Deux modèles récents en Ordonnancement

Philippe Chrétienne

Université Paris 6

Laboratoire LIP6

Plan de l'exposé

- Ordonnancement cyclique
  - Le problème central cyclique
  - Le problème à m machines identiques
    - Résolution exacte
    - Résolution approchée
- Ordonnancement avec délais de communications
  - Modèle de base et variantes
  - Résultats de complexité
  - Résolution exacte
  - Résolution approchée

Partie I

Ordonnancement cyclique

Problème d'ordonnancement cyclique

n tâches «□génériques□» : I={1, 2, .., i,.., n}

Chaque tâche générique doit être exécutée une infinité de fois ;

On note <i,k> la tâche réelle associée à la $k^{ième}$ exécution de i ;

Toutes les exécutions <i,k> de i ont la même durée p_i ;

L' itération k est définie par {<1,k> , <2,k> , … , <n,k> }

q contraintes de précédence «génériques» : $A = \{a_1, \ldots, a_q\}$

a_k est un triplet (i,j,h) où $i \in I$, $j \in I$ et h est un entier.

La contrainte générique (i,j,h) représente le sous-ensemble infini de contraintes de précédence réelles :

$\forall n \geq 0$, <i,n> précède <j,n+h>

Hypothèse (non réentrance) :

$\forall i \in I$, la contrainte (i,i,1) existe.

Contraintes de ressource génériques :

R types de ressource (renouvelable).

Disponibilité en la ressource r : B_r

Chaque exécution <i,k> de la tâche générique i requiert a_{ir} unités de la ressource r, $1 \leq r \leq R$

Exemple : problème cyclique à m machines
$R = 1$; $B = m$; $a_i = 1$.

Ordonnancement :

Un ordonnancement S attribue une date s_{ik} à chaque tâche réelle <i,k> telle que :

1) toutes les précédences réelles soient satisfaites :

$$\forall (i,j,h) \in A \ \forall n \geq 0, \ s_{j,n+h} \geq s_{in} + p_i$$

2) Toutes les contraintes de ressource réelles soient satisfaites :

$$\forall r \in \{1..R\} \ \forall t \geq 0, \ \sum_{i \in A(t)} a_{ir} \leq B_r \ .$$

(A(t) : tâches génériques dont une exécution est active à t)

Ordonnancement périodique (de période r) :

$\forall i \in I \quad \forall k \geq 0, \; s_{i,k+1} = s_{ik} + r \; ;$

Un ordonnancement périodique est donc déterminé par :
les n dates s_{i1} et la période r.

Ordonnancement K-périodique (de période r) :

$\forall i \in I \quad \forall k \geq 0, \; s_{i,k+K} = s_{ik} + r \; ;$

Un ordonnancement K-périodique est donc déterminé par :
les nK dates $s_{i1}, \ldots, s_{iK}$, $i \in I$ et la période r.

Temps de cycle d'un ordonnancement S

Notons $C_k(S) = \max\{t_{ik}(S) + p_i \ / \ i \in I\}$
la date de fin de l'itération k.

Le temps de cycle moyen de S sur les k premières itérations est défini par $C_k(S)/k$.

Le temps de cycle de S est alors défini par :

$$\lambda(S) = \limsup_{k \to \infty} C_k(S)/k$$

Le temps de cycle minimum absolu de la donnée E du problème cyclique est défini par :

$$\lambda(E) = \inf_{S \in S(E)} \{\lambda(S)\}$$

(S(E) est l'ensemble des ordonnancements de E))

Problème d'optimisation (général):

Déterminer un ordonnancement S dont le temps de cycle
$\Omega(S)$ aussi proche que possible de $\Omega(E)$.

Sous-problèmes :

Déterminer un ordonnancement périodique de période minimale.
(Si S périodique de période r, $\Omega(S)=r$)

Déterminer un ordonnancement K-périodique de temps
de cycle minimal.
(Si S K-périodique de période r, $\Omega(S)=r/K$)

Le problème central cyclique

Définition :
Problème d'ordonnancement cyclique sans contraintes de ressources.

Exemple:
8 tâches génériques {1,2,….,8}
p_1=5, p_2=1, p_3=3, p_4=2, p_5=2, p_6=1, p_7=3, p_8=2 ;

10 contraintes de précédence génériques :
{(5,1,0), (1,2,0), (2,3,1), (3,4,0), (4,5,1), (5,6,1), (5,7,0), (6,7,0), (7,8,0), (8,6,1)}.

On peut « représenter » une donnée du PCC par :

son graphe réduit (fini) R :

sommets : I,

arcs : A,

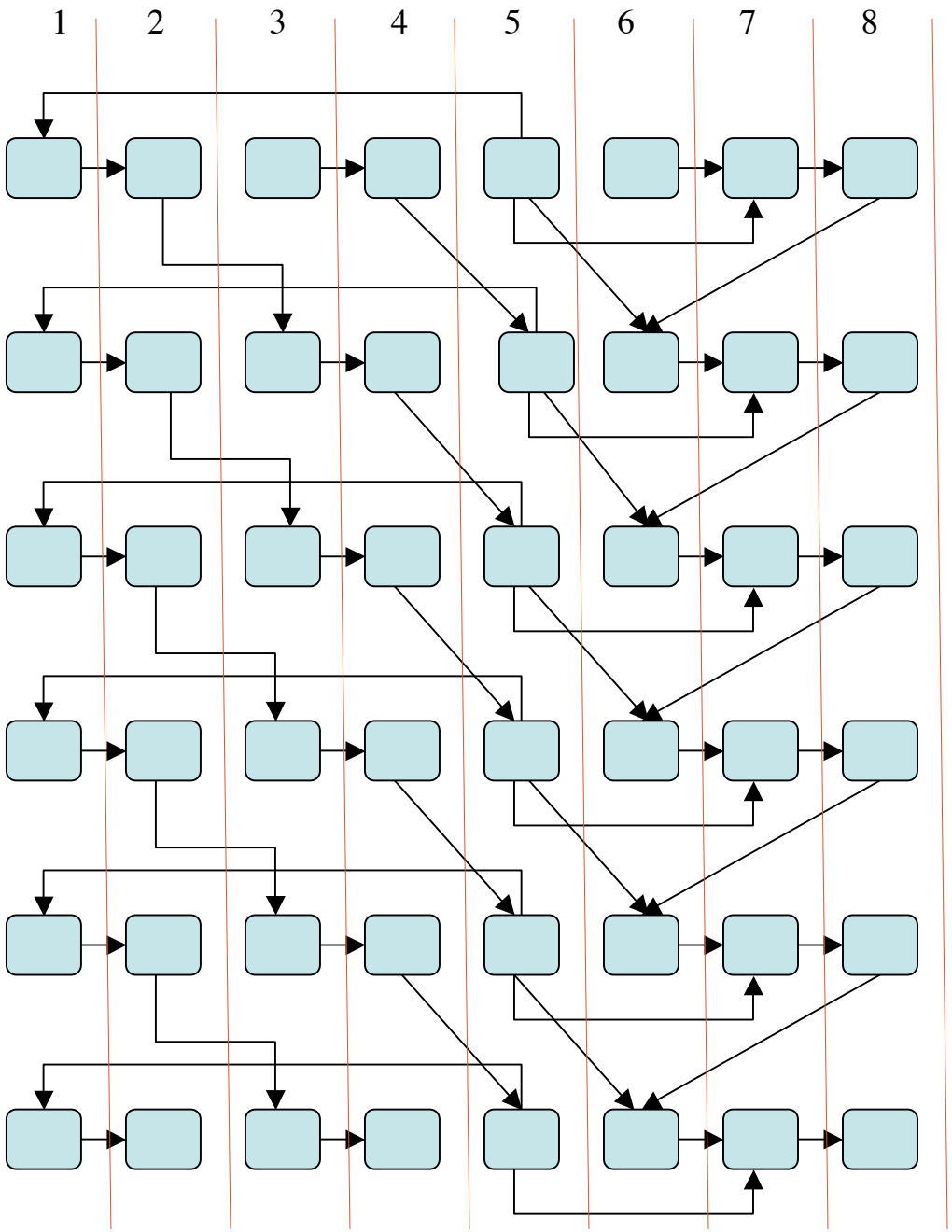
valuations de l'arc (i,j,h) : (p_i,h).

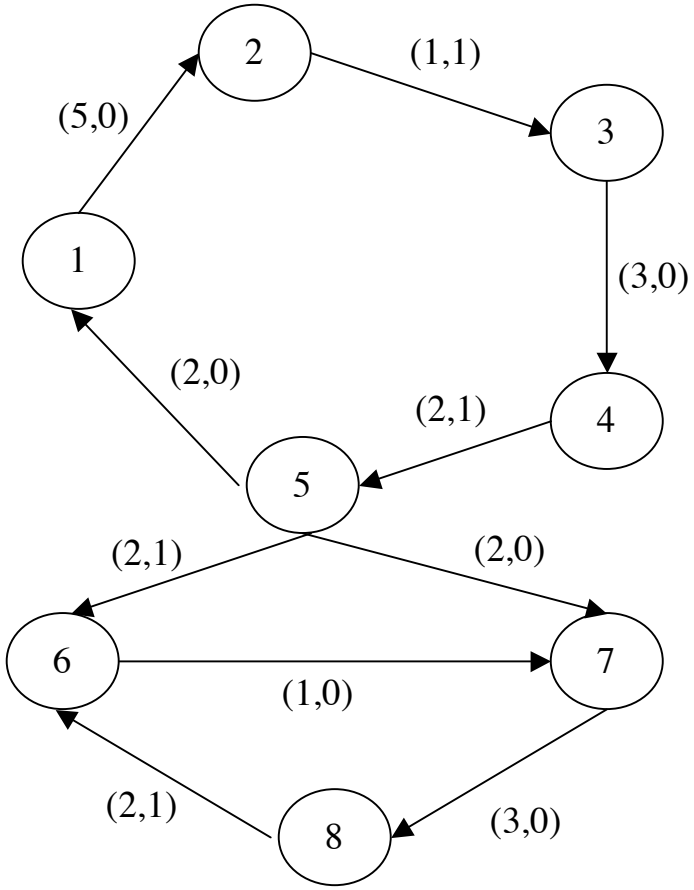
son graphe développé (infini) D :

sommets : tâches réelles,

arcs : précédences réelles,

valuation : durée de l'extrémité initiale.


1
2
3
4
5
6
7
8


1
2
3
4
5
6
7
8
(5,0)
(1,1)
(3,0)
(2,0)
(2,1)
(2,1)
(2,0)
(1,0)
(2,1)
(3,0)

Deux types de résultats:

1) Ordonnancement périodique optimal **[Ramchandani 81]**.

2) Analyse de l'ordonnancement au plus tôt **[Chrétienne 83]**.

CNS d'existence d'un ordonnancement :

Pas de circuit dans le graphe développé, c'est-à-dire pas de circuit de hauteur nulle dans le graphe réduit R.

Période d'un circuit :

Si γ est un circuit de R, sa période $\theta(\gamma)$ est définie par $P(\gamma)/H(\gamma)$ où $P(\gamma) = \sum_{a \text{ arc de } \gamma} p(a)$ et $H(\gamma) = \sum_{a \text{ arc de } \gamma} h(a)$.

Soit $\theta_R = \max \{P(\gamma)/H(\gamma) \mid \gamma \text{ circuit élémentaire de R}\}$ la période maximale d'un circuit de R.

Un circuit γ est dit critique si $\theta(\gamma) = \theta_R$

Remarque :
Différences par rapport au problème central classique :
-les circuits de R «remplacent» les chemins ;
-la période d'un circuit «remplace» la durée d'un chemin.

Structure de l'ordonnancement au plus tôt a(i,k).

1) R fortement connexe.
Pour tout i de I, la suite a(i,k) est K-périodique de période λ_R.

2) R quelconque.

Soient R(1), … , R(Q) les composantes fortement connexes de R ;

Soient $\lambda_{R(1)}$, …. , $\lambda_{R(Q)}$ les périodes maximales des graphes R(j) ;

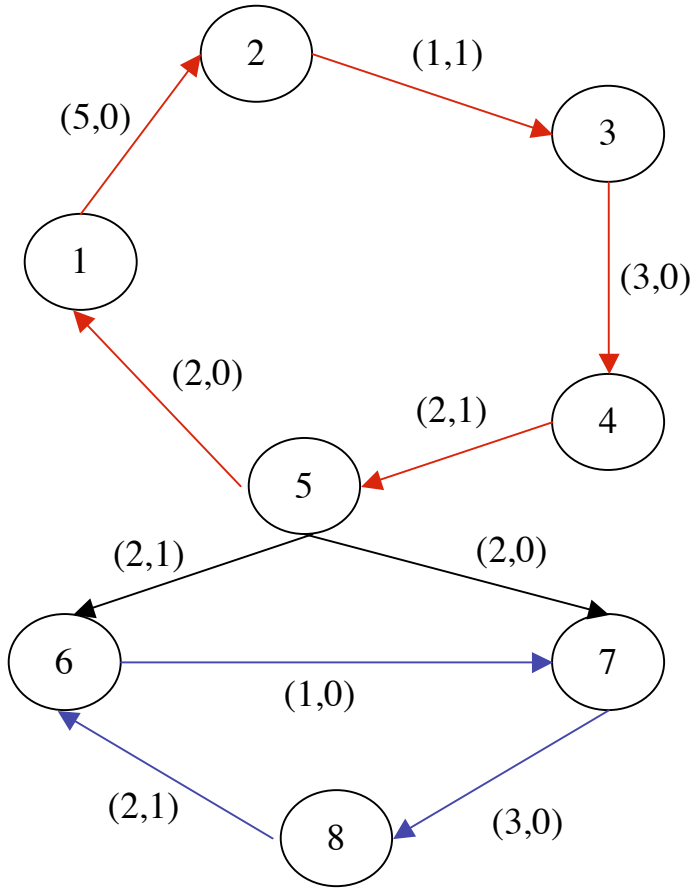
Pour tout i de R(j), la suite a(i,k) est K-périodique
de période $\max\{\lambda_{R(s)} / R(s) \text{ est ascendant de } R(j)\}$

Exemple :

Q=2.

$R_1 = \{1,2,3,4,5\}$
$\lambda_1 = 6.5$

$R_2 = \{6,7,8\}$
$\lambda_2 = 6$



Toutes les suites a(i,k) sont de période 6.5.

Existence d'un ordonnancement périodique optimal.

Soit S un ordonnancement périodique caractérisé par :
- les dates $u_1, u_2, \ldots, u_n$ des exécutions de la 1ère itération ;
- une période r .

Propriété :
S satisfait la contrainte générique (i,j,h) si : $u_j - u_i \geq p_i - rh$

Corollaire 1:
S est réalisable si u est un potentiel de R pour la valuation $p_i - rh$ de l'arc de R associé à la contrainte (i, j,h).

Corollaire 2 :
La valeur minimale de r est égale à $\square_R$.

Le problème cyclique à m machines identiques

Définition :
n tâches génériques ;
q contraintes de précédence génériques ;
m machines identiques.

Chaque <i,k> est exécutée sur une machine ;

Déterminer un ordonnancement de temps de cycle minimum.

Résolution exacte [Hanen-Munier 1991]

Prop 1 : Les ordonnancements K-périodiques sont dominants.
Question ouverte : peut-on borner K polynomialement ?

Prop 2 : Les ordonnancements périodiques ne sont pas dominants.

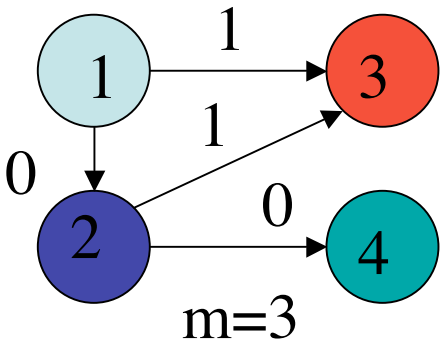
Prop 3 : $\square = \max\{(1/m)\Sigma p_i , \square_R\}$ est une borne inférieure du temps de cycle minimum absolu.

Prop 4 : Le problème est NP-difficile.

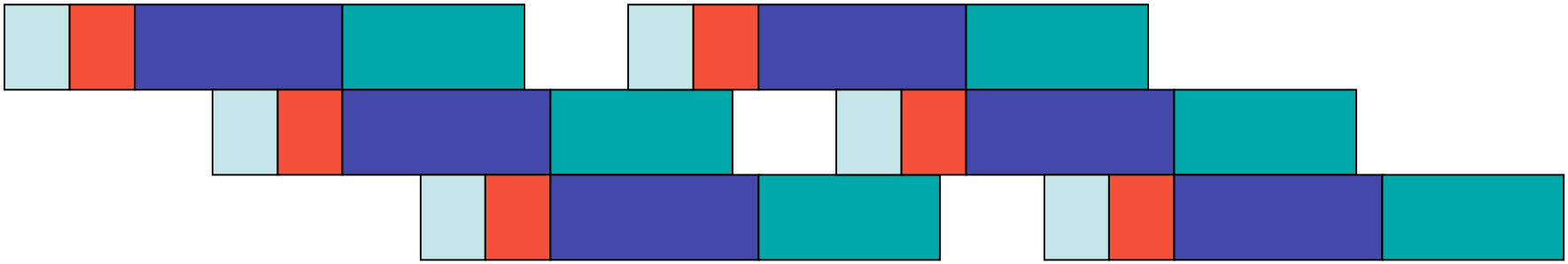
Prop 5 : Si R est un circuit, le problème est polynomial .
Il existe un ordonnancement optimal de période $\square$.

Prop 6 : Si R ne possède pas de circuit, alors le problème est polynomial. Il existe un ordonnancement de période $\square$.

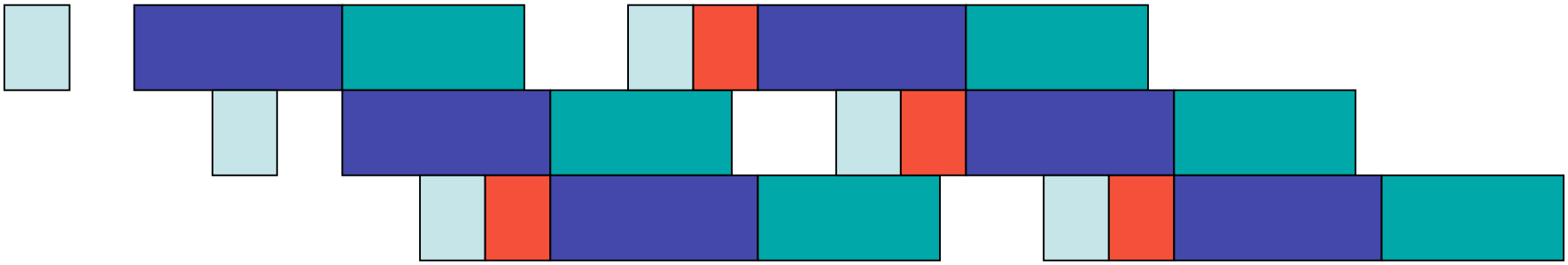
Schéma de preuve de Prop 6.



Solution initiale (en général non réalisable) : l'itération k est exécutée sur la machine (k-1) mod m +1 à la date (k-1) □.



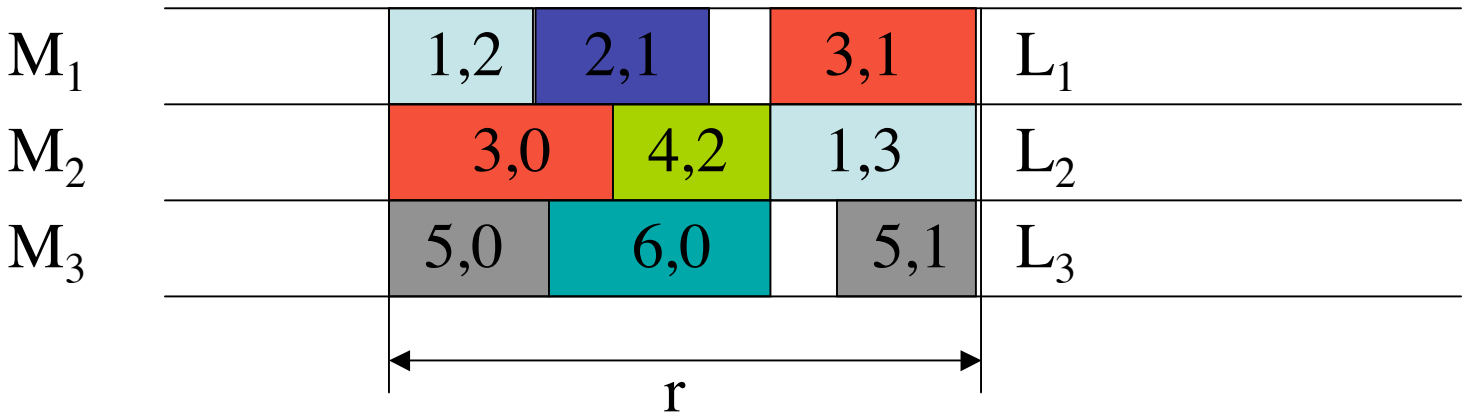
(2,3,1) n'est pas respectée. On supprime les 2 premières exécutions de 3 .



Ordonnancements périodiques

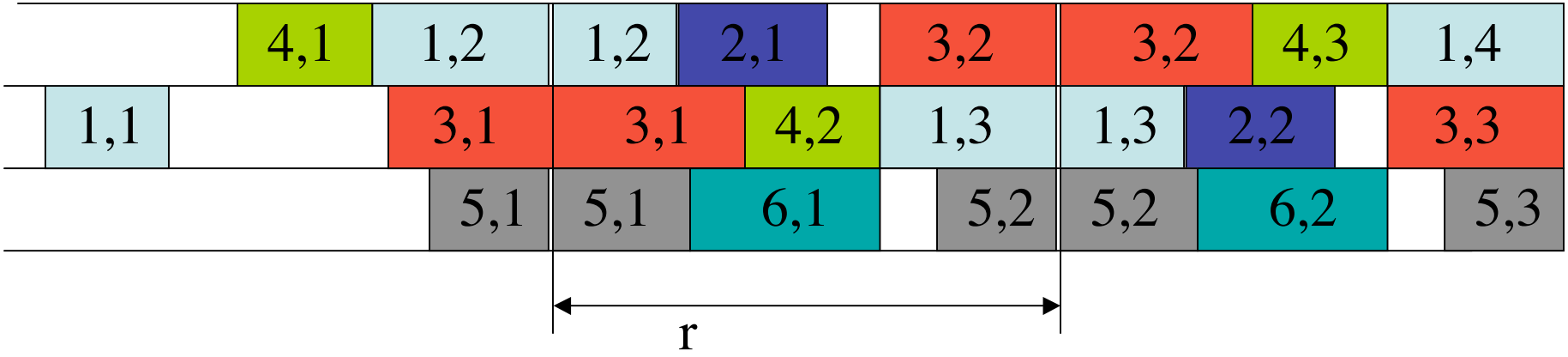
Prop 7 : Les ordonnancements périodiques à allocation périodique de ressource sont dominants.

Soit un motif d'un ordonnancement périodique S de période r
(i.e: «tranche» de durée r du régime permanent)



Donnée : n=6 , m=3,
(5,6,0), (6,1,2), (1,5,0),
(1,4,0), (4,2,0), (2,3,1),
(3,4,1))

2 types de tâches dans le motif :
Les tâches coupées : {1,3,5}
Les tâches entières : {2,4,6}



Pour assurer la continuité d'exécution des tâches coupées, il faut pour les périodes suivante et précédente (en ajustant les numéros d'occurrence des tâches) que :

$L_1 \square L_2$ $L_2 \square L_1$ $L_3 \square L_3$

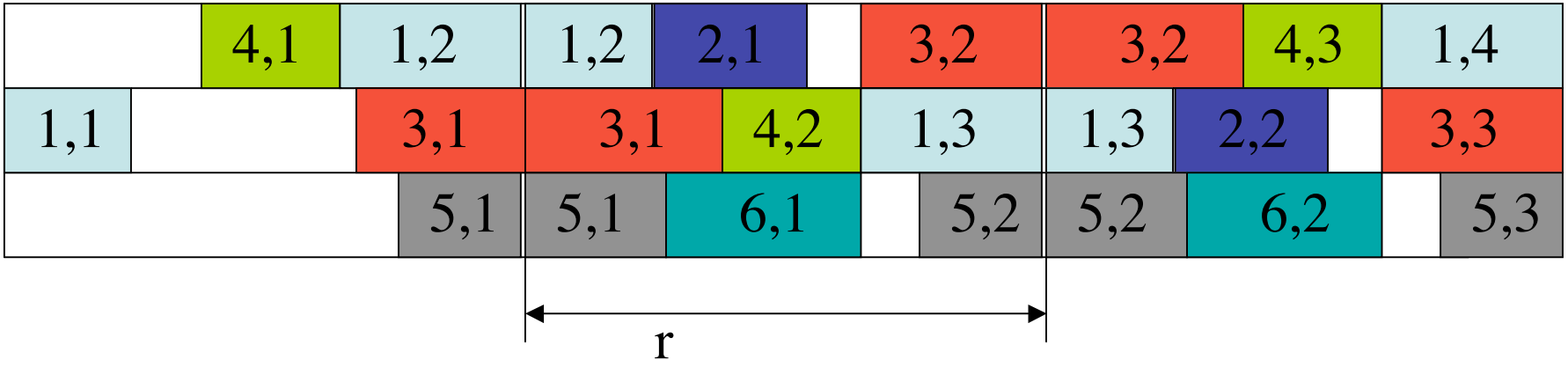
On obtient ainsi un ordonnancement de période r dont l'allocation de ressource est périodique :

m(i,k+1)=m(i,k)mod 2 +1 pour i=1,2,3, 4

m(i,k)=3 pour i=5,6

Prop 8 : Les ordonnancements à allocation circulaire sont dominants

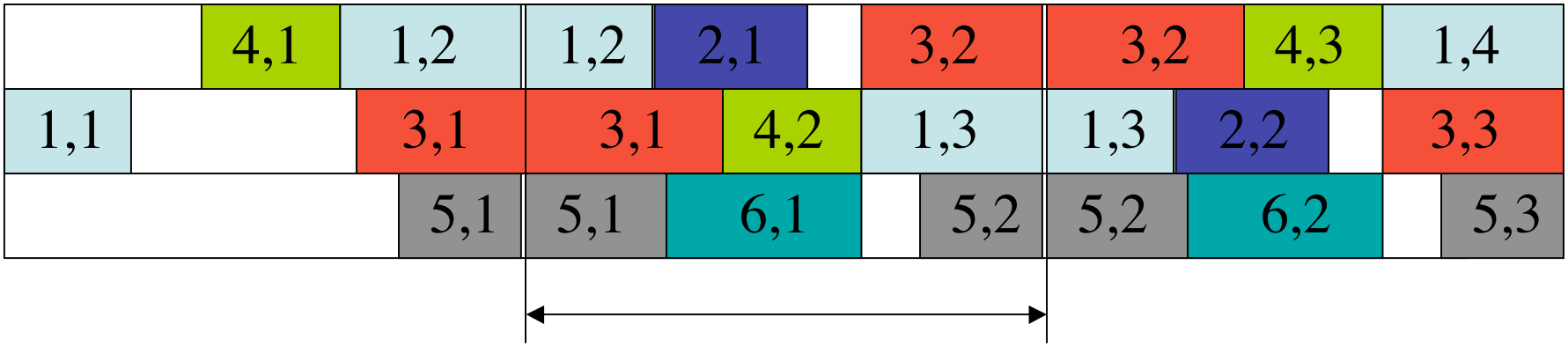
Graphe d'arbitrage d'un ordonnancement périodique à allocation périodique de ressource.



Soit S un ordonnancement périodique à allocation de ressource périodique ;

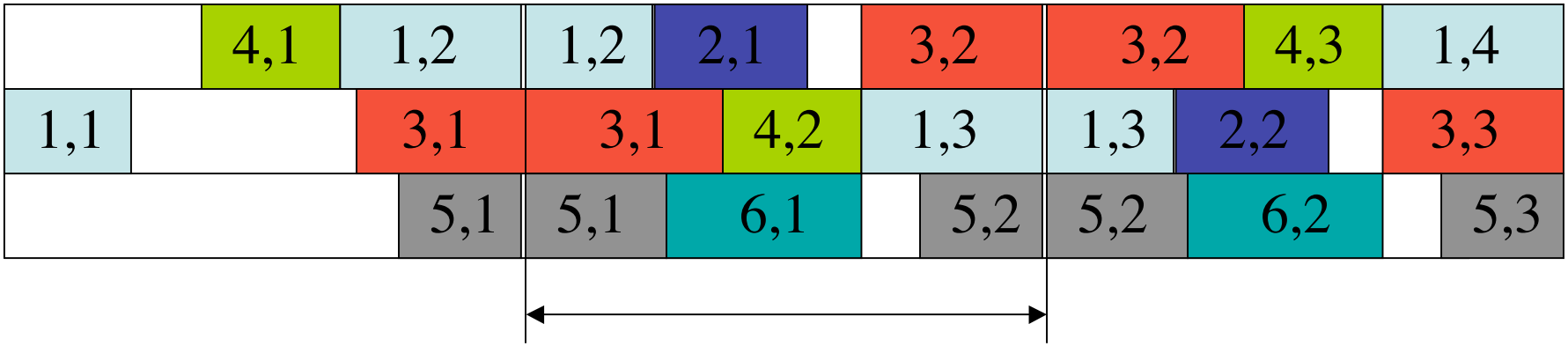
Soit C une orbite de longueur c de la fonction allocation de S (pour l'exemple : C = (1,2,1))

Soit K le sous-ensemble des k tâches génériques exécutées par les machines de C (pour l'exemple : K = {1,2,3,4})



Chaque machine de C exécute en régime permanent
une suite infinie d'exécutions du type :
$\langle j_1,h_1\rangle, \langle j_2,h_2\rangle, \ldots, \langle j_k,h_k\rangle; \langle j_1,h_1+c\rangle, \langle j_2,h_2+c\rangle, \ldots, \langle j_k,h_k+c\rangle;$
$\langle j_1,h_1+2c\rangle, \langle j_2,h_2+2c\rangle, \ldots, \langle j_k,h_k+2c\rangle; \ldots.$

Pour l'exemple :
Orbite 1 : <4,1>, <1,2>, <2,1>, <3,2>; <4,1+2>, <1,2+2>, …
Orbite 2 : <5,1>, <6,1>, <5,1+1>, <6,1+1>; ….



Pour chaque orbite C (longueur c), les contraintes de précédence :

$X(C) = \{(j_1,j_2,h_2-h_1),(j_2,j_3,h_3-h_2), \ldots , (j_k,j_1,h_1+c-h_k)\}$

sont satisfaites.

Pour l'exemple :

$X(C_1)=\{ (4,1,1), (1,2,-1), (2,3,1), (3,4,1)\}$

$X(C_2)=\{ (5,6,0), (6,5,1)\}$

Soit $X = \bigcup_C X(C)$ et $R(X)=(I, A_X)$

Les contraintes de X(C) forment un circuit de R(X) appelé circuit d'arbitrage ;
Les circuits d'arbitrage sont disjoints ;
Les circuits d'arbitrage couvrent les sommets de R(X) ;
La somme des hauteurs des circuits d'arbitrage est égale à m.

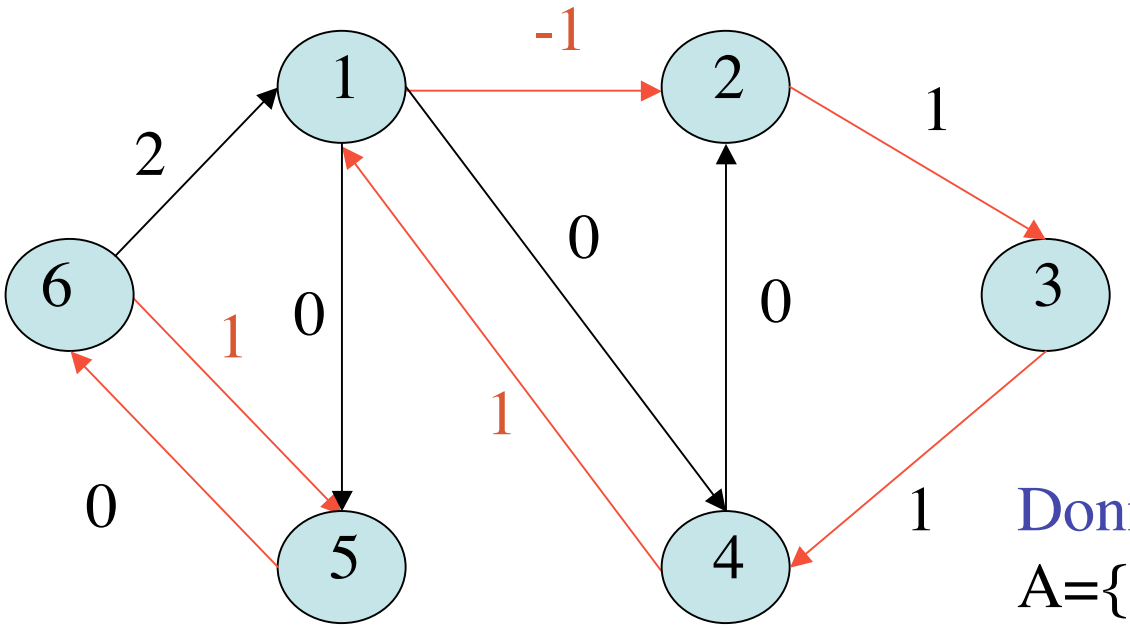
S est un ordonnancement de la donnée (I,A_X,m)
du problème cyclique

Comme tout ordonnancement de la donnée (I,A_X) du PCC satisfait la contrainte de ressource, l'ordonnancement S est dominé par la solution optimale de la donnée (I,A_X).

Pour l'exemple :

$X(C_1)$={ (4,1,1), (1,2,-1), (2,3,1), (3,4,1)}

$X(C_2)$={ (5,6,0), (6,5,1)}



Donnée : n=6 , m=3,
A={(5,6,0), (6,1,2), (1,5,0), (1,4,0), (4,2,0), (2,3,1), (3,4,1)}

Contraintes d'arbitrage de S non dans A :

X={(4,1,1), (1,2,-1), (6,5,1)}

Réciproquement, soit X une famille de contraintes (d'arbitrage) (i,j,h) telles que :
-Les contraintes de X forment des circuits disjoints ;
-Les sommets des circuits couvrent les sommets de R ;
-La somme des hauteurs des circuits est égale à m.

Tout ordonnancement de la donnée du PCC (I,A□ X) satisfait alors la contrainte de ressource.

Le problème initial peut donc se reformuler ainsi : déterminer une famille d'arbitrages X telle que la période maximale de la donnée du PCC (I,A□ X) soit minimale.

Résolution approchée

Ordonnancement périodique.

Un **algorithme approché** [Gaspéroni et al. 93] fournissant un ordonnancement de période r telle que :

$$r - r^* \leq (1 - 1/m)(r + p_{max} - 1)$$

r^* est la période optimale d'un ordonnancement périodique ;

r est la période minimale pour m=+ ($r = \square_R$);

p_{max} est la durée maximale d'une tâche.

Phase 1 :

Calcul d'un ordonnancement périodique optimal en relaxant la contrainte de ressource (PCC)

Résultat : u_i , $i \in I$ et r

Phase 2 : (on casse les circuits du graphe réduit)

Soient pour $i \in I$, $x_i = u_i \bmod r$ et $y_i = u_i \operatorname{div} r$;

Suppression dans le graphe réduit des arcs (i,j) tels que :

$$x_j < x_i + p_i ;$$

Le nouveau graphe G' est acyclique.

Phase 3 : (G' est ici le graphe de précédence d'un problème non cyclique)

Appliquer un algorithme de liste quelconque à G' ;

Résultat : ordonnancement v_i, $i \in I$ de durée π.

Phase 4 :(Définition d'un ordonnancement de période π du problème initial)

Poser : pour $i \in I$: $\sigma_i = v_i + y_i \pi$ et $r = \pi$.

L'ordonnancement périodique (π,r) est réalisable et satisfait :

$$r - r^* \leq (1 - 1/m)(r + p_{max} - 1)$$

Ordonnancement de liste [Chrétienne 93].

Un ordonnancement de liste est défini à partir d'une liste de priorité σ des tâches réelles.
$\sigma(k)$ est alors la tâche réelle de rang k dans σ.

Exemple :
σ peut être construit à partir d'une liste de priorité $(i_1, i_2, \ldots, i_n)$ des tâches génériques :
La liste de priorité des exécutions est alors :
$\langle i_1,1\rangle, \langle i_2,1\rangle, \ldots, \langle i_n,1\rangle ; \langle i_1,2\rangle, \langle i_2,2\rangle, \ldots, \langle i_n,2\rangle ; \ldots$

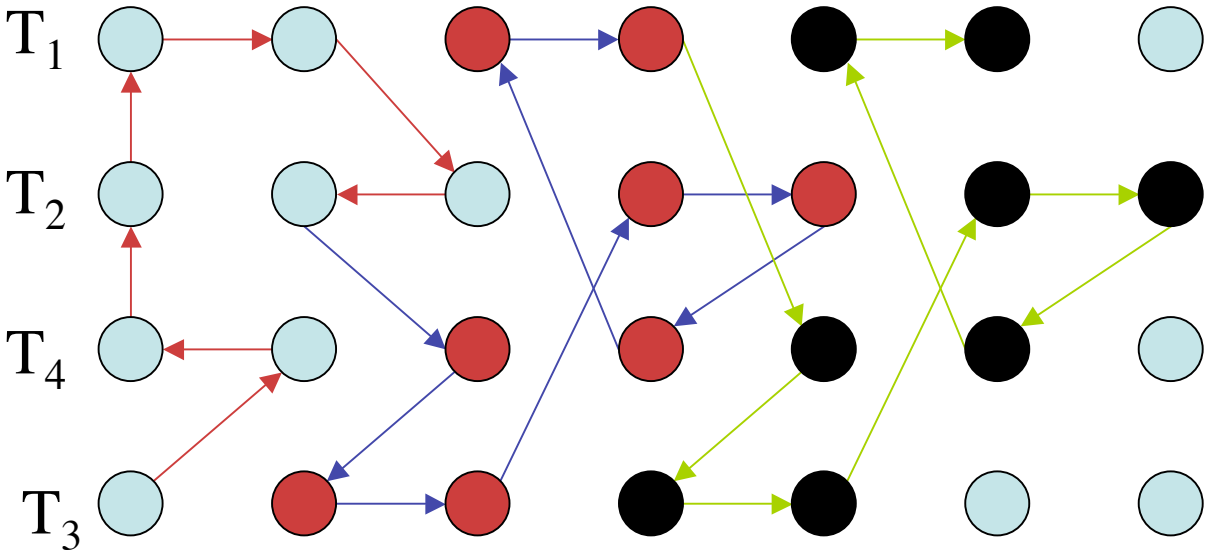
On note $S(\sigma)$ l'ordonnancement de liste obtenu en appliquant la liste σ au graphe développé de la donnée (I,A,p,m).

Prop1 (extension de la garantie de Graham) :
Un ordonnancement de liste S(□) de l'énoncé e=(I,A,p,m) satisfait : □(S(□)) ≤ (2-1/m) □(e)

Remarque : S(□) n'est pas forcément K-périodique.

Question : quel type de liste □ fournit des ordonnancements S(□) K-périodiques?

Réponse : les listes linéaires K-périodiques.



Prop 2 : Si Λ est une liste linéaire K-périodique, alors $S(\Lambda)$ est un ordonnancement de liste K-périodique.
Réciproquement, si S est un ordonnancement de liste K-périodique, il existe une liste linéaire K-périodique Λ telle que $S=S(\Lambda)$.

Propagation des garanties pour des problèmes spécifiques.

Soit un algorithme de liste fournissant la garantie ρ pour tout graphe d'un problème (non cyclique) à m machines.

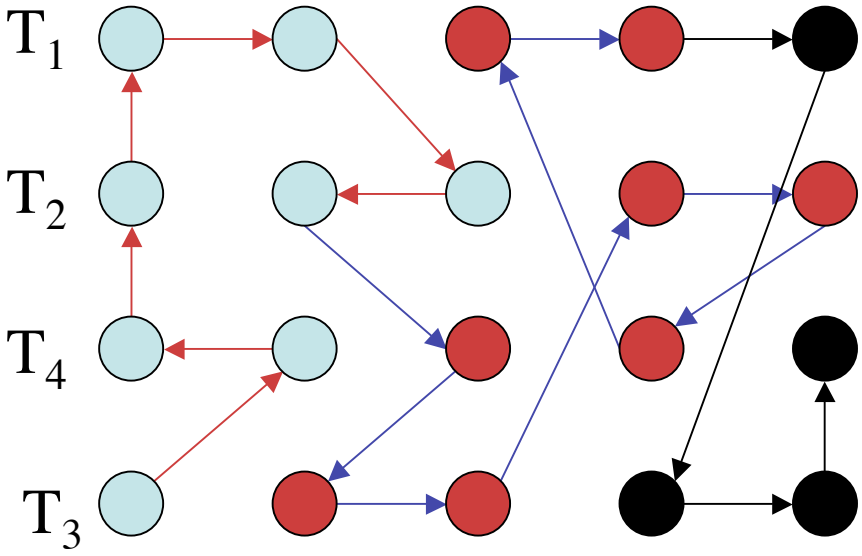
Soit G_k le sous-graphe des k premières itérations du graphe développé de la donnée e=(I,A,p,m) du problème cyclique.

Soit L_k la liste de priorité de G_k.

Prop 3 : Si la suite de listes (L_1,L_2, ….,L_k, ….) est quasi K-périodique, alors il existe une liste K-périodique □ telle que : □(S(□)) ≤ □□(e)

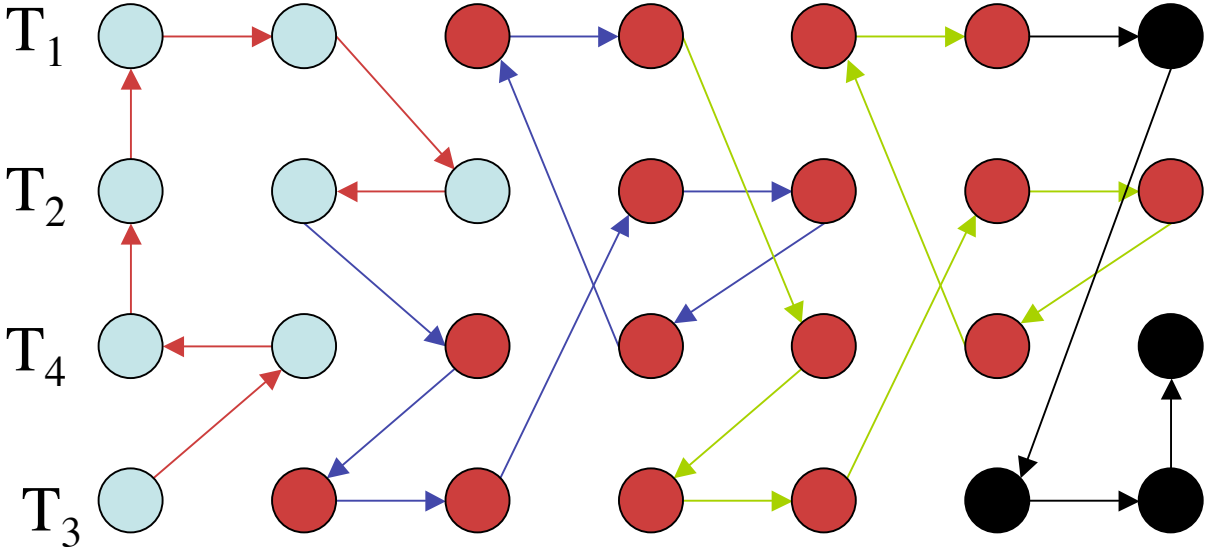
Application :
La suite (L_1,L_2, ….,L_k, ….) des listes de priorité de Coffman-Graham des graphes G_k est quasi K-périodique.

Donc, pour le problème cyclique à m machines et durées unitaires, l'algorithme de Coffman-Graham, étendu au cas cyclique, fournit la garantie (2-2/m) .



Suite $(L_1, L_2, \ldots, L_n, \ldots)$
quasi K-périodique

L_5



L_7

Partie II

Ordonnancement avec délais de communication

Plan

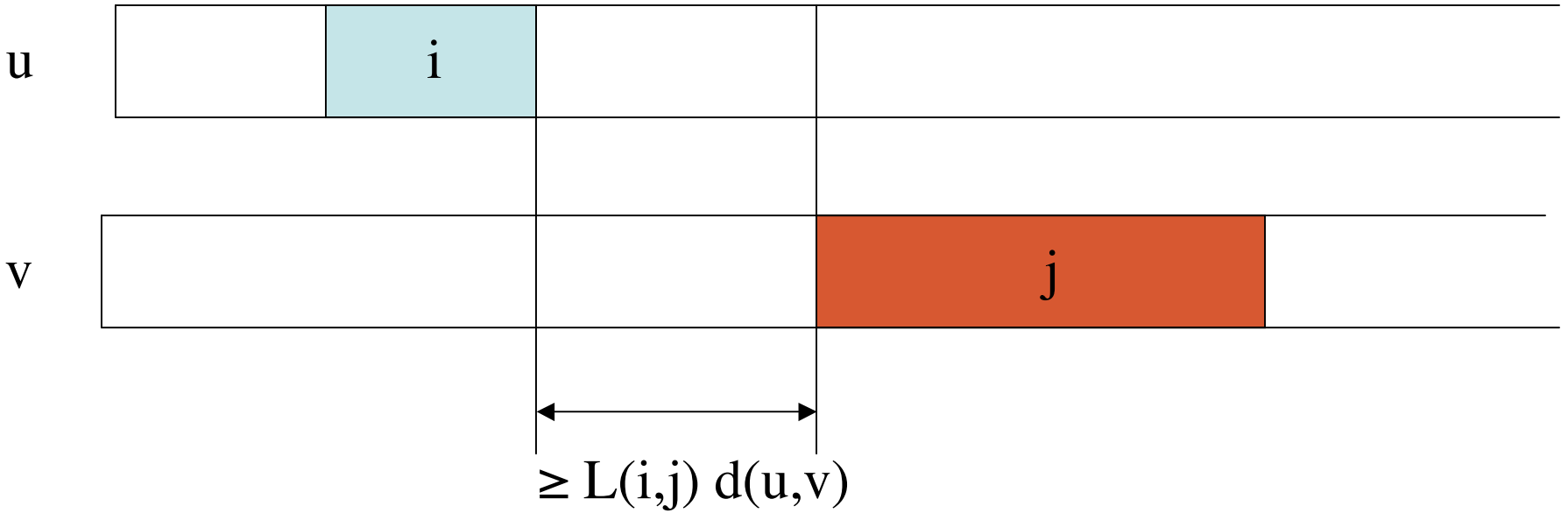
- Modèle de base et variantes
- Ressources non limitées et duplication
- Ressources non limitées sans duplication
- Ressources limitées sans duplication

Modèle de base

Les tâches sont exécutées sur les processeurs d'un réseau.
Une contrainte de précédence (i,j) correspond à un transfert de message de la tâche i à la tâche j.

On note M l'ensemble des processeurs (Card(M)=m)
Soit u, v $\in$ M.
d(u,v) est le temps de transfert (supposé connu et constant) d'un octet de u à v dans R ;
(hypothèse : $\forall$u $\in$ M, d(u,u)=0)

Soit L(i,j) la longueur du message envoyé par i à j.
Soit p(i) le processeur affecté à la tâche i ;
La durée c(i,j) du transfert (i,j) est égale à : L(i,j)$\times$ d(p(i),p(j))



Si S=(t,p) est un ordonnancement, la contrainte de précédence (i,j) s'exprime par : $t(j) \geq t(i) + p_i + L(i,j)\, d(u,v)$.

Cas particuliers :
RC : $d(u,v)=1 \ \forall u \neq v$ (réseau complet)
SCT : RC et $c_{max} \leq p_{min}$.
r.SCT : RC , $c_{max} \leq r\, p_{min}$ et $r < 1$.

Une hypothèse particulière : la duplication.

Pour limiter l'effet des délais de communication, on s'autorise à exécuter plusieurs copies d'une même tâche.

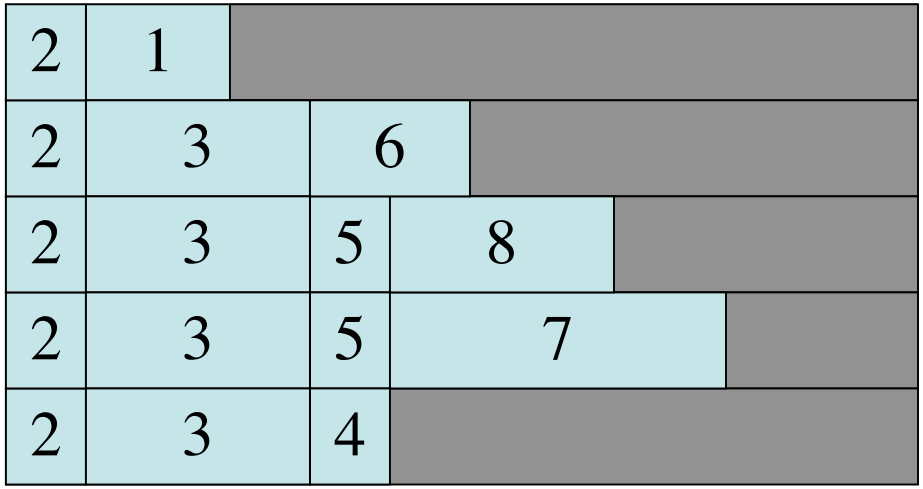
Une copie est un triplet c= <i,t,p> où :

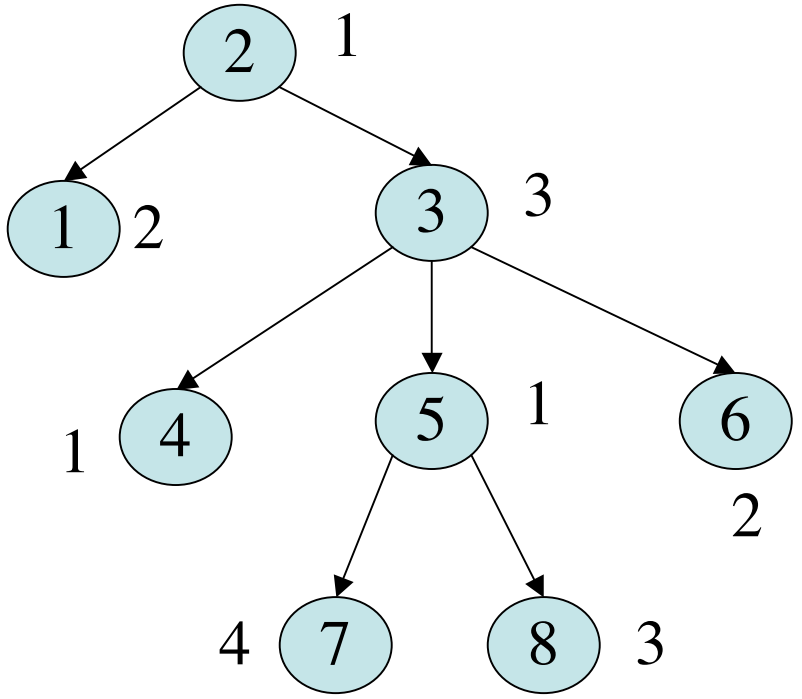
1) i est une tâche ,
2) t est la date d'exécution de c ,
3) p est le processeur affecté à c .

Un ordonnancement est une famille de copies telle que :

1) pour toute tâche i, il existe au moins une copie <i,t,p> ;
2) pour toute contrainte de précédence (i,j) :
   pour toute copie <j,u,q>
   il existe une copie <i,t,p> telle que :
   $u \geq t + p_i + c_{ij}$ si $p \neq q$
   $u \geq t + p_i$ si $p = q$

Un exemple :
arborescence, délais quelconques,
ressource non limitée : m ≥ n.





La durée de l'ordonnancement ne dépend pas des délais de communication

Complexité et résolution exacte

- Ressource non limitée et duplication
- Ressource non limitée sans duplication
- Ressource limitée avec duplication
- Ressource limitée sans duplication

Ressource non limitée et duplication

Un algorithme polynomial ($O(n^2)$) pour le cas SCT

[Colin & Chrétienne 91].

3 phases :

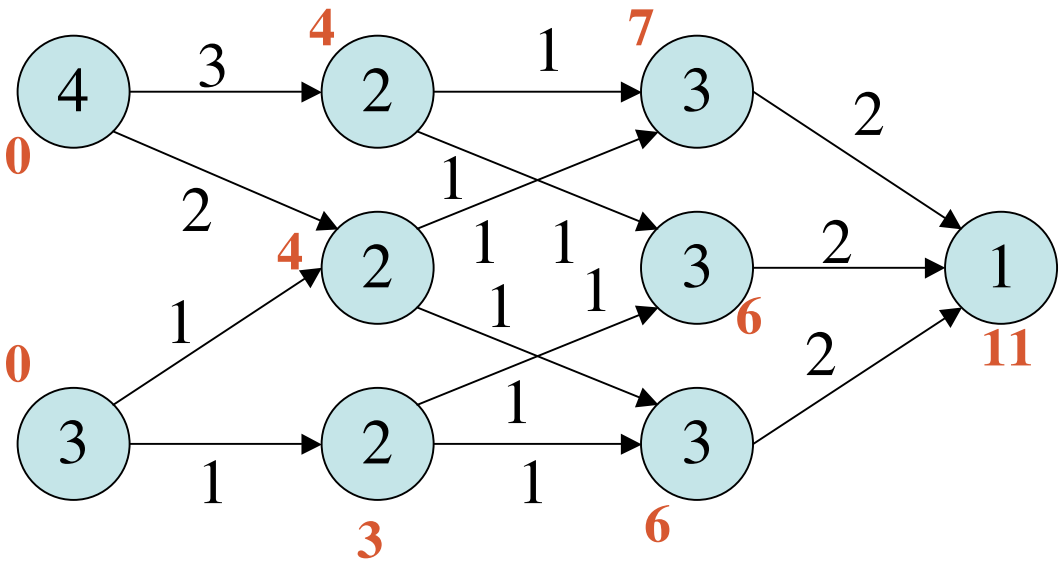
1) Calcul de r_i (date de disponibilité de toute copie de i) ;
2) Calcul d'un graphe critique ;
3) Ordonnancement au plus tôt du graphe critique.

Pour la phase 1 (parcours topologique) :

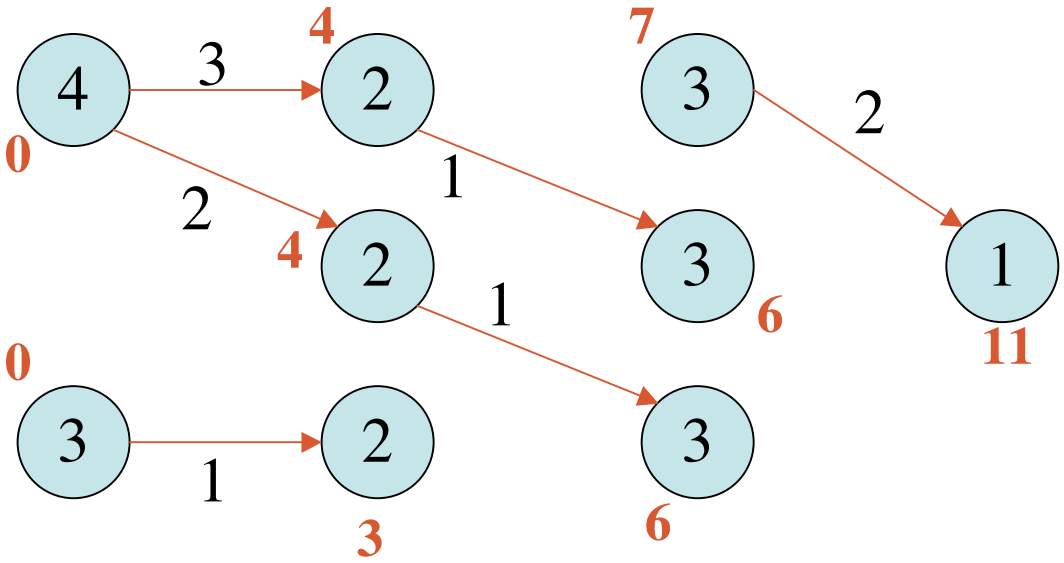
si $Pred(j)=\varnothing$ alors $r_j = 0$;

sinon, soit $s \in Pred(j)$ tel que $r_s + p_s + c_{sj}$ est maximum

$r_j = \max\{r_s + p_s, \max_{i \in Pred(j)-\{s\}} r_i + p_i + c_{ij}\}$



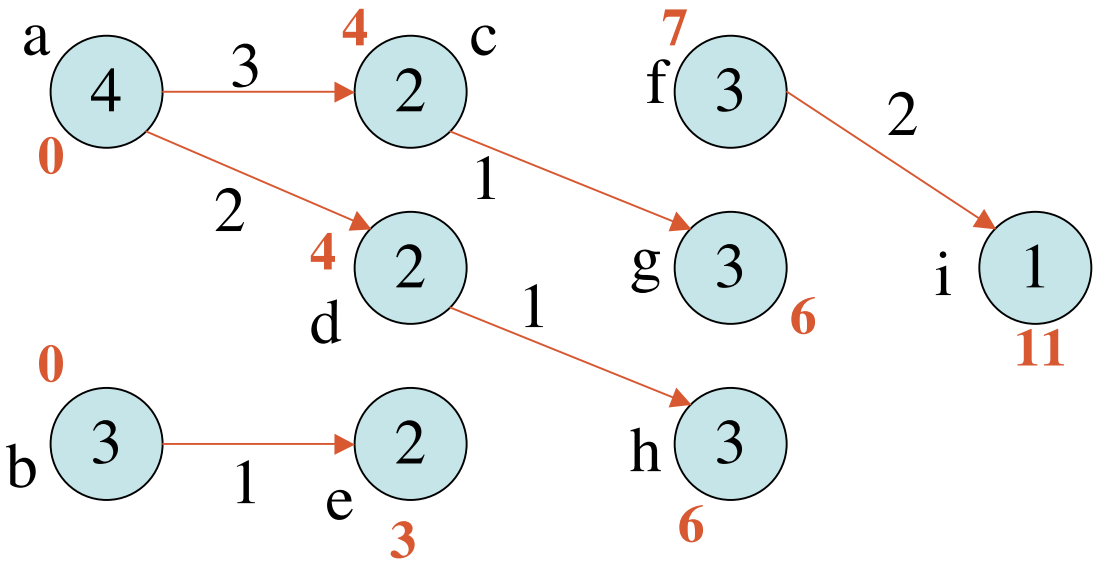
Les r_i en rouge

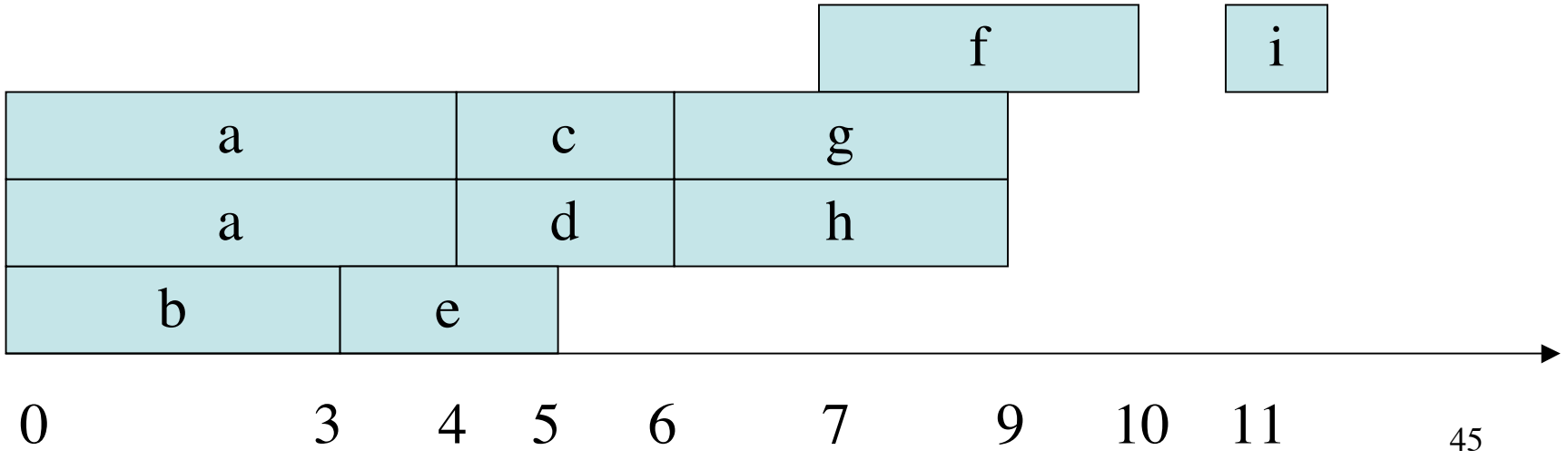


Arc critique (i,j) :

$r_i+p_i+c_{ij} > r_j$.

 forêt critique


a
4
0
3
4
2
c
2
4
2
d
1
1
7
f
3
2
g
3
6
i
1
11
0
b
3
1
2
e
3
h
3
6


f
i
a
c
g
a
d
h
b
e
0
3
4
5
6
7
9
10
11

Complexité [Papadimitriou et al. 90].

L'absence de contrainte de ressource ne suffit pas pour rendre le problème facile , en effet :

$P \mid prec, p_j=1, c>1, dup \mid C_{max}$ est NP-complet au sens fort.

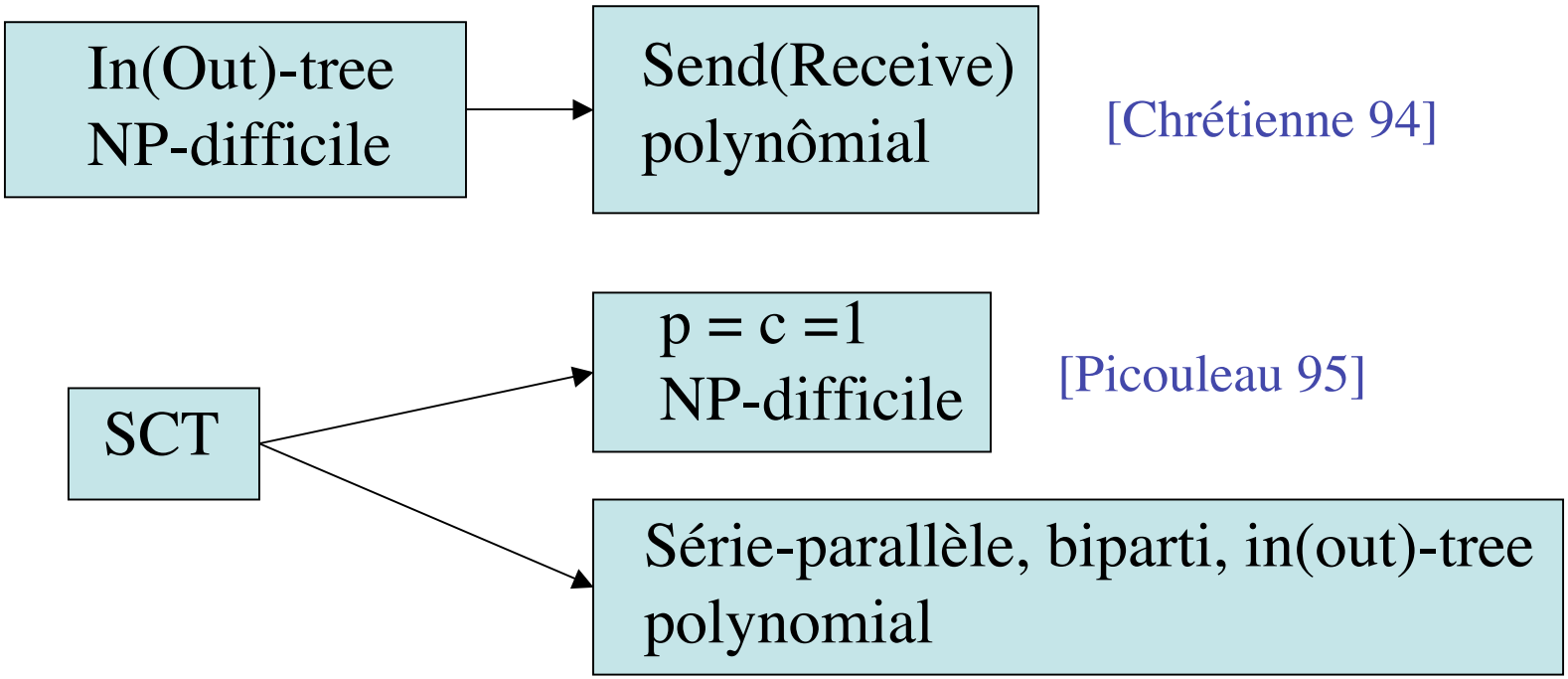
(réduction de CLIQUE)

Remarque :

Comme la preuve n'utilise pas la duplication, ce résultat implique que :

$P \mid prec, p_j=1, c>1 \mid C_{max}$ est NP-complet au sens fort.

Ressources non limitées sans duplication



[Chrétienne94] [Picouleau 95]

Ressource limitée sans duplication

$P/prec,p_j=1,c_{ij}=1/C_{max}$ est NP-complet. [Rayward-Smith 87]

$P/biparti,p_j=1,c_{ij}=1/C_{max} \leq 4$ est NP-complet. [Hoogeveen et al 92]
Pas de garantie < 5/4 en temps polynomial si P≠NP.

$P/tree,p_j=1,c_{ij}=1/C_{max}$ est NP-complet. [Veltman 93].

$Pm/tree,p_j=1,c_{ij}=1/C_{max}$ est polynomial. [Varvarigou et al. 93]

$P2(preaff)/prec,p_j=1,c_{ij}=1/C_{max}$ est NP-complet. [Veltman 93]

$P2/prec,p_j=1,c_{ij}=1/C_{max}$ est ouvert

Résolution approchée

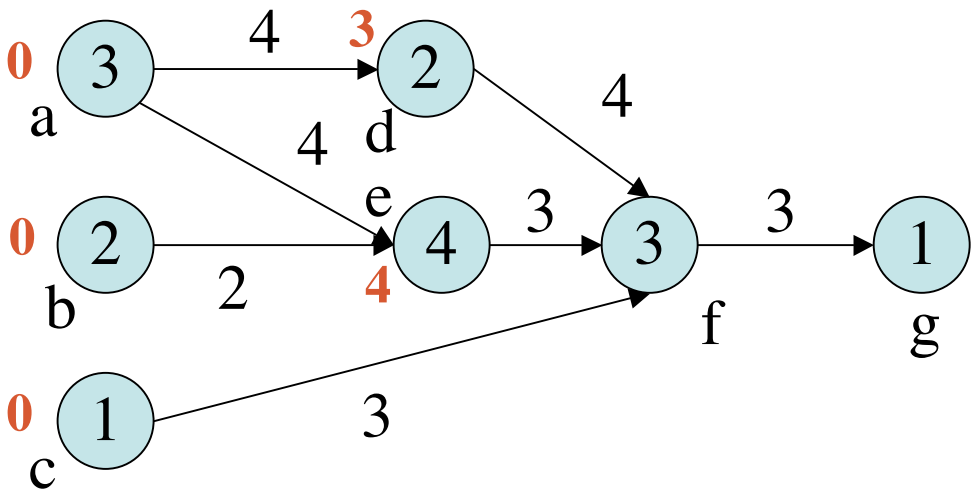
- Ressource non limitée et duplication
- Ressource non limitée sans duplication
- Ressource limitée et duplication
- Ressource limitée sans duplication

Ressource non limitée et duplication

Un algorithme de garantie 2 pour P /prec, dup/C_{max}
[Papadimitriou et al. 90]

2 phases :
1) Calcul de dates de disponibilité r_i ;
2) Calcul d'un ordonnancement tel que chaque copie de i est exécutée avant la date $2r_i$.

On suppose que (1,2, … ,n) est un ordre topologique.

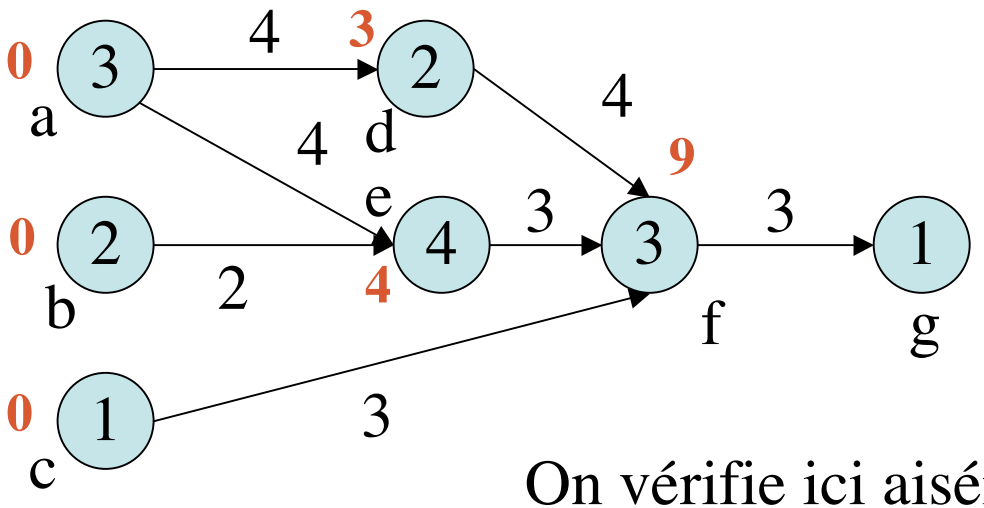


Supposons que r_a, r_b, r_c, r_d et r_e soient déjà calculées.

Soit G_f le sous-graphe des ascendants de f.
Soit t une date d'exécution d'une copie de f ;

(i,j) arc de G_f est t-critique si $r_i + p_i + c_{ij} > t$.
La tâche i est t-critique si i est l'origine d'un chemin de i à f composé d'arcs t-critiques.

Prop : Si une copie c de f est exécutable à t alors
une copie de chaque tâche t-critique i doit être exécutée après r_i et sur le même processeur que c.



On vérifie ici aisément que $8 \leq r_f \leq 11$.

Recherche dichotomique de la plus petite valeur de t telle que sur une même machine on puisse exécuter :

1) f avant t ;
2) les tâches t-critiques de G_f après leur r_i.

Pour t=9, e est la seule tâche 9-critique et l'ordonnancement 1-machine de e et f avec t_e=4 et t_f=9 est réalisable ;

Pour t=8, e et d sont 8-critiques et aucun ordonnancement 1-machine de e , d et f avec $t_e \geq r_e$, $t_d \geq r_d$ et t_f=8 n'existe.

On pose donc $r_f = 9$.

Calcul itératif de l'ordonnancement

On suppose connu un ordonnancement S' pour {1,2, … ,k-1} tel que chaque copie de i est exécutée avant la date $2r_i$.

Il existe un ordonnancement 1-machine tel que :
- chaque tâche r_k-critique i est exécutée à $u_i \geq r_i$;
- la tâche k est exécutée à la date r_k.

L'ordonnancement S de {1,2, … ,k} est constitué de :
a) l'ordonnancement partiel S' ;
b) un processeur supplémentaire qui exécute :
- une copie de chaque tâche r_k-critique i à la date u_i+r_k ;
- une copie de la tâche k à la date $2r_k$.

Autre résultat:

Un algorithme de garantie (2-1/m) pour $P/prec,p_i,c_{ij}/C_{max}$ **[Hanen-Munier 97]**

Résultats sur le nombre de duplications.

Borne inférieure :

Pour $P\ /prec,p_j=1, c_{ij}=c,dup/C_{max}$, le nombre minimum de copies est en Ω (log n) **[Jung et al. 93]**

Borne supérieure :

Calcul exact du nombre minimum de copies pour $P\ /out\text{-}tree,p_j=p, c_{ij}=c,dup/C_{max}$ **[Hanen-Munier 02]**

Ressource non limitée sans duplication

L'algorithme DSC pour P /prec/C_{max} [Gérasoulis et al. 93]

Garantie de DSC : 1 +1/g(G)

Où : $g(G) = \min\{g_i / i=1..n\}$, $g_i = \min\{u_i, v_i\}$

$u_i = \min\{p_j / j \in Pred(i)\} / \max\{c_{ji} / j \in Pred(i)\}$

$v_i = \min\{p_j / j \in Succ(i)\} / \max\{c_{ji} / j \in Succ(i)\}$

Définition :

Un cluster est une liste de tâches distinctes dont l'ordre est compatible avec le graphe de précédence.

Soit C une partition en clusters.
Si chaque cluster de C est exécuté sur un processeur distinct,
on associe à C l'ordonnancement au plus tôt du graphe CPM
G(C) où un arc (i,j) est valué par :

p_i si i et j sont dans le même cluster ;

$p_i + c_{ij}$ sinon.

C initial : A chaque tâche i correspond un cluster.

Itération de DSC :
insertion d'une tâche non encore examinée en dernière position
d'un cluster (avec éventuellement «□gel□» d'un cluster)

Pour une partition C en clusters, on définit :
- les tâches libres (tous leurs prédécesseurs ont été examinés)
- les tâches partiellement libres (au moins un prédécesseur examiné et un prédécesseur non examiné)

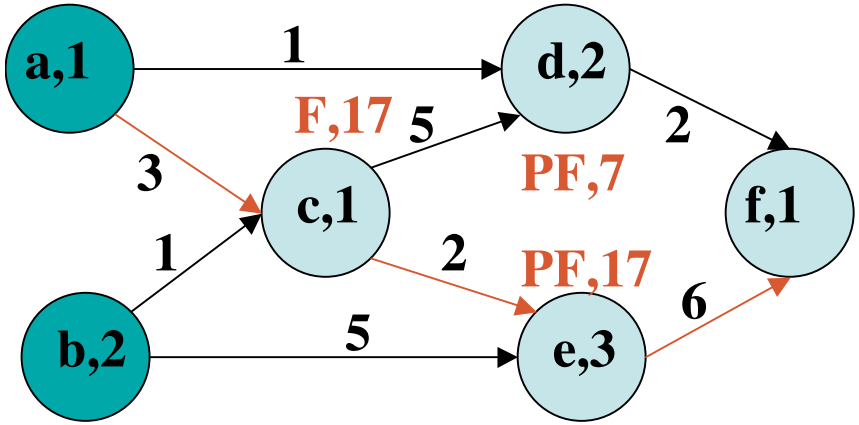
Priorité d'une tâche libre i : $r_i + q_i$ où
r_i est la date au plus tôt de i dans G(C) ;
q_i est la durée maximale d'un chemin d'origine i dans G(C)

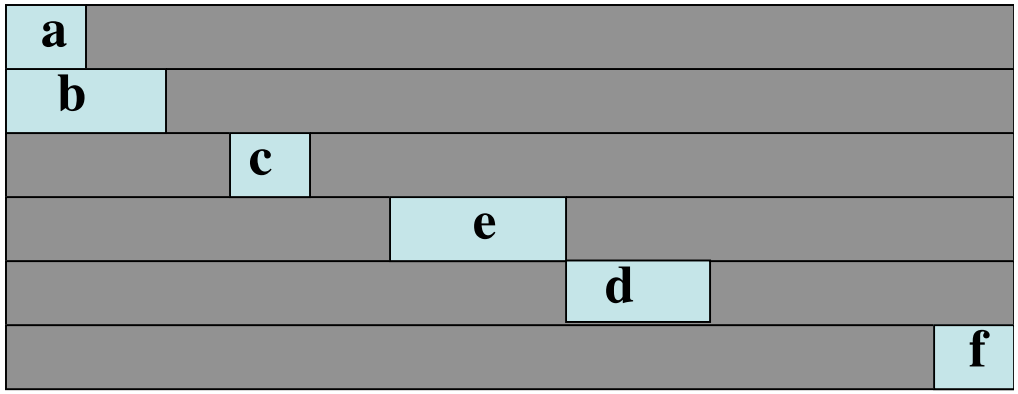
Priorité d'une tâche partiellement libre : $s_i + q_i$ où
s_i est la durée maximale d'un chemin de G(C) d'extrémité i dont tous les sommets (sauf i) sont examinés

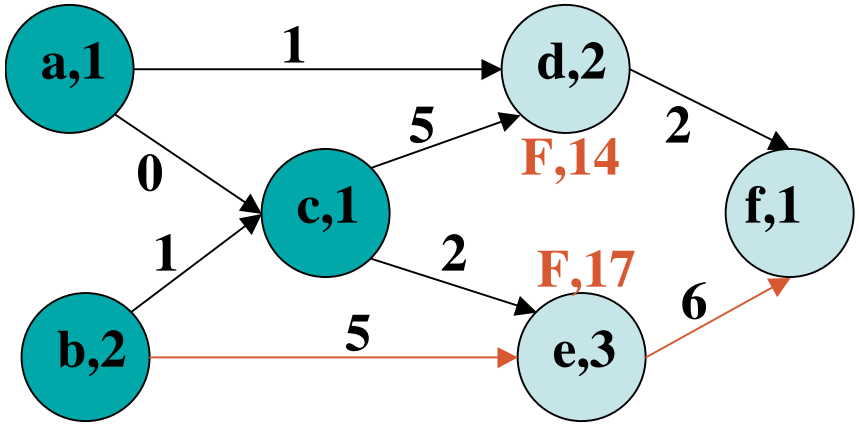
Soit j* la tâche libre de plus grande priorité (notée a) ;
Soit k* la tâche partiellement libre de plus grande priorité (notée b)

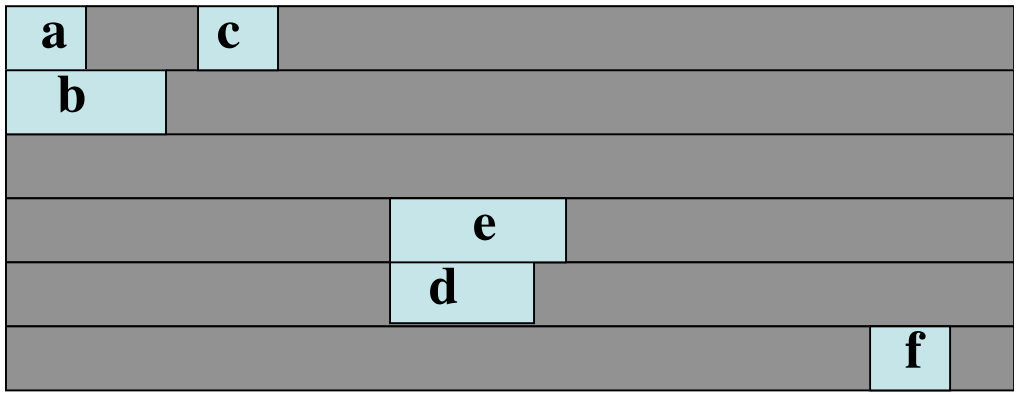
Si a≥b, on place j* en dernière position du meilleur cluster d'un de ses prédécesseurs si on améliore ainsi t_{j*} ;
Sinon, j* devient examinée mais reste « seule » dans son cluster

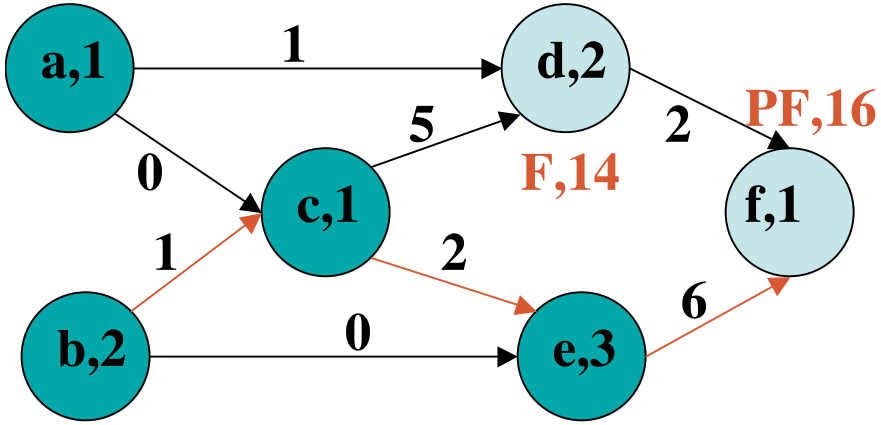
Si a<b, soit i* telle que $t_{i*}+l(i*,k*)=s_{k*}$;
Si s_{k*} diminue après affectation de k* au cluster contenant i* alors
- ce cluster est « gelé » jusqu'à ce que k* soit libre ;
- on examine j*.

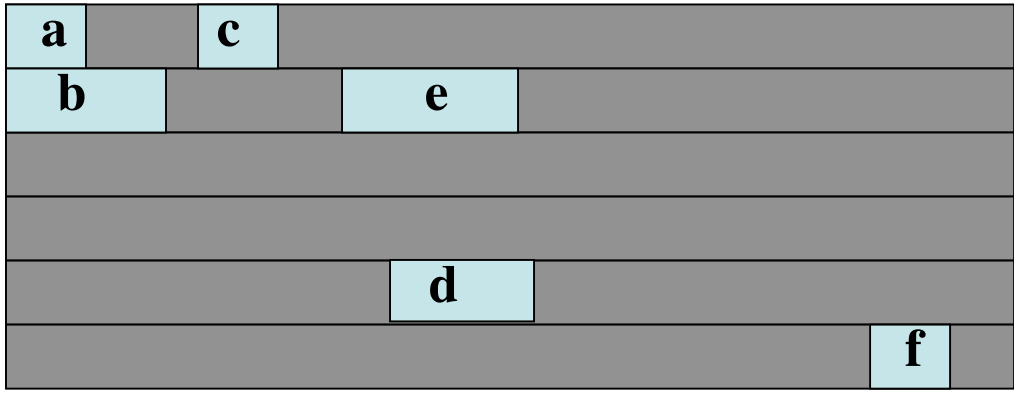

a,1
d,2
c,1
f,1
b,2
e,3
1
F,17
5
2
3
PF,7
1
2
PF,17
6
5


a
b
c
e
d
f

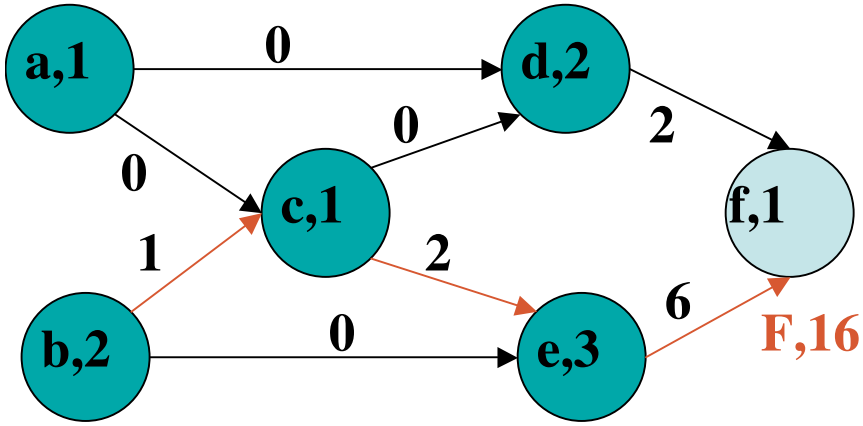

a,1
d,2
c,1
f,1
b,2
e,3
1
5
2
F,14
0
1
2
F,17
6
5

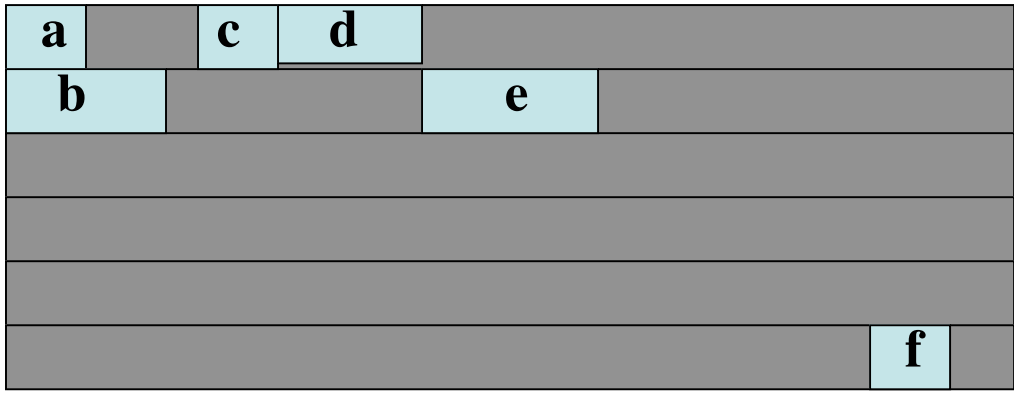

a
c
b
e
d
f

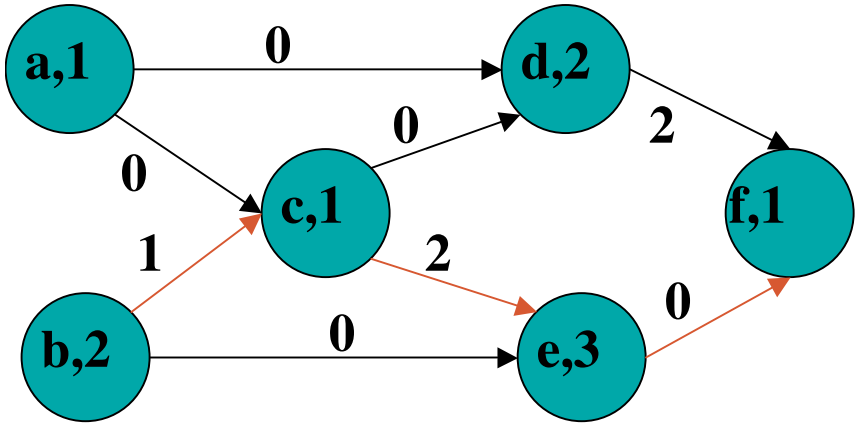

a,1
d,2
c,1
f,1
b,2
e,3
1
5
2
PF,16
F,14
0
1
2
6
0

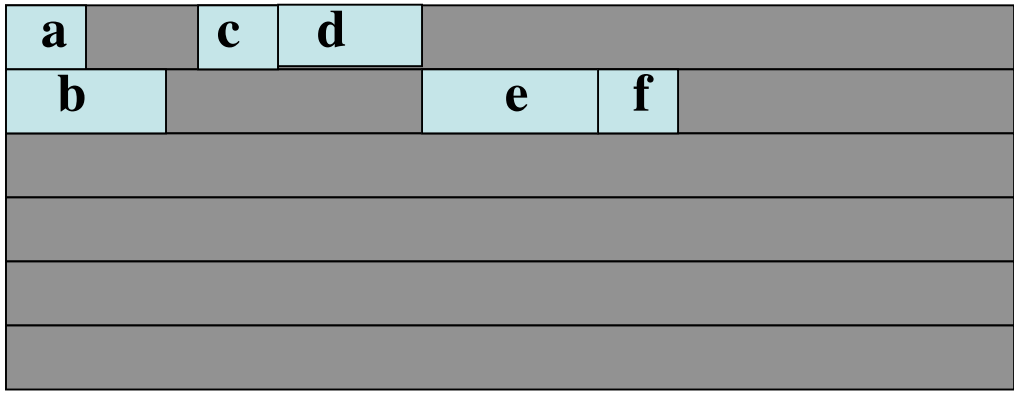

a
c
b
e
d
f

Gel du cluster {b,e} jusqu'à ce que f soit libre.









Autres résultats :

Un algorithme de garantie (1+r) pour P /prec,r-SCT/C_{max}.
[Picouleau 92]

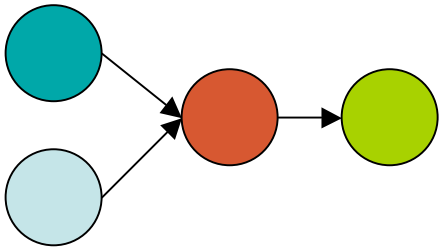
Un algorithme de garantie 4/3 pour P /prec,p_i=1,c_{ij}=1/C_{max}
[Munier et al. 97]

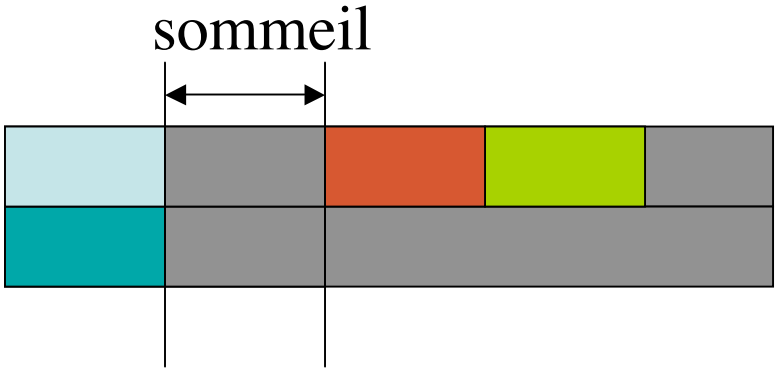
Ressource limitée sans duplication

Ordonnancement de liste (extension) pour P/prec,p_i=1,c_{ij}=1/C_{max}.
[Rayward-Smith 87]

La période t est l'intervalle [t-1,t[

Supposons les t-1 premières périodes examinées,
une tâche prête à la période t n'est pas ordonnançable à t si
elle a au moins 2 prédécesseurs exécutés pendant la période t .





Propriété 1 :

Les ordonnancements de listes sont actifs et dominants.

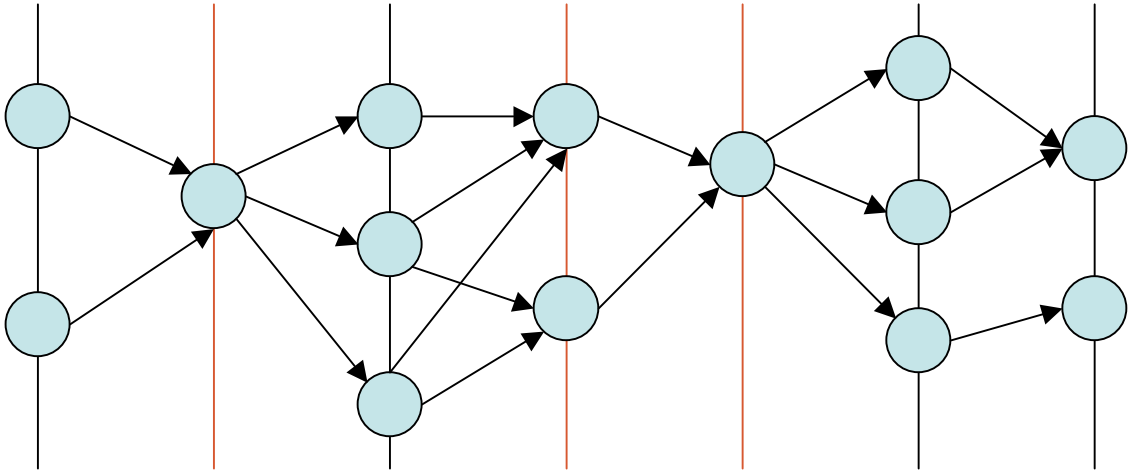
Propriété 2 :

Soit S un ordonnancement de liste possédant :

s périodes de sommeil (tous les processeurs sont oisifs) ;

r périodes semi-actives (au moins un proc. actif et un proc. oisif).

Il existe un sous-graphe G(S) de G qui est un graphe de niveau de type (u,v) avec u = s+max{(r-2s-1)/2,0}+1 et v = s.



Un graphe de niveau de type (7,3)

Propriété 3 :
Tout ordonnancement d'un graphe de niveau (u,v)
est de durée au moins égale à u+v.

Résultat :
Pour tout ordonnancement de liste LS, on a :
$C_{max}(LS) \leq (3-2/m)C_{max}^* - (1-1/m)$

Ordonnancement de liste pour $P/prec,p_i,c_{ij},d_{uv}/C_{max}$.

[Hwang et al. 89]

Ordonnancement ELS (Extended List Schedule) :

Phase 1 : Calcul de l'ordonnancement de liste S obtenu en ignorant les communications ;

Phase 2 : On conserve l'affectation des tâches aux processeurs de S et on tient compte des communications.

Garantie :

$C_{max}(ELS) \leq (2-1/m)C'_{max}{}^{*} + d_{max} \sum_{(i,j)\in G} c_{ij}$

$C'_{max}{}^{*}$ est le makespan minimum du problème sans délais de communication ;

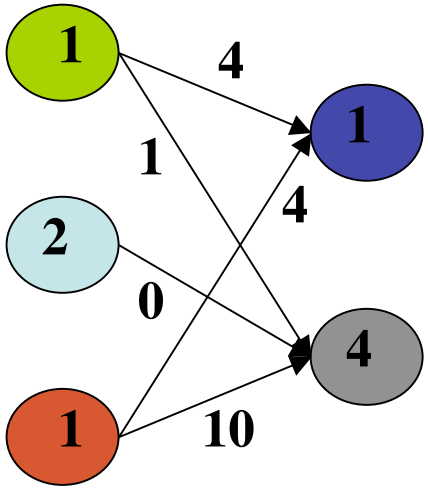
d_{max} est la distance maximale interprocesseurs

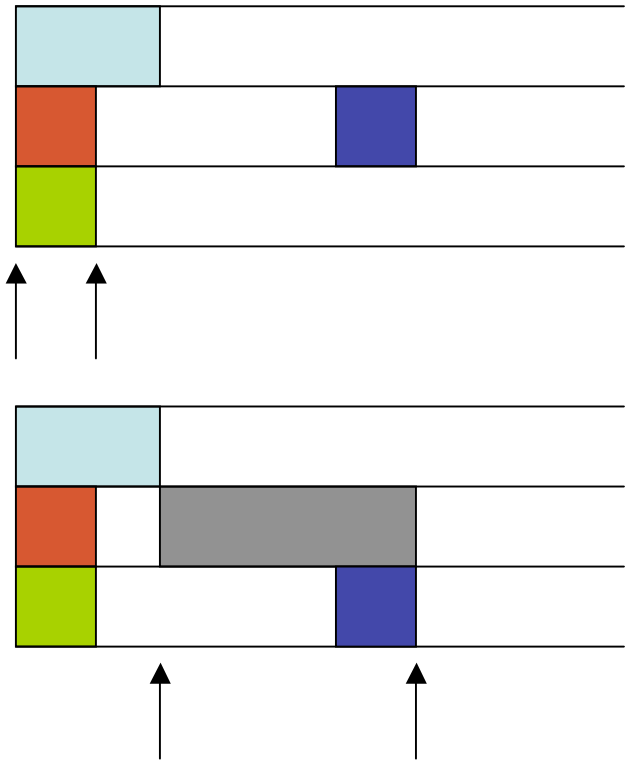
Ordonnancement de liste ETF (Earliest Task First)

Règle :
Ordonnancer d'abord la tâche prête exécutable le plus tôt.
Les instants de décisions sont les dates croissantes (au sens large) des fins de tâches

A chaque tâche prête i correspond un processeur p(i) pour lequel sa date de début t_i est minimale ;
On ordonnance alors la tache prête î telle que $t_{\hat{i}}$ est minimum sur le processeur p(î).

Exception à la règle si lors de la fin de tâche suivante,
une nouvelle tâche prête peut être exécutée encore plus tôt.





3 premières étapes pour d=0.

Etape suivante : d=1
La tâche bleu-foncé est prête et exécutable au plus tôt à t=5 sur P_2 ;

Etape suivante : d=2
La tâche grise est prête et exécutable au plus tôt à t=2 sur P_2 ;
On reporte à plus tard l'ordonnancement de la tâche bleu-foncé

Etape suivante : d=2
La tâche bleu-foncé est prête et exécutable au plus tôt à t=5 sur P_3.

Garantie de l'algorithme ETF :

$$C_{max}(ETF) \leq (2-1/m)C'_{max}{}^{*} + \square(G)$$

où :

$\square(G) = \text{Max} \{\square_{(i,j)\square\square} c_{ij} / \square \text{ chemin de } G\}$

$C'_{max}{}^{*}$ est le makespan minimum du problème sans délais de communication ;

Autres résultats (notation : $\alpha = c_{min}/p_{max}$) :

Algorithme de garantie 7/3
pour P/prec, $\alpha<1/C_{max}$ **[Hanen-Munier 95]**

Algorithme de garantie $1+(1-1/m)(2-1/(1+\alpha))$
pour P/out-tree,p_i,c_{ij}/C_{max} **[Munier 96]**

Algorithme de garantie $3-[6/(m+1)]$
pour P/prec, $p_i=1$,$c_{ij}=1/C_{max}$ **[Hanen-Munier 98]**

Conclusion

Questions en Ordonnancement Cyclique :
Durée de régime transitoire ;
Etude d'autres critères que le temps de cycle ;
Problèmes multicritères (en part. temps de cycle et durée d'itération) ;
…….

Questions en Ordonnancement avec Délais de Communications :
Approximation pour les problèmes à grands délais ;
Algorithmes efficaces pour des topologies spécifiques ;
…….

Problèmes cycliques avec délais : peu étudiés.