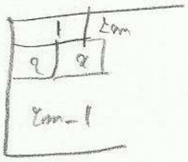
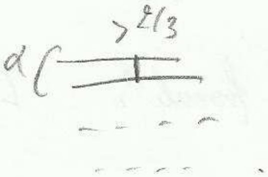
Le makespan de la machine i (son tps de fin) est ≤ à l'ancien makespan de la machine 1.

⇒ on recommence récursivement.





et LPT fait ce pattern. □

×

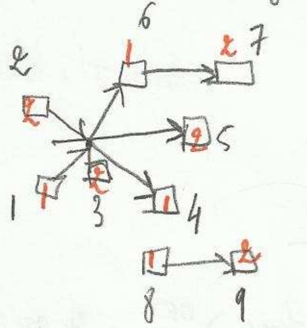
Analyse de LS sur $P|prec|C_{max}$

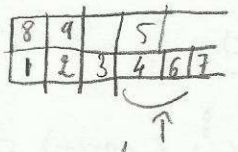
W = aire de ttes les tâches.

input: n tâches p_j, m machines identiques

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

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



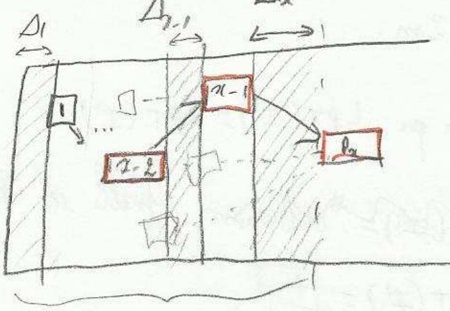


dommage, on aurait pu mettre le 6 avt.

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

idée: tjs chercher les zones pleines!

⚠ Si plusieurs prédécesseurs, tous doivent être terminés.



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.

⇒ pendant le temps Δx, ttes les machines travaillent.

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

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

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

(⚠ réflexe borne sup, borne inf) se demander pourquoi gros trou !

Rq: Où aller ? Dans LS, on voit que $C \leq OPT$ (car OPT ≥ chemin critique = + longue chaîne)) borne inf ⇒ donc ce sont les Δ_i qui nous ennuient ds LS.

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

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

Analyse tight ⇒ m̂ contre-exemple qu'avant.

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

m̂ analyse $LS \leq \frac{W}{\sum_{i=1}^{m} s_i} + \frac{p_l}{s_{i_0}} \leq \underbrace{\frac{W}{\sum s_i}}_{\leq opt} + \frac{p_{max}}{s_{i_0}} \leq \frac{W}{\sum_{i=1}^{m} s_i} + \frac{p_{max}}{s_1}$ ← vitesse de la machine la + lente

$OPT \geq \frac{p_{max}}{s_m}$ ← vitesse de la machine la + rapide

avec $s_1 \leq \dots \leq s_m$.

$\leq \underbrace{\frac{W}{\sum s_i}}_{\leq OPT} + \frac{s_m}{s_1}\underbrace{\frac{p_{max}}{s_m}}_{\leq OPT}$

$\Rightarrow$ ratio $\leq \left(1+\frac{s_m}{s_1}\right)$

⚠ Chercher les bornes inf ! TD

Partie 2 : Technique de la dual approximation.

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

→ soit produire une solution de coût $\leq \rho w$

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

$$\begin{array}{l} I \rightarrow \boxed{A} \nearrow A(I,w) \leq \rho w \\ w \rightarrow \boxed{A} \searrow FAIL \Rightarrow w < OPT(I) \end{array}$$

Propriété : (ρ dual-approx donne ρ-approx) 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 répétitions de A, on a une solution $sol_k \leq \rho\left(OPT(I) + \frac{\Delta(I)}{2^k}\right)$.

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

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

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

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

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

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

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

et du coup $\Delta^{k+1} = \dfrac{\Delta^k}{2}$ // principe de la dichotomie

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

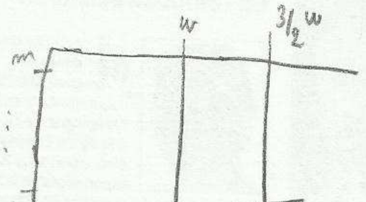
Donc, après k itérations, $sol_k \leq e\, b_{sup}^k = e\,(\underbrace{b_{inf}^k}_{\leq OPT} + \underbrace{b_{sup}{}^k - b_{inf}{}^k}_{\leq \frac{\Delta(I)}{2^k}})$ $\square$

qd k est gd, on se débarrasse du k.

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

I = instance à scheduler sur les machines

Application : $\dfrac{3}{2}$ dual approx pour $Q||C_{max}$. $\quad s_1 \leq \dots \leq s_m$. $\rightarrow$ algo récursif dès ce sens (machines rapides peuvent rattraper bêtises de machines lentes)

idée 

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

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

$I'_{Sml} = \{ j \mid \dfrac{p_j}{s_1} \leq \dfrac{w}{2} \}$

idée : les small (sml) sont faciles à ajouter à la fin.
$\Rightarrow$ greedy (= glouton)

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

Algo : $A(w, I, i)$ // essaye de scheduler w sur $i \dots m$

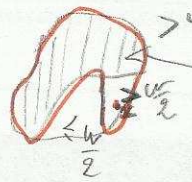
Si $i = m$, si $\sum_{j \in I} \dfrac{p_j}{s_m} > w \Rightarrow$ fail, sinon mettre I sur m.

Sinon soit $Fit_i = \{ j \mid \dfrac{p_j}{s_i} \leq w \}$, $Big_i = Fit_i \cap \{ j \mid \dfrac{p_j}{s_i} > \dfrac{w}{2} \}$ $\quad Sml_i = \{ j \mid \dfrac{p_j}{s_i} \leq \dfrac{w}{2} \}$

Soit x = + grosse tâche de Big_i

Sched x sur i. $\quad I' = I \setminus \{ Sml_i \cup \{x\} \}$; $res = A(w, I', i+1)$, si res = fail $\rightarrow$ fail, sinon lemme 1 : I faisable sur $i \dots m$ (en w) $\Rightarrow$ I' faisable sur $i+1 \dots m$ (en w).

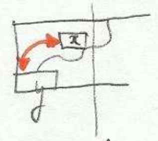
PROOF :

$I =$ 

fit 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 $y \in Big_i$ sur machine i.

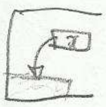
(on swappe pr revenir à la vr instance que nous)

↳ 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 $y \in Big_i$ schédulé sur i $\Rightarrow$



$\square$

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

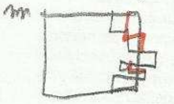
PROOF : → pour $i = m$, ok

→ i sachant $i-1$. Si $A(w, I, i)$ fail. Si c'est l'appel récursif qui fail, alors avec lemme précédent + rec, 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 $OPT(I) \geq \dfrac{W(I)}{\sum_{l=1}^{m} s_l}$ $\square$

lemme 2.4 : greedy (w, i, I_{small_i}) : tant que $\exists$ machine l, $i \leq l \leq m$ qui finit avant w.



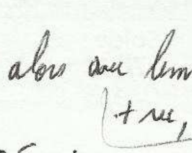
$\square$ = small

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

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

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

Or $j \in small_i \Rightarrow j \in small_l$ $l \geq i$



⚠ À traiter : $11 \to 3/2$ dual approx pour $P||C_{max}$

$?$ ↘ cq no FPTAS pour ΠKP SubsetSum avec gap red.

& part

ΠKP :

C sac à dos de taille $a_1, \dots, a_j$

n objets $\begin{cases} S_j \text{ size} \\ P_j : \text{poids} \end{cases}$

packer les objets dans les C sacs en maximisant le $\sum_{j \text{ packés}} A_j$.

Résultats Négatifs

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

lemme 1 : Si problème à valeurs entières, FPTAS ⇒ résolut° en poly (OPT(I)) (exacte).

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

⇒ 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 résolut° exacte en poly (OPT(I)) de en poly (poly(max valeurs entières), n)

donc appliqué à $\tilde{\pi} = \pi$ avec max valeurs $\leq poly(n)$, on aurait algo poly pour $\tilde{\pi}$, qui est NP-hard ☐.

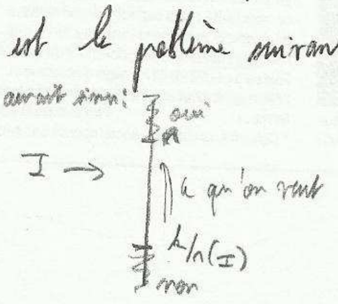
Utiliser gap réductions.

Def (gap$_r$) : Soit π problème (minimisat°). 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 $OPT(I) \leq \frac{k}{r(I)}$

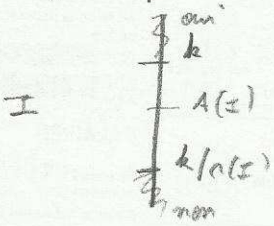
ce qu'on aurait sinon :



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

Théorème : Si gap_ρ NP-hard, alors pas de ρ' approx avec $\rho' < \rho$ (unless $P = NP$).

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



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

Sinon $\rho'(I)\, OPT(I) \geq A(I) \geq k \Rightarrow OPT(I) \geq \dfrac{k}{\rho'(I)} \overset{\text{si } \rho' < \rho}{>} \dfrac{k}{\rho(I)} \Rightarrow$ et donc "yes" $\square$

Exos : gap introduct°, reduce ac binpacking
gap introduct°, reduct°

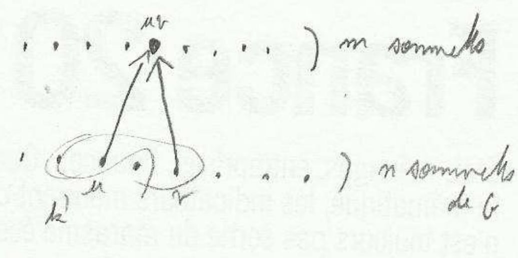
Def :
- gap introducing reduction : réduct° d'un pb de décision Π vers un pb gap.
- gap preserving reduction : réduct° depuis un pb gap vers un pb gap.

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

PROOF : réduction depuis clique

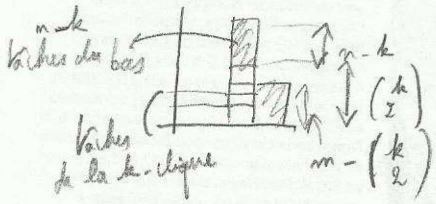
Π_{dec} = given $G = (V, E)$, clique $\geq k \Rightarrow$ oui
clique $\leq k-1 \Rightarrow$ non

Soit G' = (DAG)



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

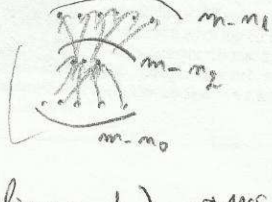
Si k cliques, ces k tâches ordonnancées à $t = 0$ libèrent $\binom{k}{2}$ tâches pour le temps 1.



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 pse =



$\forall j,\ p_j = 1$

Montrons que clique $\geq k \Rightarrow OPT \leq 3$.

⇑ dessin

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

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

(solut° de coût 3)

⇒ no idle

⇒ il faut trouver $\binom{k}{2}$ tâches dispos ⇒ k clique ou. □

×