Speed scaling power down scheduling for agreeable deadlines

C.Dürr (CNRS, LIP6)

travail en commun avec

Evripidis Bampis (LIP6)

Fadi Kacem (IBISC, Evry)

Ioannis Milis (Athens University of Economics and Business)

séminaire ENS-Lyon, 13déc2011

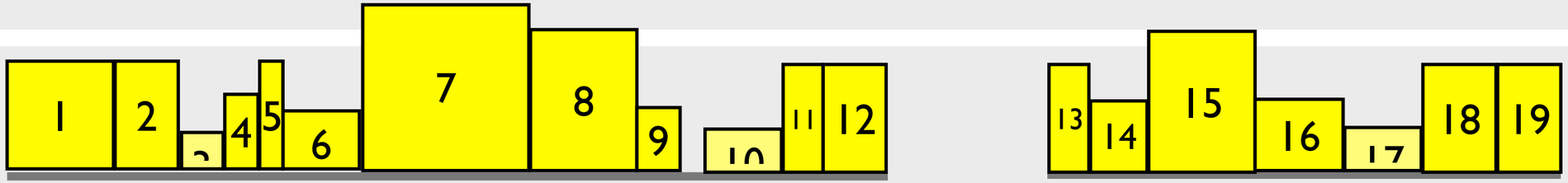
Le problème

entrée

n tâches

une seule machine

chaque tâche i=1,...,n vient avec
- une quantité de travail w_i
- et une date de relâchement r_i



un ordonnancement détermine pour tout moment t:
- mode(t) ∈ {on,off}
- s(t) une vitesse d'éxécution
- j(t) quelle tâche est exécutée (si vitesse positive)

sortie

- $w_i = \int s(t)dt$ sur tout t avec j(t)=i
- chaque tâche i doit s'exécuter pas avant r_i

contraintes

Les utilisateurs veulent minimiser le temps d'attente de leur tâche, donc min C_i-r_i, où C_i est le temps de complétude de la tâche i

L'opérateur de la machine veut minimiser l'énergie consommée par la machine durant l'exécution

objectifs opposés

Le problème

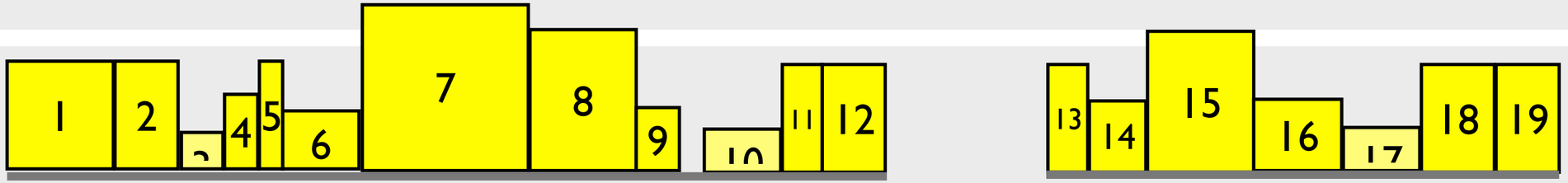
entrée

n tâches

une seule machine

chaque tâche i=1,...,n vient avec
- une quantité de travail w_i
- et une date de relâchement r_i
- et une date limite d_i, qui est $d_i=r_i+F$

F=temps d'attente maximal



un ordonnancement détermine pour tout moment t:
- mode(t) ∈ {on,off}
- s(t) une vitesse d'éxécution
- j(t) quelle tâche est exécutée (si vitesse positive)

sortie

- $w_i = \int s(t)dt$ sur tout t avec j(t)=i
- chaque tâche i doit s'exécuter dans $[r_i,d_i]$

contraintes

- L'énergie consommée est la somme de – pour paramêtres L,g,α (habituellement α=3)
- L fois le nombre de passages *off-on* — énergie pour démarrage
- g fois le temps total où le mode est *on* — énergie de base indépendante de la vitesse
- $\int s(t)^\alpha dt$ — énergie dépendante de la vitesse

objectifs minimiser l'énergie

Le problème

entrée

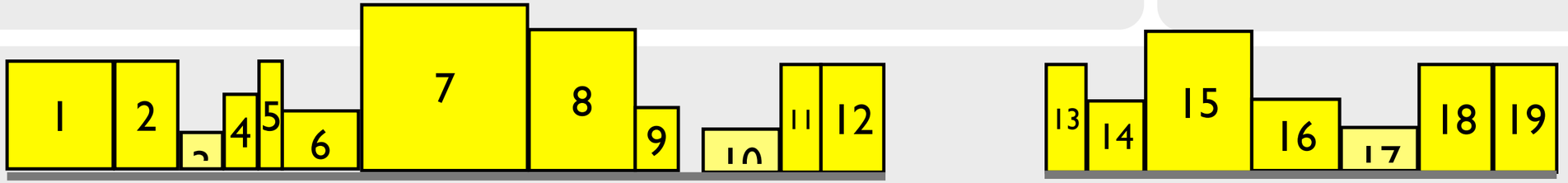
n tâches

une seule machine

chaque tâche i=1,...,n vient avec
- une quantité de travail w_i
- et une date de relâchement r_i
- et une date limite d_i

promesse
dates limites agréables

- $r_1 \leq ... \leq r_n$
- $d_1 \leq ... \leq d_n$



un ordonnancement détermine pour tout moment t:
- mode(t) ∈ {on,off}
- s(t) une vitesse d'éxécution
- j(t) quelle tâche est exécutée (si vitesse positive)

sortie

- $w_i = \int s(t)dt$ sur tout t avec j(t)=i
- chaque tâche i doit s'exécuter dans $[r_i,d_i]$

contraintes

- L'énergie consommée est la somme de – pour paramêtres L,g,α (habituellement α=3)
- L fois le nombre de passages *off-on* — énergie pour démarrage
- g fois le temps total où le mode est *on* — énergie de base indépendante de la vitesse
- $\int s(t)^\alpha dt$ — énergie dépendante de la vitesse

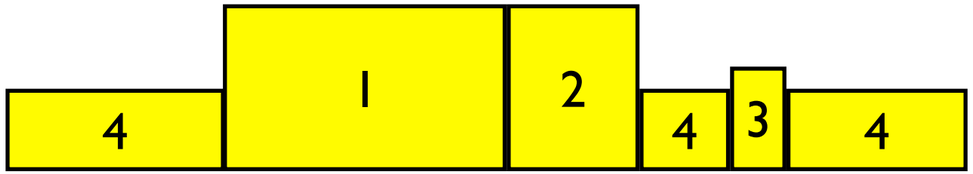
objectifs minimiser l'énergie

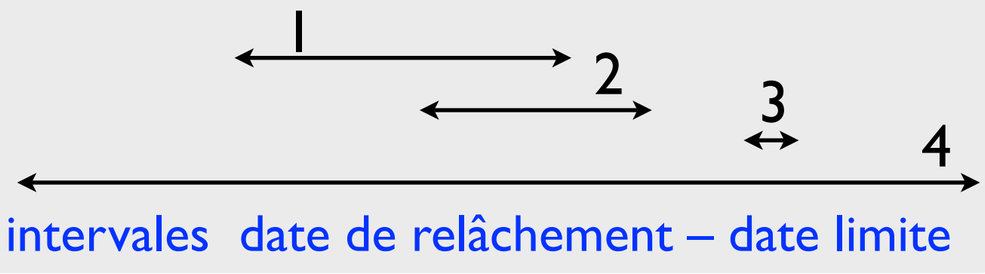
État de l'art

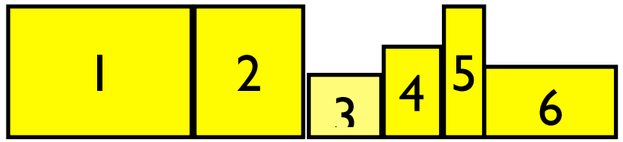
- Problème général est posé dans [Irani,Pruhs'2005]

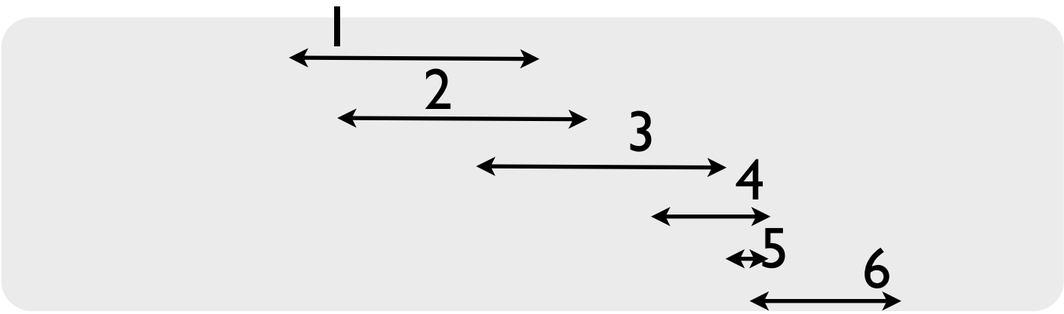
<table>
<tr><th>complexité</th><th>sans variance de vitesse
($\forall t$: $s(t)=1$)</th><th>avec variance de vitesse</th></tr>
<tr><td>sans hibernation
($g=0$, $\forall t$: mode(t)=on)</td><td>$O(n \log n)$
earliest deadline policy
[folklore]</td><td>$O(n^2 \log n)$
[Yao,Demers,Shenker'1995]
[Li,Yao,Yao'2006]
algo Y</td></tr>
<tr><td rowspan="2">avec hibernation</td><td rowspan="2">$O(n^5)$
[Baptiste,Chrobak,D.'2007]</td><td>avec dates limites agréables
$O(n^3)$</td></tr>
<tr><td>général – reste ouvert</td></tr>
</table>

Dates limites agréables donnent beaucoup de strucutre



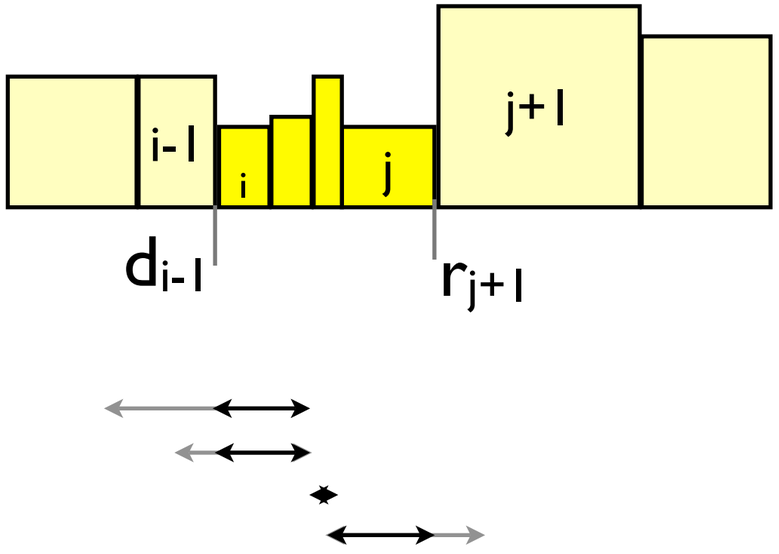






- sans perte de généralité à tout moment la tâche avec la plus petite date limite parmi ceux disponibles est exécutée.
- dans des instances générales, une tâche peut être interrompue par une plus urgente.
- dans les instances agréable, les tâches s'exécutent dans l'ordre sans interruption.

décomposition en sous instances → programmation dynamique

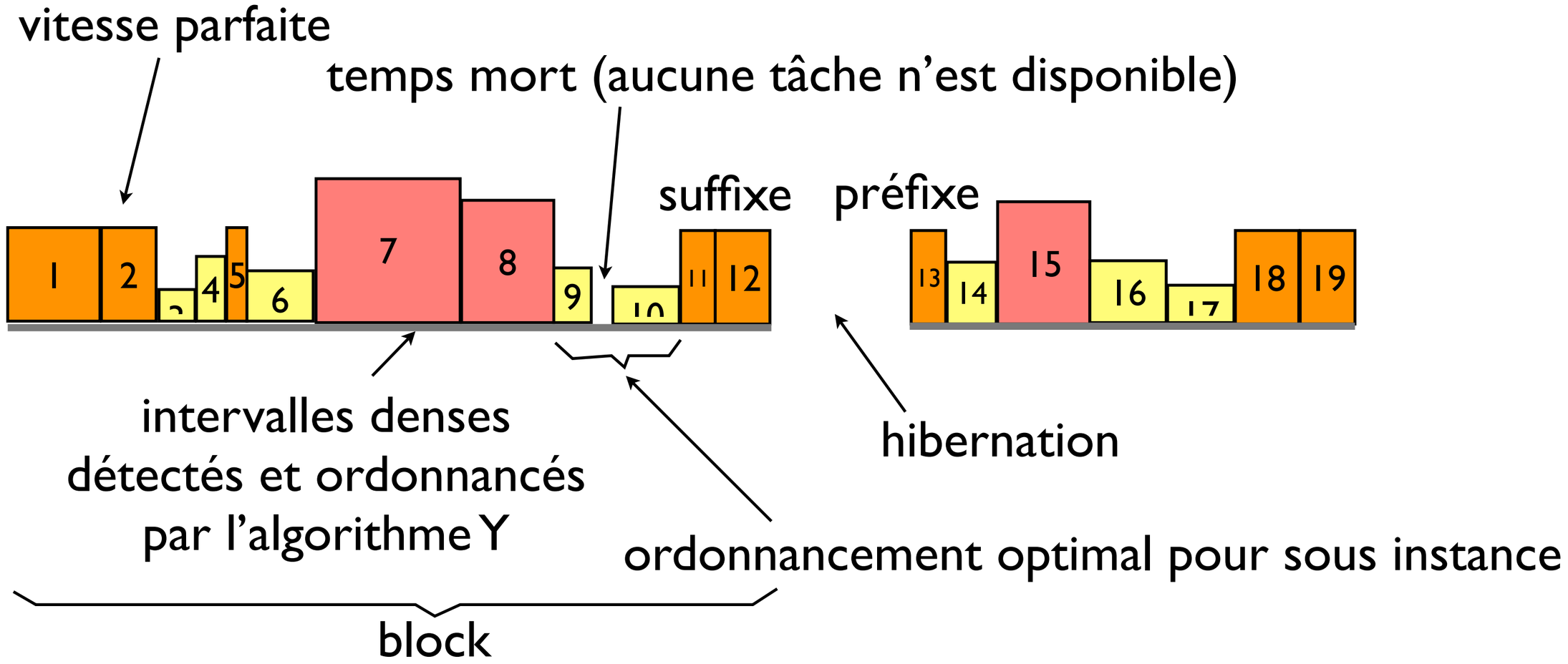


- Une sous instance est définie par une paire de tâches i,j
- consiste en toutes les tâches i,...,j, restreints à l'intervalle $[d_{i-1},r_{j+1}]$, où $d_0=-\infty$, $r_{n+1}=+\infty$
- n'a pas de solution (coût $+\infty$) si une des tâches est restreinte à l'intervalle vide,
  donc si $d_{i-1}=d_i$ ou $r_{j+1}=r_j$ ou $d_{i-1}=r_{j+1}$

Intervalle dense

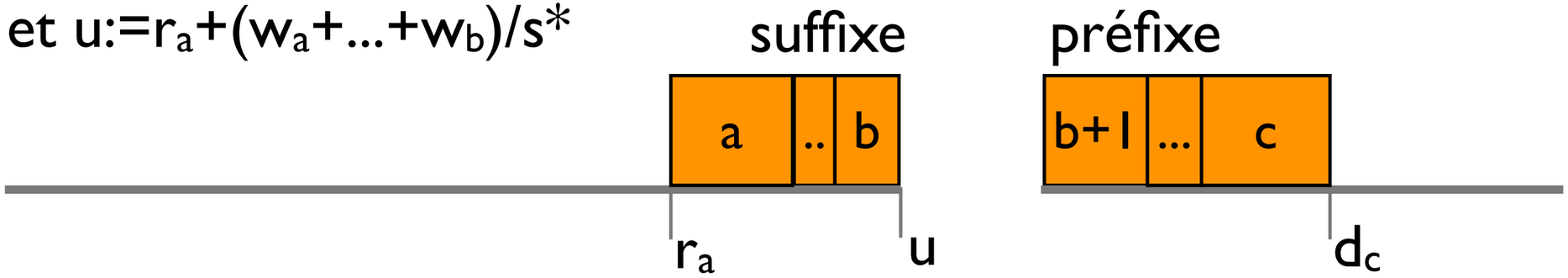
- Définition vitesse parfaite s^* : vitesse s qui minimise $(1/s)(s^{\alpha}+g)$, qui est l'énergie consommé par unité de travail. Idéalement on aimerait seulement ordonnancer à cette vitesse.
- étape 1 : calculer l'ordonnancement optimal sans hibernation (algo Y)
- Les intervalles à vitesse parfaite ou plus, doivent être ordonnancés ainsi.
- Ils séparent l'instance en sous instances indépendantes.

ordonnancements optimaux

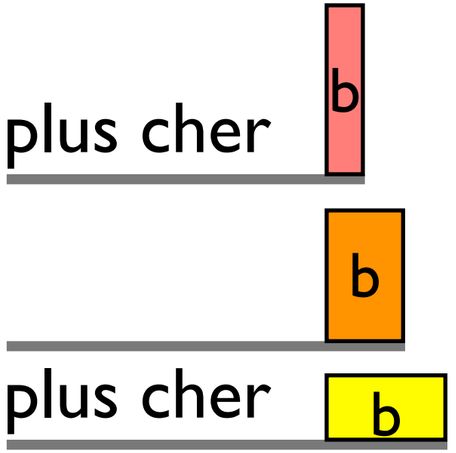


Suffixes, préfixes

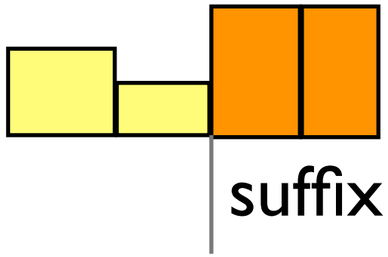
- Définition suffixe d'un block : paire de tâches (a,b),
  telle que les tâches a..b soient ordonnancés à vitesse parfaite entre r_a
  et $u:=r_a+(w_a+...+w_b)/s^*$



- Par optimalité chaque block possède un suffixe non-vide.
- Sans perte de généralité b est maximal : $r_{b+1}>u$
- Définition préfixe (symétrique)
- par balayage on calcule les fonctions $f:a\mapsto b$, $h:b+1\mapsto c$



Détail subtil



- Calculer le premier préfixe [1,h(1)] et le dernier suffixe [$f^{-1}(n)$,n]
- Ensemble avec les intervalles denses, ils décomposent en sous-instances

Programme dynamique

- Y_{ij} ordonnancement optimal sans hibernation pour sous instance (i,j)
- O_{ij} idem, mais avec hibernation

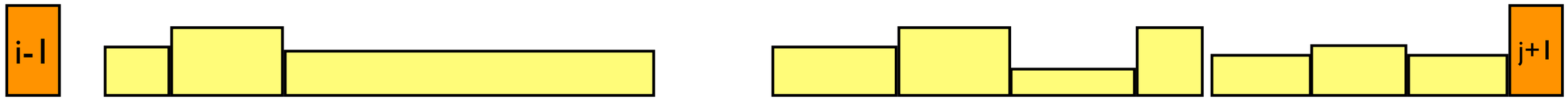
Cas de base : instance vide j=i-1

coût est $\min(L, (r_i - d_{i-1})g)$

hibernation

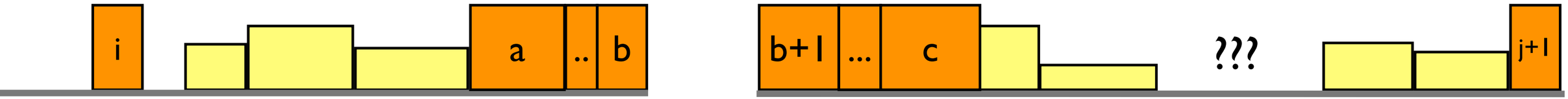
temps mort

Alternative sans hibernation



- si l'ordonnancement optimal est sans hibernation son coût est $Y_{ij} + (r_i - d_{i-1})g$

Alternative avec hibernation



- Si l'ordonnancement optimal a des périodes d'hibernation, et que la première est entourée de tâches,
  alors soit (a,b) le suffixe et (b+1,c) le préfixe associé.

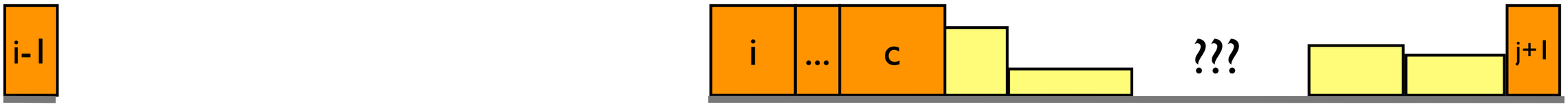
- Son coût est
  $Y_{i,a-1} + (w_a+..+w_c)g^* + L + O_{c+1,j}$

  avec $g^*=(g+(s^*)^\alpha)/s^*$

Alternative avec hibernation finale

- Si l'ordonnancement optimal a une unique période d'hibernation, et qu'elle termine en r_{j+1}, alors soit (a,j) le suffixe associé.
- Son coût est
  $Y_{i,a-1} + (w_a+..+w_j)g^* + L$

Alternative avec hibernation initiale



- Si l'ordonnancement optimal débute par une période d'hibernation, alors soit (i,c) le préfixe associé.

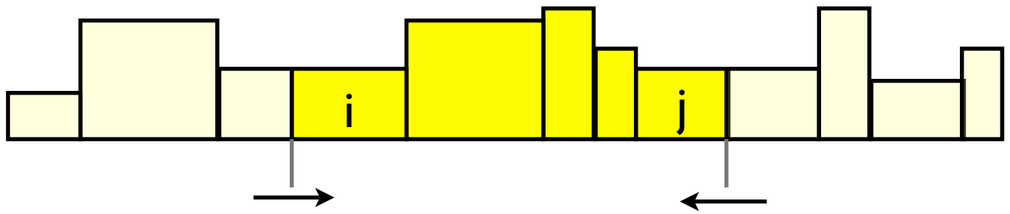
- Son coût est
  $L + (w_i+..+w_c)g^* + O_{c+1,j}$

Notre algorithme

- étape 1 : calculer les fonctions f, f^{-1}, h par balayage en $O(n)$
- étape 2 : calculer les valeurs Y_{ij}, – problème : l'algo Y est en $O(n^2)$
- étape 3 : calculer les valeurs O_{ij},
  chacune est un minimum sur $O(n)$ alternatives.
  Complexité $O(n^3)$

Calcul de (Y_{ij})

- étape 2.1 : calculer Y_{1n} en temps $O(n^2)$
- étape 2.2 : calculer $(Y_{1j})_j$ par compression droite→gauche à partir de Y_{1n}
- étape 2.3 : calculer $(Y_{ij})_i$ par compression gauche→droite à partir de Y_{1j}



- une compression considère $O(n)$ évènements. Complexité $O(n^3)$

Évènements

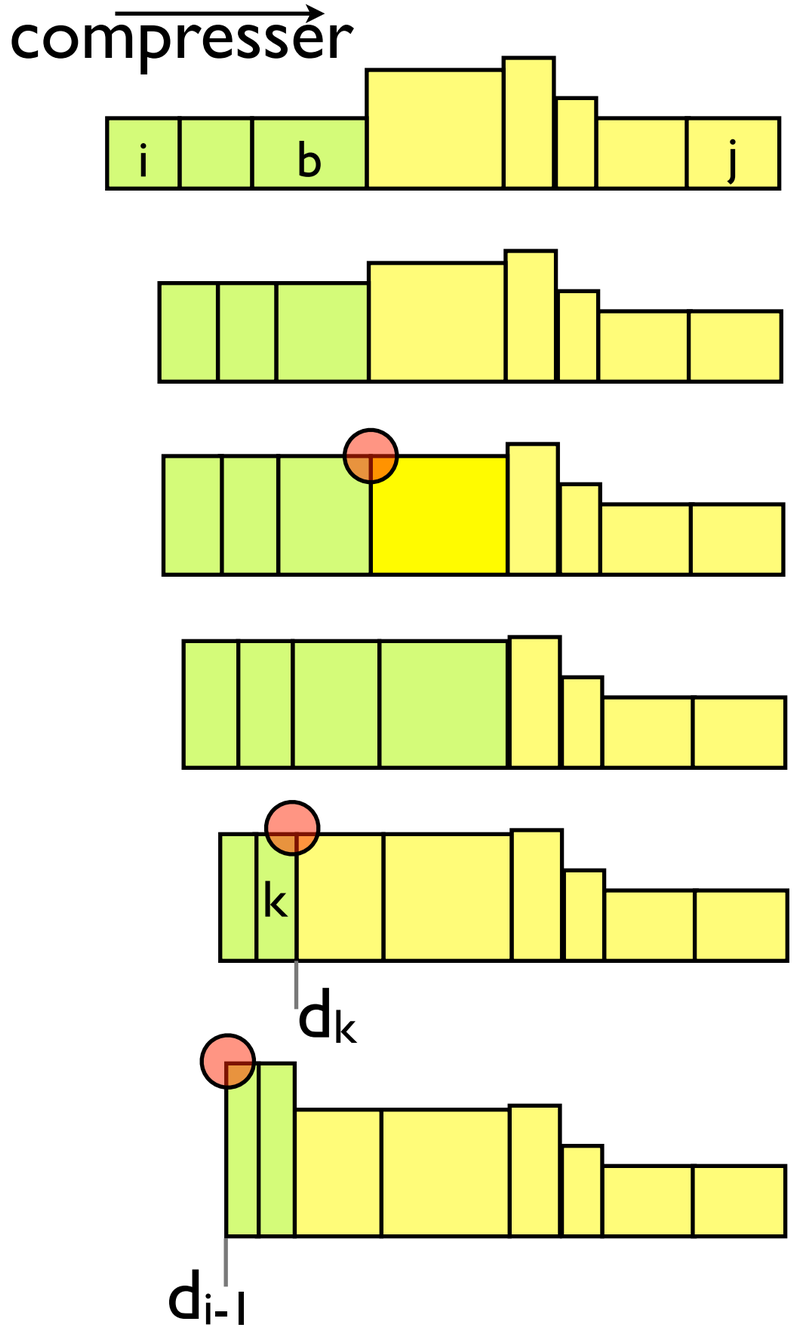
- Pour un ordonnancement des tâches i..j, considérer le block maximal des tâches i..b exécutées à même vitesse. Maintenant on augmente la vitesse jusqu'à ...

- évènement d'infaisabilité : si $d_{i-1}=d_i$ ou $r_j=r_{j-1}$ ou $d_{i-1}=r_{j-1}$ une des tâches est restreinte à un intervalle vide.
  Poser $Y_{ij}=+\infty$. Incrémenter i.

- évènement de réunion : quand la vitesse atteint celle de la tâche b+1. Agrandir le block.

- évènement de séparation : quand une tâche atteint sa date limite. Réduire le block.

- évènement de date limite : quand le début du block atteint d_{i-1}.
  Poser Y_{ij}= le coût courant. Incrémenter i.



C’est tout

merci