des inégalités valides pour $\text{Conv}(Q)$ qui assureraient le non-chevauchement, car en prenant l'enveloppe convexe des ordonnancements réalisables on englobe forcément des milieux comme C^m.

Ce qu'on entend en disant qu'elles traduisent le non-chevauchement, c'est qu'un point extrême du polyèdre qu'elles définissent est bien sans chevauchement, et c'est bien ce qui nous intéresse car les algorithmes usuellement utilisés pour résoudre des PL fournissent des points extrêmes.

5.3 Problème de séparation

Bien que très expressives comme on l'a montré, les inégalités de Queyranne n'en sont pas moins nombreuses ! Autant de contraintes que de parties dans J ça fait **un nombre exponentiel de contraintes**, soit trop pour les entrer toutes dans un solveur. Queyranne propose donc **l'algorithme de coupes** suivant.

On commence par considérer seulement les inégalités $p_j C_j \geqslant p_j^2$ pour $j \in J$, qui sont celles associées aux singletons et qui traduisent les contraintes de positivité. La résolution de ce PL nous fournit un vecteur qui peut appartenir ou non à P^Q.

• S'il y appartient, on a trouvé un vecteur qui minimise l'objectif sur un sur-ensemble de P^Q et qui appartient à P^Q, c'est donc un minimiseur sur P^Q, l'algorithme s'arrête.

• Sinon, il faut ajouter au PL une inégalité qui **coupe** ce point, c'est-à-dire une inégalité violée par ce vecteur, qui assure qu'on ne retombera pas sur cette fausse solution par la suite. Puis on recommence sur ce PL enrichi.

Le cœur de l'algorithme est de savoir si une solution est dans P^Q ou non, c'est-à-dire si elle vérifie toutes les contraintes. Formellement il s'agit du problème suivant :

SEPARATION

Entrée : $C \in \mathbb{R}^J$

Sortie : oui si C vérifie toutes les inégalités de Queyranne (i.e $C \in P^Q$)
un ensemble $S \subset J$ tel que C viole la contrainte $p * C(S) \geqslant g(S)$ sinon

Propriété 5.18

Soit $C \in \mathbb{R}^J$. *En notant* $\mathbf{\Gamma_C} := \begin{pmatrix} \mathcal{P}(J) \to \mathbb{R} \\ S \longmapsto g(S) - p * C(S) \end{pmatrix}$ *on a* $\boxed{C \in P \Leftrightarrow \max\limits_{S \in \mathcal{P}(J)} \Gamma_C(S) \leqslant 0}$

Lemme 5.19

$\forall S \in \mathcal{P}(J), \forall k \in S, \quad \Gamma(S_{\setminus\{k\}}) = \Gamma(S) - p_k[p(S) - C_k]$
$\forall S \in \mathcal{P}(J), \forall k \notin S, \Gamma(S \sqcup \{k\}) = \Gamma(S) + p_k[p(S) - C_k + p_k]$

Propriété 5.20

Si *S maximise* Γ_C, ***alors*** $k \in S \Leftrightarrow p(S) \geqslant C_k$
donc *si* $\sigma \in \mathfrak{S}_n$ *trie C par ordre croissant, S s'écrit* $\sigma([1..i])$ *pour un certain* $i \in [1..n]$.

12. Elles englobent aussi la positivité puisque pour un singleton $\{i\}$, la contrainte de Queyranne s'écrit $p_i^2 \geqslant p_i C_i$

Corollaire 5.21

On a un algorithme de séparation en ***O(n log n)****.*

Preuve : Il suffit de trier les composantes de C (étape en O(n log n)), puis de calculer $\Gamma(\sigma[1..k])$ pour chaque $k \in [1..n]$ en utilisant 5.19 pour optimiser les calculs (étape en O(n)). Si toutes les valeurs calculées sont négatives on peut répondre ok $C \in P^Q$, sinon on renvoie $\sigma[1..k]$ pour le k maximisant $\Gamma(\sigma[1..k])$ (étape en O(n)). □

Grötschel, Lovász et Schrijver [5] ont montré que la complexité de l'algorithme de coupes est polynomiale si l'on résout le problème de séparation en temps polynomial. Comme Queyranne propose un algorithme de séparation en O(n log(n)), on retrouve avec cet algorithme de coupes que ce problème est polynomial.

5.4 Deux nouvelles preuves du non-chevauchement des points extrêmes

Queyranne s'est largement appuyé sur la première description de Conv(Q) pour prouver la validité de sa formulation (i.e. la propriété 5.17), mais dans le cas de polyèdres plus compliqués on n'a pas ce genre de description et sa jolie preuve ne peut pas s'adapter. Nous proposons donc ici deux autres preuves (non issues de la littérature), que l'on reprendra pour le problème juste-à-temps.

5.4.1 Une preuve constructive

On considère un point $C \in P^Q$, dont on suppose qu'il ne traduit pas un ordonnancement réalisable, et on va montrer qu'il ne peut être extrémal en l'écrivant comme milieu de deux points de P^Q. S'il ne code pas un ordonnancement réalisable c'est nécessairement parce que deux tâches se chevauchent, c'est-à-dire qu'il existe $(i,j) \in J^2$, avec $i \neq j$ tel que $\underline{C_i \leqslant C_j < C_i + p_j}$.

On pose $\boldsymbol{\varepsilon_1} := \min\{p * C(S^1) - g(S^1) \mid S \in \mathcal{P}^*(J), i \notin S^1, j \in S^1\}$ puis $\boldsymbol{C^{+-}} := C + \frac{\varepsilon}{p_i}\mathbb{1}_i - \frac{\varepsilon}{p_j}\mathbb{1}_j$

$\boldsymbol{\varepsilon_2} := \min\{p * C(S^2) - g(S^2) \mid S \in \mathcal{P}^*(J), i \in S^2, j \notin S^2\}$ $\boldsymbol{C^{-+}} := C - \frac{\varepsilon}{p_i}\mathbb{1}_i + \frac{\varepsilon}{p_j}\mathbb{1}_j$

$\boldsymbol{\varepsilon} := \min\{\varepsilon_1, \varepsilon_2\}$.

Ainsi on a directement $C = \frac{C^{+-}+C^{-+}}{2}$, et on vérifie facilement que cette construction assure que $C^{+-} \in P^Q$ et $C^{-+} \in P^Q$. Mais attention pour conclure il faut s'assurer que $C^{+-} \neq C$, c'est-à-dire que $\varepsilon > 0$, et pour cela on a besoin des deux lemmes suivants.

Lemme 5.22

Soit $C \in P^Q$. Soit $(i,j) \in J^2$.
Si $C_i \leqslant C_j$, ***alors*** $\forall S^1 \in \mathcal{P}(J)$, $\left.\begin{array}{l} i \notin S^1 \\ j \in S^1 \end{array}\right\} \Rightarrow p * C(S^1) > g(S^1)$

Preuve : Par l'absurde on suppose qu'il existe $S^1 \in \mathcal{P}(J)$ contenant j mais pas i tel que $p * C(S^1) = g(S^1)$. On note $\tilde{S^1} = S^1_{\setminus\{j\}}$. Alors on décompose [13] $g(S^1) = g(\tilde{S^1}) + p_j p(S^1)$ et $p * C(S^1) = p * C(\tilde{S^1}) + p_j C_j$. On a supposé ces deux termes égaux, et on sait par ailleurs que $p * C(\tilde{S^1}) \geqslant g(\tilde{S^1})$ puisque $C \in P^Q$. On en déduit que $\underline{C_j \leqslant p(S^1)}_\star$. Alors on a

$$\begin{array}{lll} p * C(S^1 \cup \{i\}) = p * C(S^1) + p_i C_i & & \\ = g(S^1) + p_i C_i & & \text{supposition absurde} \\ \leqslant g(S^1) + p_i C_j & & \text{hypothèse} \\ \leqslant g(S^1) + p_i p(S^1) & & \text{par } \star \\ < g(S^1) + p_i[p(S^1) + p_i] & & \text{car } p_i > 0 \\ = g(S^1 \cup \{i\}) & & \text{cf. égalité 5.10} \\ \leqslant p * C(S^1 \cup \{i\}) & & \text{car } C \in P^Q \\ \textbf{ABSURDE} & & \end{array}$$

□

13. En utilisant 5.10 on a $g(A \sqcup \{i\}) = g(A) + g(\{i\} + p_i p(A) = g(A) + p_i^2 + p_i p(A) = g(A) + p_i p(A \sqcup \{i\})$

Lemme 5.23

Soit $C \in P^Q$. Soit $(i,j) \in J^2$.
Si $C_j < C_i + p_j$, ***alors*** $\forall S^2 \in \mathcal{P}(J),\ \left.\begin{array}{l} i \in S^2 \\ j \notin S^2 \end{array}\right\} \Rightarrow p * C(S^2) > g(S^2)$

Preuve : Par l'absurde on suppose qu'il existe $S^2 \in \mathcal{P}(J)$ contenant i mais pas j tel que $p*C(S^2) = g(S^2)$.
On montre de même que $\underline{C_i \leqslant p(S^2)}_{\star}$ et alors $p*C(S^2 \cup \{j\}) = p*C(S^2) + p_j C_j$

$= g(S^2) + p_j C_j$ — supposition absurde
$< g(S^2) + p_j[C_i + p_j]$ — hypothèse
$\leqslant g(S^2) + p_j[p(S^2) + p_j]$ — par $\star$
$= g(S^2 \cup \{j\})$ — cf. égalité 5.10
$\leqslant p*C(S^2 \cup \{j\})$ — car $C \in P^Q$
ABSURDE

□

Le lemme 5.22 nous assure que $\varepsilon_1 > 0$ et le lemme 5.23 que $\varepsilon_2 > 0$, donc $\varepsilon > 0$. Ainsi C ne peut être extrémal. Par contraposée on en déduit qu'un point extrême de P^Q est sans chevauchement.

5.4.2 Une preuve basée sur les inégalités

On fait ici une preuve directe. On considère $C \in \text{Extr}(P^Q)$ et on va montrer qu'il code un ordonnancement réalisable. Pour cela on utilise le fait qu'un point extrême sature autant d'inégalités que la dimension du polyèdre. Ici P^Q est de pleine dimension (i.e. de dimension n), notamment parce que son cône de récession est $(\mathbb{R}^+)^n$, on sait donc que C sature n inégalités. On utilise maintenant la structure particulière de ces inégalités, et notamment les propriétés de la fonction g pour connaître ces n inégalités.

Propriété 5.24

Soit $C \in P^Q$. Soit $(S,T) \in \mathcal{P}^*(J)^2$.
Si $p*C(S) = g(S)$ *et* $p*C(T) = g(T)$ ***alors*** $S \subset T$ *ou* $T \subset S$.

Preuve : Par l'absurde on suppose que $S \not\subset T$ et $T \not\subset S$. On a alors $S_{\setminus T} \neq \emptyset$ et $T_{\setminus S} \neq \emptyset$. Donc $p(S_{\setminus T})p(T_{\setminus S})$ est strictement positif, ce qui assure d'après 5.10 que $g(S \cup T) > g(S) + g(T) - g(S \cap T)$. Or par hypothèse on a $g(S) = p*C(S)$ et $g(T) = p*C(T)$, et puisque $C \in P^Q$ on a aussi $g(S \cap T) \leqslant p*C(S \cap T)$.
Donc on en déduit $g(S \cup T) > p*C(S) + p*C(T) - p*C(S \cap T) = p*C(S \cup T)$.
ABSURDE car C doit satisfaire la contrainte pour l'ensemble $S \cup T$ puisque $C \in P^Q$.

□

Corollaire 5.25

Si *C est extrême dans P^Q* ***alors*** *il existe $(S_i)_{i \in [1..n]}$ une suite de sous-ensembles imbriqués de cardinaux échelonnés de 1 à n telle que $\forall i \in [1..n],\ p*C(S_i) = g(S_i)$.*
Or *de telles inégalités assurent que C code un ordonnancement réalisable et même* .

Preuve : On sait déjà que C sature n inégalités linéairement indépendantes.
La propriété précédente assure que deux inégalités saturées ne peuvent correspondre à deux ensembles distincts de même cardinaux ; en effet si on a égalité entre $p*C$ et g sur deux ensembles, l'un est inclus dans l'autre, si en plus ils ont même cardinal ils sont égaux.
Donc les n inégalités saturées par C correspondent à des ensembles $(S_i)_{i \in [1..n]}$ de cardinaux échelonnés de 1 à n, et en utilisant à nouveau la propriété précédente on obtient leur imbrication, c'est-à-dire que $S_1 \subset S_2 ... \subset S_{n-1} \subset S_n$.
En notant $S_0 = \emptyset$ et $\sigma(i)$ pour $i \in [1..n]$ l'unique élément de $S_{i \setminus S_{i-1}}$, de sorte que $\forall i \in [1..n]$, $S_i = \sigma([1..i])$.
D'une part on a $g(S_i) = g(S_{i-1} \cup \{i\}) = g(S_{i-1}) + p_{\sigma(i)} p(S_i) = p*C(S_{i-1}) + p_{\sigma(i)} p(S_i)$ et d'autre part $g(S_i) = p*C(S_i) = p*C(S_{i-1}) + p_{\sigma(i)} C_{\sigma(i)}$, on en déduit que $C_{\sigma(i)} = p(S_i) = p(\sigma([1..i]))$ et ce pour tout $i \in [1..n]$. Donc $C = C^\sigma$, ainsi $C \in Q$ et c est bien

□

Étude polyédrale d'un problème d'ordonnancement une machine avec coûts d'avance et de retard

On revient ici sur nos problèmes d'ordonnancement juste-à-temps. Plus précisément on s'intéresse au problème avec **due-date commune** et **non restrictive**. Comme dans l'article de Queyranne [12] on va d'abord s'intéresser au polyèdre des solutions, c'est-à-dire au polyèdre des vecteurs codant les solutions pour un codage qu'on va justement déterminer [14] ; c'est pourquoi dans toute cette partie, on ne considérera fixés que n le nombre de tâches et $(p_j)_{j\in J}$ les durées de ces tâches, mais pas les coûts. On peut aussi considérer fixée d une due-date vérifiant $d \geqslant p(J)$, mais comme elle n'est pas restrictive on peut aussi simplement l'omettre.

6 Formulation (e, t, δ, l, r)

6.1 Idée et nouveau codage

On a déjà expliqué (2.1) qu'on utilisait un codage par (e, t) pour exprimer le coût de manière linéaire. Mais il faut aussi pouvoir assurer par des contraintes linéaires qu'un vecteur code bien un ordonnancement réalisable. Il y a essentiellement trois choses à garantir :

[**cohérence**] Les valeurs e_i et t_i permettent de déterminer la période d'exécution de la tâche J_i
[**non-chevauchement**] Les périodes d'exécution des tâches sont deux à deux disjointes.
[**positivité**] Les périodes d'exécution des tâches ne commencent pas avant le temps 0.

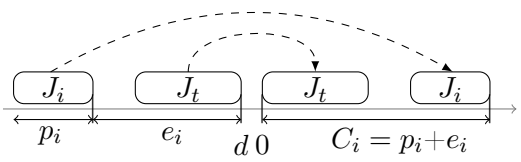
Pour la **cohérence** il faut assurer que $e_j = 0$ **ou** $t_j = 0$. Ce n'est pas exclusif puisqu'une tâche à l'heure est d'avance nulle et de retard nul. Pour gérer cette disjonction on introduit, pour tout $j \in J$, la variable δ_j qui vaut 1 si la tâche est en avance (ou à l'heure) et 0 sinon.

Mais ça ne suffit pas : pour dire "si $\delta_j = 0$, alors e_j doit être nul", il nous faut une constante qui majore e_j, qu'on appelle usuellement un "big M". Ici la positivité nous en fournit un d'office : d. La contrainte $e_j \leqslant d\delta_j$ est alors bien satisfaite par tous les (e, t, δ) codant un ordonnancement réalisable, et réciproquement un vecteur (e, t, δ) avec $\delta \in \{0, 1\}^n$ qui vérifie cette contrainte a bien $e_j = 0$ si $\delta_j = 0$.

Mais pour dire "si $\delta_j = 1$, alors t_j doit être nul", il nous faut une borne sur les t_j alors qu'a priori ils sont non bornés (une tâche peut toujours avoir lieu plus tard, c'est ce qui faisait que P^Q était non borné). Cependant si on cherche à minimiser le coût, ces solutions tardives ne nous intéressent pas, et on peut se restreindre aux ordonnancements ▭▭ et 🕐 car la propriété 4.2 reste vraie pour des coûts positifs quelconques, (à moins que $\beta = 0$, et $\forall j \in J$, $\alpha_j > 0$ mais on peut exclure ce cas car un problème juste-à-temps sans coût de retard est un problème régulier vu dans un miroir).

Se restreindre aux ordonnancements ▭▭ et 🕐 borne les retards (et les avances) par $p(J)$, qu'on choisit finalement comme "big M" pour les deux côtés,parce que dans le cas non-restrictif c'est une meilleure borne que d. Notez donc $\boldsymbol{M} := p(J)$.

Pour ce qui est du **non-chevauchement** on aimerait utiliser les inégalités de Queyranne de part et d'autre de la due-date. En effet, pour un ordonnancement 🕐 il suffit de s'assurer que les tâches en retard sont toutes à droite de d (c'est une nouvelle positivité), qu'elles ne se chevauchent pas entre elles et que les tâches en avance sont toutes à gauche de d et ne se chevauchent pas non plus entre elles. Du côté retard c'est facile parce que t joue par rapport à d le même rôle que C par rapport à 0. En revanche côté avance c'est $e+p$ qui joue le rôle de C.



14. On présente en annexe, section 10.2, deux codages et formulations envisagés mais infructueux

En notant $E' := \{j \in J \mid \delta_j = 1\}$ et $T := \{j \in J \mid \delta_j = 0\}$ on a envie d'écrire les inégalités

$$\begin{cases} \forall S \subset J,\ p * [p+e](S) \geqslant g(S) \\ \forall S \subset J,\ p * t(S) \geqslant g(S) \end{cases}$$

Mais ces inégalités ne sont même pas toujours valides : si on prend $i \in E$ et $j \in T$, la deuxième inégalité pour $S = \{i, j\}$ s'écrit $p_j t_j \geqslant \frac{1}{2}[(p_i + p_j)^2 + (p_i^2 + p_j^2)] = p_i^2 + p_j^2 + p_i p_j > p_j^2$, ce qui interdit $t_j = p_j$, même si J_j était la tâche juste après d. L'erreur vient du fait qu'on a considéré J_i une tâche en avance, alors qu'on voulait exprimer du non-chevauchement à droite de la due-date. Il faut traiter avance et retard séparément et écrire plutôt

$$\begin{cases} \forall S \subset J,\ p * [p+e](S \cap E') \geqslant g(S \cap E') & \text{(S1')} \\ \forall S \subset J,\ p * t(S \cap T) \geqslant g(S \cap T) & \text{(S2')} \end{cases}$$

Là c'est bien dans l'idée mais attention E' et T dépendent de δ, et des contraintes qui dépendent des variables c'est complètement interdit en PL ! Alors on va essayer d'utiliser notre variable δ pour faire disparaître cette intersection, d'abord sur (S1') :

$$\begin{aligned}
\text{(S1')} &\Leftrightarrow \forall S \subset J,\ \sum_{j \in J} p_j [p_j + e_j] \delta_j \geqslant \frac{1}{2} \left(\Big(\sum_{j \in J} p_j \delta_j \Big)^2 + \sum_{j \in J} p_j^2 \delta_j \right) \\
&\Leftrightarrow \forall S \subset J,\ \sum_{j \in J} p_j^2 \delta_j + \sum_{j \in J} p_j \underbrace{e_j \delta_j}_{=e_j} \geqslant \frac{1}{2} \sum_{(i,j) \in J^2} p_i p_j \delta_i \delta_j \ + \ \frac{1}{2} \sum_{j \in J} p_j^2 \delta_j \\
&\Leftrightarrow \forall S \subset J,\ \cancel{\sum_{j \in J} p_j^2 \delta_j} + \sum_{j \in J} p_j e_j \geqslant \frac{1}{2} \sum_{\substack{(i,j) \in J^2 \\ i \neq j}} p_i p_j \delta_i \delta_j \ + \ \cancel{\frac{1}{2} \sum_{j \in J} p_j^2 \underbrace{\delta_j^2}_{=\delta_j}} \ + \ \cancel{\frac{1}{2} \sum_{j \in J} p_j^2 \delta_j} \\
&\Leftrightarrow \forall S \subset J,\ \sum_{j \in J} p_j e_j \geqslant \sum_{\substack{(i,j) \in J^2 \\ i < j}} p_i p_j \delta_i \delta_j
\end{aligned}$$

Pour remplacer le terme quadratique $\delta_i \delta_j$ (le seul qui pose encore problème), on introduit de nouvelles variables booléennes, les $l_{i,j}$ pour $(i,j) \in J^<$ où $\boldsymbol{J^<} := \{(i,j) \in J^2 \mid\ i < j\}$. Pour $(i,j) \in J^<$, $l_{i,j}$ doit valoir 1 ssi δ_i et δ_j valent 1, c'est-à-dire ssi J_i et J_j sont à gauche de la due date (d'où l comme "left").

Pour (S2') on fait le même genre de calcul, mais c'est le produit $(1 - \delta_i)(1 - \delta_j)$ qui apparaît. On introduit alors les variables booléennes $r_{i,j}$ pour $(i,j) \in J^<$, qui doivent valoir 1 ssi δ_i et δ_j valent 0, c'est-à-dire ssi J_i et J_j sont à droite de la due date (d'où le r comme "right").

Pour assurer la cohérence entre δ et l, (resp. entre δ et r) on introduit de nouvelles contraintes :

$$\begin{array}{rl}
\forall (i,j) \in J^<,\ l_{i,j} \geqslant 0 & (l.1) \\
l_{i,j} \leqslant \delta_i & (l.2) \\
l_{i,j} \leqslant \delta_j & (l.3) \\
l_{i,j} \geqslant \delta_i + \delta_j - 1 & (l.4)
\end{array}
\qquad
\begin{array}{rl}
\forall (i,j) \in J^<,\ r_{i,j} \geqslant 0 & (r.1) \\
r_{i,j} \leqslant (1 - \delta_i) & (r.2) \\
r_{i,j} \leqslant (1 - \delta_j) & (r.3) \\
r_{i,j} \geqslant 1 - \delta_i - \delta_j & (r.4)
\end{array}$$

Lemme 6.1

Soit $\delta \in \{0,1\}^n$.
Si $(l, r) \in \mathbb{R}^{n(n-1)}$ *vérifie les contraintes (l.-) et(r.-) ci-dessus*
alors *on a* $\forall (i,j) \in J^<,\ l_{i,j} = 1 \Leftrightarrow \begin{cases} \delta_i = 1 \\ \delta_j = 1 \end{cases}$ *et* $\forall (i,j) \in J^<, r_{i,j} = 1 \Leftrightarrow \begin{cases} \delta_i = 0 \\ \delta_j = 0 \end{cases}$

Preuve : Soit $(i,j) \in J^<$. Si $l_{i,j} = 1$, alors on a $\delta_i \geqslant 1$ par la contrainte $(l.2)$, or $\delta_i \in \{0,1\}$, donc $\delta_i = 1$ et de même par la contrainte $(l.3)$ on a $\delta_j = 1$. Réciproquement, si $\delta_i = 1$ et $\delta_j = 1$, les contraintes $(l.2)$ ou $(l.3)$ donnent $l_{i,j} \leqslant 1$, tandis que la contrainte $(\delta.4)$ donne $l_{i,j} \geqslant -1 + 1 + 1 = 1$ donc $l_{i,j} = 1$. On montre de même l'équivalence pour $r_{i,j}$ avec les contraintes $(r.-)$ □

Maintenant qu'on s'est muni de ces nouvelles variables on peut écrire des inégalités linéaires traduisant les contraintes (S1') et (S2') :

$$\forall S \in \mathcal{P}^*(J),\ \sum_{i\in S} p_i e_i \geqslant \sum_{(i,j)\in S^<} p_i p_j l_{i,j} \quad (S1)$$

$$\sum_{i\in S} p_i t_i \geqslant \sum_{(i,j)\in S^<} p_i p_j r_{i,j} + \sum_{i\in S} p_i^2(1-\delta_i) \quad (S2)$$

Corollaire 6.2

Soit $x = (e,t,\delta,l,r) \in \mathbb{R}^{n^2+2n}$.
Si $\delta \in \{0,1\}^n$ *et si* (l,r) *vérifie les contraintes (l.-) et(r.-)* ***alors*** x *vérifie (S1)* $\Leftrightarrow$ e *vérifie (S1')*
x *vérifie (S2)* $\Leftrightarrow$ t *vérifie (S2')*

Puisqu'on a pris en compte la **positivité** dans les contraintes assurant la cohérence entre les e et les t, on a maintenant tout ce qu'il nous faut. On s'arrête donc sur ce codage des ordonnancements par un vecteur $(e,t,\delta,l,r) \in \mathbb{R}^{2n} \times [0,1]^{n^2}$. On présente dans la section suivante le polyèdre que l'on va étudier, qui est l'ensemble des vecteurs vérifiant toutes les contraintes linéaires mentionnées jusqu'ici, mais pas nécessairement l'intégrité de ses n^2 dernières composantes. Il est donc un peu plus que l'enveloppe convexe des codages des ordonnancements réalisables.

6.2 Formulation

Définition 6.3

$$\boldsymbol{P}^{\text{"e,t,}\boldsymbol{\delta}\text{,l,r"}} := \Big\{\ (e,t,\delta,l,r) \in \mathbb{R}^J \times \mathbb{R}^J \times \mathbb{R}^J \times \mathbb{R}^{J^<} \times \mathbb{R}^{J^<}\ \Big|$$

$$\forall j \in J,\ e_j \geqslant 0 \quad (e.1) \qquad e_j \leqslant M\delta_j \quad (e.2)$$

$$\forall j \in J,\ t_j \geqslant 0 \quad (t.1) \qquad t_j \leqslant M(1-\delta_j) \quad (t.2)$$

$$\forall j \in J, 0 \leqslant \delta_j \leqslant 1 \quad (\delta)$$

$$\forall (i,j) \in J^<,\ l_{i,j} \geqslant 0 \quad (l.1) \qquad l_{i,j} \leqslant \delta_i \quad (l.2) \qquad l_{i,j} \leqslant \delta_j \quad (l.3) \qquad l_{i,j} \geqslant \delta_i + \delta_j - 1 \quad (l.4)$$

$$\forall (i,j) \in J^<,\ r_{i,j} \geqslant 0 \quad (r.1) \qquad r_{i,j} \leqslant (1-\delta_i) \quad (r.2) \qquad r_{i,j} \leqslant (1-\delta_j) \quad (r.3) \qquad r_{i,j} \geqslant 1-\delta_i-\delta_j \quad (r.4)$$

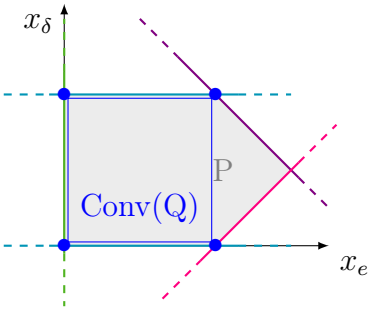
$$\forall S \in \mathcal{P}^*(J),\ \sum_{i\in S} p_i e_i \geqslant \sum_{(i,j)\in S^<} p_i p_j l_{i,j} \quad (S1)$$

$$\sum_{i\in S} p_i t_i \geqslant \sum_{(i,j)\in S^<} p_i p_j r_{i,j} + \sum_{i\in S} p_i^2(1-\delta_i) \quad (S2)$$

$$\Big\}$$

6.3 Intérêt du polyèdre $P^{\text{"e,t,}\delta\text{,l,r"}}$ et preuve

Notre but n'est pas comme chez Queyranne de montrer que le polyèdre $P^{\text{"e,t,}\delta\text{,l,r"}}$ est l'enveloppe convexe des vecteurs codant des ordonnancements réalisables (à cause des inégalités avec big M on sait déjà que ce polyèdre ne les contient pas tous), ni même celle des ordonnancements  et .
En effet, en relâchant la contrainte d'intégrité sur les variables δ, l et r on englobe des solutions qui sont hors de l'enveloppe convexe.
On illustre cette idée par le petit exemple suivant :
Si $P = \{(x_e, x_\delta) \,|\, x_e \geqslant 0,\ 0 \leqslant x_\delta \leqslant 1,\ x_\delta \leqslant 2 - x_e,\ x_\delta \geqslant x_e - 1\}$
et $Q = P \cap \mathbb{Z}^2$ on a $\text{Conv}(Q) \subsetneq P$



Notre but ici va être de montrer que les points extrêmes qu'est susceptible de nous renvoyer un solveur lorsqu'on lui demandera de minimiser une fonction coût codent bien tous des ordonnancements réalisables. On ne regarde donc pas tous les points extrêmes, seulement ceux qui minimisent une fonction de la forme $\sum_{j\in J}\alpha_j e_j + \sum_{j\in J}\beta_j t_j$, et qui sont en (δ, l, r) entiers.

Définition 6.4

On note $\boldsymbol{Extr}^* := \left\{ x^* \in \mathrm{Extr}(P^{\text{"e,t,}\delta\text{,l,r"}}) \;\middle|\; \begin{array}{l} x^* = (e,t,\delta,l,r) \text{ avec } (\delta,l,r) \in \{0,1\}^{n^2} \\ \exists (\alpha,\beta) \in (\mathbb{R}^{+*})^{2n},\ x^* \in \underset{x\in P^{\text{"e,t,}\delta\text{,l,r"}}}{\operatorname{argmin}} \langle (\alpha,\beta,0,0,0) | x \rangle \end{array} \right\}$

Remarque : On peut montrer que $\mathrm{Extr}^* = \left\{ x^* \in P^{\text{"e,t,}\delta\text{,l,r"}} \;\middle|\; \begin{array}{l} x^* = (e,t,\delta,l,r) \text{ avec } (\delta,l,r) \in \{0,1\}^{n^2} \\ \exists (\alpha,\beta) \in (\mathbb{R}^{+*})^{2n},\ x^* = \underset{x\in P^{\text{"e,t,}\delta\text{,l,r"}}}{\operatorname{argmin}} \langle (\alpha,\beta,0,0,0) | x \rangle \end{array} \right\}$

Définition 6.5

Pour $x = (e,t,\delta,l,r) \in P^{\text{"e,t,}\delta\text{,l,r"}}$ *avec* $\delta \in \{0,1\}^n$ *on note* $\boldsymbol{E'}(x) := \{ j \in J \mid \delta_j = 1 \}$
$\boldsymbol{T}(x) := \{ j \in J \mid \delta_j = 0 \}$

Lemme 6.6

Si $x = (e,t,\delta,l,r) \in Extr^*$ ***alors*** *on a* $T(x) \subsetneq J$

Remarque : Par l'absurde on suppose que $T(x) = J$, i.e. que $\delta = 0$, $e = 0$ et $l = 0$. On va montrer qu'alors on peut construire un autre point de $P^{\text{"e,t,}\delta\text{,l,r"}}$ de même coût en plaçant une tâche $i_0 \in J$ à l'heure.
Pour ce faire on considère $x' = (e', t', \delta', l', r')$ où $e' = e = 0$, $t' = t - t_{i_0}\mathbb{1}_{i_0}$, $\delta' = \mathbb{1}_{i_0}$ et où l' et r' sont définis en adéquation avec δ' (en particulier $l' = l = 0$ parce qu'il n'y a pas deux tâches en avance).
Ce vecteur vérifie les contraintes $(e.\text{-})$, $(t.\text{-})$, $(\delta.\text{-})$, $(l.\text{-})$, $(r.\text{-})$ par construction, et même les contraintes (S1) puisque e' et l' sont nuls.
Pour des $S \subset J$ ne contenant pas i_0 les contraintes (S2) sont vérifiées indifféremment par x et x' puisque les termes intervenant sont égaux, or $x \in P^{\text{"e,t,}\delta\text{,l,r"}}$ doit vérifier ces contraintes, donc x' aussi.
Il nous reste donc à traiter le cas où $S \subset J$ contient i_0. On pose alors $\tilde{S} = S_{\setminus\{i_0\}}$. On a alors :

$$\begin{aligned}
\sum_{i\in S} p_i t'_i &= \sum_{i\in\tilde{S}} p_i t_i && \text{car } t'_{i_0} = 0 \text{ et si } i \neq i_0, t_i = t'_i \\
&\geqslant \sum_{(i,j)\in\tilde{S}^<} p_i p_j r_{i,j} + \sum_{i\in\tilde{S}} p_i^2(1\text{-}\delta_i) && \text{car } x \in P^{\text{"e,t,}\delta\text{,l,r"}} \text{ satisfait les contraintes (S2)} \\
&= \sum_{(i,j)\in\tilde{S}^<} p_i p_j r'_{i,j} + \sum_{i\in\tilde{S}} p_i^2(1\text{-}\delta'_i) && \text{car seuls les } r_{i_0,j}, \text{ les } r_{i,i_0} \text{ et } \delta_{i_0} \text{ diffèrent entre } x \text{ et } x' \\
&= \sum_{(i,j)\in\tilde{S}^<} p_i p_j r'_{i,j} + \sum_{\substack{i\in\tilde{S} \\ i<i_0}} p_i p_{i_0} \underbrace{r'_{i,i_0}}_{\uparrow 0} + \sum_{\substack{j\in\tilde{S} \\ j>i_0}} p_{i_0} p_j \underbrace{r'_{i_0,j}}_{\uparrow 0} + \sum_{i\in\tilde{S}} p_i^2(1\text{-}\delta'_i) + p_{i_0}\underbrace{(1\text{-}\delta'_{i_0})}_{=0} && \text{car } \delta'_{i_0} = 1 \\
&= \sum_{(i,j)\in S^<} p_i p_j r'_{i,j} + \sum_{i\in S} p_i^2(1\text{-}\delta'_i)
\end{aligned}$$

Donc x' vérifie bien les contraintes (S2). Ainsi $x' \in P^{\text{"e,t,}\delta\text{,l,r"}}$, or en passant de x à x' on a annulé la pénalité engendrée par la tâche J_{i_0}, sans changer les autres, c'est-à-dire qu'on est passé de x un unique minimiseur à un point $P^{\text{"e,t,}\delta\text{,l,r"}}$ de coût inférieur ou égal. ABSURDE. □

Lemme 6.7

Si $x=(e,t,\delta,l,r)\in Extr^*$ ***alors*** *on a* $\forall j\in E'(x),\ e_j<M \quad et \quad \forall j\in T(x),\ t_j<M$

Remarque : Puisque $x\in \text{Extr}^*$, il existe $(\alpha,\beta)\in(\mathbb{R}^{+*})^{2n}$ tel que x soit l'unique minimiseur de $\langle\alpha|e\rangle+\langle\beta|t\rangle$. Par l'absurde on suppose qu'il existe $j\in E(x)$ tel que $e_j=M$. On aimerait alors poser $x^-=(e^-,t,\delta,l,r)$ où $e^-=e-\varepsilon\mathbb{1}_j$ pour un ε bien choisi (i.e. tel que x^- vérifie les contraintes $(e.1)$ et (S1)), car alors on aurait un point de $P^{\text{"e,t,}\delta\text{,l,r"}}$ de coût $\langle\alpha|e^-\rangle+\langle\beta|t\rangle$ moindre que celui de x (ABSURDE si $x\neq x^-$).
Pour assurer $(e.1)$ il suffit de choisir $\varepsilon\leqslant M$. Puisqu'on ne change pas δ on a $E'(x)=E'(x^-)$ et δ reste entier, ce qui nous ramène à vérifier (S1')(cf. 6.2). Comme pour tout $S\subset E'(x)$ tel que $j\in S$ on a $p*[e^-+p](S)=p*[e+p](S)-p_j\varepsilon$ il suffit que $\varepsilon\leqslant\frac{m}{p_j}$ où $m:=\min\big\{p*[e+p](S)-g(S)\ \mid\ S\subset E'(x), j\in S\big\}$.
On pose donc $\varepsilon=\min\Big(M,\frac{m}{p_j}\Big)$.
Mais pour que x^- soit témoin d'une absurdité, il faut encore que x^- soit différent de x, i.e. $\varepsilon>0$. Or pour tout $S\subset E'(x)$ tel que $j\in S$ on a $p*[e+p](S)=p*[e+p](S_{\setminus\{j\}})+p_jM+p_j^2$ et $g(S)=g(S_{\setminus\{j\}})+p_jp(S)$, sachant que $p*[e+p](S_{\setminus\{j\}})\geqslant g(S_{\setminus\{j\}})$ cela donne $p*[e+p](S)-g(S)\geqslant p_j[p_j+M-p(S)]>0$ car $p_j>0$ et $p(S)\leqslant p(J)=M$, donc $m>0$ et par conséquent $\varepsilon>0$. CQFD

Pour l'autre assertion on suppose par l'absurde qu'il existe $j\in T(x)$ tel que $t_j=M$. Dans la même démarche on pose $x^-=(e,t^-,\delta,l,r)$ avec $t^-=t-\varepsilon\mathbb{1}_j$ et $\varepsilon=\min\Big(M,\frac{1}{p_j}\min\big\{p*t(S)-g(S)\ \mid\ S\subset T(x), j\in S\big\}\Big)$
Montrer ici que $\varepsilon>0$ est un peu plus difficile : on a toujours l'égalité $g(S)=g(S_{\setminus\{j\}})+p_jp(S)$ mais $p*t(S)=p*t(S_{\setminus\{j\}})+p_jM$ sans le terme p_j^2. On obtient $p*t(S)-g(S)\geqslant p_j[M-p(S)]\geqslant p_j[p(J)-p(T(x))]$.
On utilise alors le lemme précédent : puisque $x\in\text{Extr}^*$, $T(x)\subsetneq J$, donc $p(J)-p(T(x))>0$ et donc on a bien $\varepsilon>0$. CQFD
□

Lemme 6.8

Soit $x=(e,t,\delta,l,r)\in P^{\text{"e,t,}\delta\text{,l,r"}}$ *avec* $\delta\in\{0,1\}^n$
S'il *existe* $(i,j)\in E'(x)^2$ *avec* $i\neq j$ *tq* $e_i+p_i\leqslant e_j+p_j<e_i+p_i+p_j$, $e_i\neq M$ *et* $e_j\neq M$ ***alors*** $x\notin Extr^*$.
S'il *existe* $(i,j)\in T(x)^2$ *avec* $i\neq j$ *tel que* $t_i\leqslant t_j<t_i+p_j$, $t_i\neq M$ *et* $t_j\neq M$ ***alors*** $x\notin Extr^*$.

Remarque : Considérons $(i,j)\in E'(x)^2$ avec $i\neq j$ tq $e_i+p_i\leqslant e_j+p_j<e_i+p_i+p_j$. On va adapter la preuve constructive faite en 5.4.1, et écrire x comme milieu de deux points de $P^{\text{"e,t,}\delta\text{,l,r"}}$. Pour cela on aimerait définir $e^{+-}:=e+\frac{\varepsilon}{p_i}\mathbb{1}_i-\frac{\varepsilon}{p_j}\mathbb{1}_j$ pour un ε bien choisi, i.e. tel que $x^{+-}:=(e^{+-},t,\delta,l,r)\in P^{\text{"e,t,}\delta\text{,l,r"}}$.
$e^{-+}:=e-\frac{\varepsilon}{p_i}\mathbb{1}_i+\frac{\varepsilon}{p_j}\mathbb{1}_j$ $\qquad$ $x^{-+}:=(e^{-+},t,\delta,l,r)\in P^{\text{"e,t,}\delta\text{,l,r"}}$
Pour assurer les contraintes $(e.1)$, il suffit de prendre $\varepsilon\leqslant p_ie_i$ et $\varepsilon\leqslant p_je_j$.
Pour assurer les contraintes $(e.2)$, il suffit de prendre $\varepsilon\leqslant p_i[M-e_i]$ et $\varepsilon\leqslant p_j[M-e_j]$.
Pour assurer les contraintes (S1'), il suffit de prendre $\varepsilon\leqslant m_1:=\min\{p*[e+p](S^1)\mid S^1\in\mathcal{P}^*(J), i\in S^1, j\notin S^1\}$
$\varepsilon\leqslant m_2:=\min\{p*[e+p](S^1)\mid S^2\in\mathcal{P}^*(J), i\notin S^2, j\in S^2\}$
et d'après le lemme 6.2 cela suffit pour assurer les contraintes (S1).
On pose alors $\varepsilon=\min(m_1,m_2,M-e_i,M-e_j,e_i,e_j)$, mais pour conclure il nous faut $\varepsilon>0$.
Les termes $M-e_i$ et $M-e_j$ sont positifs par les contraintes $(e.2)$, et non nuls par hypothèse.
Supposons que $e_i=0$, l'inégalité stricte de l'hypothèse devient alors $e_j+p_j<p_i+p_j$ donc $e_j<p_i$. Mais la contrainte (S1) pour $S=\{i,j\}$ (i.e. $p_ie_i+p_je_j\geqslant p_ip_j$) donne $p_je_j\geqslant p_ip_j$ soit $e_j\geqslant p_i$. ABSURDE
Supposons que $e_j=0$, l'inégalité large de l'hypothèse devient alors $e_i+p_i\leqslant p_j$ donc $e_i<p_j$. Mais la contrainte (S1) pour $S=\{i,j\}$ donne ici $p_ie_i\geqslant p_ip_j$ soit $e_i\geqslant p_j$. ABSURDE
Pour les deux termes restants, on applique les lemmes 5.23 et 5.22 au vecteur $C^e:=e+p$, puisque m_1 et m_2 sont à C^e ce que ε_1 et ε_2 sont à C dans 5.4.1. On peut bien les utiliser car l'hypothèse se réécrit $C^e_i\leqslant C^e_j<C^e_i+p_j$ et que le lemme 6.2 assure que e vérifie les contraintes (S1'), ce qui revient à dire que C^e satisfait toutes les inégalités de Queyranne. On en déduit $m_1>0$ et $m_2>0$. CQFD

On montre facilement le deuxième point en faisant pour les t ce qu'on a fait pour les e. La seule partie qui diffère un peu est la stricte positivité de t_i et t_j. Puisque $t_i\leqslant t_j$ il suffit de montrer que $t_i>0$.
Supposons que $t_i=0$. L'inégalité stricte de l'hypothèse devient alors $t_j<p_j$, mais la contrainte (S2) pour $S=\{i,j\}$ donne ici $p_jt_j\geqslant p_ip_j+p_i^2+p_j^2>p_j[p_i+p_j]$ donc $t_j>p_i+pj>p_j$ ABSURDE.

Propriété 6.9

Si $x \in Extr^*$ ***alors*** x *code un ordonnancement réalisable* $\square\!\square$ *et* $\odot$.

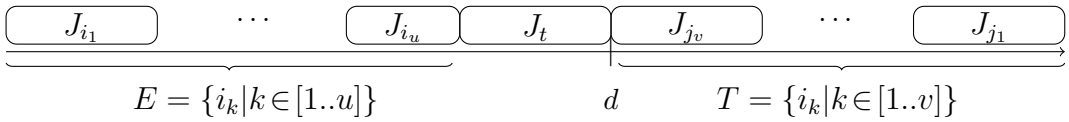
Remarque : Si $x \in \text{Extr}^*$, alors il vérifie les contraintes $(e.\text{-})$, $(t.\text{-})$, $(\delta.\text{-})$, $(l.\text{-})$, $(r.\text{-})$, ce qui nous permet de déterminer de manière unique des périodes d'exécution pour chaque tâche. De plus d'après le lemme 6.7, ses composantes sont toutes différentes de M, ce qui nous permet d'utiliser le lemme 6.8 : par contraposée on en déduit que lesdites périodes d'exécution ne se chevauchent pas, et sont bien d'un côté ou de l'autre de la due date.
Donc x code un ordonnancement $\mathcal{S}$ sans tâche à cheval, pour montrer qu'il est $\odot$ il suffit de montrer qu'il est $\square\!\square$ (car on sait qu'au moins une tâche est en avance, en effet si elles étaient toutes en retard on aurait un $t_j = M$, ce qui est impossible d'après le lemme 6.7).
Par définition de Extr*, $\mathcal{S}$ est optimal pour des poids α β.strictement positifs, donc on a la dominance stricte des ordonnancements $\square\!\square$ et en particulier $\mathcal{S}$ est $\square\!\square$. □

Propriété 6.10

Réciproquement *un ordonnancement réalisable* $\square\!\square$ *et* $\odot$ *est codé par un vecteur de* $Extr^*$.

Remarque : On considère $x^{\mathcal{S}} = (e, t, \delta, l, r)$ le codage de $\mathcal{S}$ un ordonnancement $\square\!\square$ et $\odot$. On sait déjà que $x^{\mathcal{S}} \in P^{\text{"e,t,}\delta\text{,l,r"}}$, on veut maintenant montrer que $x^{\mathcal{S}} \in \text{Extr}^*$. Il suffit pour cela de construire des poids $(\alpha, \beta) \in (\mathbb{R}^+)^{2n}$ tels que $\mathcal{S}$ soit le seul ordonnancement optimal.

En effet si $x^{\mathcal{S}}$ est barycentre de deux points x' et x'' de $P^{\text{"e,t,}\delta\text{,l,r"}}$, ceux-ci ont forcément un δ booléen puisque le δ de $x^{\mathcal{S}}$ est à valeurs dans $\{0, 1\}$ et que le leur est à valeurs dans $[0, 1]$. Ils codent donc aussi des ordonnancements, qu'on appelle respectivement $\mathcal{S}$ ' et $\mathcal{S}$ ".
Par optimalité de $\mathcal{S}$ on sait que $\mathcal{S}$ ' et $\mathcal{S}$ " ont des coûts plus élevés, mais puisque ce coût correspond à la fonction linéaire $\varphi = x \mapsto \langle (\alpha, \beta, 0, 0, 0) \,|\, x \rangle$ et que $x^{\mathcal{S}}$ est sur le segment $[x', x'']$ on a finalement l'égalité des coûts : $\mathcal{S}$, $\mathcal{S}$ ' et $\mathcal{S}$ " sont tous trois optimaux. Or il n'y a qu'un seul optimum donc $\mathcal{S} = \mathcal{S}' = \mathcal{S}''$, et $x^{\mathcal{S}} = x' = x''$. On en déduit que $x^{\mathcal{S}}$ est un point extrême de $P^{\text{"e,t,}\delta\text{,l,r"}}$ et par suite que $x \in \text{Extr}^*$ en tant que minimiseur de φ.



On construit donc de tels α et β en s'appuyant sur la description de $\mathcal{S}$ par les notations illustrées ci-dessus.

On pose : $\forall k \in [1..v]$, $\boldsymbol{\beta_{j_k}} = k * p_{j_k}$	ainsi $\beta_{j_k} = k \leqslant n$ et$(\beta_{j_k}/p_{j_k})_{k\in[1..v]} \nearrow\nearrow$
$\boldsymbol{\alpha_t} = (n+1) * p(T)$	ainsi $(\beta_{k_j}/p_{k_j})p(T) < \alpha_t$
$\forall k \in [1..v]$, $\boldsymbol{\alpha_{j_k}} \in \left]\beta_{j_k} p(T)/p_t \,,\, p_{j_k}\alpha_t/p_t\right[$	ainsi $\alpha_{j_k}/p_{j_k} < r_t := \alpha_t/p_t$ et $\alpha_{j_k} p_t > \beta_{j_k} p(T)$
$\forall k \in [1..u]$, $\boldsymbol{\alpha_{i_k}} = k/(u+1) * r_t * p_{i_k}$	ainsi $\alpha_{i_k}/p_{i_k} < r_t$
$\boldsymbol{z} = \sum_{j\in E} \alpha_j e_j + \sum_{j\in T} \beta_j t_j$	c'est le coût de l'ordonnancement $\mathcal{S}$
$\forall k \in [1..u]$, $\boldsymbol{\beta_{i_k}} = (z+1)/p_{i_k}$	ainsi $p_{i_k} * \beta_{i_k} > z$
$\boldsymbol{\beta_t} = (z+1)/p_t$	ainsi $p_t * \beta_t > z$

Montrons que $\mathcal{S}$ est optimal pour ces poids, et qu'il est le seul. Soit $\mathcal{S}^*$ un ordonnancement optimal $\odot$ et z^* son coût. Puisqu'aucun poids n'est nul, $\mathcal{S}^*$ est nécessairement $\square\!\square$.
Si J_t est en retard dans $\mathcal{S}^*$, alors $z^* \geqslant \beta_t\, p_t = z{+}1$. ABSURDE. Donc $t \in E'(\mathcal{S}^*)$, et comme pour tout $j \in J_{\setminus\{t\}}$, et a fortiori pour tout $j \in E'(\mathcal{S}^*)$, $\alpha_t/p_t > \alpha_j/p_j$ on en déduit que J_t est aussi la tâche à l'heure de $\mathcal{S}^*$.
S'il existe $k \in [1..u]$ tel que J_{i_k} est en retard dans $\mathcal{S}^*$, alors $z^* \geqslant \beta_{i_k}\, p_{i_k} = z{+}1$. ABSURDE. Donc $E(\mathcal{S}) \subset E(\mathcal{S}^*)$.
S'il existe $k \in [1..v]$ tel que J_{j_k} est en avance dans $\mathcal{S}^*$, alors elle est nécessairement placée avant J_t, son avance est donc supérieure à p_t et sa pénalité supérieure à $p_t\, \alpha_{j_k} > \beta_{j_k}\, p(T(\mathcal{S})) \geqslant \beta_{j_k}\, p(T(\mathcal{S}^*))$. Cela signifie que la tâche J_{j_k} aurait une pénalité moindre en étant placée après toutes les tâches en retard de $\mathcal{S}^*$, et que ce faisant on obtiendrait un ordonnancement de coût moindre que celui de $\mathcal{S}^*$. ABSURDE.
Donc $T(\mathcal{S}) \subset T(\mathcal{S}^*)$ et comme $J = T(\mathcal{S}) \sqcup E'(\mathcal{S}) = T(\mathcal{S}^*) \sqcup E'(\mathcal{S}^*)$ on a $E(\mathcal{S}) = E(\mathcal{S}^*)$ et $T(\mathcal{S}) = T(\mathcal{S}^*)$

On en déduit que $\mathcal{S} = \mathcal{S}^*$ par la dominance stricte des ordonnancements $\nearrow\searrow^r$. (Attention aux ratios à considérer ici : puisque les coûts sont non symétriques le ratio d'une tâche en avance est α_j/p_j, tandis que celui d'une tâche en retard β_j/p_j).
Ainsi $\mathcal{S}$ est le seul ordonnancement 🕛 optimal. Or s'il y avait aussi un ordonnancement optimal avec une tâche à cheval sur la due-date, on pourrait avancer ou retarder cet ordonnancement et obtenir ainsi deux ordonnancements encore optimaux et 🕛. On en déduit que le seul ordonnancement optimal est $\mathcal{S}$. □

7 Problème de séparation associé

Pour gérer les inégalités de type (S1) et (S2) qui sont en nombre exponentiel, on doit fournir un algorithme de séparation. Étant donné un vecteur x, cet algorithme doit nous confirmer que $x \in P^{\text{"e,t,}\delta\text{,l,r"}}$ ou nous donner des contraintes de type (S1) et (S2) transgressées par x.

Les deux familles d'inégalités sont relativement indépendantes, intuitivement l'une gère le non-chevauchement des tâches en avance, et l'autre celui des tâches en retard. On a donc deux problèmes de séparation analogues mais indépendants. On détaille le deuxième, le premier se traitant par la même méthode. Formellement il s'agit de résoudre le problème suivant :

SEPARATION_2

Entrée : $x = (e, t, \delta, l, r)$ un point qui vérifie les inégalités de $P^{\text{"e,t,}\delta\text{,l,r"}}$ sauf éventuellement celles de type (S1) ou (S2)

Sortie : oui si x vérifie toutes les inégalités de type (S2)
un ensemble $S \subset J$ tel que x viole la contrainte (S2) sinon

On pose $\Gamma_2 = \begin{pmatrix} \mathcal{P}(J) & \longrightarrow & \mathbb{R} \\ S & \longmapsto & \sum\limits_{\substack{(i,j)\in S^2 \\ i<j}} p_i p_j r_{i,j} - \sum\limits_{i\in S} p_i^2(1\text{-}\delta_i)\text{-}p_i t_i \end{pmatrix}$ (resp. $\Gamma_1 = \begin{pmatrix} \mathcal{P}(J) & \longrightarrow & \mathbb{R} \\ S & \longmapsto & \sum\limits_{\substack{(i,j)\in S^2 \\ i<j}} p_i p_j l_{i,j} - \sum\limits_{i\in S} p_i e_i \end{pmatrix}$).

Notre approche consiste à chercher à maximiser Γ_2 : si le maximum trouvé est négatif, alors toutes les contraintes de type (S2) sont satisfaites, sinon on a trouvé un ensemble S qui maximise Γ_2 et qui est donc associé à une des contraintes (S2) les plus transgressées.

Puisque tous les coefficients $(p_j)_{j\in J}$ sont positifs les fonctions Γ_1 et Γ_2 sont super-modulaires. Notre problème de séparation est donc **polynomial** puisque maximiser une fonction super-modulaire équivaut à minimiser une fonction sous-modulaire, et qu'il existe plusieurs algorithmes polynomiaux pour ce problème (cf. [11] pour un survol). Mais ces algorithmes sont coûteux, et avec une évaluation de la fonction qui se fait en $O(n^2)$, on aurait au mieux une complexité en $O(n^7 \log n)$ avec l'algorithme de Iwata et Orlin [15].

Notre fonction n'étant pas une fonction super-modulaire quelconque, on peut espérer une meilleure complexité pour cette maximisation. Une première tentative a été une approche gloutonne inspirée de l'algorithme de séparation que donne Queyranne [12] : on exprime facilement la variation de la Γ_2 lorsqu'on ajoute à un ensemble S un élément $k \in J_{\setminus S}$, ou lorsqu'on lui retire un élément. On part alors de l'ensemble $S = \{1\}$, et à chaque étape on choisit d'ajouter le nouvel élément qui offre la plus grande augmentation. Si aucun n'offre une variation positive on passe dans une deuxième phase où l'on retire à chaque étape l'élément qui offre la plus grande augmentation. Si plus aucun élément de l'ensemble courant n'offre de variation positive ou si l'ensemble courant a été vidé, on a terminé, en $O(n^2)$. Malheureusement ce qu'on a obtenu est un genre de maximum local (au sens où les ensembles dont l'indicatrice est un sommet voisin de $\mathbb{1}_S$ sur l'hypercube ont tous des valeurs plus petites), mais pas un maximum global. Il y a des contre-exemples, cependant dans des cas où le maximum global est négatif, c'est-à-dire des cas où l'on n'a pas vraiment besoin de maximiser pour notre séparation.

Cette piste reste donc à creuser, mais dans le souci de présenter un code complet on s'est rabattu sur une solution plus rapide à comprendre et à coder (mais de complexité **exponentielle !**) : on ramène la maximisation de Γ_2 au problème MAX-CUT dans un graphe complet, dont on code la

résolution par un PLNE compact. L'astuce utilisée pour se ramener à MAX-CUT (utilisée pour la modélisation des verres de spin), ainsi que la formulation compacte de MAX-CUT pour les graphes complets sont développées dans [1]. Je présente ici les calculs ramenant la maximisation de Γ_2 à la recherche d'une coupe minimale dans un graphe complet aux pondérations quelconques. [15].

$$
\begin{aligned}
\max_{S\in\mathcal{P}(J)} \Gamma_2(S) &= \max_{x\in\{0,1\}^J} \left(\sum_{(i,j)\in J^<} \underbrace{p_i p_j r_{i,j}}_{:=q_{i,j}} \; x_i x_j + \sum_{i\in J} \underbrace{p_i^2(1-\delta_i) - p_i t_i}_{:=c_i} \; x_i \right) \\
&= \max_{x\in\{0,1\}^J} \left(\sum_{(i,j)\in J^<} q_{i,j} \; x_i x_j + \sum_{i\in J} c_i \; x_i \right) \\
&= \max_{s\in\{-1,1\}^J} \left(\sum_{(i,j)\in J^<} q_{i,j} \left(\frac{s_i+1}{2}\right)\left(\frac{s_j+1}{2}\right) \; + \; \sum_{i\in J} c_i \left(\frac{s_i+1}{2}\right) \right) \\
&= \max_{s\in\{-1,1\}^J} \left(\sum_{(i,j)\in J^<} \frac{q_{i,j}}{4} \; [s_i s_j + s_i + s_j + 1] \; + \; \sum_{i\in J} \frac{c_i}{2} \; [s_i + 1] \right) \\
&= \max_{s\in\{-1,1\}^J} \left(\sum_{(i,j)\in J^<} \frac{q_{i,j}}{4} \; s_i s_j \; + \; \sum_{i\in J} \left[\sum_{j>i} \frac{q_{i,j}}{4} + \sum_{j<i} \frac{q_{j,i}}{4} + \frac{c_i}{2} \right] s_i \; + \; \frac{1}{4} \underbrace{\sum_{(i,j)\in J^<} q_{i,j}}_{:=Q} \; + \; \frac{1}{2} \underbrace{\sum_{i\in J} c_i}_{:=C} \right) \\
&= \frac{1}{4} \max_{s\in\{-1,1\}^J} \left(\sum_{(i,j)\in J^<} \underbrace{\frac{q_{i,j}}{:=\omega_{i,j}}} \; s_i s_j \; + \; \sum_{i\in J} \underbrace{\left[2c_i + \sum_{j>i} q_{i,j} + \sum_{j<i} q_{j,i} \right]}_{:=\omega_{0,i}} \; s_i \right) + \frac{Q}{4} + \frac{C}{2} \\
&= \frac{1}{4} \max_{\substack{s_0=1 \\ s\in\{-1,1\}^J}} \left(\sum_{(i,j)\in J^<} \omega_{i,j} \underbrace{s_i s_j}_{\substack{\uparrow \\ =1 \text{ si } s_i=s_j \\ =-1 \text{ sinon}}} + \sum_{i\in J} \omega_{0,i} \underbrace{s_0 s_i}_{\substack{\uparrow \\ =1 \text{ si } s_i=s_0 \\ =-1 \text{ sinon}}} \right) + \frac{Q}{4} + \frac{C}{2} \\
&= \frac{1}{4} \max_{\substack{s_0=1 \\ s\in\{-1,1\}^J}} \left(\sum_{(i,j)\in J^<} \omega_{i,j} - 2 \sum_{\substack{(i,j)\in J^< \\ s_i\neq s_j}} \omega_{i,j} \; + \; \sum_{i\in J} \omega_{0,i} - 2 \sum_{\substack{i\in J \\ s_i\neq s_@}} \omega_{0,i} \right) + \frac{Q}{4} + \frac{C}{2} \\
&= \frac{-2}{4} \min_{\substack{s_0=1 \\ s\in\{-1,1\}^J}} \left(\sum_{\substack{(i,j)\in (J^0)^< \\ s_i\neq s_j}} \omega_{i,j} \right) \; + \; \frac{1}{4} \underbrace{\sum_{(i,j)\in J^<} \omega_{i,j}}_{=Q} \; + \; \frac{1}{4} \sum_{i\in J} \omega_{0,i} \; + \; \frac{Q}{4} + \frac{C}{2} \\
&= -\frac{1}{2} \min_{\substack{s_0=1 \\ s\in\{-1,1\}^J}} \left(\sum_{\substack{(i,j)\in (J^0)^< \\ s_i\neq s_j}} \omega_{i,j} \right) \; + \; \frac{Q}{4} \; + \; \frac{1}{4} \left[\underbrace{\sum_{i\in J} 2c_i}_{=2C} + \underbrace{\sum_{i\in J} \sum_{j>i} q_{i,j}}_{=Q} + \underbrace{\sum_{i\in J} \sum_{j<i} q_{j,i}}_{=Q} \right] \; + \; \frac{Q}{4} + \frac{C}{2} \\
&= -\frac{1}{2} \left(\begin{array}{l} \text{valeur d'une } \textbf{coupe min} \\ \text{dans le graphe complet non orienté} \\ \text{d'ensemble de sommets } J\cup\{0\} \\ \text{et d'arêtes pondérées par } \omega \end{array} \right) \; + \; Q + C
\end{aligned}
$$

15. C'est du fait des poids négatifs que notre recherche d'une coupe "min" se ramène à MAX-CUT, et non à MIN-CUT

8 Résultats expérimentaux

Puisque notre formulation contient des variables entières, nous avons utilisé un algorithme de branchement, comme on utiliserait un algorithme de Branch-and-Bound pour résoudre un PLNE. Mais ici, à chaque nœud de l'arbre de branchement, on doit résoudre un PL qui présente un nombre exponentiel de contraintes de type (S1) et (S2). Donc à chaque nœud utilise un algorithme de coupes analogue à celui présenté dans la section 5.3. On parle alors d'algorithme de **Branch-and-Cut**.

Comme l'a fait Queyranne, on commence notre algorithme de coupes avec les contraintes de positivité (ici les contraintes (S2) pour les singletons), mais aussi avec les contraintes (e,-), (t,-),(δ,-),(l,-), et (r,-), qui sont en nombre polynomial. Puis on ajoute au fur et à mesure des contraintes de type (S1) et de type (S2) grâce à l'algorithme de séparation décrit dans la section 7.

Nous avons codé notre algorithme de Branch-and-Cut en utilisant le logiciel **CPLEX (version 12.6.3**) et l'interface Concert Technology. On fournit à CPLEX nos contraintes "fixes", ici les contraintes (e,-), (t,-),(δ,-),(l,-), et (r,-), ainsi que notre algorithme de séparation. Il gère ensuite tout seul le branchement et l'itération de l'algorithme de coupe à chaque nœuds. Puisqu'on a choisi de résoudre le problème de séparation par une formulation PLNE de MAX-CUT dans un graphe complet, on fait appel à CPLEX aussi au sein de la séparation, mais cette utilisation est indépendante de la première.

Pour tester notre code, on a utilisé le **benchmark** proposé par Biskup et Feldmann [2] et disponible en ligne sur http ://people.brunel.ac.uk/ mastjjb/jeb/orlib/schinfo.html. Il est constitué de sept séries de 10 problèmes sans due-date (donc non restrictif), pour $n = 10, 20, 50, 100, 200$ et 1000, auxquels on peut ajouter une due-date $d = h * p(J)$ pour $h = 0.2$, 0.4, 0.6 ou 0.8. afin d'obtenir un panel d'instances restrictives. Les auteurs proposent pour les 280 instances ainsi construites les bornes supérieures fournies par leur heuristique, et précisent quand celles-ci coïncident avec la valeur optimale.

Pour notre problème non restrictif, on ne génère pas de due-date, mais on calcule a posteriori pour quelle valeur minimale de d notre solution est encore valide. Plus précisément on calcule la proportion de la durée des tâches en avance par rapport à la durée globale des tâches, ce qui donne la valeur minimale du coefficient h pour lequel cette solution est valide (pour la due-date $d = h * p(J)$). Cela nous permet de choisir à quelle borne supérieure nous comparer.

On présente ci-dessous les résultats obtenus en lançant notre algorithme sur un ordinateur Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz. Les valeurs de bornes supérieure sont celles fournies par Biskup et Feldmann [2] (obtenues par une méthode exacte pour n=10 et par leurs heuristiques au delà).

Pour les instances de tailles n=10 et 20, on a laissé notre algorithme se dérouler jusqu'au bout, donc les valeurs de l'objectif présentées dans les deux tables suivantes sont les valeurs optimales.

n	k	p(J)	valeur de l'objectif	proportion en avance	borne supérieure	temps en secondes	nombre de coupes	nombre de nœuds
10	1	116	818	0,66	818	<1	10	5
	2	129	615	0,40	615	<1	12	0
	3	125	793	0,54	793	<1	14	10
	4	102	803	0,67	803	<1	13	0
	5	94	521	0,54	521	<1	15	4
	6	88	755	0,50	755	<1	13	0
	7	103	1083	0,60	1083	1,4	18	4
	8	79	540	0,77	540	<1	13	4
	9	92	554	0,67	554	1,3	12	12
	10	127	671	0,69	671	<1	11	0

Dans cette table on précise ici en plus l'écart relatif entre la borne supérieure et notre valeur. On observe alors que les heuristiques de Biskup et Feldmann [2] donnent de très bons résultats sur ces instances.

n	k	p(J)	valeur de l'objectif	proportion en avance	borne supérieure	écart relatif	temps en secondes	nombre de coupes	nombre de nœuds
20	1	217	2986	0,44	2986	0	73	56	269
	2	237	2980	0,70	2980	0	76	54	107
	3	233	3583	0,47	3600	0,47%	65	68	747
	4	230	3040	0,70	3040	0	90	48	145
	5	188	2173	0,53	2206	1,5%	83	77	354
	6	207	3010	0,53	3016	0,19%	68	96	364
	7	244	3878	0,67	3900	0,56%	71	53	250
	8	202	1638	0,53	1638	0	161	52	144
	9	139	1965	0,51	1992	1,3%	107	76	505
	10	216	1972	0,62	1995	1,1%	71	33	100

Pour les instances de taille n=50, on a d'abord mis une limite de temps de quinze minutes, puis on a laissé tourné l'algorithme plus longtemps sur deux instances dont la résolution semblait parmi les mieux engagées au bout d'un quart d'heure. Les différentes exécutions présentées pour l'instance $k = 2$ diffèrent par leur branchement (en particulier le nombre de nœuds n'est pas le même), cependant elles présentent les même valeurs. L'instance pour $k = 3$ a été résolue en un peu moins de deux heures, mais pour l'instance $k = 4$ on a interrompu l'algorithme après une heure quarante-cinq.

n	k	p(J)	borne inférieure	gap CPLEX	borne supérieure	écart relatif	temps en secondes	nombre de coupes	nombre de nœuds
50	1	549	6894	-	17990	158%	1826.31	139	0
	2	512	6392	65,64%	14132	121%	1902.59	172	12
		512	6392	65,64%	14132	121%	1912.11	172	27
		512	6392	65,64%	14132	121%	1902.51	172	22
	3	535	10400	20.62%	16947	59%	2085.8	233	20
	4	478	9057	79.27%	14105	55%	2133.6	195	31
	5	542	5897	-	14650	148%	1856.31	170	0
	6	548	7356	87.01%	14075	91%	1850.51	163	10
	8	638	11945	44.47%	21367	78%	1614.72	111	4
	9	457	9108	83,79%	13952	53%	2135.03	172	10
	10	505	9257	82.25%	14377	55%	2137.32	183	10
50	3	535	16495.4	0	16947	0,01%	1h50	604	30 007
	4	478	12274	13.28%	14105	14%	1h45	748	12733

Conclusion

Grâce à des inégalités inspirées des inégalités de Queyranne [12], on a pu traduire le non chevauchement des tâches pour notre problème d'ordonnancement juste-à-temps avec due-date commune non restrictive, et ainsi obtenir une formulation PLNE (non compacte) du problème.

Si Queyranne a pu démontrer la validité de ses inégalités par une preuve reposant sur une description géométrique, il nous a fallu changer de méthode pour prouver que notre formulation était correcte, mais dans les deux cas on s'appuie sur des propriétés de dominance connues pour ces problèmes d'ordonnancement.

Notre formulation, comme elle est actuellement codée, permet de résoudre de manière exacte des petites instances (pour une vingtaine de tâches), mais elle peut être largement améliorée notamment au niveau de la séparation. En effet par manque de temps on a utilisé un algorithme exponentiel (la résolution d'un PLNE), alors qu'on sait que notre problème de séparation est polynomial puisque la maximisation d'une fonction super-modulaire l'est. De plus on a émis une idée d'algorithme en $O(n^2)$ qui, même s'il n'est pas exact, pourrait servir d'heuristique dans la séparation : on ajouterait les inégalités violées qu'il fournirait jusqu'à ce qu'il n'en trouve plus, alors on utiliserait un algorithme exact mais plus coûteux pour s'assurer que toutes les inégalités sont satisfaites.

On peut aussi espérer l'adapter au cas restrictif en ajoutant simplement les contraintes $e_j \leqslant d$ pour tout $j \in J$, mais la preuve que les points extrêmes sont sans chevauchement demande à être retravailler.

Par la suite on pourra essayer de retrouver la polynomialité du problème pour des poids indépendants des tâches, en étudiant les solutions obtenues en résolvant le PL relâché (i.e. sans les contraintes d'intégrité) pour voir si une étape d'arrondi permettrait de trouver une solution optimale comme c'est le cas dans le problème du sac-à-dos lorsque les coûts sont tous identiques.

On pourra aussi s'intéresser aux autres formulations PLNE du problème, qu'on a ici laisser de côté, dans le but d'obtenir finalement un codage pour lequel on connaît la description minimale du polyèdre des solutions.

ANNEXES

9 Quelques définitions

9.1 Définitions d'analyse convexe

Définition 9.1

Soit $A \subset \mathbb{R}^n$.
A *est* ***convexe*** $\Leftrightarrow \forall (x,y) \in A^2, \forall \theta \in]0,1[,\ \theta x + (1\text{-}\theta) y \in A$
A *est un* ***cône*** $\Leftrightarrow \forall x \in A, \forall \lambda \in \mathbb{R}^{+*},\ \lambda x \in A$

Définition 9.2

Soit $A \subset \mathbb{R}^n$.
On note ***Conv(***A***)*** *le plus petit convexe contenant* A.
On note ***Cone(***A***)*** *le plus petit cône contenant* A.
On note ***CC(***A***)*** *le plus petit cône convexe contenant* A.
On note ***CC°(***A***)*** *le plus petit cône convexe contenant* $A \cup \{0\}$.

Définition 9.3

Soit $A \subset \mathbb{R}^n$. *Soit* $u \in \mathbb{R}^n$
u *est une* ***direction de récession*** *de* $A \Leftrightarrow \forall x \in A, \forall \lambda \in \mathbb{R}^{+*},\ x + \lambda u \in A$
On note $\mathbf{0}^+(A)$ *le* ***cône de récession*** *de* A*, c'est-à-dire l'ensemble de ses directions de récession.*

Remarque : Si $C \subset \mathbb{R}^n$ est convexe, alors $0^+(C)$ est un cône convexe contenant 0.

Définition 9.4

Soit C *un convexe de* $\mathbb{R}^n$. *Soit* $x \in \mathbb{R}^n$
x *est un* ***point extrême*** *de* $C \Leftrightarrow \forall (y,z) \in C^2, \forall \theta \in]0,1[,\ x = \theta y + (1\text{-}\theta) z \Rightarrow x = y = z$
On note ***Extr(***C***)*** *l'ensemble des points extrêmes de* C.

9.2 Définitions de sous-modularité

Définition 9.5

Soit J *un ensemble fini non vide. Soit* g *une fonction de* $\mathcal{P}(J)$ *dans* $\mathbb{R}$.
g *est* ***sous-modulaire*** $\Leftrightarrow \forall (A,B) \in \mathcal{P}(J)^2,\ g(A \cup B) \leqslant g(A) + g(B) - g(A \cap B)$
$\Leftrightarrow \forall (X,Y) \in \mathcal{P}(J)^2, \forall z \in Y^C, X \subset Y \Rightarrow\ g(X') - g(X) \geqslant g\big(Y'\big) - g(Y)$
où $X' := X \cup \{z\}$ *et* $Y' := Y \cup \{z\}$

Remarque : L'équivalence entre les deux définitions est montrée dans [11]

Définition 9.6

Un polyèdre P *est dit* ***sous-modulaire*** *s'il existe une fonction* g *sous-modulaire définie sur* $\mathcal{P}(J)$ *et vérifiant* $g(\emptyset) = 0$ *telle que* $P = \big\{x \in \mathbb{R}^n \,|\, \forall S \in \mathcal{P}(J),\ \langle \mathbb{1}_S | x \rangle \leqslant g(S)\big\}$

Définition 9.7

Soit J *un ensemble fini non vide. Soit* g *une fonction de* $\mathcal{P}(J)$ *dans* $\mathbb{R}$.
g *est* ***sous-modulaire*** $\Leftrightarrow \forall (A,B) \in \mathcal{P}(J)^2,\ g(A \cup B) \geqslant g(A) + g(B) - g(A \cap B)$
$\Leftrightarrow \forall (X,Y) \in \mathcal{P}(J)^2, \forall z \in Y^C, X \subset Y \Rightarrow\ g(X') - g(X) \leqslant g\big(Y'\big) - g(Y)$
où $X' := X \cup \{z\}$ *et* $Y' := Y \cup \{z\}$

10 Catalogue de contre-exemples pour des formulations qui ne marchent pas

10.1 Formulations pour le problème étudié par Queyranne [12]

10.1.1 Une formulation avec variables disjonctives

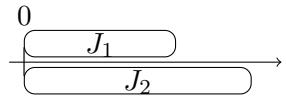
Pour gérer le non-chevauchement des tâches dans le problème étudié par Queyranne, on peut d'abord penser qu'il suffit d'introduire des variables disjonctives $(a_{i,j})_{(i,j)\in J^<}$ qui indiquent si J_i a lieu avant J_j ($a_{i,j}=1$ dans ce cas). On considère alors le polyèdre suivant :

$$P^{C,a} = \left\{ (C,a) \in \mathbb{R}^n \times \mathbb{R}^{J^<} \;\middle|\; \begin{array}{ll} \forall j \in J,\ C_j \geqslant p_j & (1) \\ \forall (i,j) \in J^<,\ 0 \leqslant a_{i,j} \leqslant 1 & (2) \\ \forall (i,j) \in J^<,\ C_i - p_i - C_j \geqslant -p(J) * a_{i,j} & (3) \\ \forall (i,j) \in J^<,\ C_j - p_j - C_i \geqslant -p(J) * (1-a_{i,j}) & (4) \end{array} \right\}$$

En effet pour $(i,j)\in J^<$, si $a_{i,j}=1$ la contrainte (4) s'écrit $C_j - p_j - C_i \geqslant 0$ soit $C_j \geqslant C_i + p_j$, on retrouve la contrainte de non-chevauchement voulue, et si $a_{i,j}=0$ c'est la contrainte (3) qui donne $C_i \geqslant C_j + p_j$. Mais la contrainte (4) dans le cas où $a_{i,j}=0$ donne $-C_i \geqslant -p(J) - C_j + p_j$ soit $C_i \leqslant p(J) + (C_j - p_j)$. Cette contrainte est valide pour un ordonnancement $\square\!\!\square$ puisqu'alors on a nécessairement $C_i \leqslant p(J)$, et que la contrainte (1) assure que $C_j - p_j \geqslant 0$. De même la contrainte (3) est valide pour un ordonnancement $\square\!\!\square$ même quand $a_{i,j}=1$. Résoudre le problème de Queyranne revient alors à minimiser une fonction linéaire des C_i sur les points entiers de $P^{C,a}$ (en fait on n'a besoin de l'intégrité seulement pour les $a_{i,j}$, mais si les durées des tâches sont entières les C_i le sont aussi dans un ordonnancement $\square\!\!\square$).
Mais résoudre un PLNE n'est pas facile, à moins d'être sûr qu'en résolvant le relâché continu on tombe sur un point entier,ce qui n'est pas le cas ici d'après le petit contre-exemple suivant.

Contre-exemple

Pour le problème où $n=2$, $p_1=2$ et $p_2=3$, et où on a donc $p(J)=5$
on considère le vecteur solution défini par $C_1=2$, $C_2=3$ et $a_{1,2}=3/5$



$C_1 - p_1 - C_2 = 2-2-3 = -5*(3/5) = -p(J)*a_{1,2}$ *donc la contrainte (3) est saturée*
$C_2 - p_2 - C_1 = 3-3-2 = -5*(2/5) = -p(J)*(1-a_{1,2})$ *donc la contrainte (4) est saturée*
Donc (C,a) est un point extrême de $P^{C,a}$ en tant que point de $P^{C,a}$ saturant deux inégalités.
Pourtant il n'est pas entier, et il ne représente pas un ordonnancement valide.

La conclusion n'est pas que cette formulation avec des variables disjonctives est fausse, mais qu'elle n'apporte rien, au contraire elle nous incite à utiliser un algorithme exponentiel alors que le problème est dans **P**.

10.1.2 Une formulation avec des vecteurs triés

Une autre approche est de se dire que l'on a besoin de connaître (en plus des valeurs C_i) l'ordre dans lequel les tâches sont effectuées, autrement dit le $\sigma \in \mathfrak{S}_n$ qui rend $\left(C_{\sigma(j)}\right)_{j\in[1..n]}$ croissant.

En admettant qu'on sait obtenir de manière linéaire à partir de C le vecteur $\overrightarrow{C}$ de ses composantes triées (i.e. $\overrightarrow{C} = \left(C_{\sigma(j)}\right)_{j\in[1..n]}$), on imagine être capable d'exprimer le non-chevauchement.

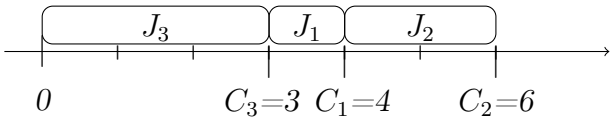
Mais si on essaye on écrit $\overrightarrow{C}_2 \geqslant \overrightarrow{C}_1 + p_k$, où k doit être l'indice de la première tâche, i.e. $k=\sigma(1)$, malheureusement on ne connaît pas $\sigma(1)$. Tout le problème est là **connaître les valeurs triées est moins fort que connaître le tri**.

Une autre façon de se convaincre que les $(\overrightarrow{C}_i)$ ne peuvent exprimer (à eux seuls) le non-chevauchement, est qu'un même vecteur $\overrightarrow{C}$ peut correspondre tout autant à un C codant un ordonnancement réalisable qu'à un C codant un chevauchement.

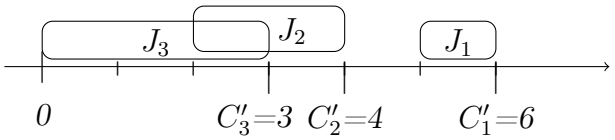
Contre-exemple

Avec $n = 3$, $p_1 = 1$, $p_2 = 2$, $p_3 = 3$

le vecteur $C = (6, 4, 3)$ *correspond à*



le vecteur $C' = (4, 6, 3)$ *correspond à*



pourtant $\overrightarrow{C} = \overrightarrow{C'} = (3, 4, 6)$

10.2 Formulations pour notre problème juste-à-temps

10.2.1 Une formulation avec des variables adaptées à la gestion de projet

Si l'on modélise un projet par un ensemble de tâches à réaliser, liées par des contraintes de précédence uniquement, on est ramené à un problème d'ordonnancement avec une infinité de machines. Il n'est alors plus question des contraintes de non-chevauchement puisque les tâches ne sont plus en conflit pour utiliser **la** machine, elles sont en revanche soumises aux contraintes de précédence, qui ont la même forme : si J_i doit être exécutée avant J_j, la contrainte s'écrit $C_j \geqslant C_i + p_j$. La différence par rapport au cas une machine c'est qu'on n'a **plus de contraintes avec alternative** : soit J_i et J_j sont liées par la contrainte de précédence $C_j \geqslant C_i + p_j$, soit elles sont liées par $C_i \geqslant C_j + p_i$, soit J_i et J_j ne sont pas liées.

Dans ce cadre il existe une formulation adaptée à la minimisation d'une fonction coût type juste-à-temps, i.e $\sum_{j \in J} \alpha_j [C_j - d_j]^+ + \beta_j [d_j - C_j]^+$ où les $(d_j)_{j \in J}$ sont les dates d'échéance souhaitées.

Cette formulation est développée sous forme d'exercice dans [4] et consiste essentiellement à introduire, en plus des variables $(C_j)_{j \in J}$, des variables $(F_j)_{j \in J}$ représentant la pénalité engendrée par les tâches (i.e. $F_j = \alpha_j e_j + \beta_j t_j$ pour notre codage).

On obtient assez facilement les contraintes $\forall j \in J,\ F_j \geqslant \alpha_j(d_j - C_j)$ et $\forall j \in J,\ F_j \geqslant \beta_j(C_j - d_j)$ en écrivant
$$\begin{aligned} F_j &= \alpha_j [C_j - d_j]^+ + \beta_j [d_j - C_j]^+ \\ &= \alpha_j \max[0, C_j - d_j] + \beta_j \max[0, d_j - C_j] \\ &= \max[0, \alpha_j(C_j - d_j)] + \max[0, \beta_j(d_j - C_j)] \\ &= \max[\alpha_j(C_j - d_j), \beta_j(d_j - C_j)] \end{aligned}$$
On montre ensuite que ça suffit c'est-à-dire que les points extrêmes susceptibles d'être atteints en minimisant $\sum F_j$ vérifieront bien $F_j = \max[\alpha_j(C_j - d_j), \beta_j(d_j - C_j)]$.

On a donc essayé de transposer cela à notre cas en remplaçant les contraintes de précédence par des contraintes essayant de traduire le non-chevauchement, et on a considéré le polyèdre [16] suivant :

$$P^{\text{"FC"}} = \left\{ (C, F) \in \mathbb{R}^n \times \mathbb{R}^n \;\middle|\; \begin{array}{l} \forall j \in J,\ \ F_j \geqslant \alpha_j(d - C_j) \\ \forall j \in J,\ \ F_j \geqslant \beta_j(C_j - d) \\ \forall S \in \mathcal{P}^*(J),\ \ p * C(S) \geqslant g(S) \end{array} \right\}$$

16. Notez qu'ici le polyèdre dépend des poids α et β, puisqu'ils sont nécessaires à la variable F_j

Mais les variables C_i ne peuvent traduire le non-chevauchement. Dans l'idée on peut dire qu'elles sont adaptées quand le "0 est important",c'est-à-dire quand les tâches vont se caler à gauche parce qu'on est dans un problème régulier, mais pas aux problèmes juste-à-temps (non restrictifs) où c'est "d qui est important", et qui doit servir d'origine parce que c'est autour de d que les tâches se collent.

En pratique dans les instances où d est très grand devant $\max\{g(S)|S \in \mathcal{P}(J)\}$, comme dans l'exemple suivant, les tâches ont toute liberté quant à leur chevauchement.

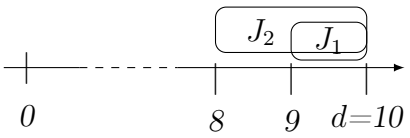
Contre-exemple

Pour l'instance :

$p_1 := 1 \quad \alpha_1 := 1 \quad \beta_1 := 2 \quad d := 10 \quad g(\{1\}) = p_1^2 = 1 \quad g(\{1,2\}) = p_1^2 + p_2^2 + p_1p_2 = 7$

$p_2 := 2 \quad \alpha_2 := 2 \quad \beta_2 := 3 \qquad\qquad g(\{2\}) = p_2^2 = 4$

il existe un point extrême avec chevauchement :



$C_1 := 10$

$C_2 := 10 \qquad C_1p_1 = 9 \geqslant 1$

$F_1 := 0 \qquad C_2p_2 = 16 \geqslant 4 \qquad F_1 = 0 = \alpha_1(d - C_1) \quad F_2 = 0 = \alpha_2(d - C_2)$

$F_2 := 0 \qquad C_1p_1 + C_2p_2 = 25 \geqslant 7 \quad F_1 = 0 = \beta_1(C_1 - d) \quad F_2 = 0 = \beta_2(C_2 - d)$

Remarque : Le point (F, C) est un point extrême de $P^{\text{"FC"}}$ car il sature quatre contraintes linéairement indépendantes. Bien que ces contraintes aient l'air deux à deux redondantes parce qu'on lit deux fois $F_i \geqslant 0$, il s'agit bien de contraintes linéairement indépendantes puisqu'elles correspondent aux vecteurs [17] $(1, 0, \alpha_1, 0) = (1, 0, 1, 0)$, $(1, 0, -\beta_1, 0) = (1, 0, -2, 0)$, $(0, 1, 0, \alpha_2) = (0, 1, 0, 2)$ et $(0, 1, 0, -\beta_2) = (0, 1, 0, -3)$, qui sont bien linéairement indépendants. □

10.2.2 Une formulation avec une nouvelle fonction $\tilde{g}$ pour le non-chevauchement

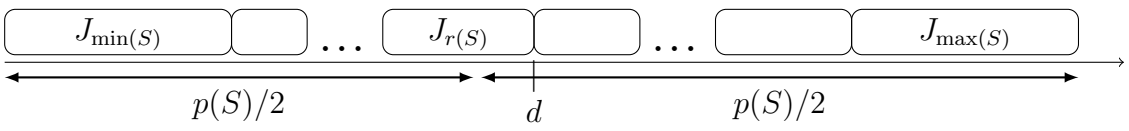
Puisque Queyranne [12] a réussi a traduire le non-chevauchement en considérant les coûts ω^S pour tous les $S \subset J$, on peut espérer faire de même dans le problème juste-à-temps. On veut donc, pour chaque $S \subset J$, minorer le coût engendré par les tâches de S pour $\omega_j = p_j$, par le coût minimal d'un ordonnancement de ces tâches pour $\omega_j = p_j$, qu'on note alors $\tilde{\boldsymbol{g}}(S)$, Autrement dit on s'intéresse au polyèdre suivant :

$$P^{\text{"e,t,}\delta\text{"}} = \left\{ (e,t,\delta) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{Z}^n \;\middle|\; \begin{array}{l} \forall j \in J,\ 0 \leqslant \delta_j \leqslant 1 \\ \forall j \in J,\ e_j \leqslant M\ \delta_j \\ \forall j \in J,\ t_j \leqslant M(1 - \delta_j) \\ \forall S \in \mathcal{P}^*(J),\ \alpha * e(S) + \beta * t(S) \geqslant \tilde{g}(S) \end{array} \right\}$$

Comme Queyranne utilise la règle de Smith pour définir g, on utilise un résultat de Hall et Posner [7] pour calculer ces valeurs $\tilde{g}(S)$. En effet, ils identifient le cas $\omega_j = p_j$ comme polynomial en proposant l'algorithme suivant :

1. Trier les tâches par durées décroissantes, et former ainsi un bloc.
2. Trouver à partir de quelle tâche la moitié de $p(J)$ est écoulée, et placer le bloc de sorte que d coïncide avec la fin de celle-ci.

On suppose dans la suite de cette section que $(p_j)_{j \in [1..n]}$ est décroissante de sorte que l'étape 1 devient superflue. Pour simuler l'étape 2, on pose pour chaque $S \subset J$, $\boldsymbol{r}(S) := \min\left\{ i \in S \mid \sum_{\substack{j \in S \\ j \leqslant i}} p_j > p(S)/2 \right\}$.



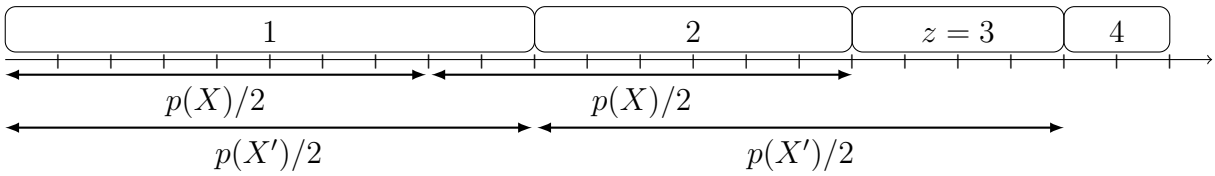
17. Une contrainte linéaire s'écrit toujours $\langle u|x\rangle \leqslant c$, et l'ensemble des points la vérifiant est alors un demi-espace délimité par un hyperplan (affine) de vecteur normal u, on explicite ici les vecteurs u correspondant à ces 4 inégalités

On peut alors écrire $\tilde{g}(S) = \sum_{\substack{j\in S\\ j\leqslant r(S)}} p_j * p\big(S\cap]j..r(S)]\big) \;+\; \sum_{\substack{i\in S\\ i>r(S)}} p_i * p\big(S\cap]r(S)..i]\big)$.

En notant $E'(S) = S \cap [1..r(S)]$ les tâches en avance dans l'ordonnancement fourni par l'algorithme pour l'entrée S, et $T(S) = S\cap]r(S)..n]$ celles en retard, on a aussi $\tilde{g}(S) = g\big(E'(S)\big)+g\big(T(S)\big)$. Cela ne signifie pas que $\tilde{g}$ est la somme de deux fonctions super-modulaires, mais cette écriture nous donnait bon espoir que $\tilde{g}$ soit super-modulaire. On a essayé de le montrer en s'appuyant sur la caractérisation $\forall X\subset Y\subset J,\ \forall z\in Y^C,\ \tilde{g}\big(X\cup\{z\}\big) - \tilde{g}(X) \leqslant \tilde{g}\big(Y\cup\{z\}\big) - \tilde{g}(Y)$ (cf. définition 9.7).
Après une énorme disjonction de cas sur l'ordre entre $r(X)$, $r(Y)$ et z (pour $X\subset Y\subset J$ et $z\in Y^C$), on a trouvé le contre-exemple suivant qui atteste que **$\tilde{g}$ n'est pas super-modulaire**.

Contre-exemple

$p_1 := 10$	$X := \{1,2\}$	$p(X) = 10+6 = 16$	$p(X)/2 = 8 \in [0, p_1[$	*donc* $r(X) = 1$
$p_2 := 6$	$Y := \{1,2,4\}$	$p(Y) = 16+2 = 18$	$p(Y)/2 = 9 \in [0, p_1[$	*donc* $r(Y) = 1$
$p_3 := 4$	$X' := X\cup\{z\}$	$p(X')/2 = p(X)/2 + p_z/2 = 10 \in [p_1, p_1+p_2[$		*donc* $r(X') = 2$
$p_4 := 2$	$Y' := Y\cup\{z\}$	$p(Y')/2 = p(Y)/2 + p_z/2 = 11 \in [p_1, p_1+p_2[$		*donc* $r(Y') = 2$



$\left.\begin{array}{l}\tilde{g}(X) = 10{*}0 + 6{*}6 = 36\\ \tilde{g}(X') = 10{*}6 + 6{*}0 + 4{*}4 = 76\end{array}\right\}$ *donc* $\tilde{g}(X') - \tilde{g}(X) = 40$

$\left.\begin{array}{l}\tilde{g}(Y) = 10{*}0 + 6{*}6 + 28 = 36 + 16\\ \tilde{g}(Y') = 10{*}6 + 6{*}0 + 4{*}4 + 26 = 76 + 12\end{array}\right\}$ *donc* $\tilde{g}(Y') - \tilde{g}(Y) = 40 + (12-16) = 40 - 4$

On a donc ici $\tilde{g}\big(X\cup\{z\}\big) - \tilde{g}(X) > \tilde{g}\big(Y\cup\{z\}\big) - \tilde{g}(Y)$, *alors que* $X\subset Y\subset J$ *et* $z\in Y^C$.

Si la fonction $\tilde{g}$ avait été super-modulaire, il aurait encore fallu montrer que les points extrêmes de $P^{\text{"e,t,}\delta\text{"}}$ (ou du moins ceux qu'on peut atteindre en minimisant le coût) codent bien des ordonnancements sans-chevauchement. On aurait alors pu résoudre notre problème par un algorithme de Branch-and-Cut, dont le problème de séparation aurait consisté à maximiser la fonction super-modulaire $\tilde{\Gamma} := S \mapsto \tilde{g}(S) - \alpha * e(S) - \beta * t(S)$, ce qui est un problème polynomial. La difficulté du problème (comme la complexité de l'algorithme) aurait alors résidé dans la partie branchement de l'algorithme, c'est-à-dire dans le choix de δ, ce qui est cohérent avec la dominance $\nearrow\searrow^r$: une fois qu'on a décidé de la répartition entre avance et retard des tâches, il suffit de deux tris pour obtenir un ordonnancement optimal.

11 Notations

11.1 Notations générales

Dans tout le document J désigne un ensemble fini non vide puisqu'il modélise notre ensemble de tâches. On notera $n = \#J$ et on assimilera J à $[1..n]$. Puisqu'on travaille le plus souvent dans $\mathbb{R}^J$, les vecteurs considérés sont implicitement de taille n.

$[1..n]$	désigne l'ensemble des entiers compris entre 1 et n
x_i	désigne la i-ème composante du vecteur x, pour $i \in [1..n]$
$\frac{x}{y}$	désigne le vecteur quotient composante par composante i.e. $\frac{x}{y}_i = \frac{x_i}{y_i}$
$\mathbb{1}$	désigne le vecteur $(1, 1, \ldots, 1)$
$\mathbb{1}_S$	désigne le vecteur indicateur du sous ensemble S de $[1..n]$ i.e. $X_i = 1 \Leftrightarrow i \in S$
$\mathbb{1}_{\{i\}}$	désigne donc le i-ème vecteur de la base canonique pour $i \in [1..n]$
$\langle x \vert y \rangle$	désigne le produit scalaire du vecteurs x par le vecteur y i.e $x.y = \sum_{i=1}^{n} x_i y_i$
$x(S)$	désigne la somme des composantes dans S d'un vecteur x i.e $x(S) = \langle x \vert \mathbb{1}_S \rangle = \sum_{i \in S} x_i$
$x * y(S)$	désigne, pour deux vecteurs x et y, et pour $S \subset J$, la somme $\sum_{i \in S} x_i y_i$
$[t]^+$	désigne la partie positive d'un réel t i.e $[t]^+ = \max(t, 0)$
$[t]^-$	désigne la partie négative d'un réel t i.e $[t]^- = \max(-t, 0)$
$\mathcal{P}(X)$	désigne les parties de l'ensemble X
$\mathcal{P}^*(X)$	désigne les parties non vide de l'ensemble X
$\mathfrak{S}_n$	désigne l'ensemble des permutations de $[1..n]$
$f_{/X}$	désigne la restriction de la fonction f à l'ensemble X
$X_{\setminus Y}$	désigne l'ensemble X privé de l'ensemble Y
X^C	désigne le complémentaire de l'ensemble X
$X^<$	désigne $\{(i, j) \in X \times X \mid i < j\}$ pour $(X, <)$ un ensemble ordonné
$\searrow$	sera utilisé comme abréviation de décroissant (au sens large)
$\searrow\!\!\searrow$	sera utilisé comme abréviation de strictement décroissant

11.2 Notations d'analyse convexe

$\mathrm{Conv}(A)$	désigne l'enveloppe convexe de l'ensemble de points $A \subset \mathbb{R}^n$
$\mathrm{CC}^\circ(E)$	désigne le cône convexe contenant 0 engendré par l'ensemble de vecteurs $E \subset \mathbb{R}^n$
$\mathrm{Extr}(C)$	désigne l'ensemble des points extrémaux d'un convexe $C \subset \mathbb{R}^n$
$0^+(C)$	désigne le cône de récession d'un convexe $C \subset \mathbb{R}^n$

11.3 Notations pour l'ordonnancement

Pour l'instance :

J	désigne l'**ensemble des tâches**
n	désigne le **nombre de tâches** i.e. $n = \#J$
p_j	désigne la **durée** de la tâche $j \in J$
α_j	désigne le **coût d'avance** pour la tâche $j \in J$
β_j	désigne le **coût de retard** pour la tâche $j \in J$
ω_j	désigne le **coût** pour la tâche $j \in J$ dans le cas symétrique i.e. $\omega_j = \alpha_j = \beta_j$
r_j	désigne le **ratio** prix/durée pour la tâche $j \in J$ dans le cas symétrique i.e. $r_j = \frac{\omega_j}{p_j}$
d_j	désigne la **due date** de la tâche $j \in J$
d	désigne la **due date** si celle-ci est commune

Pour un ordonnancement $\mathcal{S}$ (S comme "scheduling") :

$C_j(\mathcal{S})$ [17]	désigne le **date de fin** de la tâche $j \in J$
$e_j(\mathcal{S})$ [17]	désigne l'**avance** de la tâche $j \in J$, i.e. $e_j = [d - C_j]^+$
$t_j(\mathcal{S})$ [17]	désigne le **retard** de la tâche $j \in J$, i.e. $t_j = [C_j - d]^+$
$E(\mathcal{S})$	désigne l'ensemble des (indices des) tâches strictement en avance i.e. $E(\mathcal{S}) = \{j \in [1..n] \mid C_j < d\} = \{j \in [1..n] \mid e_j > 0\}$
$E'(\mathcal{S})$	désigne l'ensemble des (indices des) tâches en avance ou à l'heure i.e. $E'(\mathcal{S}) = \{j \in [1..n] \mid C_j \leqslant d\} = \{j \in [1..n] \mid e_j \geqslant 0\}$
$T(\mathcal{S})$	désigne l'ensemble des (indices des) tâches strictement en retard i.e. $T(\mathcal{S}) = \{j \in [1..n] \mid C_j > d\} = \{j \in [1..n] \mid t_j > 0\}$

Références

[1] W. Ben-Ameur, A. R. Mahjoub, and J. Neto, *Le problème de coupe maximum*, Hermès, 2006, pp. 17–60.

[2] D. Biskup and M. Feldmann, *Benchmarks for scheduling on a single machine against restrictiveand unrestrictive common due dates*, Computers and operations research, Vol 28 (2001), pp. 787–801.

[3] M. R. Garey, R. E. Tarjan, and G. T. Wilfong, *One-processor sceduling with symmectric earliness and tardiness penalties*, Mathematics of Operations Research, Vol 13 (1998), pp. 330–348.

[4] G. GOThA, *Modèles et algorithmes en ordonnancement*, Ellipses, 2004.

[5] M. Grötschel, L. Lovasz, and A. Scrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, Vol 1 (1981), pp. 169–197.

[6] N. G. Hall, W. Kubiak, and S. P.Sethi, *Earliness-tardiness sceduling problems, 2 : Deviation of completion times about a common due date*, Operations Research, Vol 39 (1991), pp. 847–856.

[7] N. G. Hall and M. E. Posner, *Earliness-tardiness sceduling problems, 1 : Weighted deviation of completion times about a common due date*, Operations Research, Vol 39 (1991), pp. 836–846.

[8] J. A. Hoogeveen and S. van de Velde, *Scheduling around a small common due date*, European Journal of Operational Research, Vol 55 (1991), pp. 237–242.

[9] H. G. Kahlbacher, *Termin- und Ablaufplanung : ein analytischer Zugang*, PhD thesis, Kaiserslautern University of Technology, Germany, 1992.

[10] J. J. Kanet, *Minimizing the average deviation of job completion times about a common due date*, Naval research logistics quarterly, Vol 28 (1981), pp. 643–651.

[11] S. T. McCormick, *Submodular Function Minimization*, Elsevier, 2006, ch. 7, p. 321–391.

[12] M. Queyranne, *Structure of a simple scheduling polyhedron*, Mathematical Programming, Vol 58 (1993), pp. 263–285.

[13] R. T. Rockafellar, *Convex Analysis*, Princetion University Press, 1970.

[14] A. Schrijver, *Combinatorial optimization*, Algoritms and Combinatorics, Springer, 2002.

[15] S.Iwata and J. Orlin, *A simple combinatorial algorithme for submodular functions minimization*, Proceedings of the twentieth annual ACM-SIAM Synopsium on discrete algorithms, (2009), pp. 1230–1237.

1. Le $\mathcal{S}$ sera souvent implicite

[16] W. E. Smith, *Various optimizers for single-stage production*, Naval research logistics quarterly, Vol 3 (1956), pp. 59–66.

[17] F. Sourd, *Ordonnancer juste-à-temps*, mémoire d'habilitation à diriger des recherches, Université Pierre et Marie Curie, Avril 2008.