1. RÉSULTATS POSITIFS (FPTAS)

1.1 Préliminaires

1.1.1 Notations

Notation à trois champs

La notation à trois champs est une notation de la forme :

P Q R		$c_{max} = \sum c_i$
Machines	Contraintes	Fonction objectif

Remarque 1.1.1.0.1.
Dans cette notation, le premier champs peut prendre trois valeurs :
- Q chaque machine i a une vitesse s_i. Ordonnancer la tâche j sur i nécessite $\frac{p_j}{s_i}$ unités de temps.
- R plus général, le temps est donné par x_{ij}
- P toutes les machines sont identiques

Exemple $(P|C|C_{MAX})$.
Étant donné, m machines identiques, n tâches indépendantes[1] avec p_j la durée de la tâche j. On cherche à ordonnancer les n tâches sur les m machines (sans préemptions) pour minimiser $\max_{1 \leq j \leq n} C_j$ où c_j est le "completion time" de la tâche j[2].

1.1.2 Définitions

NP-hard sens fort, sens faible

On considère Π_{dec} :
Entrée : $x_i, 1 \leq i \leq n$ avec $\forall i, 0 \leq x_i \leq C$
Question :

1.1.3 Différents problèmes

1.2 Analyse classiques de gloutons

1.2.1 Technique de SWAP et de restructuration d'un optimal

Exemple sur $1||\sum \omega_j c_j$. On a une machine, chaque tâche a :

$$\begin{cases} p_j & : \text{processing time} \\ \omega_j & : \text{poids} > 0 \end{cases}$$

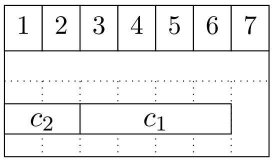
1. Pas de règles de priorités ou d'exécutions pré-requises.
2. makespan, noté C_{max}

On cherche à minimiser $\sum_{j=1}^{n} \omega_j c_j$.

Exemple.
On définit deux taches comme suit :

c_1	$p_1 = 4$	$\omega_1 = 2$
c_2	$p_2 = 2$	$\omega_2 = 2$

Considérons l'ordonnancement suivant :



En exécutant c_2 en premier, le coût est donné par :

$$\begin{aligned} \text{coût} &= 2 \times 2 + 2 \times 6 \\ &= 16 \end{aligned}$$

Le coefficient 6 est la date de fin de c_1.

Intuition

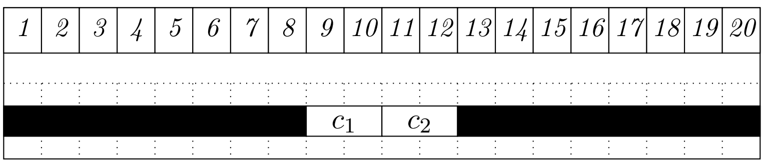
Si tous les $\omega_j = 1$:

$$\begin{aligned} \text{coût} &= c_1 + c_2 + c_3 \\ &= p_1 + (p_1 + p_2) + (p_1 + p_2 + p_3) \\ &= 3p_1 + 2p_2 + p_3 \end{aligned}$$

Quand $\omega_j = 1$, on s'aperçoit qu'il faut ordonner par p_j croissant [3]
Si $\forall j = 1, \ldots n, \quad p_j = 1$ Le coût est alors donné par coût $= \omega_1 + 2\omega_2 + 3\omega_3$, on cherchera donc à ordonnancer par ω_j décroissant.
Il existe un algorithme polynomial pour $1||\sum \omega_j c_j$, il s'agit de "SMITH RULE" : Ordonnancer par $\frac{p_j}{\omega_j}$ croissant.

Preuve : SMITH RULE est optimale

Lemme 1.2.1.1. *Soit S une solution de la forme :*

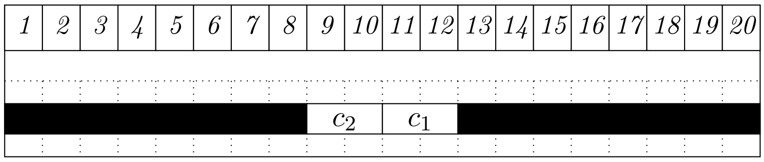


Si :

$$\frac{p_2}{\omega_2} \leq \frac{p_1}{\omega_1}$$

Alors il existe S' de la forme :

3. SPT : Shortest processing time (first)



telle que coût$(S') \leq$ *coût*(S)

Démonstration.

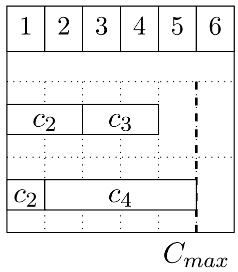
$$\begin{array}{rrcl} & \text{coût}(S) & = & X + \omega_1(t+p_1) + \omega_2(t+p_1+p_2) + Y \\ & \text{coût}(S') & = & X + \omega_2(t+p_2) + \omega_1(t+p_2+p_1) + Y \\ \Rightarrow & \Delta & = & \text{coût}(S) - \text{coût}(S') \\ & & = & \omega_2 p_1 - \omega_1 p_2 \geq 0 \end{array}$$

Car $\omega_2 p_1 \geq \omega_1 p_2$ □

Lemme 1.2.1.2 (Restructuration optimale)**.** *Soit OPT,* $\exists S_1, S_2, \ldots, S_r$ *:*

$$\begin{array}{rcl} \textit{coût}(OPT) & \geq & \textit{coût}(S_1) \\ & \geq & \textit{coût}(S_r) \end{array}$$

Exemple : $P2||C_{max}$



Théorème 1.2.1. $P2||C_{max}$ *est NP-hard*

Démonstration. On fait une réduction à partir de 2−partition :

Données : Un ensemble d'éléments $S = \{x_i,\ 1 \leq i \leq n\}$, une fonction ω à valeurs dans $\mathbb{N}$ affectant un poids à chacun des éléments de S :

$$\omega \left\{ \begin{array}{rcl} S & \longrightarrow & \mathbb{N} \\ x_i & \longmapsto & \omega(x_i) \end{array} \right.$$

On pose, sans perte de généralité : $\sum_S x_i = 2a$

Question : Est il possible de partitonner S en deux sous ensembles S_1 et S_2 tels que :

- $S_1 \cap S_2 = \emptyset$
- $S_1 \cup S_2 = S$
- $\sum_{S_1} = \sum_{S_2} = a$

Il est possible à partir d'une instance quelconque de 2−partition de construire une instance de $P_2||C_{max}$ de la manière suivante :

- n tâches
- chaque tâche j est telle que $p_j = x_j$

S'il existait un algorithme polynomial permettant de résoudre $P_2||C_{max}$, on pourrait alors résoudre 2−partition en temps polynomial en répondant OUI s'il existe un ordonnancement tel que $C_{max} \leq a$ et non sinon. □

Remarque 1.2.1.0.2. $P2||C_{max}$ est NP-hard sens faible

On va résoudre $\overline{P_2||C_{max}}$ en temps $\mathbf{poly}(n)$ en utilisant la programmation dynamique.

$PD(j, l_1, l_2)$ = étant donné que la machine i est chargée de 0 à l_i, $PD(j, l_1, l_2)$ ordonnance les tâches $j, \ldots, n$ de façon optimale.

Algorithm 1 Programmation dynamique

```
function PD(j, l1, l2)
    if j = n + 1 then
        return max(l1, l2) ;
    else
        return min(PD(j + 1, l1 + pj, l2), PD(j + 1, l1, l2 + pj)) ;
```

$PD(1, 0, 0)$ donne une solution optimale en temps $O(n^3c^2)$. Pour la programmation dynamique, la complexité est donnée par le nombre de valeurs possibles des paramètres multiplié par le temps d'exécution d'un traitement. Le nombre de valeurs possibles est donné par :

$$\begin{array}{cccc} 1 \leq j \leq n & 0 & \leq l_1 \leq & n \times \max(p_j) = nc \\ nx & (nc)^2 & & \end{array}$$

Dans le cas général (pour $P2||C_{max}$), cet algorithme est pseudo polynomial, c'est à dire $\mathbf{poly}(n, C)$. Et donc, si on se restreint à $\overline{P2||C_{max}}$, $C = \mathbf{poly}(n)$ et l'algorithme devient polynomial.

Remarque 1.2.1.0.3. Un problème NP-hard sens fort ne peut pas avoir d'algorithme pseudo polynomial [4]

Remarque 1.2.1.0.4. $\forall 0 \leq x \leq nC$, on lance $S_1 = KP(\text{taches}, x)$ [5] et $S_2 = \{1, \ldots, n\} \backslash S_1$ on obtient une solution σ_x. On retourne alors la meilleure solution de σ_x pour $1 \leq x \leq nC$

Théorème 1.2.2. *$P||C_{max}$ est NP-hard (sens fort)*

Démonstration. On fait une réduction depuis 3−partition :

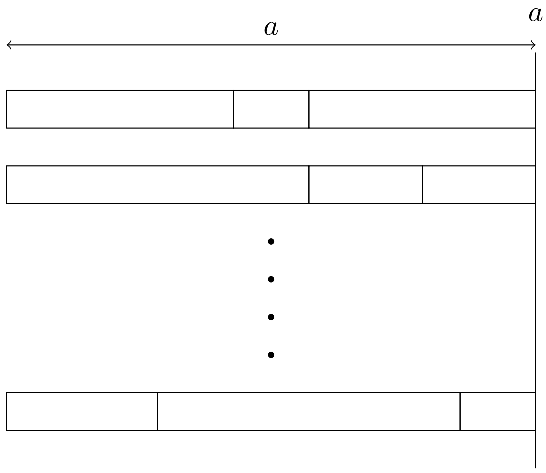
Données : Un ensemble d'éléments $S = \{x_i, 1 \leq i \leq n\}$, une fonction ω à valeurs dans $\mathbb{N}$ affectant un poids à chacun des éléments de S :

$$\omega \left\{ \begin{array}{ccc} S & \longrightarrow & \mathbb{N} \\ x_i & \longmapsto & \omega(x_i) \end{array} \right.$$

Question : Étant donné $x_j, 1 \leq j \leq n$, peut on partitionner les x_j en $\frac{n}{3}$ triplets $x_1^l, x_2^l, x_3^l, 1 \leq l \leq \frac{n}{3}$.

On a $\sum_{j=1}^{n} x_j = \frac{na}{3}$, donc $\sum_{t=1}^{3} x_t^l = a$, on pose alors $m = \frac{n}{3}$ machines, n tâches $p_j = x_j$, si 3−partition retourne oui, alors $OPT_{P||C_{max}} \leq a$

4. Sauf si $P = NP$
5. KP = Knapsac Problem, sac à dos



S'il retourne non, alors $OPT_{P||C_{max}} \geq a+1$ $\square$

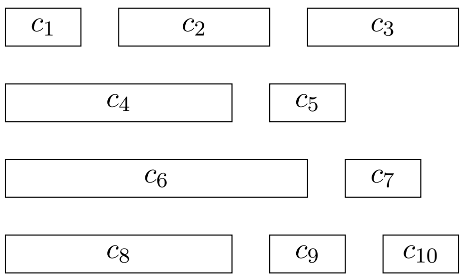
1.2.2 $2 - \frac{1}{m}$ *approximation pour* $P||C_{max}$

On va définir l'algorithme "List Scheduling" (LS) :

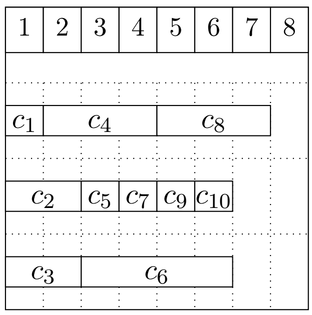
Algorithm 2 List Scheduling

function LS($t_1, \ldots, t_n$)
  Ordonner les tâches selon un critère de son choix ;
  for $1 \leq i \leq n$ **do**
    ordonnancer la tâche i pour qu'elle termine le plus tôt possible ;

Exemple. Le nombre de machines m est fixé à 3. La liste des tâches triées par ordre dichotomique est la suivante :



L'ordonnacement correspondant :



Propriété 1.2.2.1. List Scheduling est une $2 - \frac{1}{m}$ approximation pour $P||C_{max}$.

Démonstration. Soit x une tâche qui atteint le makespan, soit s_x la date de début de x, on a donc $LS = s_x + p_x$. Par définition de LS, $\forall t \in [0, s_x]$, toutes les machines travaillent.

Soit :

$$W = \sum_{j=1}^{n} p_j,\ W \geq ms_x + P_x \Rightarrow s_x \leq \tfrac{W - p_x}{m}$$

donc :

$$\begin{aligned} LS &\leq \frac{W - p_x}{m} + p_x \\ &= \frac{W}{m} + p_x\left(1 - \frac{1}{m}\right) \end{aligned}$$

□

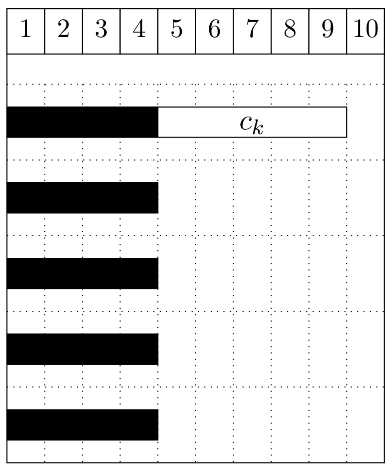
Lemme 1.2.2.1 (Bornes inférieures classiques)**.** *Voici quelques bornes classiques :*
- $\frac{W}{m} \leq OPT$
- $p_x \leq OPT$

On a donc :

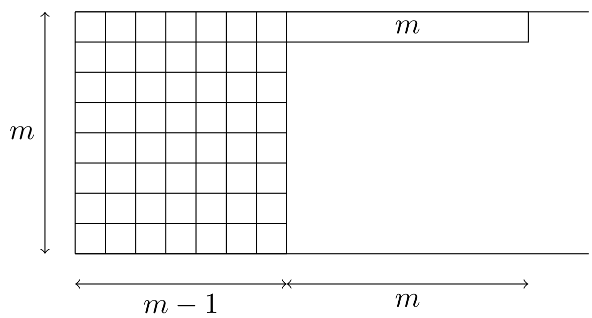
$$LS \leq OPT\left(2 - \frac{1}{m}\right)$$

Théorème 1.2.3. *La borne est tight (= atteinte)*

Démonstration. **Intuition :** Le tri peut être mauvais et une tâche très grande peut être mise tout à la fin de l'ordonnacement. On se retrouverait alors dans un cas de figure de ce genre :



On prend une tâche de taille m et $m(m-1)$ tâches de taille 1. Si LS trie par p_j croissant,



$OPT = m, \quad LS = 2m-1 \Rightarrow \rho = \frac{2m-1}{m} = 2 - \frac{1}{m}$ □

Théorème 1.2.4. *Soit* LPT[6] $= LS$ *où l'on trie par* p_j *décroissant. LPT est une* $\left(\frac{4}{3} - \frac{1}{3m}\right)$ *approximation.*

Intuition : "Taquiner la formule"

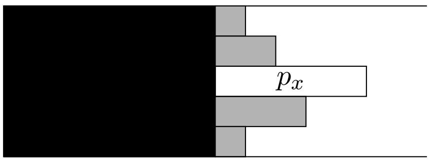
$$\begin{array}{rcl} LS & \leq & \dfrac{W}{m} + p_x \\ & \leq & OPT \end{array}$$

Le résultat est trivial si $p_x \leq \dfrac{OPT}{3}$.

Démonstration. Soit x la tâche qui atteint le makespan de LPT. $W \leq ms_x + p_x$ avec $s_x \leq \frac{W-p_x}{m}$

$$\begin{array}{rcl} LPT & \leq & s_x + p_x \\ & \leq & \dfrac{W}{m} + p_x\left(1 - \dfrac{1}{m}\right) \end{array}$$

□



- Cas 1 : si $p_x \leq \frac{OPT}{3}$

$$\begin{array}{rcl} LPT & \leq & OPT + \frac{OPT}{3}\left(1 - \frac{1}{m}\right) \\ & \leq & OPT\left(\frac{4}{3} - \frac{1}{3m}\right) \end{array}$$

- Cas 2 : si $p_x > \frac{OPT}{3}$
  Soit $I' = \{j / s_j \leq s_x\}$, on a $LPT(I') = LPT(I)$[7] On va montrer que :

  $$LPT(I) = LPT(I') = OPT(I') \leq OPT(I)$$

  Intuition : Toutes les tâches j avant x vérifient $p_j \geq p_x$ et donc $p_j > \frac{OPT}{3} \quad \forall j = 1, \ldots, x-1$. Ceci d'écrit :

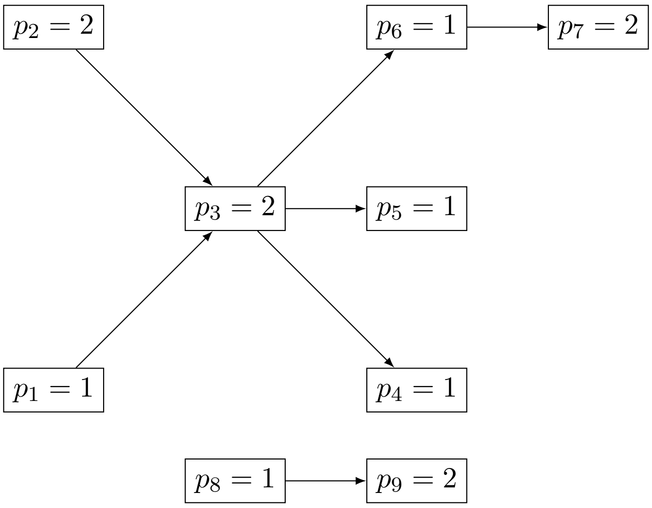
  $$\forall j \in I', \quad p_j > \frac{OPT(I')}{3} > \frac{OPT(I)}{3}$$

  On a alors $|I'| \leq 2m$.
  On va prouver que $LPT(I') = OPT(I')$, pour ce faire, quelle est la structure de $OPT(I')$

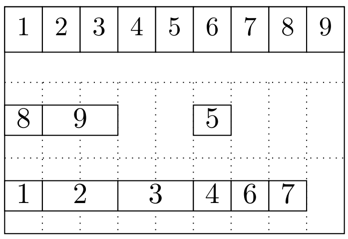
6. Longest Processing Time (first)
7. On supprime toutes les tâches positionnées après p_x, qui n'influent pas sur le temps d'exécution optimal.

Analyse de List Scheduling sur le problème $P|Prec|C_{max}$

Input : n tâches p_j, m machines identiques, Prec est le graphe de précédence[8]



Output : Un ordonnancement des tâches minimisant le temps total d'exécution.



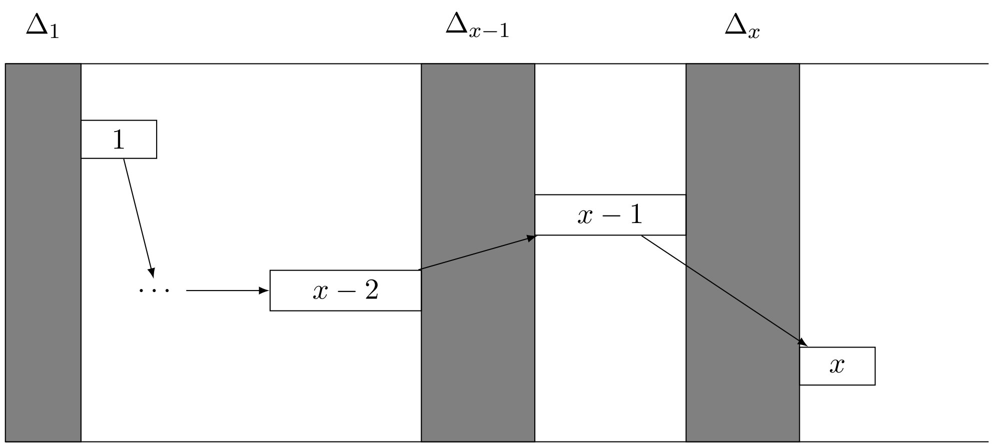
On définit le *List Scheduling* pour un graphe :

Algorithm 3 List Scheduling sur Graphe

while Pas Fini **do**
  Prendre une tâche prête ;
  L'ordonnancer au plus tôt ;

Théorème 1.2.5. *LS est une* $2 - \frac{1}{m}$ *approximation pour* $P|Prec|C_{max}$

Considérons l'ordonnancement réalisé par LS :



8. Il s'agit d'un DAG (=Directed Acyclic Graph), s'il existe une arête $j_1 j_2$ dans l'arbre, alors la tache j_1 doit être terminée pour que j_2 commence.

Soit x une tâche qui atteint le makespan, soit $x-1$ le dernier prédécesseur de x. On sait que pendant le temps Δ_x, toutes les machines sont occupées [9]. Soit $x-2$ le dernier prédécesseur de $x-1$, de la même manière, pendant Δ_{x-1}, toutes les machines travaillent. On procède ainsi jusqu'à la tâche 1.

La valeur de LS est donnée par :

$$LS = \sum_{i=1}^{x} \Delta_i + \sum_{i=1}^{x} p_i$$

On pose $\sum_{i=1}^{x} p_i = C$ que l'on définit comme la taille de la chaîne définie par récursion. Intéressons nous aux bornes, on voit que $C \leq OPT$ car ($OPT \geq$ chemin critique + longueur de la chaîne), de plus, on peut écrire :

$$W \geq m \sum_{i=1}^{x} \Delta_i + C$$

On en déduit :

$$\sum_{i=1}^{x} \Delta_i \leq \frac{W-C}{m}$$

Donc dans LS, on obtient :

$$\begin{aligned} LS &\leq \frac{W-C}{m} + C \\ &\leq \frac{W}{m} + C(1-\frac{1}{m}) \\ &\leq \frac{mOPT}{m} + OPT(1-\frac{1}{m}) \\ &\leq OPT(2-\frac{1}{m}) \end{aligned}$$

La borne est atteinte par le même exemple que précédemment.

Remarque 1.2.2.0.5. LS sur $Q||C_{max}$ [10] On réalise la même analyse et on arrive à :

$$LS \leq \frac{W}{\sum_{i=1}^{m} s_i} + \frac{p_x}{s_{i_0}} \leq \frac{W}{}$$

1.3 Dual approx

Définition 1.3.0.1. Une ρ [11] dual approx A est un algorithme qui, $\forall J, \omega$ [12], va soit produire une solution de coût $\leq \rho\omega$, soit dire "FAIL" et prouver que $\omega < OPT(I)$

Propriété 1.3.0.2. Soit $b_{sup}(I)$ et $b_{inf}(I)$ tels que :

$$b_{inf}(I) \leq OPT(I) \leq b_{sup}(I)$$

Soit $Delta(I) = b_{sup}(I) - b_{inf}(I)$ Alors, $\forall k$, en k répétitions de A, on a une solution :

$$sol_k \leq \rho(OPT(I) + \frac{\Delta(I)}{2^k})$$

9. Sinon on aurait pu lancer x plus tôt.
10. Chaque machine a une vitesse différente.
11. $\rho > 1$
12. Appelé le guess

L'idée est de réaliser une recherche dichotomique du plus petit ω_f qui ne sera pas rejeté par l'algorithme. Autrement dit on cherche ω_f tel que :

$$A \leq \rho\omega_f \quad \text{et} \quad \omega_f - \epsilon \text{ rejeté}$$

$$\begin{array}{rrcl} \Rightarrow & \omega_f - \epsilon & < & OPT \\ \Rightarrow & \omega_f & < & OPT + \epsilon \\ \Rightarrow & A & \leq & \rho(OPT + \epsilon) \end{array}$$

On va montrer que après les itérations, on a :
- une solution $s_k \leq \rho^{b^k_{sup}}$
- $b^k_{inf} \leq OPT(I)$
- $\Delta^k = b^k_{sup} - b^k_{inf}$

Corollaire 1.3.1. *Quand le problème est à valeurs entières, on obtient une "vraie" ρ approximation.*

Algorithm 4 3/2 dual approx pour $Q||C_{max}$

function $A(\omega, I, i)$
  if i = m **then**
    if $\frac{\sum_{j\in I} P_j}{s_m} > \omega$ **then**
      return FAIL ;
    else
      Mettre I sur m ;
  else
    $Fit_i \leftarrow \{j | \frac{p_j}{s_i} \leq \omega\}$;
    $\text{Big}_i \leftarrow Fit_i \cap \{j | \frac{p_j}{s_i} > \frac{\omega}{2}\}$;
    $\text{Sml}_i \leftarrow \{j | \frac{p_j}{s_i} \leq \frac{\omega}{2}\}$;
    $x \leftarrow \max(\text{Big}_i)$;
    ordonnancer x sur i
    $I \leftarrow I \backslash \{I_{sml_i} \cup \{x\}\}$;
    $res = A(\omega, I', i+1)$;
    if res =FAIL **then**
      return FAIL ;
    else
      Essayer d'ajouter I_{Sml} avec algorithme glouton
      if Algorithme glouton retourne une erreur **then**
        return Fail
function GREEDY(ω, i, I_{small})
  while $\exists$ machine $l(i \leq l \leq m)$, qui finit avant ω **do**
    Ajouter $x \in I_{small}$ sur l ;
  if Il reste des tâches **then**
    return FAIL

Lemme 1.3.0.2. *I faisable sur $i \dots m$ (en ω) $\Rightarrow$ I' faisable sur $i+1 \dots m$ (en ω)*

Lemme 1.3.0.3. *$\forall i, A(\omega, I, i)$ échoue alors I n'est pas faisable sur $i, \dots, m$*

On fait une preuve par récurrence, si $i = m$ alors c'est évident. On s'intéresse au cas ou $i - 1$ échoue mais i ne échoue pas.

Si c'est l'appel récursif qui plante, alors avec le lemme précédent le lemme est vérifié. Si c'est le remplissage glouton qui retourne une erreur, alors on a :

$$W(I) = \sum_{j \in J} p_j > \omega(\sum_{l=i}^{m} s_l$$

Donc $OPT(I) \geq \frac{W(I)}{\sum_{l=i}^{m} s_l} > \omega$.

1.4 Un peu de LP

2. RÉSULTATS NÉGATIFS

2.1 Utiliser la NP-difficulté

Lemme 2.1.0.4. *Si le problème est à valeurs entières, l'existence d'un FPTAS implique une résolution exacte en* $\boldsymbol{poly}(OPT(I))$

Avec FPTAS, $\forall\epsilon$ je peux avoir $A \leq (1+\epsilon)OPT(I)$ en $\mathbf{poly}(\frac{1}{\epsilon}n)$.

Pour avoir une solution exacte, il faut que $A \leq OPT(I) + \epsilon OPT(I)$ avec $\epsilon OPT(I) < 1$. On force donc $\epsilon < \frac{1}{OPT(I)}$.

Donc lancer le FPTAS avec un tel ϵ fournit une solution exacte en $\mathbf{poly}(OPT(I))$

Corollaire 2.1.1. *Soit* Π *un problème de minimisation. Si* Π *est NP-hard et polynomialement borné, il n'existe pas de FPTAS (sauf si* $P = NP$*)*

Si Π est NP-hard sens fort et OPT(I) est pseudo polynomialement borné, alors il n'existe pas de FPTAS. Si on avait FPTAS, on aurait une résolution exacte en temps $\mathbf{poly}(OPT(I))$ donc en temps $\mathbf{poly}(\mathbf{poly}(\text{max valeurs instance}), n)$. Donc appliqué à $\overline{\Pi} = \Pi$ avec max valeurs $\leq \mathbf{poly}(n)$.

2.2 Utiliser les gap réductions

Définition 2.2.0.2. gap_r
Soit Π un problème de minimisation

Définition 3.0.0.3. Un schéma d'approximation est un algorithme qui $\forall\epsilon, \forall I$, appelé $A(I,\epsilon)$ fournissant une $(1+\epsilon)-$approximation en temps :
- PTAS : polynomial à ϵ fixé ($O(n^{\frac{1}{\epsilon}})$)
- EPTAS : $f(\frac{1}{\epsilon})$ est polynomial
- FPTAS : polynomial en n et $\frac{1}{\epsilon}$

3.1 Les ingrédients pour faire un AS

Il existe plusieurs méthodes pour réaliser un AS :
- Simplification de l'instance :
  - partitionner
  - arrondi
  - ...
- Résolution :
  - Enumération
  - Programmation Dynamique
  - LP
  - ...
- Techniques de Guess

3.1.1 Simplification de l'instance

L'idée générale :
À la fin, on a besoin d'écrire :

$$\begin{aligned} A(I) &\leq \mu A(I') \\ &\leq \mu\rho OPT(I') \\ &\leq \mu\rho\lambda OPT(I) \end{aligned}$$

Exemple. On s'intéresse au problème du sac à dos, on cherche un **FPTAS** pour ce dernier.

Notations : n objets de prix p_j et de taille s_j, la capacité du sac à dos est notée C.

Rappel : Il existe deux sorte de programmation dynamique pour ce problème.

1. PD1
2. PD2

La polynomialité est assurée par la présence de l'argument V (respectivement P) qui est borné apr nS_{max} (respectivement nP_{max}).

Idée : On va choisir PD2 et définir $I' = I$ avec $p'_j =$On cherche alors à résoudre exactement I', puis montrer que

Résolution :

1. On résouts I' à l'aide de PD2, ainsi P évolue en valeur entière de 1 en 1, et P est borné par $\frac{np_{max}}{d}$

2. On définit $A(I) = \{j | j \in PD2(I')\}$, on a bien $A(I) \geq A(I')$
3. Soit $S = \{j | j \in OPT(I)\}$ il est évident que $OPT(I') \geq$ coût de S dans I'. Or le coût de $S \geq OPT(I) - n^*d$. Avec n^* le nombre d'objet dans $OPT(I)$.
4. Donc :

$$\begin{aligned} A(I) &\geq A(I') \\ &= OPT(I') \\ &\geq OPT(I) - n^*d \\ &\geq OPT(I) - nd \end{aligned}$$

On veut $nd \leq \epsilon OPT(I)$, donc on choisit $nd \leq \epsilon_{p_{max}} \leq \epsilon OPT(I)$:

$$d = \frac{\epsilon p_{max}}{n}$$

La complexité est alors donnée par $O(\frac{n^3}{\epsilon})$ [1]

3.1.2 Résolution

On s'intéresse à $P||C_{max}$, et on cherche à utiliser une programmation dynamique basique. La complexité de l'algorithme est donnée par : $O(m\times$ nombre de sous ensembles de $X\times$ nombre de configuration possibles)

On va chercher à rendre le nombre de subset et le nombre de configuration constants.

1. Définition de I' : on commence par définit I_{Big} en cherchant les petites tâches que l'on peut rajouter facilement à la fin par un algorithme glouton, c'est à dire les tâches que l'on peut rajouter à la fin sans modifier le ratio, et en cherchant à rendre constant le nombre de grosses tâches par machine.

Lemme 3.1.2.1. *Pour ϵ fixé. soit :*

$$I_{Big} = \left\{ J | p_j > \frac{\epsilon LS}{2} \right\}$$

et soit :

$$I_{Sml} = I \backslash I_{Big}$$

Si on a :

$$A(I_{Big}) \leq (1+\epsilon) OPT(I_{Big})$$

on peut déduire :

$$A(I) \leq (1+\epsilon) OPT(I)$$

De plus pour résoudre I_{Big}, il faut au maximum : $\lambda = \left\lceil \frac{2}{\epsilon} \right\rceil$ taches par machine.

Démonstration. Soit S la solution retournée par $A(I_{\text{Big}})$, on rajoute alors les petites tâches à l'aide de LS. On peut alors envisager deux cas :

(a) le makespan augmente, alors :

$$A(I) \leq \frac{W}{m} + \epsilon \frac{LS}{2} \leq (1+\epsilon) OPT(I)$$

1. On peut descendre à une complexité $O(n + f(\frac{1}{\epsilon}))$.

(b) le makespan n'augmente pas :

$$\begin{aligned} A(I) &= A(I') \\ &\leq (1+\epsilon)OPT(I') \\ &\leq (1+\epsilon)OPT(I) \end{aligned}$$

Dans I_{Big}, si dans un ordonnancement,, il y a x tâches sur une machine alors :

$$C_{max} > x\epsilon\frac{LS}{2}$$

et avec $C_{max} > LS$, tout branchement est inutile., donc il faut :

$$x\epsilon\frac{LS}{2} \leq LS \Rightarrow x \leq \frac{2}{\epsilon}$$

Une autre formulation est la suivante : Dans toute solution optimale de I_{Big}, il y a au plus $\left\lceil \frac{2}{\epsilon} \right\rceil$ tâches par machine. □

2. Étape 2 : $I_{\text{Big}} \to I'$ en utilisant un arrondi géométrique On définit alors :

$$p'_j = \left\{ \max x | \frac{\epsilon LS}{2}(1+\epsilon)^x \leq p_j \right\}$$

On cherche à borner x_max qui est le nombre de tranches.
Il faut :

$$\begin{aligned} \frac{\epsilon LS}{2}(1+\epsilon) &= p_{max} \quad \leq LS \\ \Rightarrow \quad (1+\epsilon)^{x_max} &\leq \frac{2}{\epsilon} \\ \Rightarrow \quad x_{max} &\leq \left(1+\frac{1}{\epsilon}\right) ln\left(\frac{2}{\epsilon}\right) \end{aligned}$$

3. Résolution exacte de I' avec $PD(i,X)$
4. Bilan : on peut écrire ce qui suit :

$$\begin{aligned} A(I') &= OPT(I') \\ &\leq OPT(I_{\text{Big}}) \end{aligned}$$

De plus :

$$\begin{aligned} A(I_{\text{Big}}) &\leq (1+\epsilon)A(I') \\ \Rightarrow \quad A(I_{\text{Big}}) &\leq (1+\epsilon)OPT(I_{\text{Big}}) \end{aligned}$$

On a donc un PTAS.

Exemple (Introduction :). le ski rental problem.

Entrée : x le nombre de jour restant à vivre **But :** Sachant que louer des skis coûte 30€ par jour et une paire de ski coûte 300€ à acheter, on cherhce à minimiser les dépenses pour skier tous les jours.

En mode offline, le problème n'est pas très intéressant et retourne $\min(30x, 300)$. En mode online, on cache une partie de l'entrée : x, on cherche à déterminer $j_0 =$ nombre de jours de location avant achat afin de minimise :

$$\rho(x) = \max \frac{\text{Coût Algo}(j_0, x)}{\text{coût OPT offline}(x)}$$

Si on choisit un j_0, le plus grand $\rho(x)$ est atteint pour $x = j_0 + \epsilon$:

$$\rho(x) \leq \frac{30j_0 + 300}{\min(30j_0, 300)}$$

On veut j_0 qui minimise :

$$\frac{a + b}{min(a, b)} \geq 2$$

ce qui est atteint pour $a = b$ et donc $a = 10$.