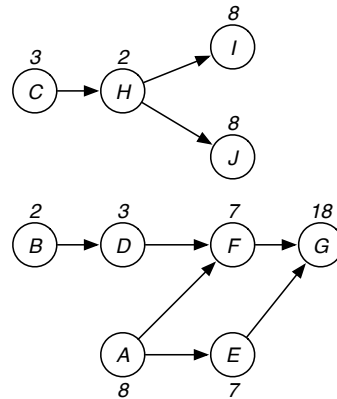


## Ordonnancement – 2

## 1 Anomalies avec les ordonnancements de liste

On considère le graphe de tâches suivant, où les lettres représentent les noms des tâches, et les nombres associés le poids des tâches.



▷ **Question 1** *Quel est le makespan obtenu par l'ordonnancement de liste se basant sur le chemin critique, sur 2 processeurs ? Est-ce optimal ?*

▷ **Question 2** *Supposons maintenant que le poids des tâches est diminué d'une unité (A a maintenant un poids de 7, B un poids de 1, ...). Montrez que le makespan obtenu avec l'ordonnancement basé sur le chemin critique augmente. Montrez que le makespan obtenu avec n'importe quelle heuristique de liste augmente.*

▷ **Question 3** *On retourne aux poids initiaux, et on suppose maintenant que nous avons 3 processeurs. Montrez que le makespan obtenu avec l'ordonnancement de liste basé sur le chemin critique augmente. Montrez que le makespan obtenu avec n'importe quelle heuristique de liste augmente.*

## 2 Ordonnancement sur un ensemble de processeurs hétérogènes (sans communications)

On dispose de  $p$  processeurs et d'un ensemble de  $n$  tâches indépendantes  $T_1, \dots, T_n$ . On notera  $p_{ij}$  le temps de calcul de la tâche  $T_j$  sur le processeur  $P_i$ . Dans le cas où les processeurs vont simplement à des vitesses différentes (c'est-à-dire quand  $p_{ij} = p_j/s_i$  où  $s_i$  représente la vitesse du processeur  $i$  et  $p_j$  la quantité de travail nécessaire au traitement de la tâche  $T_j$ ), le problème est plus simple et on a le même type de résultat que dans le cas homogène. Dans le cas contraire, la meilleure approximation actuellement connue est une 2-approximation.

▷ **Question 4** *Montrer que décider de l'existence d'un ordonnancement dont le temps d'exécution est 3 pour un ensemble de tâches indépendantes  $\{T_1, \dots, T_n\}$  sur les processeurs  $P_1, \dots, P_p$  est un problème NP-complet. (Indication : on pourra se ramener à 3DM.)*

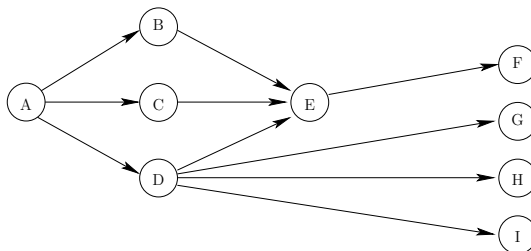
**Définition 1** (3-Dimensional-Matching (3DM)). Étant donné trois ensembles  $A = \{a_1, \dots, a_n\}$ ,  $B = \{b_1, \dots, b_n\}$  et  $C = \{c_1, \dots, c_n\}$  ainsi qu'un ensemble  $F = \{T_1, \dots, T_p\}$  de triplets de  $A \times B \times C$ , trouver un sous-ensemble  $F'$  de  $F$  tel que tout élément de  $A \cup B \cup C$  apparaît dans exactement un triplet de  $F'$ .

### 3 Ordonnement de Coffman et Graham

Soient 2 machines identiques,  $n$  tâches  $(T_i), i = 1 \dots n$  de même durée et  $\prec$  un ordre partiel strict sur les tâches représentant les contraintes de précedence. On note  $\sigma = (\mu, \tau)$  un ordonnancement où  $\mu(i)$  est la machine effectuant  $T_i$  et  $\tau(i)$  est la date de début d'exécution de  $T_i$ .

Lorsque  $T_i \prec T_j$ , on dit que  $T_j$  est un successeur de  $T_i$ . De plus, lorsqu'il n'existe pas de tâche  $T_k$  telle que  $T_i \prec T_k \prec T_j$ , on dit que  $T_j$  est successeur direct de  $T_i$ . On définit de même les notions de prédécesseur et de prédécesseur direct.

▷ **Question 5** Donner un ordonnancement optimal pour le graphe de tâches suivant.



Étant donnée une fonction de priorité  $p$  (supposée injective) sur les tâches, on s'intéresse à l'ordonnement de liste  $\sigma_p = (\mu_p, \tau_p)$  défini comme suit : à chaque instant, on choisit parmi toutes les tâches prêtes celle de priorité maximale et on l'exécute sur la machine 1. De même, la seconde tâche prête la plus prioritaire est exécutée sur la machine 2.

▷ **Question 6** Pour commencer, quelle condition doit vérifier  $p$  pour que les tâches soient exécutées dans un ordre compatible avec les contraintes de précedence ?

▷ **Question 7** Montrer que la machine 1 n'est jamais inactive, et que si  $T_i$  est effectuée sur la machine 1, toutes les tâches  $T_j$  effectuées après (ou au même moment) sont de priorité plus petite que  $T_i$ .

Pour simplifier, on suppose que dans  $\sigma_p$ , lorsqu'il n'y a pas de tâche prête à être effectuée sur la machine 2, on effectue une tâche "fantôme" sans prédécesseur, de priorité plus faible que les tâches initiales. On définit une suite de couples de tâches  $(D_k, J_k)$ , effectuées au même moment, respectivement par la machine 1 et 2.  $D_0$  est la dernière tâche calculée par la machine 1, de même  $J_0$  est la dernière tâche effectuée par  $P_2$ .  $J_k$  (si elle existe) est la tâche la plus tardive effectuée avant  $D_{k-1}$  sur la machine 2, qui soit de priorité plus petite que  $D_{k-1}$ . On note  $F_k$  l'ensemble des tâches effectuées strictement après  $D_{k+1}$  et strictement avant  $D_k$ , plus la tâche  $D_k$ , et on note  $E_k$  l'ensemble des tâches de  $F_k$  sans prédécesseur dans  $F_k$ .

▷ **Question 8** Donner pour l'exemple précédent, l'ordonnement  $\sigma_p$ , les tâches  $D_k$  et  $J_k$  et les ensembles  $F_k$  et  $E_k$ , en supposant que la priorité  $p$  suit l'ordre alphabétique (décroissant) des noms des tâches. Même question pour une priorité compatible avec la précedence (à préciser) conduisant à un ordonnancement optimal.

▷ **Question 9** Montrer que toute tâche de  $F_k$  est de priorité plus grande que  $D_k$  et que toute tâche  $T_i$  de  $F_{k-1}$  est successeur de  $D_k$ . En déduire que les tâches de  $E_{k-1}$  sont les successeurs directs de  $D_k$  et que tout autre successeur direct de  $D_k$  a une priorité plus faible.

On considère  $\prec_l$  l'ordre lexicographique et on suppose que la priorité  $p$  vérifie en plus la propriété suivante :  $p(T_j) < p(T_i)$  si et seulement si  $l(T_j) \prec_l l(T_i)$  où  $l(T_j)$  (resp.  $l(T_i)$ ) est la liste des priorités des successeurs directs de  $T_j$  (resp.  $T_i$ ) classé par ordre décroissant.

▷ **Question 10** Montrer alors que toutes les tâches de  $F_k$  (notamment celles sans successeur dans  $F_k$ ) sont prédécesseurs de toutes les tâches de  $F_{k-1}$ . Conclure en donnant un algorithme permettant de construire une telle priorité  $p$  et un ordonnancement de durée minimale.

## 4 Réponses aux exercices

### ▷ Question 1, page 1

On calcule le bottom level et les temps d'exécution :

Task	A	B	C	D	E	F	G	H	I	J
Bottom level	33	30	13	28	25	25	18	10	8	8
$\sigma$	0	0	5	2	8	8	15	15	17	25
Processeur	$P_1$	$P_2$	$P_2$	$P_2$	$P_1$	$P_2$	$P_1$	$P_2$	$P_2$	$P_2$

Le makespan est de 33, le temps séquentiel est de 66. Il n'y a pas de période où un processeur n'est pas utilisé, on a donc un ordonnancement optimal.

### ▷ Question 2, page 1

Task	A	B	C	D	E	F	G	H	I	J
Bottom level	30	26	10	25	23	23	17	8	7	7
$\sigma$	0	0	3	1	7	13	19	5	6	13
Processeur	$P_1$	$P_2$	$P_2$	$P_2$	$P_1$	$P_1$	$P_1$	$P_2$	$P_2$	$P_2$

Le makespan est de 36. En fait, n'importe quel heuristique de liste mène à un makespan d'au moins 36. C'est un peu chiant à montrer : il n'y a pas d'autre manière que de tester tous les cas. Au temps 0, soit on ordonnance  $A$  et  $B$ , soit  $A$  et  $C$ , soit  $B$  et  $C$ . On explore ainsi un arbre des solutions, et on arrive finalement à prouver le résultat.

### ▷ Question 3, page 1

Avec 3 processeurs, il n'y a qu'un seul ordonnancement de liste : il n'y a aucune liberté.

Task	A	B	C	D	E	F	G	H	I	J
$\sigma$	0	0	0	2	8	3	5	5	13	20
Processeur	$P_1$	$P_2$	$P_3$	$P_2$	$P_1$	$P_3$	$P_2$	$P_3$	$P_2$	$P_1$

Le makespan est de 38.

### ▷ Question 4, page 1

Soit  $I = (A, B, C, F)$  une instance de 3DM. Notre instance  $I'$  sera composée de  $p = |F|$  processeurs et de  $2n + p$  tâches, où  $n = |A| = |B| = |C|$ . Le processeur  $P_i$  est associé au triplet  $T_i$  pour  $i \in \llbracket 1, p \rrbracket$ . Une tâche  $J_e$  est associée à chaque élément  $e$  de  $A \cup B \cup C$ . Enfin, nous aurons également besoin de tâches factices  $(X_x)_{1 \leq x \leq p-n}$  pour arriver au bon nombre de tâches. La durée de la tâche  $J_e$  est de 1 sur tout processeur  $P_i$  tel que  $e \in T_i$ , et de 3 autrement. Les tâches  $X_x$  sont de durée 3, et ce quelque soit le processeur sur lequel elles sont exécutées.

Montrons qu'il existe un ordonnancement de durée 3 pour  $I'$  si et seulement s'il existe un couplage 3-dimensionnel de l'instance  $I$ .

⇒ S'il existe un ordonnancement de durée 3, alors les  $p - n$  tâches  $X$  étant de durée 3, les  $3n$  tâches restantes (celles qui sont associées aux éléments de  $A \cup B \cup C$ ) doivent être exécutées sur des processeurs où leur temps de calcul est 1. Autrement dit, chaque processeur  $P_i$  se voit assigner le traitement de trois tâches  $J_a, J_b, J_c$  dont le temps d'exécution est 1, ce qui signifie que  $T_i = (a, b, c)$ . On peut donc en extraire un couplage 3-dimensionnel solution du problème initial.

⇐ S'il existe un couplage 3-dimensionnel  $F'$  de l'instance  $I$ , alors en associant pour tout  $T_i = (a, b, c) \in F'$  les tâches  $J_a, J_b, J_c$  au processeur  $P_i$  et les autres  $p - n$  tâches  $X$  restantes aux  $p - n$  processeurs restants, on obtient bien un ordonnancement de durée 3.

### ▷ Question 5, page 2

$P_2$		C	B	E	F	
$P_1$	A	D	G	H	I	

▷ **Question 6, page 2**

$$T' \prec T \Rightarrow p(T') > p(T)$$

▷ **Question 7, page 2**

À cause des temps d'exécution unitaires,  $P_1$  et  $P_2$  sont disponibles pour exécuter des tâches au même moment. Supposons que  $P_1$  et  $P_2$  deviennent libre pour exécuter au temps  $t$ , et que  $T$  est la tâche de plus haute valeur  $p(T)$  parmi toutes les tâches non encore exécutées au temps  $t$ . Pour tout  $T'$  tel que  $T' \prec T$  la fonction de priorité  $p$  respectant  $T' \prec T \Rightarrow p(T') > p(T)$ , on a que au temps  $t$ , tous les prédécesseurs de  $T$  ont déjà été exécuté, et donc  $T$  est prêt à être exécuté. Puisque l'algorithme assigne les tâches libres de plus haute valeur  $p$  à  $P_1$  et  $P_2$  pour qu'elle s'exécute, et puisque  $P_1$  reçoit les tâches avant  $P_2$ , on voit que  $T$  va être exécuté sur  $P_1$  à  $t$ . On a donc la propriété suivante : si  $T$  est exécuté sur  $P_1$  et  $\tau(T) \leq \tau(T'), T \neq T'$ , alors  $p(T) > p(T')$ . On déduit également que  $P_1$  ne sera jamais inactif puisqu'on lui assigne les tâches libres en priorité.

▷ **Question 8, page 2**

Cas où les tâches sont triées par ordre alphabétique :

$P_2$	$\emptyset_1$	C	$\emptyset_2$	G	H	$\emptyset_3$
$P_1$	A	B	D	E	F	I

Couples  $(D_k, J_k)$  :

- $D_0 = I, J_0 = \emptyset_3$
- $D_1 = D, J_1 = \emptyset_2$
- $D_2 = A, J_2 = \emptyset_1$

$F_k$  et  $E_k$  :

- $F_0 = \{E, G, H, F, I\}, E_0 = \{E, G, H, I\}$
- $F_1 = \{B, C, D\}, E_1 = \{B, C, D\}$
- $F_2 = \{A\}, E_2 = \{A\}$

Cas où les tâches ont un ordre conduisant à un ordonnancement optimal. On prend les valeurs suivantes pour  $p$  :  $p(A) = 9, p(B) = 7, p(C) = 6, p(D) = 8, p(E) = 5, p(F) = 4, p(G) = 3, p(H) = 2, p(I) = 1$ .

$P_2$	$\emptyset_1$	B	G	H	I	
$P_1$	A	D	C	E	F	

Couples  $(D_k, J_k)$  :

- $D_0 = F, J_0 = I$
- $D_1 = E, J_1 = H$
- $D_2 = C, J_2 = G$
- $D_3 = A, J_3 = \emptyset_1$

$F_k$  et  $E_k$  :

- $F_0 = \{F\}, E_0 = \{F\}$
- $F_1 = \{E\}, E_1 = \{E\}$
- $F_2 = \{B, C, D\}, E_2 = \{B, C, D\}$
- $F_3 = \{A\}, E_3 = \{A\}$

▷ **Question 9, page 2**

Par définition on a  $\forall T \in F_k, p(T) \geq p(D_k)$ , car :  $\forall T \in F_k, \sigma(T) > \sigma(D_{k+1}) = \sigma(J_{k+1})$ , or  $p(J_{k+1}) < p(D_k)$  et  $\sigma(J_{k+1})$  est maximal. Donc il n'existe pas sur  $P_2$  de tâche ayant une priorité inférieure à  $D_k$ , et il ne peut pas y en avoir sur  $P_1$ , sinon soit  $D_k$  aurait été exécuté avant, soit cela veut dire que  $p$  ne respecte pas les dépendances.

Montrons que  $\forall T \in F_{k-1}, D_k \prec T$ . Soit  $n_k$  le nombre de tâches sur  $P_1$  dans  $F_k$  (c'est-à-dire la durée de  $F_k$ ). Soit  $T \in F_k$  avec  $\tau(T)$  minimal, c'est-à-dire  $\tau(T) = \tau(D_{k+1}) + 1$ . On a  $p(D_{k+1}) > p(T) \geq p(D_{k+1}) > p(J_{k+1})$ , alors  $T$  a été appelé par  $P_2$  pour être exécuté au temps  $\tau(D_{k+1})$ , mais n'a pas été exécuté. Cela veut dire qu'à ce moment là, un prédécesseur  $T'$  de  $T$  n'avait pas encore été exécuté. Donc,  $\tau(T') \geq \tau(D_{k+1})$ . Mais cela implique que  $p(T') > p(T)$  par la définition de  $p$ . Puisque  $T$  est exécuté au temps  $\tau(T)$ ,  $T'$  doit forcément s'exécuter avant  $T$  et  $\tau(T') \leq \tau(T) - 1 = \tau(D_{k+1})$ . Donc  $\tau(T') = \tau(D_{k+1})$  et  $p(T') > p(J_{k+1})$ , il n'y a ainsi qu'une seule possibilité pour  $T'$  :  $D_{k+1}$ . On a donc  $D_{k+1} \prec T$ .

On suppose maintenant que pour un  $j$  fixé tel que  $1 \leq j < n_k$ , la propriété suivante est vérifiée :  $T \in F_k$ ,  $\tau(T) \leq \tau(D_k) - n_k + j$  alors on a  $D_{k+1} \prec T$ . Soit  $T' \in D_k$ , avec  $\tau(T') = \tau(D_k) - n_k + j + 1$ . Puisque  $p(D_{k+1}) > p(T') \geq p(D_k) > p(J_{k+1})$ , alors, comme précédemment,  $T'$  a été appelé par  $P_2$  au temps  $\tau(D_{k+1})$  pour être exécuté, mais n'était pas libre. Ainsi, un prédécesseur  $T''$  de  $T'$  n'avait pas été exécuté à cet instant, et on doit avoir  $\tau(T'') \leq \tau(T') - 1 = \tau(D_k) - n_k + j$ . Si  $\tau(T'') = \tau(D_k) - n_k$ , alors puisque  $p(T'') > p(T') \geq p(D_k) > p(J_{k+1})$ , on doit avoir  $T'' = D_{k+1}$  et on obtient  $D_{k+1} \prec T'$ . On suppose donc que  $\tau(T'') \geq \tau(D_k) - n_k + 1$ . Grâce à l'hypothèse d'induction, puisque  $\tau(D_k) - n_k + 1 \leq \tau(T'') \leq \tau(D_k) - n_k + j$ , on voit que  $T'' \in F_k$  et  $D_{k+1} \prec T''$ . Par transitivité on a bien  $D_{k+1} \prec T'$ , et on peut conclure que  $D_{k+1} \prec T \forall T \in F_k$ .

Puisque  $D_{k+1} \prec T, \forall T \in F_k$ , on voit facilement que  $S(D_{k+1}) \cap F_k = E_k$ .

### ▷ Question 10, page 2

L'induction commence par dire que  $D_{k+1} \prec X, \forall X \in F_k$ .

Supposons que pour un certain  $j$ ,  $0 \leq j \leq n_{k+1} - 1$ , nous ayons montré que si  $T \in F_{k+1}$  avec  $\tau(D_{k+1}) - j \leq \tau(T) \leq \tau(D_{k+1})$ , alors  $T \prec X$  pour tout  $X \in F_k$ . Soit  $X' \in D_{k+1}$  avec  $\tau(X') = \tau(D_{k+1}) - j - 1$ . Puisque  $X' \in F_{k+1}$  alors  $p(X') > p(D_{k+1})$ . Ainsi, on doit avoir  $l(X') \succ_l l(D_{k+1})$ . S'il existe  $X'' \in S(X') \cap F_{k+1}$ , alors par l'hypothèse d'induction, puisque  $\tau(X'') > \tau(X') = \tau(D_{k+1}) - j - 1$ ; c'est-à-dire  $\tau(X'') \geq \tau(D_{k+1}) - j$ , alors  $X'' \prec X$  pour tout  $X \in F_k$ , et par transitivité,  $X' \prec X$ . On peut donc supposer que  $S(X') \cap F_{k+1}$  est vide. On a d'après la question précédente  $p(T) < p(D_k)$  si  $\tau(T) > \tau(D_k)$ . De plus, on voit que  $\tau(J_k) < \tau(D_k)$ . Si  $l(X') \succ_l l(D_{k+1})$  et que  $X'$  n'a pas de successeur dans  $F_{k+1}$ , alors on doit avoir  $S(X') \cap F_k = E_k$ . Ce qui implique que  $X' \prec X, \forall X \in F_k$ .

Algorithme, on va assigner les valeurs  $\{1, 2, \dots, n\}$  aux  $n$  tâches :

1. Une tâche  $T_0$  est arbitrairement choisie parmi les tâches telles que  $S(T_0) = \emptyset$ . On a alors  $p(T_0) = 1$
2. Supposons que pour  $k \leq n$ , les entiers  $1, 2, \dots, k-1$  ont déjà été assignés. Pour chaque tâche  $T$  pour lesquelles ont a déjà assigné  $p$  à tous ses successeurs, soit  $l(T)$  la liste des priorités des successeurs de  $T$  triée par ordre décroissant. Au moins une tâche  $T^*$  doit vérifier  $l(T^*) \leq l(T)$  parmi toutes ces tâches  $T$ . On choisit  $T^*$  et on lui assigne  $p(T^*) = k$ . Et on répète ce processus jusqu'à ne plus avoir de tâche sans priorité.

Preuve de l'optimalité. Soit une liste de priorité quelconque  $L$ . Toutes les tâches dans  $F_{k+1}$  doivent être exécutées avant les tâches de  $F_k$ . Puisque  $F_{k+1}$  contient  $2n_{k+1} - 1$  tâches (les  $F_{k+1}$  contiennent un nombre impaire de tâches, et  $n_{k+1}$  est le nombre de tâches s'exécutant sur  $P_1$  dans  $F_{k+1}$ ), cela nécessite au moins  $n_{k+1}$  unités de temps. Donc, pour exécuter le graphe de tâches il faut au moins  $\sum_{k=0}^m n_k$  unités de temps (où  $m$  est le nombre d'ensembles  $F_k$ ), quelque-soit la liste  $L$ . Puisque  $w(l) = \sum_{k=0}^m n_k$ , on a que  $w(L) \geq w(l)$ , donc notre algorithme de liste est optimal.