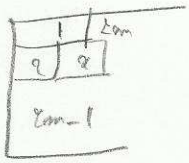
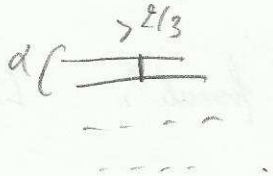
Le makespan de la machine i (son t_p de fin) est $\leq$ à l'ancien makespan de la machine 1.

$\Rightarrow$ on recommence récursivement.





et LPT fait ce pattern. □

×

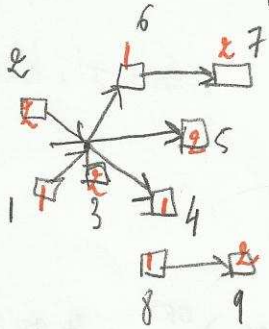
Analyse de LS sur $P|prec|C_{max}$

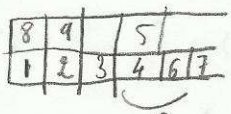
W = aire de toutes les tâches.

input: n tâches p_j, m machines identiques

prec = graphe de précédence = DAG = Directed Acyclic Graph (pas forcément connexe)

Si $\exists$ un arc entre j_1 et j_2, alors la tâche j_1 doit être terminée pour que j_2 commence.



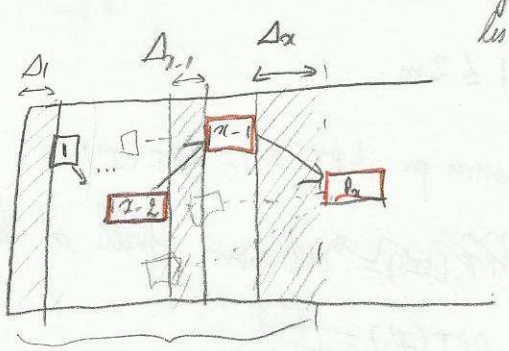


dommage, on aurait pu mettre le 6 avt.

LS + graphe: tant que PAS FINI prendre une tâche prête et l'ordonnancer au plus tôt.

idée: ici on cherche les zones pleines!

⚠ si plusieurs prédécesseurs, tous doivent être terminés.



Il n'est pas plein forcément

Théorème: LS est une $2-\frac{1}{m}$ approx pour $P|prec|C_{max}$

PROOF: Considérons l'ordo réalisé par LS.

Soit p_x une tâche qui atteint le makespan.

Soit $x-1$ le dernier prédécesseur de x.

$\Rightarrow$ pendant le temps Δ_x, toutes les machines travaillent.

Soit $x-2$ le dernier prédécesseur de $x-1$ ... jusqu'à la tâche 1!

$$W \geq m \sum_{i=1}^{x} \Delta + C. \quad (1)$$

$$LS = \sum_{i=1}^{x} \Delta_i + \underbrace{\sum_{i=1}^{x} p_i}_{C = \text{longueur chaîne.}}$$

(⚠ réflexe borne sup, borne inf)
ne demandez pourquoi pas bornal

Rq: Où aller ? Dans LS, on voit que $C \leq OPT$

(car OPT > chemin critique = + longue chaîne). } borne inf. $\Rightarrow$ donc ce sont les Δ_i qui nous ennuient dans LS.

Donc, de (1), on tire $\sum_{i=1}^{\alpha} \Delta_i \leq \frac{W-c}{m}$

Donc, dans LS, on obtient $LS \leq \frac{W-c}{m} + c = \underbrace{\frac{W}{m}}_{\leq opt} + \underbrace{c\left(1-\frac{1}{m}\right)}_{\leq opt}$ □.

Analyse tight $\Rightarrow$ même exemple qu'avant.

Rq : LS sur $Q||C_{max}$ Q → chaque machine i a une vitesse Δ_i.
(on met la tâche sur la machine qui la fera terminer au + tôt)

même analyse $LS \leq \frac{W}{\sum_{i=1}^{m} \Delta_i} + \frac{p_\alpha}{\Delta_{i_0}} \leq \underbrace{\frac{W}{\sum \Delta_i}}_{\leq opt} + \frac{p_{max}}{\Delta_{i_0}} \leq \frac{W}{\sum_{i=1}^{m} \Delta_i} + \frac{p_{max}}{\Delta_1}$

Δ_1 ← vitesse de la machine la + lente

avec $\Delta_1 \leq \dots \leq \Delta_m$.

$OPT \geq \frac{p_{max}}{\Delta_m}$ (Δ_m ← vitesse de la machine la + rapide)

$\leq \underbrace{\frac{W}{\sum \Delta_i}}_{\leq OPT} + \frac{\Delta_m}{\Delta_1} \underbrace{\frac{p_{max}}{\Delta_m}}_{\leq OPT}$

$\Rightarrow$ ratio $\leq \left(1 + \frac{\Delta_m}{\Delta_1}\right)$

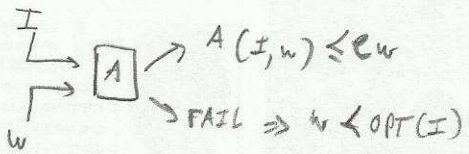
⚠ chercher les bornes inf ! Tips

Partie 2 : Technique de la dual approximation.

pb de minimisation : une e ($e>1$) dual appro A est un algorithme qui, $\forall I$, $\forall w$ ("le guess") :

→ soit produire une solution de coût $\leq e\,w$

→ soit donne "FAIL" et prouve que $w < OPT(I)$.



Propriété : (e dual-appro donne e-appro) Soit $b_{sup}(I)$ et $b_{inf}(I)$ / $b_{inf}(I) \leq OPT(I) \leq b_{sup}(I)$.

Soit $\Delta(I) = b_{sup}(I) - b_{inf}(I)$.

Alors, $\forall k$, en k itérations de A, on a une solution $sol_2 \leq e\left(OPT(I) + \frac{\Delta(I)}{2^k}\right)$

PROOF : (idée = recherche dichotomique du plus petit w_f qui ne sera pas rejeté par l'algo.

$\Rightarrow A \leq e\,w_f$ et $w_f - \varepsilon$ rejeté $\Rightarrow w_f - \varepsilon < OPT \Rightarrow w_f \leq OPT + \varepsilon \Rightarrow A \leq e(OPT + \varepsilon)$)

On va maq après les itérations, on a : → une solution $s_k \leq e\, b_{sup}^k$

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

→ $\Delta^k = b_{sup}^k - b_{inf}^k = \frac{\Delta(I)}{2^k}$

$k \to k+1$ Soit $w_{k+1} = \frac{b_{inf}^k + b_{sup}^k}{2}$ (si $k=0$, trivial)

↳ si $A(I, w_{k+1})$ acceptant, $b_{sup}^{k+1} = w_{k+1}$ $b_{inf}^{k+1} = b_{inf}^k$ (la borne inf ne bouge pas)

et donc $\Delta^{k+1} = \frac{\Delta^k}{2}$ // principe de la dichotomie

↳ si $A(I, w_{k+1})$ rejeté $b_{inf}^{k+1} = w_{k+1}$ $b_{sup}^{k+1} = b_{sup}^k$ $\Delta^{k+1} = \frac{\Delta^k}{2}$.

Donc, après k itérations, $sol_k \leq e\, b_{sup}^k = e\,(\underbrace{b_{inf}^k}_{\leq OPT} + \underbrace{b_{sup}^k - b_{inf}^k}_{\leq \frac{\Delta(I)}{2^k}})$ □

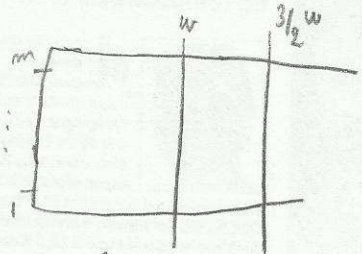
qd k est grd, on se débarrasse de k.

Corollaire : Quand le problème est à valeurs entières, on obtient une "vraie" e approx ($A \leq e\,OPT$), en $\log(e\,\Delta(I))$ itérations (car $A \leq e\,OPT + e\,\frac{\Delta(I)}{2^k}$, $e\,\frac{\Delta(I)}{2^k} < 1$).

I = instance à scheduler sur les machines

Application : $\frac{3}{2}$ dual approx pour $Q\|C_{max}$. $s_1 \leq \dots \leq s_m$. → algo identique dans ce sens (machines rapides peuvent rattraper bêtises de machines lentes)

idée



idée : les small (sml) et faciles à ajouter à la fin.
=> greedy (= glouton)

Soit $I'_{Big} = \{ j \mid \frac{p_j}{s_1} > \frac{w}{2} \} \cap Fit'$

$Fit' = \{ j \mid \frac{p_j}{s_1} \leq w \}$.

$I'_{sml} = \{ j \mid \frac{p_j}{s_1} \leq \frac{w}{2} \}$.

Algo : $A(w, I, i)$ // usage de scheduler ce sous $i \dots m$

si $i = m$, si $\frac{\sum_{j \in I} p_j}{s_m} > w$ => fail, sinon mettre I sur m.

sinon soit $Fit_i = \{ j \mid \frac{p_j}{s_i} \leq w \}$, $Big_i = Fit_i \cap \{ j \mid \frac{p_j}{s_i} > \frac{w}{2} \}$ $sml_i = \{ j \mid \frac{p_j}{s_i} \leq \frac{w}{2} \}$

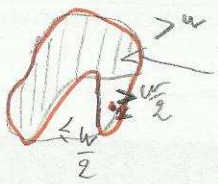
Soit x = + grosse tâche de Big_i

Check x sur i. $I' = I \setminus \{ I_{sml_i} \cup \{x\} \}$; $res = A(w, I', i+1)$, si res = fail → fail sinon

essayer d'ajouter I_{sml_i} avec algo greedy
↳ si pb = fail 2
(cf lemme 24)

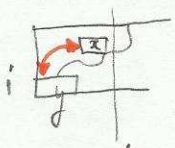
lemme 1 : I faisable sur $i \dots m$ (en w) $\Rightarrow$ I' faisable sur $i+1 \dots m$ (en w).

PROOF :

$I =$ 

fait pas Soit $\tilde{I} = I' \cup \{x\} = I \setminus I_{small}$.

I faisable sur $i \dots m \Rightarrow \tilde{I}$ faisable sur $i \dots m$.

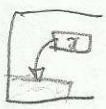
$\exists$ solution S^* de $\tilde{I}$: 

Si S^* a une tâche $y \in B_{ij}$ sur machine i. (on swappe pr revenir à la même instance que nous)

↳ si $y = x$, gagné, S^* restreinte à $i+1 \dots m$ est une solution de I'.

↳ sinon, swappe x et y.

La solution (après SWAP) restreinte à $i+1 \dots m$ ~~restreinte~~ nous prouve que I' est faisable sur $i+1 \dots m$.

Si S^* n'a pas de $y \in B_{ij}$ schédulé sur $i \Rightarrow$ 

□

lemme 2 : $\forall i$, $A(w, I, i)$ fail $\Rightarrow$ I pas faisable sur $i \dots m$.

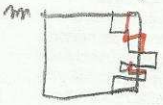
PROOF : → pour $i = m$, ok

→ i sachant $i+1$. Si $A(w, I, i)$ fail. Si c'est l'appel récursif qui fail, alors par lemme précédent (+ ou, c'est ok.

Si c'est le remplissage greedy qui FAIL, alors $w(I) = \sum_{j \in I} p_j > w \left(\sum_{l=1}^{m} s_l \right)$

donc $OPT(I) \geq \dfrac{w(I)}{\sum_{l=1}^{m} s_l}$ □

lemme 4 : greedy (w, i, I_{small_i}) : tant que $\exists$ machine l, $i \leq l \leq m$ qui finit avant w.

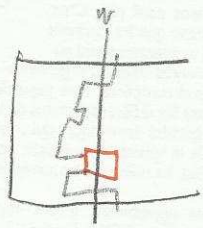


→ si reste tâches, FAIL ajoute x ($\in I_{small_i}$) sur l.

lemme 3 : Si pas fail, $A(w, I, i) \leq \frac{3w}{2}$.

PROOF : le seul endroit où l'on peut dépasser w est dans greedy.

On $j \in sol_i \Rightarrow j \in sol_l$ $l \geq i$



⚠ À chercher : 1| → 3/2 dual approx pour PL ...
2| → Cq no FPTAS pour ΠKP SubsetSum avec gap red.

ΠKP : C sac à dos de taille $a_1, \dots, a_j$
n objets { s_j size, p_j : poids
& part
placer les objets ds les c sacs en maximisant la $\sum_{j \text{ placés}} p_j$.

Résultats Négatifs

Utiliser la NP-hardness (⇒ pour "no FPTAS").

Lemme 1 : Si problème à valeurs entières, FPTAS ⇒ résolution (exacte) en poly(OPT(I)).

PROOF : Avec FPTAS, ∀ε, je peux avoir $A \leq (1+\varepsilon)OPT(I)$ en $poly(\frac{1}{\varepsilon}, n)$

⇒ pour avoir une solution exacte, il faut que $A \leq OPT(I) + \underbrace{\varepsilon OPT(I)}_{<1}$

$\varepsilon < \frac{1}{OPT(I)}$ donc lancer la FPTAS avec un tel ε formant une solution exacte en poly(OPT(I)).

Corollaire : Soit π pb de minimisation.

a) Si π est NP hard et OPT(I) poly borné ⇒ NO FPTAS (unless P=NP)
(par ex : $\max p_j \leq poly(n)$).

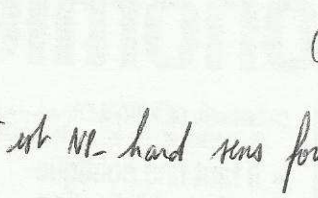
Si π est NP-hard sens fort et OPT(I) est pseudo poly borné ⇒ no FPTAS (unless P=NP).

b) Si on avait FPTAS, on aurait résolution exacte en poly(OPT(I)) de en poly(poly(max valeurs entrées), n)
donc appliqué à $\tilde{\pi} = \pi$ avec max valeurs ≤ poly(n), on aurait algo poly pour $\tilde{\pi}$, qui est NP-hard □.

Utiliser gap réductions.

Def (gap_r) : Soit π problème (minimisation). Soit $r : \mathcal{I} \to \mathbb{R}$, $I \mapsto r(I)$.
gap_r est le problème suivant : • input : I, k
output : yes si $OPT(I) \geq k$, no si $OPT(I) \leq \frac{k}{r(I)}$

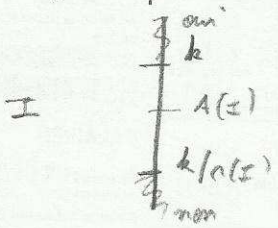
ce qu'on aurait sinon :



Remarque : On étend la définition de NP-hard pour les pb de gap.

Théorème : Si gap_ρ NP-hard, alors pas de λ'-approx avec $\lambda' < \rho$ (sauf si P=NP).

PROOF : Supposons que l'on ait A (polynômial), une λ'-approx, $\lambda' < \rho \Rightarrow$ comment résoudre gap_ρ ?



On résout $A(I)$. Si $A(I) < k \Rightarrow OPT(I) \leq A(I) < k \Rightarrow$ NO

Sinon $\lambda'(I)\, OPT(I) \geqslant A(I) \geqslant k \Rightarrow OPT(I) \geqslant \frac{k}{\lambda'(I)} > \frac{k}{\lambda(I)}$ (si $\lambda' < \lambda$) $\Rightarrow$ il donne "yes" □.

Exos : gap introduct°, reduce au bin packing
gap introduct°, reduct°

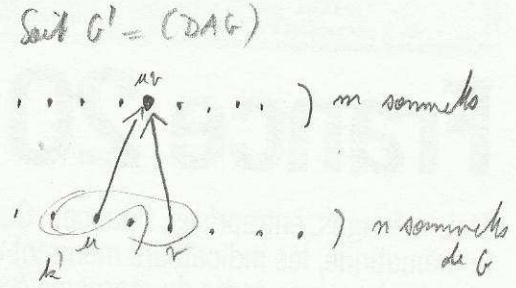
Def :
- gap introducing reduction : réduct° d'un pb de décision TC vers un pb gap.
- gap preserving reduction : réduct° depuis un pb gap vers un pb gap.

Théorème : gap 4/3 est NP-hard pour $P|prec|C_{max}$.

PROOF : réduction depuis clique

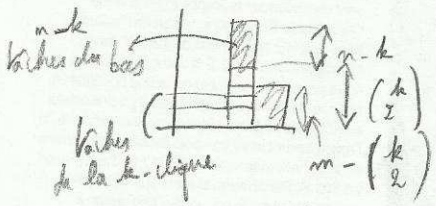
π_{clic} : given $G=(V,E)$, clique $\geqslant k \Rightarrow$ oui
clique $\leq k-1 \Rightarrow$ non

Soit $G' = (DAG)$

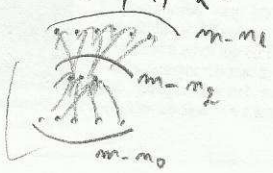


- $n' = n+m$
- $m' = 2m$
- m_a (# machines)
- pour chaque $uv \in E$

Si k cliques, les k tâches ordonnancées à $t=0$ libèrent $\binom{k}{2}$ tâches pour le temps 1.



Définition de l'instance : Soit $n_0 = k$, $n_1 = \binom{k}{2} + (n-k)$
$n_2 = m - \binom{k}{2}$
Soit $m_a = \max(n_0, n_1, n_2) + 1$
Soit $p_{ij} =$



$\forall j,\ p_j = 1$

Montrons que clique $\geqslant k \Rightarrow OPT \leq 3$.
↳ dessin

clique $\leq k-1 \Rightarrow OPT > 4$ (car opt $\leq 3 \Rightarrow$ clique $\geqslant k$) DANS TOUTE sol ≤ 3, on est obligés de faire ?
(no idle $\Rightarrow$) On est obligés de choisir k tâches du bas :

au temps 1, or est obligé de mettre les $n-k$ vaches restantes du bas (mais car pas de sommets isolés ds G).

(solut° de coût 3)

⇒ no idle

⇒ il faut trouver $\binom{k}{2}$ vaches dispos ⇒ k cliques ou $\square$

$\times$