Production planning

Recherche opérationnelle
Dimitri Watel - ENSIIE

2024

In this chapter, we will focus on two classic scheduling problems. Scheduling encompasses all problems that involve deciding the order of jobs or events. These are situations encountered in many industrial contexts. One can think of project management, a production line, the organization of classes, computations on a machine, moving, and so on.

Part of the scheduling problems involves assuming that we need to execute jobs (undetermined, which is not important here) and that we have machines to perform these jobs. Each job must go through all the machines but does not take the same amount of time on each machine. A job cannot be executed on two machines simultaneously, and a machine can only handle one job at a time. Other specific constraints may be added. The objective is to process all jobs on all machines in the minimum amount of time.

1 Two machines with precedence constraint

In this section, we assume that we have two machines, M_1 and M_2, and that each job must be executed first on machine M_1 and then on machine M_2.

The following example shows that it is possible to have two different durations depending on the order. The jobs are denoted as j_k (where j stands for *job*). The table indicates the duration of each job on each machine.

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

Let's consider the order j_1, j_2, j_3, j_4, j_5. We obtain the following Gantt chart, where each line indicates which job is being processed by the machine at each time step.

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

If we consider the order j_4, j_3, j_5, j_1, j_2, we obtain a shorter duration. This is an optimal solution.

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 5 1 1 1	2

There is a very simple algorithm to find the optimal solution. It is called the Johnson algorithm. In the following algorithm, we denote $t_i(k)$ as the duration of job j_k on machine M_i.

Algorithm 1 Johnson algorithm

$A \leftarrow \{k | t_1(k) \leq t_2(k)\}$
$B \leftarrow \{k | t_1(k) > t_2(k)\}$
Sort A by increasing order on t_1
Sort B by decreasing order on t_2
return concat A and B

This algorithm is polynomial. Its complexity is in the order of $O(n \log n)$ where n is the number of jobs. Johnson published and proved the optimality of his algorithm in the following paper:

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`

There are three steps in the proof.

Theorem 1.1. *There exists an optimal solution where the order of jobs on machine M_1 and on machine M_2 is the same.*

Proof. Let there be an optimal solution with an order O_1 of jobs on machine 1 and an order O_2 of jobs on machine 2.

Suppose there exist two jobs j_k and j_l such that

- j_k is before j_l on M_1
- j_l is before j_k on M_2

According to the precedence constraints on the machines, in the Gantt chart of the jobs, jobs j_l and j_k appear on M_2 after the completion of job j_l on M_1.

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

We can assume without loss of generality that j_k arrives just after j_l in O_2. In fact, for any successive jobs in O_2, if they are in the same order in O_1, then $O_1 = O_2$.

If, on M_2, there is a moment of inactivity between j_l and j_k, then we can move j_k to the left until that inactivity is eliminated. It can be seen in the following example (where inactivity is represented by a dash) that this does not introduce any violation of the constraints, since j_k always completes before j_l on machine M_1 and we have not altered the other jobs.

Once the inactivity is removed, we can swap jobs j_l and j_k on M_2 without creating any conflict. The following table represents these two steps.

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 -	

We then have a new order of jobs that is valid and whose total duration is less than or equal to the initial duration. In fact, it is equal because the initial order was supposed to be optimal. Thus, we have another solution that is also optimal. We can therefore replicate these steps to produce different optimal orders until $O_2 = O_1$. We then have an optimal solution with the desired property. □

The previous theorem indicates that there is no need to focus on solutions where the orders are different.

Theorem 1.2. *If O is a job order, where o_i is the i^{th} job in the order. If, for all $i < k$, $\min(t_1(o_i), t_2(o_k)) \leq \min(t_2(o_i), t_1(o_k))$, then O is optimal.*

Remark 1. The proof here is somewhat long and comes from the previously cited article. It is not necessary to master it or to know it by heart. It is presented here for your general knowledge (and because the original article is not easy to follow).

Proof. Let O^* be an optimal order. We can transition from O^* to O by performing only successive swaps of jobs. For example, to go from the order (j_2, j_3, j_1, j_4) to (j_1, j_3, j_2, j_4), we can go through the following orders:

- (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)

Indeed, if $O^* \neq O$, then there exist two jobs o_i and o_k such that $i < k$ and o_k is before o_i in the order O^*. There necessarily exist two such jobs that are successive in O^* (otherwise, $O^* = O$). We can therefore perform the swap and obtain a new order O_1. We can repeat this operation until we obtain O.

Let us denote $(O_0 = O^*, O_1, O_2, \ldots, O_p = O)$ as the different successive orders. We will show that the hypothesis made on the different jobs implies that, for all $q \in \{1, 2, \ldots, p\}$, O_q ends at the same time or later than O_{q+1}. By transitivity, we will have that O^* ends at the same time or later than O, and thus, by the optimality of O^*, we will conclude that O is also optimal.

It is important to note that, in O_q, we always exchange two jobs o_i and o_k such that $i < k$ and o_i is just after o_k in the order O_q, which will allow us to use the hypothesis of the lemma.

To simplify the reading of the following, we will denote the two orders O_q and O_{q+1} respectively as B and A. We denote a_i (respectively b_i) as the i^{th} job of the order A (resp. B). We want to show that A finishes before B. We assume that jobs k and $k+1$ have been swapped between A and B (thus $a_k = b_{k+1}$ and $b_k = a_{k+1}$). We also know that a_k is a job o_i and a_{k+1} is a job o_l of the order O such that $i < l$. We now check that $\min(t_1(a_k), t_2(a_{k+1})) \leq \min(t_2(a_k), t_1(a_{k+1}))$ by assumption.

Let's start by determining when order A finishes on machine M_2. We assume that, in A, each job is scheduled as early as possible: on machine M_1, each job is scheduled after the end of the previous one, and there is no inactivity time. On machine M_2, let x_i be the duration of the inactivity time just before the start of job a_i on M_2. In the following example, $x_1 = 5$, $x_3 = 1$, and $x_2 = x_4 = x_5 = 0$.

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

Thus, machine M_2 finishes at time $\sum_{i=1}^{n}(t_2(a_i)+x_i) = \sum_{i=1}^{n} t_2(a_i) + \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} t_2(j_i) + \sum_{i=1}^{n} x_i$.

Similarly, if in the solution resulting from order B, we denote y_i as the duration of the idle time just before the start of job b_i on M_2, then machine M_2 finishes at time $\sum_{i=1}^{n} t_2(j_i) + \sum_{i=1}^{n} y_i$.

We are left to show that $\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} y_i$ to prove the desired result. We can easily verify the following equalities:

$$x_1 = t_1(a_1)$$

$$x_2 = \max(t_1(a_1) + t_1(a_2) - t_2(a_1) - x_1, 0)$$

$$x_3 = \max(\sum_{l=1}^{3} t_1(a_l) - \sum_{l=1}^{2} t_2(a_l) - \sum_{l=1}^{2} x_l, 0)$$

$$\ldots$$

$$x_i = \max(\sum_{l=1}^{i} t_1(a_l) - \sum_{l=1}^{i-1} t_2(a_l) - \sum_{l=1}^{i-1} x_l, 0)$$

And then

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

Thus, we set

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

And then

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

Similarly in B we have the following inequalities

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

$$\sum_{l=1}^{i} y_l = \max_{i=1}^{n} L_i$$

Because $a_l = b_l$ if $l \notin \{k, k+1\}$, we have

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

Thus, if we prove that $\max(K_k, K_{k+1}) \leq \max(L_k, L_{k+1})$, then $\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} y_i$. Finally, by subtracting $\sum_{l=1}^{k+1} t_1(a_l) - \sum_{l=1}^{k-1} t_2(a_l)$ from both elements of the inequality, and remembering that $a_k = b_{k+1}$ and $b_k = a_{k+1}$, we obtain

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

This last inequality is precisely the hypothesis made about jobs a_k and a_{k+1}. Therefore, we have A ending before B, and as explained above, O ends before O^*, which means it is an optimal order. □

Theorem 1.3. *If O is the order returned by Johnson's algorithm, it satisfies the assumptions of Theorem 1.2.*

Proof. This proof is left as an exercise in the tutorial. □

This last theorem, coupled with the previous one, shows that Johnson's algorithm is optimal. Therefore, we have a polynomial algorithm.

Remark 2. These results are only valid with two machines. The algorithm is not intended to work with three machines. In general, adding machines or constraints (for example, *job* 4 *must be completed by tomorrow on machine* M_2 *because the client has paid for us to expedite job* 4) makes the problem more difficult. We do not know of a polynomial algorithm to solve the more general version of the problem. However, there are cases, such as with three machines, where Johnson's algorithm can be helpful. An example is provided in the exercises of the tutorial.

2 Two machines without precedence constraint

In this section, no order is imposed on the machines. Thus, each job must go through each of the machines but can do so in a different order than the other jobs. We will show that, in this case, there is also a polynomial-time algorithm to solve the problem. Let's first consider the example from the previous section:

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

A solution would be the following

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

It is impossible to get a better solution because the machines have no inactivity. It is worth noting that, unlike the first problem, a solution is not simply a sequence of jobs. For each job, it is necessary to specify the time at which it starts on M_1 and the same on M_2. Given a solution O, we denote o_{ik} as the start time of job j_i on M_k.

In the following, we set

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

If $T_1 < T_2$, then we preprocess the instance by adding dummy jobs. Each dummy job satisfies $t_1(k) = 1$ and $t_2(k) = 0$. By adding $T_2 - T_1$ dummy jobs, we have $T_1 = T_2 = T$. We act similarly if $T_2 < T_1$.

We now show the following theorem:

Theorem 2.1. *One can build in polynomial time an optimal solution with duration* $\max(M, T)$*.*

Lemma 2.1. *The duration of any solution is at least* $\max(M, T)$*.*

Proof. Let's assume we have a feasible schedule O. The finishing time t_f of the order O is after the completion of the execution of each machine. At best, each machine must work for a duration T. Therefore, $T \leq t_f$. Moreover, each job must be completed, and since a job cannot be executed on both machines at the same time, the order O finishes after a duration of $t_1(i) + t_2(i)$ for every job i. Thus, $M \leq t_f$. □

Lemma 2.2. *If $M \geq T$, we can construct in polynomial time an optimal solution whose duration is M.*

Proof. Let i such that $t_1(i) + t_2(i) = M$. We can construct a solution of duration M in this way, where A corresponds to all the jobs except 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	-	-

□

Lemma 2.3. *If $T \geq M$ and if we have 3 jobs, we can construct an optimal solution in constant time whose duration is T.*

Remark 3. Building an optimal solution in constant time is not difficult here; it just involves enumerating all possible solutions. However, proving that we can construct a solution of duration T for any instance is a bit more complicated.

Proof. We will show that there is always a solution without any machine being inactive.

First, let's assume that $t_1(1) \geq t_2(2)$. If $t_2(1) \geq t_1(3)$ then there exists a solution of duration T with the following solution:

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

If on the other hand $t_2(1) < t_1(3)$, the following solution is achievable and has a duration of 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

This solution is indeed feasible. In fact, job 1 does not run on both machines at the same time because $t_2(1) < t_1(3)$. Similarly, the same holds for job 2 because $t_1(3) + t_1(1) > t_2(2) + t_2(1)$. Finally, there is no conflict for job 3 because
$t_1(3) \leq M - t_2(3) \leq T - t_2(3) = t_2(1) + t_2(2)$.

Let's assume now that $t_1(1) < t_2(2)$. We will first show that we can reduce the case to one where $t_1(1) \geq t_2(2)$. We consider the following new instance:

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

We notice that this is the same instance as the original instance where we exchanged M_1 and M_2 and where we exchanged j_1 and j_2. Therefore, both have exactly the same optimal solutions. In this new instance, we denote $t'_k(i')$ as the time of job j'_i on machine M'_k. We observe that $t'_1(1') \geq t'_2(2')$. We can apply, in this instance, one of the two solutions of duration T proposed at the beginning of the proof depending on whether $t'_2(1') \geq t'_1(3')$ or not. □

Lemma 2.4. *If $T \geq M$ and if we have more than 3 jobs, we can construct an optimal solution in constant time whose duration is T.*

Proof. We can always simplify the instance to get only 3 left jobs:

Create 3 empty sets J_1, J_2, J_3
for k from 1 to n **do**
  for p from 1 to 3 **do**
    if $\sum_{i \in J_p \cup \{k\}} t_1(i) + t_2(i) \leq T_1$ **then**
      Add j_k to J_p
      Continue the outer loop

In the example at the beginning of the section, we would have $J_1 = \{j_1, j_2\}$, $J_2 = \{j_3, j_4\}$ and $J_3 = \{j_5\}$.

If this algorithm assigns all jobs to J_1, J_2, or J_3, then we obtain a new instance, a super-instance, with 3 super-jobs. We define $t_k(J_p) = \sum_{j_i \in J_p} t_k(i)$. In the example, we would then have

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

With lemma 2.3, we can find a solution in time T for this 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

In this solution of the super-instance, each super-job J_k is not executed simultaneously on both machines. If we replace this super-job with the jobs contained in the original instance on both machines, then these jobs cannot be on the same machine at the same time either. In the example, we obtain the following result.

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

It remains to show that we correctly place all the jobs in J_1, J_2, J_3 and we obtain the result. Suppose the contrary, let j_k be the first job that is not placed in J_1, J_2, J_3. Then, for all $p \in [\![1; 3]\!]$,

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

If we sum the 3 inequalities for the 3 values of p

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

However, each job is in at most one set J_p and j_k is in no such set.

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

By inequalities (1) and (2)

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

By inequalities (2) and (3)

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

Finally, for every $p \neq p'$ and each job j_l placed in $J_{p'}$, $\sum_{i\in J_p} t_1(i)+t_2(i)+t_1(l)+t_2(l) > T$. Indeed, otherwise, j_l would have been placed in J_p instead of $J_{p'}$. Thus, particularly

$$\sum_{i\in J_1} t_1(i)+t_2(i)+\sum_{i\in J_2} t_1(i)+t_2(i) > T$$
$$\sum_{i\in J_1} t_1(i)+t_2(i)+\sum_{i\in J_3} t_1(i)+t_2(i) > T$$
$$\sum_{i\in J_2} t_1(i)+t_2(i)+\sum_{i\in J_3} t_1(i)+t_2(i) > T$$
$$2\cdot\sum_{p=1}^{3}\sum_{i\in J_p} t_1(i)+t_2(i) > 3T$$

We thus have a contradiction with inequality 4, so all the jobs are in J_1, J_2, and J_3. □

Proof of the theorem. Using lemmas 2.1, 2.2, and 2.4, we can prove that there always exists a solution of duration $\max(M,T)$ and that it can be constructed in polynomial time.