Ordonnancement

Recherche opérationnelle
Dimitri Watel - ENSIIE

2024

Dans ce chapitre on va s'intéresser à deux problèmes classiques d'ordonnancement. L'ordonnancement regroupe l'ensemble des problèmes qui consistent à décider de l'ordre de tâches ou d'évènements. Ce sont des situations que l'on retrouve dans beaucoup dans l'industrie. On peut penser à la gestion d'un projet, à une ligne de production, à l'organisation de cours, à des calculs sur une machine, à un déménagement, ...

Une partie des problèmes d'ordonnancement consiste à supposer qu'on doit exécuter des tâches (indéterminées, ce n'est pas important ici) et qu'on a des machines pour exécuter ces tâches. Chaque tâche doit passer sur toutes les machines mais ne prend pas la même durée sur chaque machine. Une tâche ne peut pas être exécutée sur deux machines en même temps, et une machine ne peut traiter qu'une tâche à la fois. D'autres contraintes spécifiques peuvent s'ajouter. L'objectif est de faire passer toutes les tâches sur toutes les machines en un minimum de temps.

1 Deux machines avec contrainte de précédence

Dans cette partie, on suppose qu'on dispose de deux machines, M_1 et M_2 et qu'il faut exécuter chaque tâche d'abord sur la machine M_1, puis sur la machine M_2.

L'exemple suivant montre qu'il est possible, selon l'ordre, d'avoir deux durées différentes. Les tâches sont notées j_k (où j signifie *job*). Le tableau indique la durée de chaque tâche sur chaque machine.

Machine \ Tâche	j_1	j_2	j_3	j_4	j_5
M_1	5	3	2	1	4
M_2	3	1	4	2	5

Considérons l'ordre j_1, j_2, j_3, j_4, j_5. On obtient le diagramme de gantt suivant, où chaque ligne indique, pour chaque pas de temps quelle tâche est traitée par la machine.

M_1	1	1	1	1	1	2	2	2	3	3	4	5	5	5	5					
M_2	-	-	-	-	-	1	1	1	2	-	3	3	3	3	4	4	5	5	5	5

Si on considère l'ordre j_4, j_3, j_5, j_1, j_2. On obtient une durée plus courte. Il s'agit ici d'une solution optimale.

M_1	4	3	3	5	5	5	5	1	1	1	1	1	2	2	2					
M_2	-	4	4	3	3	3	3	5	5	5	5	1	1	1	1	2				

Il existe un algorithme très simple pour trouver la solution optimale. Il s'agit de l'algorithme de Johnson. Dans l'algorithme suivant, on note $t_i(k)$ la durée de la tâche j_k sur la machine M_i.

Algorithme 1 Algorithme de Johnson

1: $A \leftarrow \{k | t_1(k) \leq t_2(k)\}$
2: $B \leftarrow \{k | t_1(k) > t_2(k)\}$
3: Trier A par ordre croissant sur t_1
4: Trier B par ordre décroissant sur t_2
5: **Renvoyer** la concaténation de A et B.

Cet algorithme est polynomial. Sa complexité est de l'ordre de $O(n \log n)$ où n est le nombre de tâches. Johnson a publié et démontré l'optimalité de son algorithme dans le papier suivant:

S. M. Johnson. "Optimal two- and three-stage production schedules with setup times included". In: *Naval Research Logistics Quarterly* 1.1 (1954), pp. 61–68. DOI: https://doi.org/10.1002/nav.3800010110

La preuve se fait en trois temps.

Théorème 1.1. *Il existe une solution optimale où l'ordre des tâches sur la machine M_1 et sur la machine M_2 est le même.*

Proof. Soit une solution optimale avec un ordre O_1 des tâches sur la machine 1 et un ordre O_2 des tâches sur la machine 2.

Disons qu'il existe deux tâches j_k et j_l telles que

- j_k est avant j_l sur M_1
- j_l est avant j_k sur M_2

D'après les contraintes de précédences sur les machines, dans le diagramme de Gantt des tâches, les tâches j_l et j_k apparaissent sur M_2 après la fin de la tâche j_l sur M_1.

M_1	$\cdots$	k	k	$\cdots$	l	l	l	$\cdots$												
M_2	$\cdots$								$\cdots$	l	l	$\cdots$	k	k	k					

On peut supposer sans perte de généralité que j_k arrive juste après j_l dans O_2. En effet, pour toute tâche successive dans O_2, si elles sont dans le même ordre dans O_1 alors $O_1 = O_2$.

Si, sur M_2, entre j_l et j_k, il existe un moment où M_2 est inactive, alors on peut déplacer j_k sur la gauche jusqu'à ce que ce moment d'inactivité disparaisse. On peut voir sur l'exemple suivant (où l'inactivité est décrite par un tiret) que cela n'introduit pas de violation des contraintes puisque j_k se termine toujours avant j_l sur la machine M_1 et qu'on a pas touché aux autres tâches.

Une fois l'inactivité supprimée, on peut échanger les tâches j_l et j_k sur M_2 sans créer de conflit. Le tableau suivant représente ces deux étapes.

M_1	$\cdots$	k	k	$\cdots$	l	l	l	$\cdots$																
M_2	$\cdots$								$\cdots$	l	l	-	k	k	k									
M_2	$\cdots$								$\cdots$	l	l	k	k	k	-									
M_2	$\cdots$								$\cdots$	k	k	k	l	l	-									

On a alors un nouvel ordre des tâches qui est valide et dont la durée totale est inférieur ou égale à la durée initiale. Elle est en fait égale car l'ordre initial était supposé optimal. On a donc un autre autre lui aussi optimal. On peut donc reproduire ces étapes pour produire différents ordres optimaux jusqu'à ce que $O_2 = O_1$. On a alors une solution optimale avec la propriété recherchée. □

Le théorème précédent indique qu'il n'est pas nécessaire de s'intéresser aux solutions où les ordres sont différents.

Théorème 1.2. *Si O est un ordre des tâches, où o_i est la $i-ieme$ tâche de l'ordre. Si, pour tout $i < k$, $\min(t_1(o_i), t_2(o_k)) \leq \min(t_2(o_i), t_1(o_k))$ alors O est optimal.*

Remarque 1. La preuve est ici un peu longue et est issue de l'article cité plus haut. Il n'est pas nécessaire de la maîtriser ni de la connaître par coeur. Elle est ici pour votre culture générale (et parce que l'article original n'est pas simple à lire).

Proof. Soit O^* un ordre optimal. On peut passer de O^* à O en effectuant uniquement des échanges de tâches successives. Par exemple pour passer de l'ordre (j_2, j_3, j_1, j_4) à (j_1, j_3, j_2, j_4), on peut passer par les ordres suivants:

- (j_2, j_3, j_1, j_4)
- (j_3, j_2, j_1, j_4)
- (j_3, j_1, j_2, j_4)
- (j_1, j_3, j_2, j_4)

En effet, si $O^* \neq O$ alors il existe deux tâches o_i et o_k telles que $i < k$ et telles que o_k est avant o_i dans l'ordre O^*. Il existe nécessairement deux tâches de la sorte qui sont successives dans O^* (sinon $O^* = O$). On peut donc effectuer l'échange. On obtient alors un nouvel ordre O_1. On peut recommencer l'opération jusqu'à obtenir O.

Notons $(O_0 = O^*, O_1, O_2, \ldots, O_p = O)$ les différents ordres successifs. Montrons que l'hypothèse faite sur les différentes tâches montrent que, pour tout $q \in [\![1; p]\!]$, O_q se termine en même temps ou après O_{q+1}. Par transitivité, on aura que O^* se termine en même temps ou après O et donc, par optimalité de O^*, on en déduira que O est aussi optimal.

Il est important de remarquer que, dans O_q, on échange toujours deux tâches o_i et o_k telles qu $i < k$ et telles que o_i est juste après o_k dans l'ordre O_q, ce qui permettra d'utiliser l'hypothèse du lemme.

Afin de simplifier la lecture de la suite, on notera les deux ordres O_q et O_{q+1} respectivement B et A. On notera a_i (respectivement b_i) la i-ieme tâche de l'ordre A (resp. B). On veut montrer que A se termine avant B. On suppose que les tâches k et $k+1$ ont été échangées entre A et B (donc $a_k = b_{k+1}$ et $b_k = a_{k+1}$). On sait enfin que a_k est une tâche o_i et que a_{k+1} est une tâche o_l de l'ordre O telles que $i < l$. On vérifie donc $\min(t_1(a_k), t_2(a_{k+1})) \leq \min(t_2(a_k), t_1(a_{k+1}))$ par hypothèse.

Commençons par déterminer à quel moment se termine l'ordre A sur la machine M_2. On suppose que, sur A, chaque tâche est placée au plus tôt : sur la machine M_1, chaque tâche est placée après la fin de la suivante, il n'y a pas de moment d'inactivité. Sur la machine M_2, notons x_i la durée d'inactivité juste avant le début de la tâche a_i sur M_2. Dans l'exemple suivant, $x_1 = 5$, $x_3 = 1$ et $x_2 = x_4 = x_5 = 0$.

M_1	1	1	1	1	1	2	2	2	3	3	4	5	5	5	5									
M_2	-	-	-	-	-	1	1	1	2	-	3	3	3	3	4	4	5	5	5	5	5			

La machine M_2 se termine donc à l'instant $\sum_{i=1}^{n}(t_2(a_i) + x_i) = \sum_{i=1}^{n} t_2(a_i) + \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} t_2(j_i) + \sum_{i=1}^{n} x_i$.

De la même manière, si, dans la solution issue de l'ordre B, on note y_i la durée d'inactivité juste avant le début de la tâche b_i sur M_2, alors la machine M_2 se termine à l'instant $\sum_{i=1}^{n} t_2(j_i) + \sum_{i=1}^{n} y_i$.

Il ne nous reste donc qu'à montrer que $\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} y_i$ pour démontrer le résultat souhaité. On vérifie aisément les égalités suivantes:

$$
\begin{aligned}
x_1 &= t_1(a_1) \\
x_2 &= \max(t_1(a_1) + t_1(a_2) - t_2(a_1) - x_1, 0) \\
x_3 &= \max(\sum_{l=1}^{3} t_1(a_l) - \sum_{l=1}^{2} t_2(a_l) - \sum_{l=1}^{2} x_l, 0) \\
&\cdots \\
x_i &= \max(\sum_{l=1}^{i} t_1(a_l) - \sum_{l=1}^{i-1} t_2(a_l) - \sum_{l=1}^{i-1} x_l, 0)
\end{aligned}
$$

Et donc

$$\sum_{l=1}^{i} x_l = \max(\sum_{l=1}^{i} t_1(a_l) - \sum_{l=1}^{i-1} t_2(a_l), \sum_{l=1}^{i-1} x_l)$$

Ainsi, posons

$$K_i = \sum_{l=1}^{i} t_1(a_l) - \sum_{l=1}^{i-1} t_2(a_l)$$

Alors

$$\sum_{l=1}^{i} x_l = \max_{i=1}^{n} K_i$$

De même, dans l'ordre B, on a les égalités suivantes

$$L_i = \sum_{l=1}^{i} t_1(b_l) - \sum_{l=1}^{i-1} t_2(b_l)$$
$$\sum_{l=1}^{i} y_l = \max_{i=1}^{n} L_i$$

Puisque $a_l = b_l$ si $l \notin \{k, k+1\}$, on a

$$K_i = L_i \text{ si } i \notin \{k, k+1\}$$

Ainsi, si on prouve que $\max(K_k, K_{k+1}) \leq \max(L_k, L_{k+1})$ alors $\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} y_i$. Pour finir, en soustrayant $\sum_{l=1}^{k+1} t_1(a_l) - \sum_{l=1}^{k-1} t_2(a_l)$ aux deux éléments de l'inégalité, et en se rappelant que $a_k = b_{k+1}$ et $b_k = a_{k+1}$, on obtient

$$\max(K_k, K_{k+1}) \leq \max(L_k, L_{k+1})$$
$$\max(-t_1(a_{k+1}), -t_2(a_k)) \leq \max(-t_2(a_{k+1}), -t_1(a_k))$$
$$\min(t_2(a_{k+1}), t_1(a_k)) \leq \min(t_1(a_{k+1}), t_2(a_k))$$

Cette dernière inégalité est exactement l'hypothèse faite sur les tâches a_k et a_{k+1}. On a donc bien A qui se termine avant B et donc comme expliqué plus haut, O qui se termine avant O^*, et qui est donc un ordre optimal. □

Théorème 1.3. *Si O est l'ordre renvoyé par l'algorithme de Johnson, il vérifie les hypothèses du théorème 1.2.*

Proof. Cette preuve est donnée en exercice en TD. □

Ce dernier théorème couplé au précédent montre que l'algorithme de Johnson est optimal. On a donc un algorithme polynomial.

Remarque 2. Ces résultats ne sont valables qu'avec deux machines. On constate que, de toute manière, l'algorithme n'est pas prévu pour marcher avec trois machines. De manière générale, si on rajoute des machines ou des contraintes (par exemple *la tâche* 4 *doit passer avant demain sur la machine* M_2 *car le client a payé pour qu'on aille plus vite sur la tâche* 4) alors le problème devient plus difficile. On ne connait pas d'algorithme polynomial pour résoudre de version plus générale du problème. Cependant, il existe des cas, par exemple à 3 machines, où l'algorithme de Johnson peut aider. Un exemple est donné en execice de TD.

2 Deux machines sans contrainte

Dans cette section, on n'impose aucun ordre sur les machines. Ainsi, chaque tâche doit passer sur chacune des machines mais peut le faire dans un ordre différent des autres tâches. On va montrer que, dans ce cas, il existe aussi un algorithme polynomial pour résoudre le problème. Considérons dans un premier temps l'exemple de la section précédente :

Machine \ Tâche	j_1	j_2	j_3	j_4	j_5
M_1	5	3	1	2	4
M_2	3	1	4	2	5

Une solution serait la suivante:

M_1	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5
M_2	3	3	3	3	4	4	5	5	5	5	5	1	1	1	2

Il est impossible de faire mieux car les machines n'ont aucun temps de pause. On peut noter que, contrairement au premier problème, une solution ne se résume pas à un ordre des tâches. Il faut indiquer, pour chaque tâche l'instant où elle démarre sur M_1 et de même sur M_2. Connaissant une solution O, on note o_{ik} l'instant de démarrage de la tâche j_i sur M_k.

Dans la suite, on pose

- $M = \max_{k=1}^{n}(t_1(k) + t_2(k))$
- $T_1 = \sum_{k=1}^{n} t_1(k)$
- $T_2 = \sum_{k=1}^{n} t_2(k)$
- $T = \max(T_1, T_2)$

Si $T_1 < T_2$, alors on prétraite l'instance en rajoutant des tâches fictives. Chaque tâche fictive vérifie $t_1(k) = 1$ et $t_2(k) = 0$. En rajoutant $T_2 - T_1$ tâches fictives, on a $T_1 = T_2 = T$. On agit de manière similaire si $T_2 < T_1$.

On va montrer le théorème suivant:

Théorème 2.1. *On peut construire en temps polynomial une solution optimale dont la durée est* $\max(M, T)$.

Lemme 2.1. *La durée de toute solution réalisable est au moins* $\max(M, T)$.

Proof. Supposons qu'on dispose d'un ordonnancement O réalisable. L'instant de fin t_f de l'ordre O est après la fin de l'exécution de chaque machine. Au mieux chaque machine doit travailler pendant une durée T. Donc $T \leq$

t_f. De plus chaque tâche doit être terminée et puisqu'une tâche ne peut être exécutée sur les deux machines en même temps, l'ordre O se termine après une durée $t_1(i)+t_2(i)$ pour toute tâche i. Donc $M \leq t_f$. □

Lemme 2.2. *Si $M \geq T$, on peut construire en temps polynomial une solution optimale dont la durée est M.*

Proof. Soit i tel que $t_1(i) + t_2(i) = M$. On peut construire une solution de durée M ainsi où A correspond à toutes les tâches sauf i.

M_1	-	A	A	$\cdots$	A	A	i	i	$\cdots$	i	i	i
M_2	i	i	i	$\cdots$	i	i	A	A	$\cdots$	A	-	-

□

Lemme 2.3. *Si $T \geq M$ et si on dispose de 3 tâches, on peut construire en temps constant une solution optimale dont la durée est T.*

Remarque 3. Construire une solution optimale en temps constant n'est pas difficile ici, il suffit d'énumérer toutes les solutions possible. Mais démontrer qu'on peut construire une solution de durée T quelle que soit l'instance est un peu plus compliqué.

Proof. On va montrer qu'il existe toujours une solution sans qu'aucune machine ne soit inactive.

Supposons dans un premier temps que $t_1(1) \geq t_2(2)$. Si $t_2(1) \geq t_1(3)$ alors il existe une solution de durée T avec la solution suivante :

M_1	1	1	1	$\cdots$	1	1	2	2	$\cdots$	2	2	3	3	3
M_2	2	2	2	$\cdots$	2	3	3	3	$\cdots$	3	1	1	1	1

Si par contre $t_2(1) < t_1(3)$, la solution suivante est réalisable et de durée T.

M_1	2	2	2	$\cdots$	2	2	1	1	$\cdots$	1	1	3	3	3
M_2	3	3	3	$\cdots$	3	3	3	2	$\cdots$	2	2	2	1	1

Cette solution est bien réalisable. En effet, la tâche 1 ne s'exécute pas sur les deux machines en même temps car $t_2(1) < t_1(3)$. Ensuite, il en est de même pour la tâche 2 car $t_1(3) + t_1(1) > t_2(2) + t_2(1)$. Enfin, il n'y a pas de conflit pour la tâche 3 car
$t_1(3) \leq M - t_2(3) \leq T - t_2(3) = t_2(1) + t_2(2)$.

Supposons maintenant que $t_1(1) < t_2(2)$. On va montrer d'abord qu'on peut se ramener à un cas où $t_1(1) \geq t_2(2)$. On considère la nouvelle instance suivante:

Machine \ Tâche	j'_1	j'_2	j'_3
M'_1	$t_2(2)$	$t_2(1)$	$t_2(3)$
M'_2	$t_1(2)$	$t_1(1)$	$t_1(3)$

On remarque qu'il s'agit de la même instance que l'instance d'origine où on a échangé M_1 et M_2 et où on a échangé j_1 et j_2. Les deux ont donc exactement les mêmes solutions optimales. Dans cette nouvelle instance, on note $t'_k(i')$ le temps de la tâche j'_i sur la machine M'_k. On remarque que $t'_1(1') \geq t'_2(2')$. On peut appliquer, dans cette instance, l'une des deux solutions de durée T proposées en début de preuve en fonction de si $t'_2(1') \geq t'_1(3')$ ou non. □

Lemme 2.4. *Si $T \geq M$ et s'il y a plus de 3 tâches alors on peut construire en temps polynomial une solution optimale dont la durée est T.*

Proof. On peut toujours se ramener à 3 tâches avec l'algorithme suivant:

Créer 3 ensembles J_1, J_2, J_3 vides
Pour k de 1 à n **Faire**
  Pour p de 1 à 3 **Faire**
    Si $\sum_{i \in J_p \cup \{k\}} t_1(i) + t_2(i) \leq T_1$ **Alors**
      Ajouter j_k à J_p
      Continuer la boucle extérieure

Dans l'exemple en début de section, on aurait $J_1 = \{j_1, j_2\}$, $J_2 = \{j_3, j_4\}$ et $J_3 = \{j_5\}$.

Si cet algorithme place toutes les tâches dans J_1, J_2 ou J_3 alors on obtient une nouvelle instance, une super-instance, avec 3 super-tâches.
On pose $t_k(J_p) = \sum_{j_i \in J_p} t_k(i)$. Dans l'exemple, on aurait donc

Machine \ Tâche	J_1	J_2	J_3
M_1	8	3	4
M_2	4	6	5

Avec le lemme 2.3, on peut trouver une solution en temps T pour cette super-instance.

M_1	J_1	J_1	J_1	J_1	J_1	J_1	J_1	J_1	J_2	J_2	J_2	J_3	J_3	J_3	J_3
M_2	J_2	J_2	J_2	J_2	J_2	J_2	J_3	J_3	J_3	J_3	J_3	J_1	J_1	J_1	J_1

Dans cette solution de la super-instance, chaque super-tâche J_k n'est pas exécutée en même temps sur les deux machines. Si on remplace, dans les deux machines, cette super-tâche par les tâches de l'instance d'origine qu'elle contient alors ces tâches ne peuvent pas non plus être sur la même machine en même temps. Dans l'exemple, on obtient le résultat suivant.

M_1	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5
M_2	3	3	3	3	4	4	5	5	5	5	5	1	1	1	2

Il reste donc à montrer qu'on place bien toutes les tâches dans J_1, J_2, J_3 et on obtient le résultat. Supposons le contraire, soit j_k la première tâche qui n'est pas placée dans J_1, J_2, J_3. Alors, pour tout $p \in [\![1; 3]\!]$,

$$\sum_{i \in J_p \cup \{k\}} t_1(i) + t_2(i) > T$$

Alors, si on somme sur les 3 valeurs de p

$$\sum_{p=1}^{3} \sum_{i \in J_p} t_1(i) + t_2(i) + 3 \cdot (t_1(k) + t_2(k)) > 3T \tag{1}$$

Or, chaque tâche est dans au plus un ensemble J_p, et j_k n'est dans aucun de ces ensembles:

$$\sum_{p=1}^{3} \sum_{i \in J_p} t_1(i) + t_2(i) + t_1(k) + t_2(k) \leq \sum_{i=1}^{n} t_1(i) + t_2(i)$$

$$\sum_{p=1}^{3} \sum_{i \in J_p} t_1(i) + t_2(i) + t_1(k) + t_2(k) \leq 2T \tag{2}$$

D'après les inégalités (1) et (2)

$$2T + 2 \cdot (t_1(k) + t_2(k)) > 3T$$

$$t_1(k) + t_2(k) > \frac{T}{2} \tag{3}$$

D'après les inégalités (2) et (3)

$$\sum_{p=1}^{3} \sum_{i \in J_p} t_1(i) + t_2(i) < \frac{3T}{2} \tag{4}$$

Enfin, on sait que pour $p < p'$, et chaque tâche j_l placée dans $J_{p'}$, on a $\sum_{i \in J_p} t_1(i) + t_2(i) + t_1(l) + t_2(l) > T$. En effet, sinon, j_l aurait été placée dans J_p au lieu de $J_{p'}$. Donc on a en particulier

$$\sum_{i \in J_1} t_1(i) + t_2(i) + \sum_{i \in J_2} t_1(i) + t_2(i) > T$$

$$\sum_{i \in J_1} t_1(i) + t_2(i) + \sum_{i \in J_3} t_1(i) + t_2(i) > T$$

$$\sum_{i \in J_2} t_1(i) + t_2(i) + \sum_{i \in J_3} t_1(i) + t_2(i) > T$$

$$2 \cdot \sum_{p=1}^{3} \sum_{i \in J_p} t_1(i) + t_2(i) > 3T$$

On a donc une contradiction avec l'inégalité 4, donc toutes les tâches sont dans J_1, J_2 et J_3. □

Preuve du théorème. En utilisant les lemmes 2.1, 2.2 et 2.4, on prouve bien qu'il existe toujours une solution de durée $\max(M, T)$ et qu'on peut la construire en temps polynomial.