Ordonnancement sur une machine

Module MOP-Cours 2

Spécialité IAD

Philippe Chrétienne

Plan

- Critères Min-Max
  - 1 / prec / f_{max}
  - 1 / prec , pmtn,r_j / f_{max}
  - 1 / prec , r_j , p_{mtn},L_{max}
  - 1 / r_j , q_j / C_{max}
- Critères Min-Sum
  - 1 / / $\sum w_j C_j$
  - 1 / r_j , pmtn / $\sum w_j C_j$

Notations

Enoncé d'un problème : E

Paramètres de E :

- les jobs : J= $\{J_1,\ldots,J_n\}$
- les durées p_j ;
- les contraintes de précédence : (J_i,J_j)
- les dates de disponibilité r_j
- les deadlines d_j ;
- les durées de latence q_j ;
- la préemption : pmtn
- la fonction coût : $f_j(C_j)$.

E = <J,G,p,r,d,f>

f*(E) coût minimum d'un ordonnancement de E.

Algorithme de résolution d'un problème : A

Solution fournie par A pour l'énoncé E : A(E)

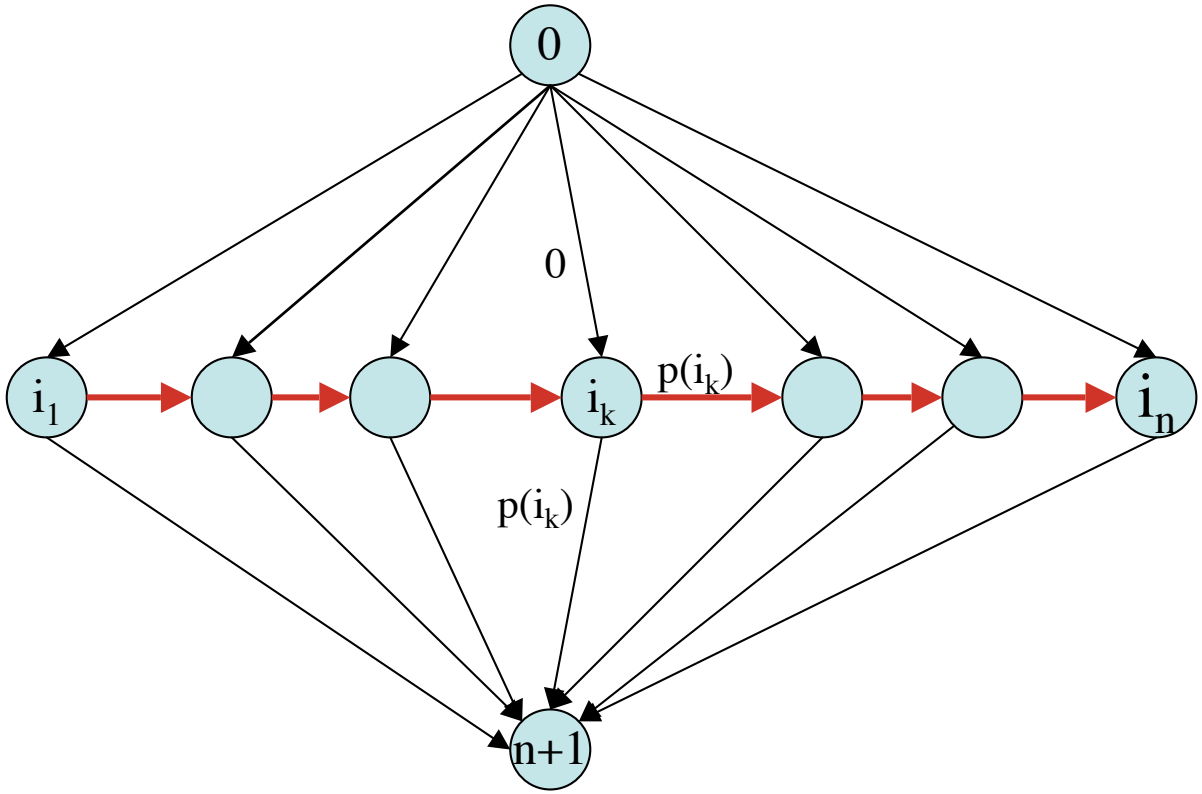
1 / prec / f_{max}

Un coût $f_j(C_j)$ est attaché à la terminaison du job J_j à la date C_j.
Les fonctions f_j sont croissantes au sens large.

Idée de base de l'algorithme :
En raison de la monotonie des fonctions de coût,
il existe un ordonnancement optimal sans temps mort.

Les solutions sans temps mort sont donc dominantes.
Une solution sans temps mort du problème est donc une séquence $s=(i_1,i_2,...,i_n)$ des n jobs compatible avec « prec ».

L'ordonnancement associé à $s=(i_1,i_2,...,i_n)$ est alors l'ordonnancement au plus tôt pour les contraintes de précédence associées à s.



Remarque :

Quelle que soit la séquence réalisable s, la dernière tâche se termine à la date $P = \Sigma_{i=1..n}\, p_i$.

L'algorithme de Lawler construit un séquencement optimal de la dernière à la première tâche.

Algorithme A1(E) ;
Si Card(J)=1 alors Retourner((J$_1$)) ;
Choisir une sortie J_k de G dont le coût $f_k(\sum_J p_j)$ est minimum ;
J := J-{J_k} ; G := G(J)
Ê := <J,p,G,f> ;
Retourner ((A1(Ê),J_k)).

Propriété de A1.
Soit E un énoncé, Ô= A1(E) et soit J_k le dernier job de Ô.
Il existe une solution optimale de E dont le dernier job est J_k .
Preuve :
Soit O* une solution optimale dont le dernier job est J_i où i≠k .
Notons $O^*=O_1\, J_k\, O_2\, J_i$ la séquence des jobs dans O*.
La séquence $O' = O_1\, O_2\, J_i\, J_k$ est réalisable ;
La solution O' satisfait $f_{max}(O') \leq f_{max}(O^*)$. QED.

Preuve de A1 (récurrence sur n).
P(n): Si Card(J)=n, A1(E) est un ordonnancement optimal.

P(1) est vraie.
Supposons que P(n-1) soit vraie.

Considérons un énoncé E à n jobs.
Soit $\hat{O} = A1(E) = O_1\ J_k$.
O_1 est optimal pour <J-{J_k} ,p,G(J -{J_k}),f> (induction et récursivité)
D'après la propriété précédente, il existe un ordonnancement optimal $O^* = O_2\ J_k$.

On a alors :
$f_{max}(\hat{O}) = \max\{f_{max}(O_1), f_k(P)\} \leq \max\{f_{max}(O_2), f_k(P)\} = f_{max}(O^*)$
Il en résulte que $\hat{O}$ est optimal. QED.

$1 / prec, pmtn, r_j / f_{max}$

Première étape :
Rendre les dates de disponibilité r_j « compatibles avec G » .

Propriété :
Soit O un ordonnancement quelconque.
Si $(J_i,J_j) \in$ prec alors : $S_j \geq r_i + p_i$.
$\max\{r_i+p_i, r_j\}$ est donc aussi une date de disponibilité pour J_j .

Algorithme pour modifier les dates de disponibilité r_j :

on ajuste les dates de disponibilité dans l'ordre d'une liste topologique (par rapport à prec) des jobs.

Supposons que $(J_1,J_2,\ldots,J_n)$ soit une liste topologique de prec.
(i.e : $(J_i,J_j) \in$ prec $\Rightarrow i < j$)

Algorithme B1(E)

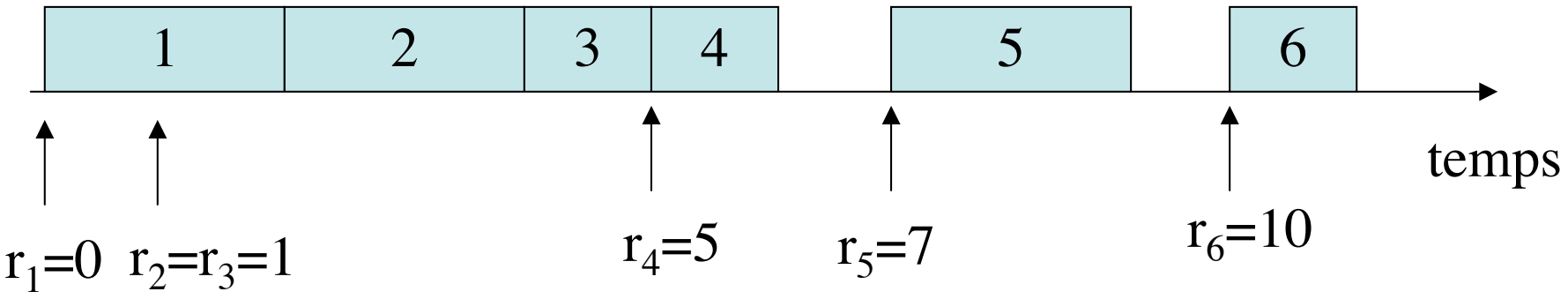
$r'_1 := r_1$;
Pour j de 2 à n faire $r'_j := \max(r_j, \max_{(i,j) \in prec} (r'_i + p_i))$ FinPour ;
Retourner(r').

Propriété (des dates r'_j):
Les valeurs r'_j sont des dates de disponibilité pour E.
Pour tout J_j, $r'_j \geq r_j$;
Pour tout $(J_i,J_j) \in$ prec : $r'_j \geq r'_i$.

On supposera dans la suite que les dates de disponibilité r_j sont compatibles avec prec.
Donc, si $r_1 \leq r_2 \leq \ldots \leq r_n$, la séquence (1,2, ...,n) est réalisable.

Deuxième étape :
Calcul de l'ordonnancement au plus tôt non préemptif O pour la séquence (1,2,...,n).

Exemple :
n=6 ;
$r_1 = 0, r_2 = r_3 = 1, r_4 = 5, r_5 = 7, r_6 = 10$;
$p_1 = p_2 = 2, p_3 = p_4 = 1$; $p_5=2$; $p_6 = 1$;



Un bloc B possède une date de début notée début(B), une date de fin notée fin(B), un ensemble de jobs noté jobs(B).

Algorithme B2(E) ;
% retourne la liste des blocs de O%
%construction du premier bloc%
i := ArgMin{r_j,j∈J} ; t := Min{r_j,j∈J} ;
début(B) := t ; jobs(B) := {i} ;
Tantque i < n et $r_{i+1} \leq t + p_i$ faire
    jobs(B) := jobs(B) ∪ {J_{i+1}} ;
    t := t + p_i ;
    i := i + 1
FinTantque ;
fin(B) := t +p_i ;
Si (i=n) alors retourner((B))
    sinon %appel récursif pour les autres blocs%
    J := J-jobs(B) ; G := G(J) ;
    Ê := <J,G,p,r,f> ;
    Retourner ((B,B2(Ê)))
FinSi

Propriétés de l'ordonnancement O :

Soient :

- $(B_1, B_2, \ldots, B_q)$ la suite des blocs obtenus ;
- $u_1, u_2, \ldots, u_q$ les dates de début des blocs ;
- $v_1, v_2, \ldots, v_q$ les dates de fin des blocs.

Propriété 1:

Les dates u_k sont des dates de disponibilité r_j.

Si le job J_j appartient à jobs(B_k) , alors $u_k \leq r_j \leq v_k$.

Propriété 2 :

Soit O* un ordonnancement optimal de l'énoncé E du problème initial (préemptif) .

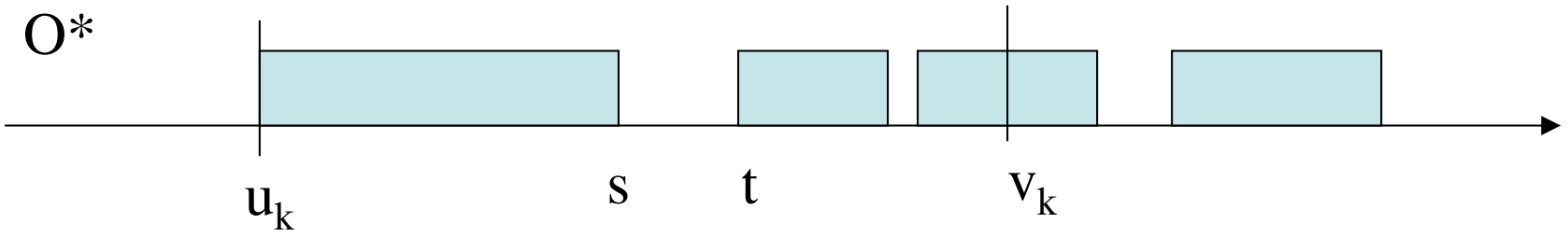
Dans O*, les jobs de B_k sont exécutés dans $[u_k, v_k]$.

Preuve de la propriété 2 :

Supposons que dans O*, $[u_k,v_k]$ soit le premier intervalle non totalement occupé par les jobs de B_k.

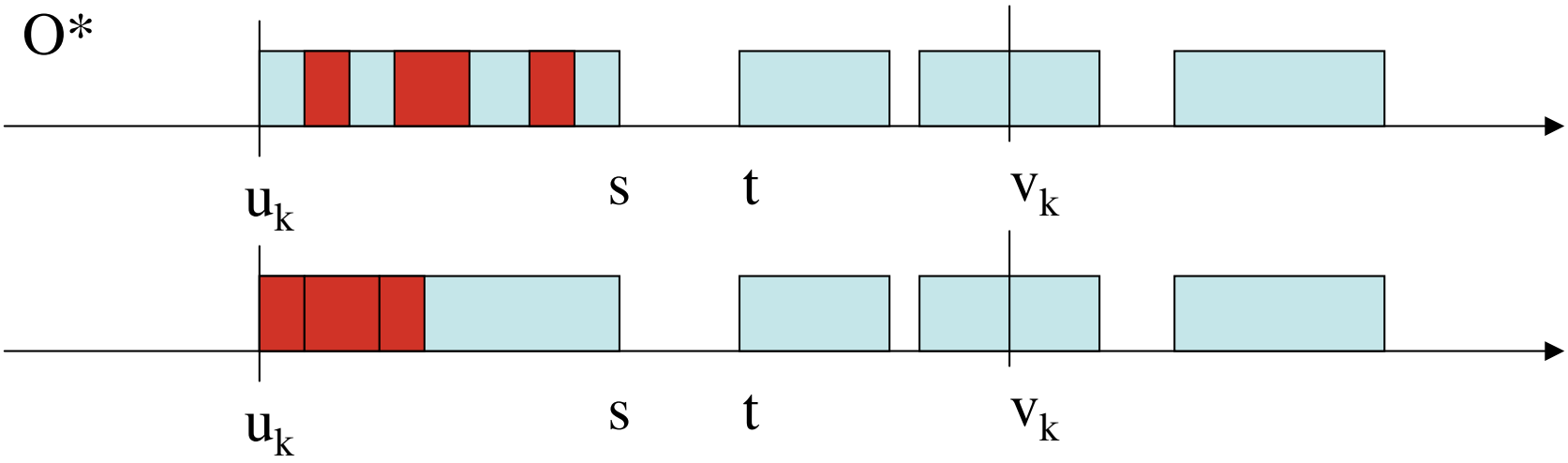
Comme les jobs de B_{k+1}, ...,B_q ne sont pas disponibles avant u_{k+1} (Propriété 1), les jobs de B_k ne sont pas totalement exécutés dans $[u_k,v_k]$.

Soit donc [s,t] le premier temps-mort de $[u_k,v_k]$ dans O*.



Montrons qu'il existe un job J_p de B_k tel que :
$r_p \leq s$ et J_p n'est pas terminé à la date s dans O*.
Dans le cas contraire :

- les jobs J_i de B_k tels que $r_i \leq s$ sont terminés à s dans O*;
- on peut donc exécuter ces jobs dans l'ordre compatible avec (1,2,...,n) et sans préemption dans $[u_k,s]$;
- l'ordonnancement précédent est par définition le début de l'ordonnancement du bloc B_k dans O.

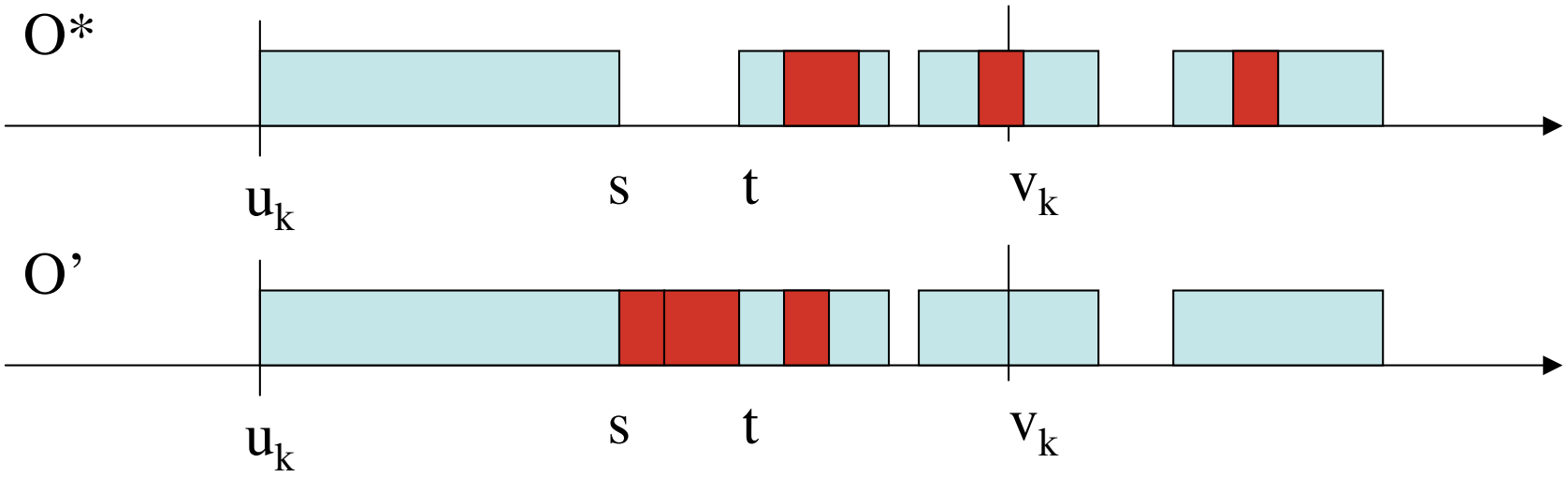


En rouge, premier job de B_k dans la séquence (1,2...,n) dont la date de disponibilité est u_k.

Soit alors T l'ensemble des jobs exécutés après s dans O.
Comme $\text{Min}\{r_j / J_j \in T\} > s$, O comporte un temps mort dans $[u_k, v_k]$. Contradiction.

Soit R_p la durée résiduelle de J_p au temps s.
On peut alors :

- exécuter J_p, sur l'intervalle $[s, s+\text{Min}\{R_p, t-s\}]$;
- supprimer dans O* les $\text{Min}\{R_p, t-s\}$ dernières unités de temps d'exécution de J_p;



L' ordonnancement O' obtenu satisfait : $f_{max}(O') \leq f_{max}(O^*)$.
O' est donc un ordonnancement optimal.

En itèrant la transformation tant qu'il existe un temps-mort dans l'ordonnancement optimal en cours, on obtient un ordonnancement optimal tels que les jobs de chaque bloc B sont exécutés dans la fenêtre [début(B),fin(B)]. QED

D'après la propriété 2, on peut résoudre de manière indépendante les problèmes associés à chaque bloc B.

Soit [deb(B),fin(B)] la fenêtre de B.
Soit E(B) l'énoncé associé aux seuls jobs de jobs(B).
Soit O*(B) un ordonnancement optimal de E(B).
Soit G(B) le graphe de précédence des jobs de jobs(B).
Soit S(B) l'ensemble des jobs sans successeurs dans G(B).

Propriété 3:

2 bornes inférieures de f*(J) :

1) $f_l(\text{fin}(B)) = \text{Min}_{S(B)} (f_j(\text{fin}(B)))$;

2) $f^*(E(B-\{J_l\}))$

où $E(B-\{J_l\}$ est l'énoncé associé aux jobs de $B-\{J_l\}$.

Preuve :

En remplaçant dans O*(B) les morceaux de J_l par des temps-morts, on obtient un ordonnancement O' de $E(B-\{J_l\})$ tel que :

$f_{max}(O^*(B)) = \max(f_{max}(O'), f_l(C_l^*))$ où C_l^* est la date de fin de J_l dans O*(B)

En effet les jobs de $B-\{J_l\}$ ont la même date de terminaison dans O*(B) et O'.

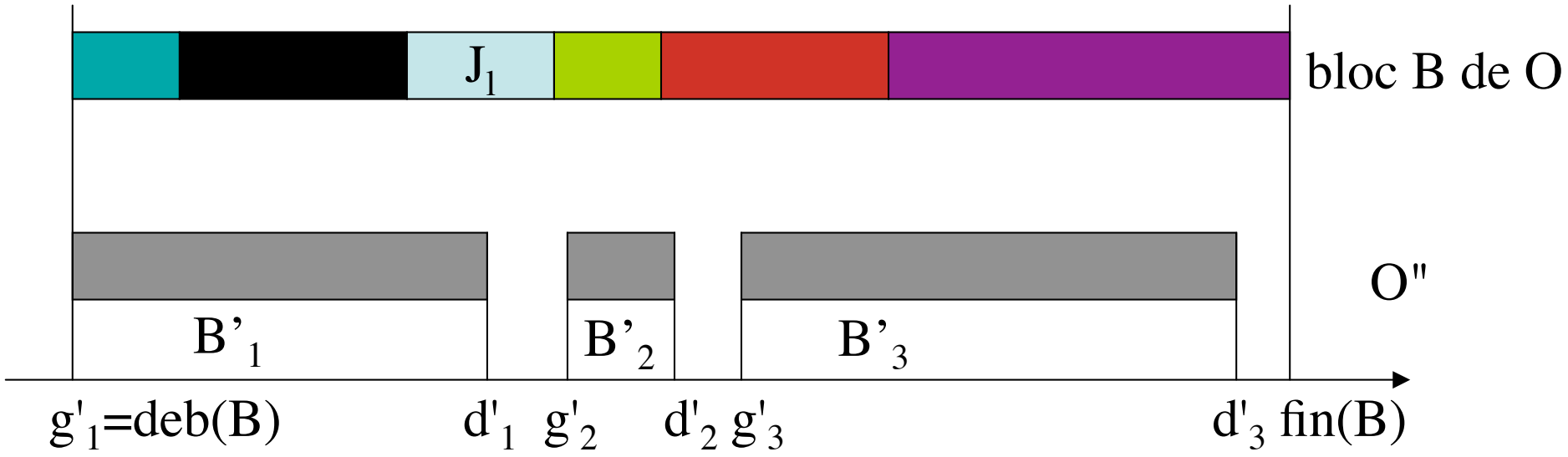
Il en résulte que : $f^*(E(B)) \geq f_{max}(O') \geq f^*(E(B-\{J_l\}))$. QED.

Construction (récursive) d'un ordonnancement optimal.

Soient :

- $B'_1, B'_2, \ldots, B'_q$ les blocs associés à l'ordonnancement non préemptif au plus tôt O' des jobs de B-$\{J_1\}$;
- $g'_1, g'_2, \ldots, g'_q$ les dates de début des blocs de O' ;
- $d'_1, d'_2, \ldots, d'_q$ les dates de fin des blocs de O';

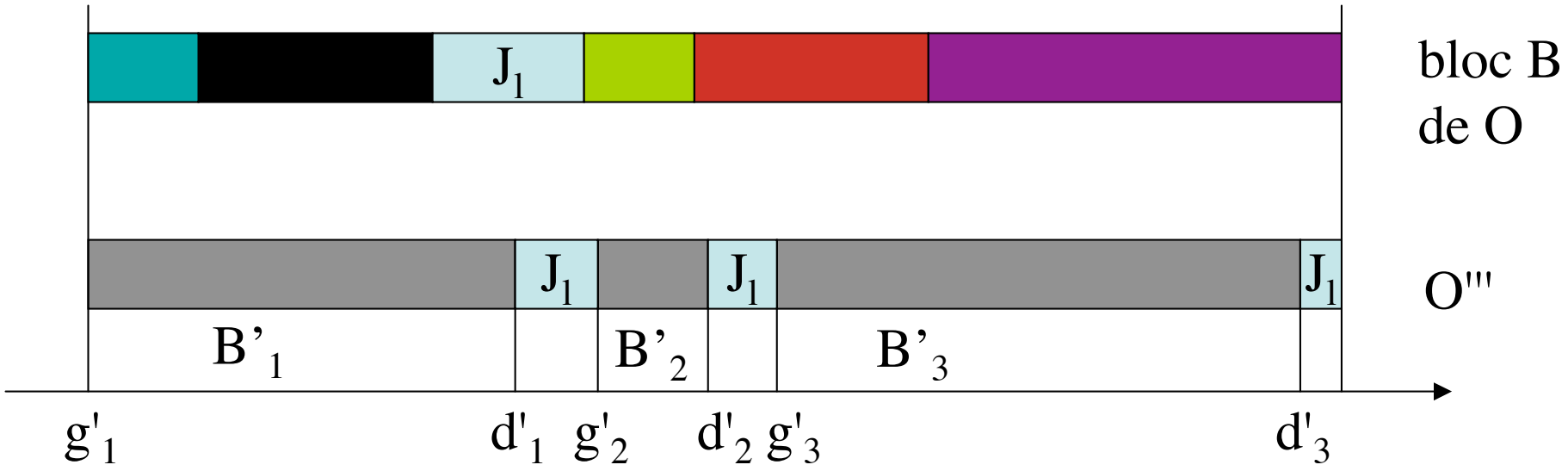
Soit O" un ordonnancement (préemptif) optimal pour E(B-$\{J_1\}$)
(rappel : les jobs de B'_k sont placés dans $[g_k,d_k]$ (Propriété 2)).



En remplaçant dans la fenêtre [deb(B),fin(B)] les temps morts de O" par des morceaux d'exécution de J_1, on obtient un ordonnancement O'''

- réalisable pour E(B) car les prédécesseurs de J_1 sont exécutés dans le bloc B'_1 (les r_i étant les mêmes pour E(B) et E(B-{J_1}));
- dont le coût, égal à max(f*(E(B-{J_1}),$f_1(C_1^*)$), est au plus égal à max(f*(E(B-{J_1}),f_1(fin(B))).

O''' est donc un ordonnancement optimal (Propriété 3).



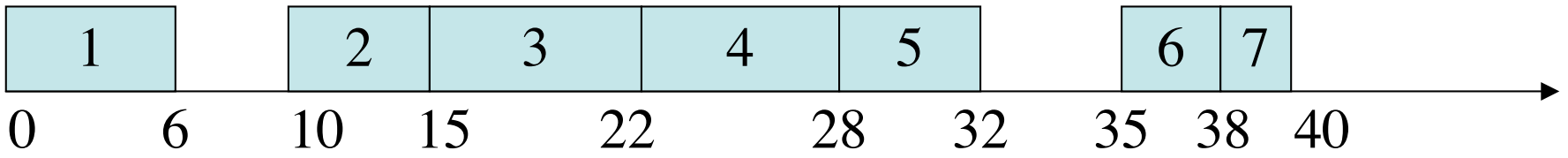
Exemple :

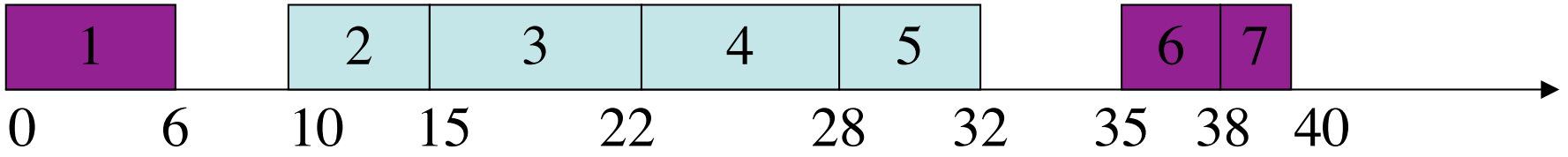
i	1	2	3	4	5	6	7
p_i	6	5	7	6	4	3	2
r_i	0	10	11	13	20	35	35
d_i	23	33	26	14	19	39	40

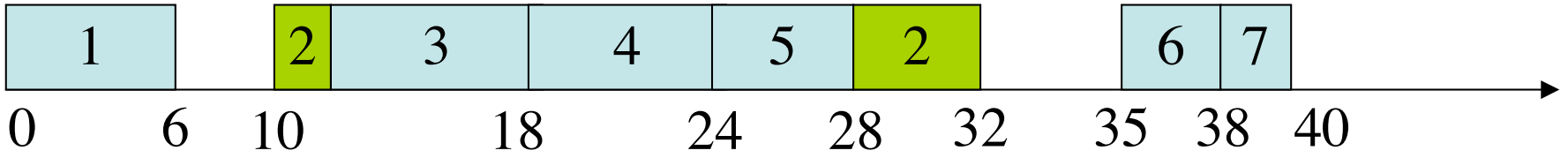
$f_i(t) = Max\{0, t-d_i\}$ (retard absolu du job J_i)

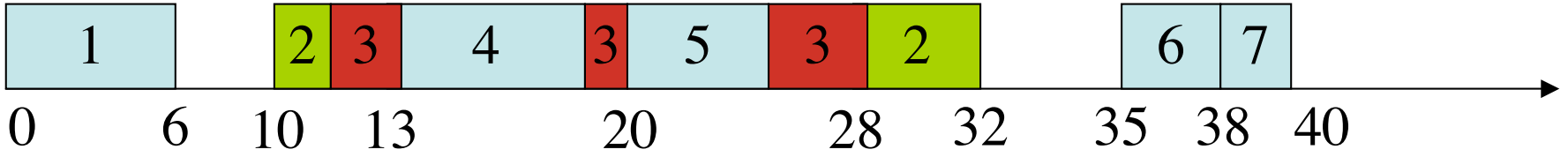
prec = $\varnothing$ (jobs indépendants)

i	1	2	3	4	5	6	7
p_i	6	5	7	6	4	3	2
r_i	0	10	11	13	20	35	35
d_i	23	33	26	14	19	39	40









Algorithme (récursif) B3(E(B))
(pour ordonnancer un seul bloc B de B2(E)).

Algorithme B3(E(B)) ;
Si Card(jobs(B))=1
  alors O := placerjob(jobs(B),deb(B),fin(B),O)
  sinon
    Soit J_1 une sortie de G(B) de coût f_1(fin(B)) minimum ;
    $(B_1,B_2,\ldots,B_p)$:= B2(E(B-{J_1})) ;
    Pour k de 1 à p faire
      O' := B3(E(B_k)) ; %appel récursif%
      O[deb(B_k),fin(B_k)] := O' ;
      remplacertempsmort(J_1,deb(B),fin(B),O)
    FinPour ;
FinSi ;
Retourner(O).

Remarques sur B3(E)

La procédure placerjob(J_i,u,v,O) place le job J_i dans la fenêtre [u,v] d'un ordonnancement O;

La procédure remplacertempsmort(J_i,u,v,O) remplace les temps morts d'un ordonnancement O par des morceaux d'exécution du job J_i .

Algorithme général A2(E) pour 1 / prec,r_j,pmtn / f_{max} .

Algorithme A2(E) ;
(B_1,B_2,…,B_p) := B2(E) ;
Pour k de 1 à p faire
  O*.[deb(B_k),fin(B_k)]:= B3(E(B_k))
FinPour ;
Retourner(O*).

$1 / prec, r_j, pmtn / L_{max}$

Etape 1 : Modification des deadlines (mous).

Supposons que $(J_1,J_2,\ldots,J_n)$ soit une liste topologique de prec.

Algorithme de modification des deadlines :
Algorithme B4(E)
$d'_n := d_n$;
Pour i de n-1 à 1 faire $d'_i := \min(d_i, \min_{(i,j) \in prec} (d'_j - p_j))$ FinPour.

Propriété des dates d'_j :
Les valeurs d'_j sont des deadlines telles que :
- pour tout J_j, $d'_j \leq d_j$;
- $(J_i,J_j) \in prec \Rightarrow d'_i \leq d'_j$.

Algorithme A3(E) :

d' := B4(E) ;

t := $Min_J \{r_j\}$;

Tantque (tous les jobs ne sont pas terminés) faire

Choisir un job J_i tel que ($r_i \leq t$) dont le deadline d'_i est minimum ;

Exécuter J_i jusqu'au plus petit instant $u > t$ tel que :

- soit J_i est terminé à u;
- soit u est une date de disponibilité.

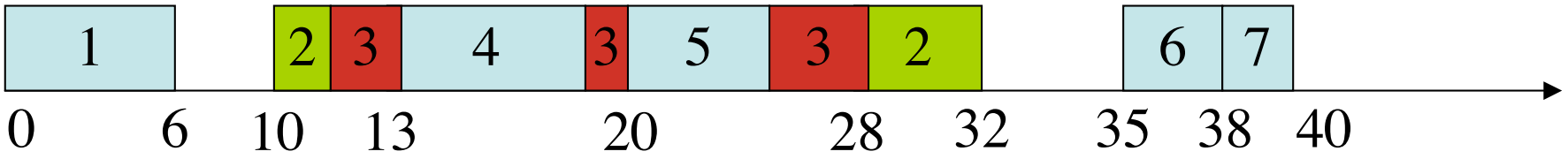
t := u ;

FinTantque.

Exemple pour 1 / prec,pmtn,r_j / L_{max}

i	1	2	3	4	5	6	7
p_i	6	5	7	6	4	3	2
r_i	0	10	11	13	20	35	35
d_i	33	43	36	24	29	42	50

$f_i(t) = t-d_i$ (retard algébrique du job J_i)
prec = ∅ (jobs indépendants)



$1 / r_j , q_j / C_{max}$

Définition du problème.
n jobs non préemptifs indépendants.
Pour chaque job J_j :
• une date de disponibilité r_j ;
• une durée de latence q_j (temps de séchage, durée de livraison,…).

La durée de latence q_j représente un temps minimum nécessaire entre la fin d'exécution de J_j sur la machine et la terminaison globale.
La machine n'est donc pas requise par J_j entre C_j et C_j+q_j.
Nouvelle définition de C_{max} : $C_{max} = Max_J\{C_j + q_j\}$

Il faut minimiser C_{max} .

Complexité de $1 / r_j , q_j / C_{max}$

Propriété :
$1 / r_j, q_j / C_{max}$ est NP-complet au sens fort

Preuve :
Réduction pseudo-polynomiale de 3-PARTITION

Remarque :
$C_{max} \leq C \Leftrightarrow \forall\ j, C_j \leq C - q_j$.

Il en résulte que les problèmes de décision
$1 / r_j , q_j / C_{max}$ et $1 / r_j , d_j / -$
sont équivalents en complexité.

Montrons que $1 / r_j, d_j / -$ est NP-complet au sens fort.

3-PARTITION:

Donnée:

$A=\{a_1,\ldots\ldots, a_{3n}\}$;

$s:A\rightarrow\aleph$ telle que :

- $s(A)=nB$ et
- $\forall i\in\{1,..,3n\}\ B/4 < s(a_i) < B/2$

Question:

Peut-on réaliser une partition de A en n classes de même poids?

Si la réponse est « oui », chaque classe contient exactement 3 éléments de A et est de taille B.

On montre que 3-PARTITION $\propto_{ps}$ OMS

Réduction pseudo-pôlynomiale :

Enoncé de 3-PARTITION:

I=(A,s) où Card(A)=3n ;

s(A)=nB ;

$\forall a \in A$, $B/4 < s(a) < B/2$.

Enoncé f(I) de OMS:

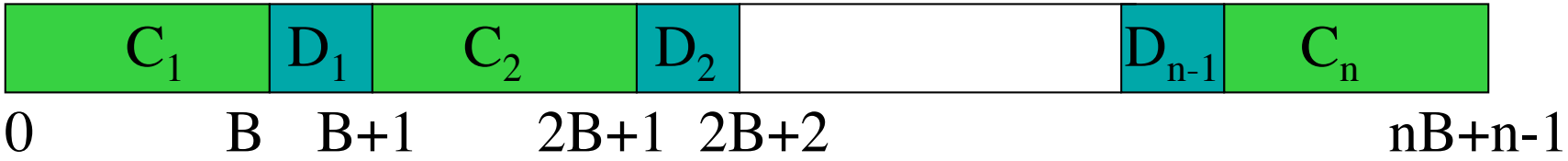
3n tâches T_i:
($p_i = s(a_i)$, $r_i=0$, $d_i=nB+n-1$)

n-1 tâches D_j:
($p_j=1$, $r_j = jB+j-1$, $d_j = jB+j$)

Supposons l ’énoncé (A,s) à réponse « oui ».

Soit alors C_1,..,C_n les sous-ensembles de 3 tâches T_i associées aux n classes de A de poids B.

L ’ordonnancement de f(I) suivant est réalisable:



Réciproquement, supposons qu ’il existe un ordonnancement de l ’énoncé f(I) de OMS.

Les n-1 tâches D_j sont bloquées dans les intervalles [jB+j-1,jB+j].

L ’espace libre restant de l ’intervalle [0, nB+n-1] est juste suffisant pour les autres tâches T_i.

On peut donc partitionner A en n classes de poids B.

Résolution de 1 / r_j , q_j / C_{max}

1) Recherche d'une bonne solution approchée par l'algorithme de Jackson ;
2) Propriétés fondamentales de la solution de Jackson ;
3) Mise en œuvre d'une méthode arborescente fondée sur 2).

L'algorithme de Jackson construit une solution notée O_0 en appliquant la règle simple suivante (règle de Jackson) :

Dès que la machine est libre et que l'ensemble des tâches prêtes est non vide, ordonnancer la tâche prête de plus grande durée de latence.

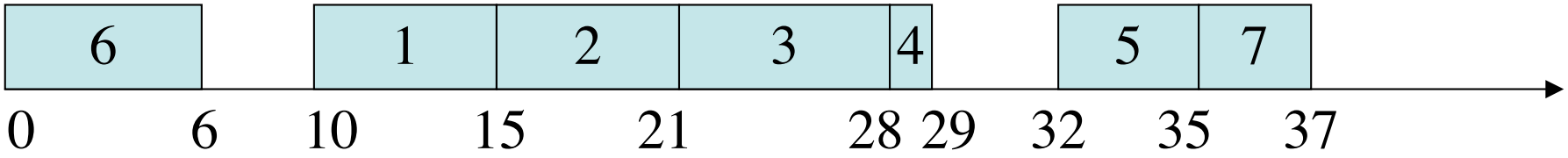
Algorithme de Jackson :

Algorithme B5(E) ;
Si J ≠ ∅
  alors u := $Min_J\{r_j\}$;
        Choisir J_i dans J tel que r_i=u et q_i maximum ;
        O.[u,u+p_i]:=J_i ; J:=J-{J_i} ;
        Pour tout Jj de J faire r_j := Max{r_j,u+p_i}
        Ê := <J,p,r,q> ;
        Retourner(B5(Ê)) ; %appel récursif%
FinSi.

La notation O.[u,v]:= J_i signifie que la fenêtre [u,v] est affectée au job J_i.

Exemple :

j	1	2	3	4	5	6	7
r_j	10	13	11	20	32	0	32
p_j	5	6	7	1	3	6	2
q_j	7	26	24	21	8	17	0



Propriétés de la solution de Jackson

Borne inférieure :

Soit I une partie quelconque de J , la valeur

$$h(I) = \text{Min}_I \{r_j\} + \sum_I p_j + \text{Min}_I q_j$$

est une évaluation par défaut de la durée optimale f*.

Il en résulte que : $\text{Max}_I\, h(I) \leq f^*$.

Solution de Jackson et évaluation par excès de f*.

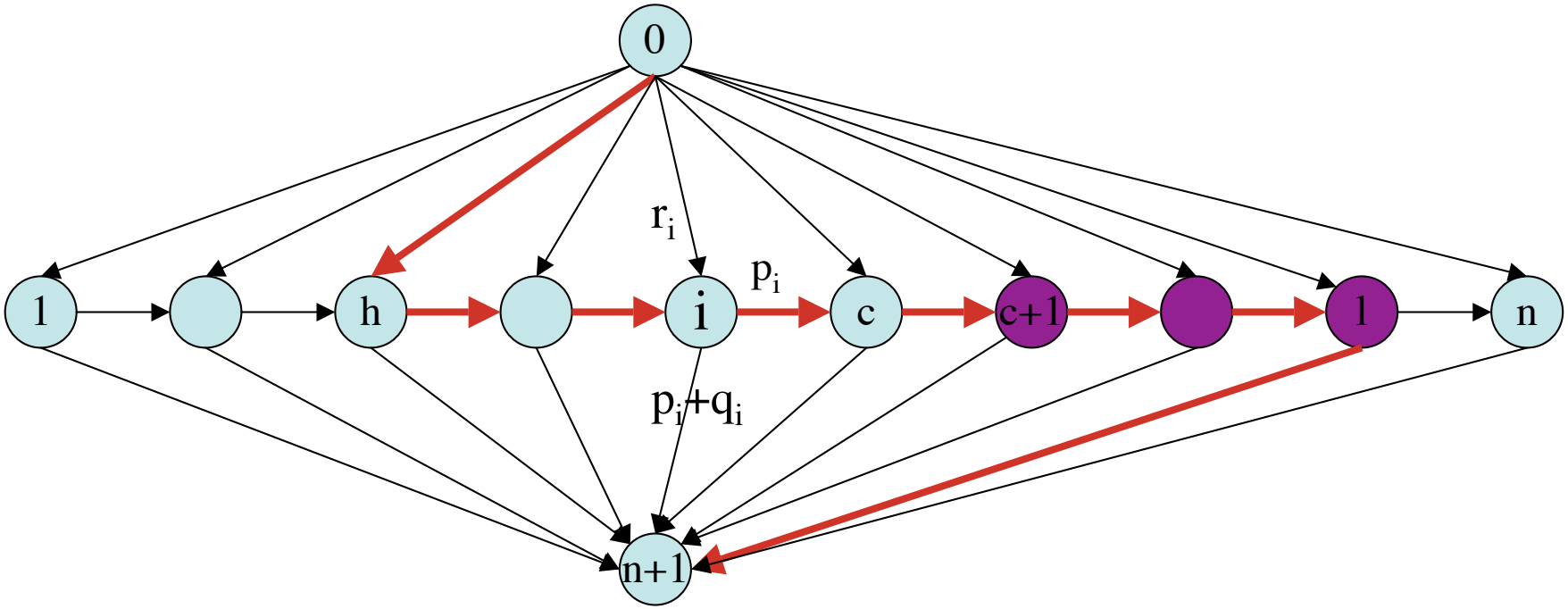
Soit O_0 la solution de Jackson et f_0 sa valeur.

1) On a : $f_0 - f^* \leq \text{Max}_J \{p_j\}$
2) Si O_0 n'est pas optimal, on peut déterminer une partie D de J et un job c dans J-D tels que : $f^* - h(D) < p_c$.

Preuve :

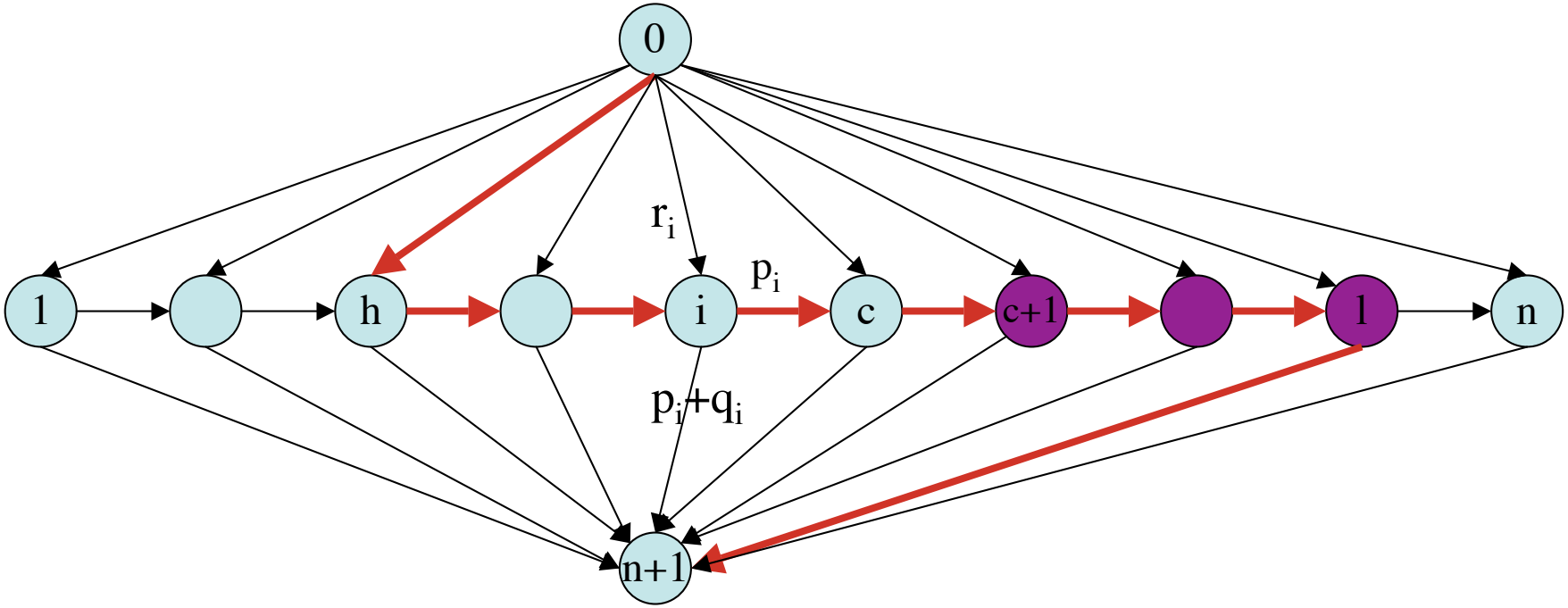
Supposons que le séquencement de O_0 soit $(J_1, J_2, \ldots, J_n)$.

O_0 est donc l'ordonnancement au plus tôt du graphe valué $G(0_0)$ suivant :



Définition d'un chemin critique « maximal » de $G(O_0)$.

l : plus grand indice par lequel passe un chemin critique.
h : plus petit indice tel que le chemin (0,h,h+1,…,l,n+1) est critique.
c : plus grand indice de h..l tel que $q_c < q_l$
(c = h-1 si cet indice n'existe pas)
D = {c+1..l} . (sommets violets)



Dans O_0, le slot $[r_h-1, r_h]$ est un temps mort.

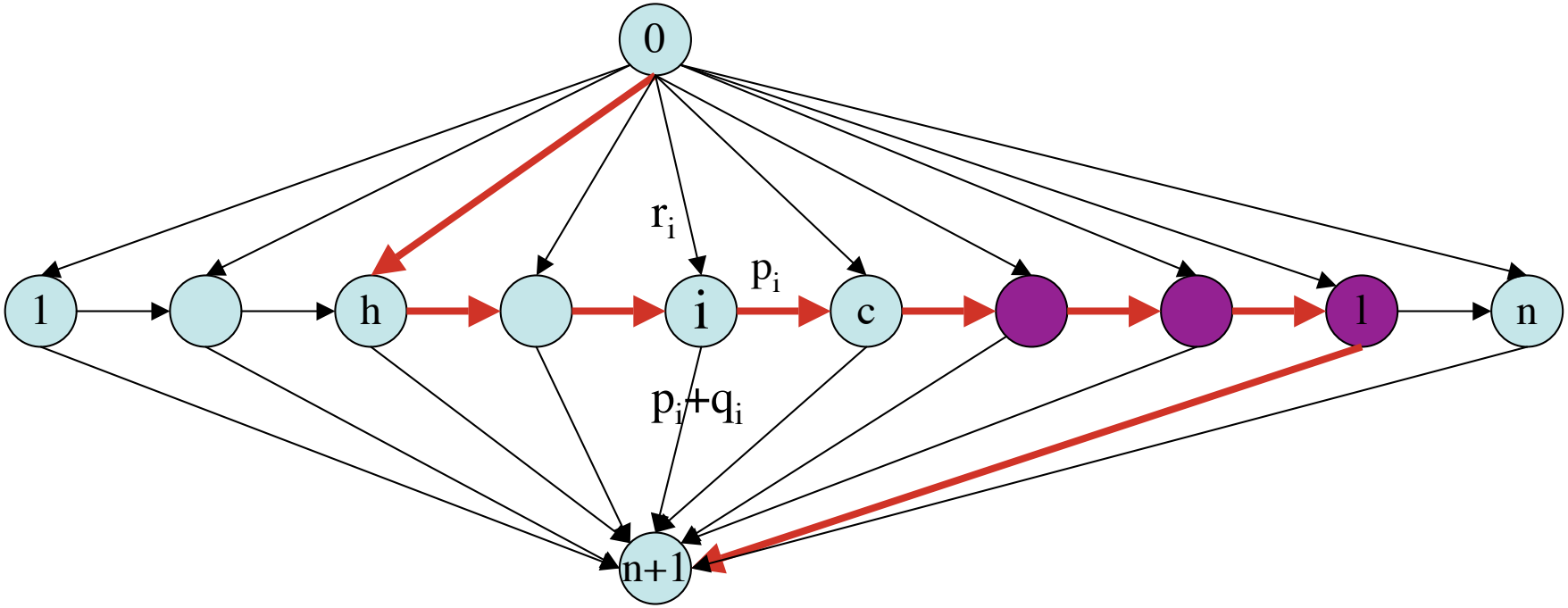
Sinon le job J_{h-1} en exécution sur ce slot se termine en r_h.

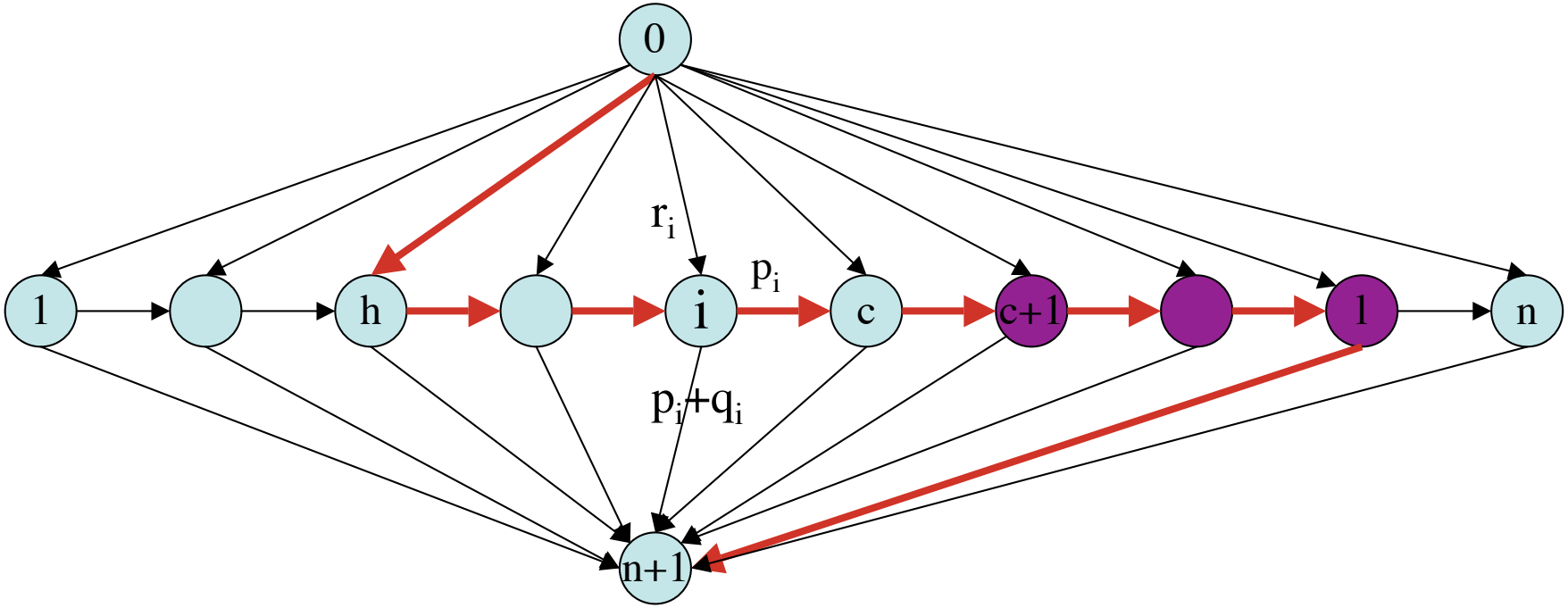
On a donc : $t_{h-1} + p_{h-1} + \sum_{h..l} p_i + q_l = r_h + \sum_{h..l} p_i + q_l = f_0$.

t_{h-1} est la longueur d'un plus long chemin de 0 à h-1.

Ce chemin concaténé au chemin (h,..,l,n+1) est critique.

Contradiction avec la définition de h.





On a : $r_h = \text{Min}_{h..l} \{r_i\}$.

Evident si h=1;

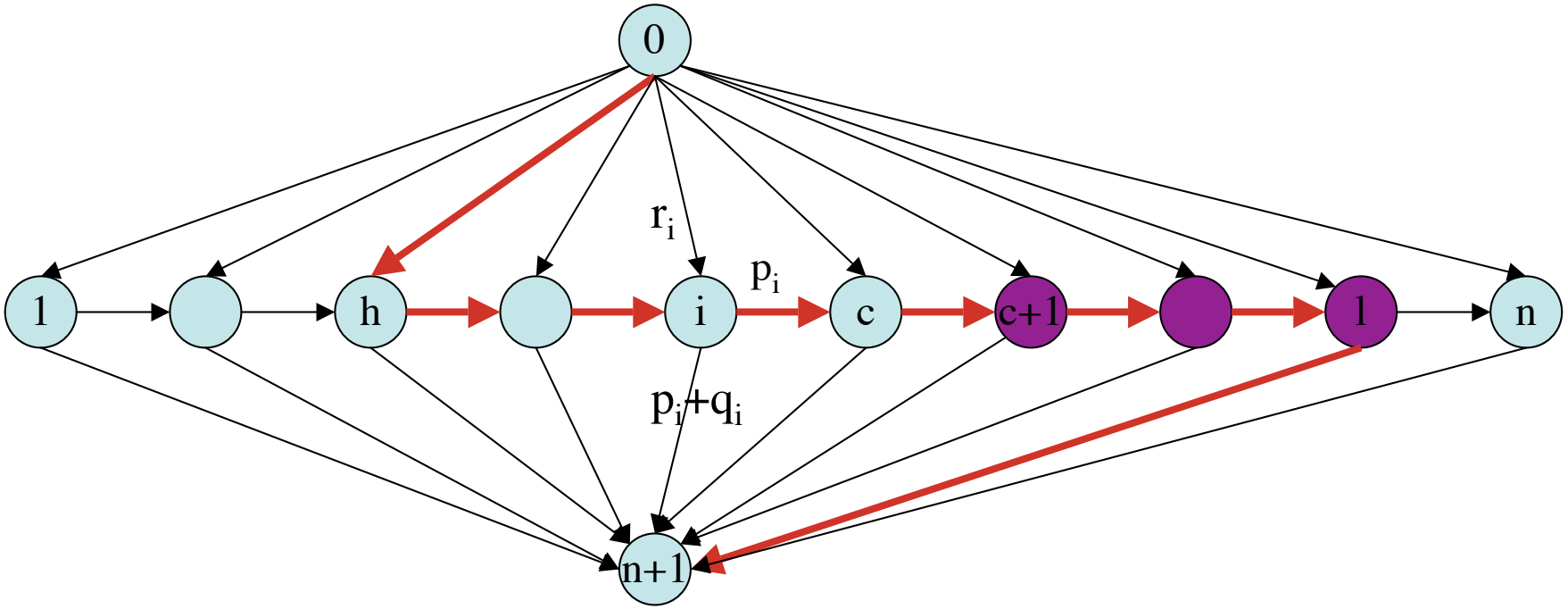
Sinon, l'intervalle $[t_{h-1}+p_{h-1}, t_h]$ est un temps mort de O_0 ;

Donc , lors de la terminaison de J_{h-1}, il vient :

$t_h=r_h=\text{Min}_{h..n} \{r_i\}$ et donc $r_h = \text{Min}_{h..l} \{r_i\}$.

Supposons $q_l = \text{Min}_{h..l} \{q_i\}$.
On a alors D = h..l ;
O_0 est alors optimal car $f_0 = h(D)$

Supposons $q_l \neq \text{Min}_{h..l} \{q_i\}$.
D'après la définition de c et D :
(1) $q_l = \text{Min}_D \{q_i\}$;
(2) $q_c < q_l$.



Il en résulte que pour tout j dans D : $q_j > q_c$.
Montrons que pour tout j dans D : $r_j > t_c$.
Supposons j dans D tel que $t_c \geq r_j$;
A la date t_c, j est disponible et $q_j > q_c$. Contradiction.

On a donc : $t_c < \text{Min}_D \{r_j\}$.

Comme $t_c = r_h + p_h + \ldots + p_{c-1}$ et $q_l = \text{Min}_D \{q_j\}$:

$$h(D) = \text{Min}_D \{r_j\} + \sum_D p_j + \text{Min}_D\, q_j$$

$$> t_c + \sum_D p_j + q_l = r_h + \sum_{h..l} p_j - p_c + q_l = f_0 - p_c.$$

D'où : $h(D) > f_0 - p_c$.

Conclusion :

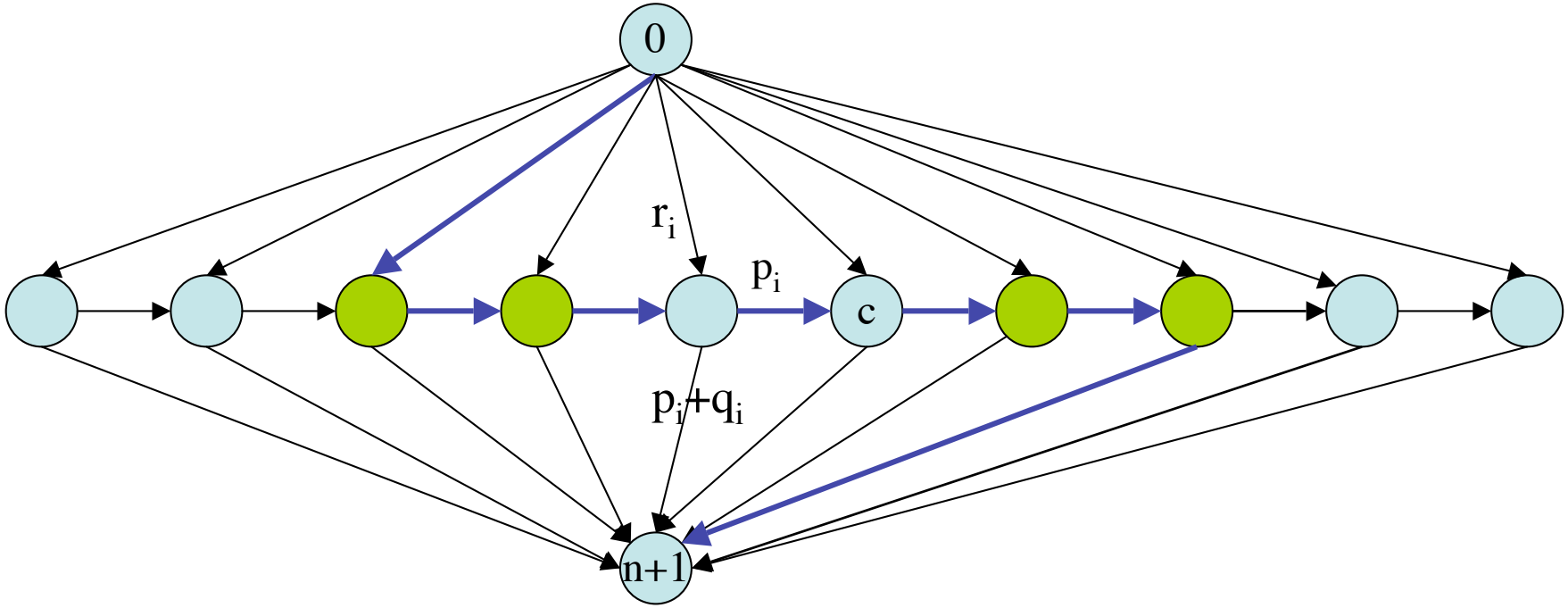
Le couple (c,D) associé à la solution de Jackson O_0 satisfait :

$f_0 - h(D) \leq p_c - 1$.

Il en résulte que dans une solution optimale :

- soit le job c est exécuté avant tous les jobs de D
- soit le job c est exécuté après tous les jobs de D.

Dans le cas contraire, il existerait dans le graphe G(O*)
un chemin de 0 à n+1 de longueur $\geq h(D)+p_c > f^*$.



Graphe G(O*) associé au séquencement optimal.
Paire (c,D) associée à O_0 .
Sommets de D en vert.
Hypothèse : c exécuté ni avant D ni après D ;
Longueur du chemin bleu ≥ h(D) + p_c > f_0 ≥ f*. Contradiction

Propriété (Calcul de Max $I \subset J$ {h(I)}).
Soit E' l'énoncé associé à E pour lequel la préemption est autorisée.
$\text{Max}_{I \subset J}$ {h(I)} est la valeur du makespan de la solution fournie par l'algorithme de Jackson préemptif pour l'énoncé E'.

Propriété :
L'algorithme de Jackson préemptif calcule un ordonnancement optimal pour le problème $1 / r_j, q_j, \text{pmtn} / C_{max}$.

Règle simple de l'algorithme de Jackson préemptif :
A chaque fin de job ou nouvelle date disponibilité on exécute le job disponible de latence maximale.

Algorithme de Jackson préemptif :

Algorithme B6(E') ;

t := $Min_J\{r_j\}$; ReadyJobs:={j∈J/r_j=t}; U:=J ;

Tantque U ≠ ∅ faire

  Si ReadyJobs(t) ≠ ∅,

    alors Choisir J_i de latence maximum dans ReadyJobs;

      Déterminer le plus petit instant u (>t) tel que :

        - soit 'J_i est terminé à u'

        - soit 'u est une date de disponibilité' ;

      O.[t,u] := J_i ;

      Si J_i est terminé à u alors U := U-{J_i} ;

    sinon Soit u := Min(r_j / j∈U et r_j > t)

      O.[t,u] := ∅ ;

  FinSi

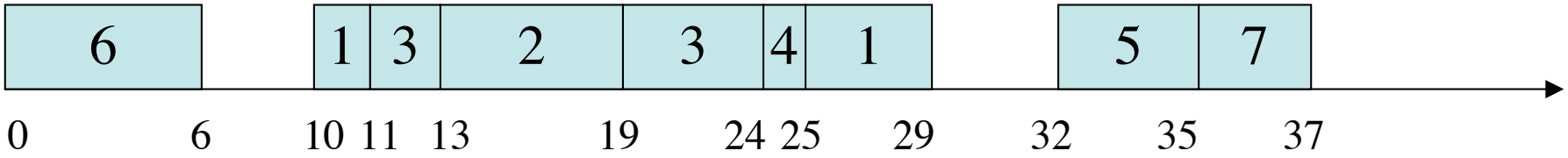
t := u ;

FinTantque ;

Retourner(O).

Exemple :

j	1	2	3	4	5	6	7
r_j	10	13	11	20	32	0	32
p_j	5	6	7	1	3	6	2
q_j	7	26	24	21	8	17	0



Algorithme Branch & Bound pour $1 / r_j , q_j / C_{max}$

Chaque sommet S de l'arborescence de recherche correspond à un énoncé de « $1 / r_j , q_j / C_{max}$ » que l'on note E(S)=(r,p,q).

On note (c(S),D(S)) le couple (c,D) associé à E(S).

La racine correspond à l'énoncé initial E^0.

La fonction d'évaluation par défaut est définie par :
$g(S) = Max_{I \subset J} \{h(I)\}$
Rappel : g(S) = B6(E(S)') (algorithme de Jackson préemptif)

Séparation d'un sommet S.

On sépare le sommet S en 2 nouveaux problèmes :

- le problème 'avant' pour lequel c(S) est exécuté avant tous les jobs de D(S) ;
- le problème 'après' pour lequel c(S) est exécuté après tous les jobs de D(S) ;

Le problème 'avant' est associé au nœud S^{av} et défini par le triplet (r,p,q') où :

- pour $j \neq c(S)$: $q'_j = q_j$;
- $q'_{c(S)} = \text{Max}\{q_{c(S)}, p(D(S))+\text{Min}_{D(S)}\{q_j\}$.

Le problème 'après' est associé au nœud S^{ap} et défini par le triplet (r',p,q) où :

- pour $j \neq c(S)$: $r'_j = r_j$;
- $r'_{c(S)} = \text{Max}\{r_{c(S)}, p(D(S))+\text{Min}_{D(S)}\{r_j\}$.

Soit ê la valeur de la meilleure solution connue.
Soit S(ê) l'ensemble des solutions de durée inférieure à ê.

Propriété :
Pour tout job J_k de J\D tel que $p_k > ê - h(D)$, alors dans toute solution de S(ê) :
- soit J_k est exécuté avant tous les jobs de D ;
- soit J_k est exécuté après tous les jobs de D.

Propriétés de dominance :
Soit $K = \{J_k \in J\text{-}D \,/\, p_k > ê - h(D)\}$ et $k \in K$.
Si $r_k + p(D) + p_k + p_l \geq ê$, alors dans toute solution de S(ê) ,
J_k est exécuté après tous les jobs de D ;
Si $\text{Min}_D\, r_j + p(D) + p_k + q_k \geq ê$, alors dans toute solution
S(ê), J_k est exécuté avant tous les jobs de D.